2014

An XML-based system for management and query of video databases with user identifiable and annotated scenes

Zheng Li
Iowa State University

An XML-based system for management and query of video databases with user identifiable and annotated scenes

by

Zheng Li

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Shashi K. Gadia, Major Professor
Leslie Miller
Manimaran Govindarasu

Iowa State University

Ames, Iowa

2014

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

ABSTRACT

Video is one type of document. However, unlike traditional documents and databases that are searchable, video is not. At the same time, video is typically lengthy and contains large amount of useful and/or useless information. Therefore, it is desired to make the video manageable and searchable.

Metadata and scene annotation are two keys to resolve this challenge. In this thesis we designed and implemented an XML-based Movie Database Manager (MDM) as a proof of concept. The video information is well organized in its natural way and stored in one single XML file. Videos are divided into user defined scenes with annotations for quick and exact query with build-in query engine. Users can also create, edit, and delete scenes annotations without physically split them in the video files. MDM integrates the query builder to help users run pre-defined queries or create and maintain their own ones. These queries can be stored in the same XML file. A full functional video player is also implemented to help user play their query results, create / edit / delete scene annotations.

CHAPTER 1. INTRODUCTION

Videos are wildly used to record information nowadays. Videos can be considered as documents, but with great difference that unlike most of the documents which are searchable, video contents are not. Videos are long as well. This intrinsic feature of video raises the challenge: how can one keep track of personal or public collection of videos?

Videos are also complicated. Two distinct but essential concepts are containers and codecs. Container describes the structure of the video, basically means where the various pieces are stored, how they are interleaved, and which codecs are used by which pieces. The container can also include synchronization information such as audio track, subtitles, and metadata [1]. The container is normally reflected by its file extension of the video file such as .asf, .avi, .mp4, etc. Codec is a way of encoding audio or video into a stream of bytes. This process is usually lossy [2]. Optimized method can compress the video and audio to a relatively small size while at the same time maintain the quality. However, more resources are required to decode and playback the video. Based on various purposes, codecs can be classified as:

- Video capture, which includes H.264/MPEG-4 AVC, MJPEG, and DV/HDV etc.
- Disc-based delivery, which includes MPEG-2, Microsoft VC-1 etc.
- Streaming for Web, which includes WMV, H.264/MPEG-4 etc.

Scene is another important aspect of video. Commercial movies, public speeches, and homemade recordings are all composed of scenes. Scene is part of video and generally is considered as some actions occurred in a continuous time in some specific location [3]. However, different definitions might be applied so that a scene might not be limited in single location or continuous time but on specific subject. One video or movie might

contain hundreds to thousands scenes based on different scenarios or features. Therefore another question arises: how can people find the scenes that satisfy their interests without go through the whole video?

There are a number of commercial video management programs available in the market such as Movie Collector, Movie Label, and My Movies etc. All these management programs have similar functionalities and mainly focus on the commercial movies. As the web-streaming and homemade digital videos increase rapidly, a methodical way to find the desired information in video becomes important. However, none of these products provide the ability to looking for specific scene in the whole movie and hence make them less capable to solve the challenge.

Another type of software is scene splitter. This type of software tends to split the big video files into smaller pieces. There are a couple of different algorithms to achieve this goal by sudden optical change, by date of shooting, or by black frame between scenes. None of these algorithms works perfect and split results are barely consistent. In addition, these programs break the video file integrity and create hundreds to thousands small files for each video and hence make video management even more difficult.

A user-friendly video management system should enable users to identify and refer to any scenes they wanted or interested. To address the challenges, scene identification and their annotation with metadata are the key. The question is how to store the information. Relational database requires the information to be dismantled and atomized. This breaks the natural structure of the information and scatters the information into pieces [4]. The information about videos might be incomplete, missing. For example, the video might not

have any audios, and hence the metadata won't contain this part of information. In addition, video scenes can contain sub-scenes, which are of interest on their own right, giving rise to a nested scene hierarchy. All these make the relational database inappropriate for this circumstance.

Here we report an XML-based video management system Movie Database Manager (MDM), which gives user ability to review video information, manage scenes, query the video information, and movie/scene playback. Although video is normally considered as a broader definition, we use video and movie interchangeable from now on. The functionalities of this system are implemented with DOM (Document Object Model), XPath (XML Path Language), and XQuery. DOM is used as standard accessing and manipulating tools for the XML file [5]. XPath provides the ability to navigate the XML file and select nodes by different criteria [6]. XQuery is a query language similar to SQL but for xml documents. It is used to query and transform a collection of data, which can be structured or unstructured, to fulfill more complicated query tasks [7].

Currently the system is still a proof of conception, so several assumptions have been made. Frist of all, we assume that the video database is comprehensive, which means it contains different type of videos, different file format, etc. However, current implementation use 250 movie trailers as video source and all files are in .mp4 format. Secondly, we assume that the scene annotation is accurate while current setting randomly generates scenes from video files and random annotations have been generated for the scene. Annotating of video scenes is a very time consuming and tedious process and not possible within the human resource limits that we have at hand. However, it is to be noted that it is perhaps a fraction of the time it takes to produce actual videos. It is hoped

that some industry practices would evolve to address this issue. This would also help users to customize their own user-defined scenes.

The rest of the document is organized as follows. In Section 2, the data storage model has been described. How we structure the xml file to store different types of information has been discussed. Section 3 is focus on the program implementation and user interface introduction. Various functions of MDM are discussed. Section 4 briefly investigates the previous and similar work, and Section 5 concludes the project description and discusses some future thoughts.

Unlike relational database, XML does not have to comply with the data uniformity. Simple object such as atomic values or very complex objects such as nested or hierarchy structure can be saved in the same XML instance [4]. Most importantly, the data structure in the XML file can offer natural ways of storing object. We end this section with an informal example.

When creating scene annotations, user could encounter different situations. The scenes could be continuous, scattered, overlapping, or nested. In order to give an example, we hypothesize that a database {M} consisting of a single video M with the title "Fun" is given. Figure 1 shows a scenario consisting of multiple steps where these situations occur. In each step the user interaction is shown on the left and the resulting internal representation is shown on the right. The user has a context ranging from the whole video to a scene that has been previously insolated. In the given context the user identifies scenes and associates the desired annotations with them. The result is a hierarchical partitioning of the video. Sometimes in some part of the hierarchy the level is

incremented by one. Logically, a scene identified with particular annotations will consist of a sequence of finitely many disjoint clips. In this document the term clip stands for an arbitrary sequence of frames without a break. In the example, annotations “eating”, “playing”, and “singing” denoted in short as “E”, “P”, and “S” are used by the user. Whereas a clip is analogous to an interval on real line, a scene is analogous to a finite union of intervals introduced in [8,9].

Continuous or scattered scene annotations are kept as they are, as shown in Figure 1(a). While for every two overlapped scene annotations, they will be split into three new scene annotations. Non-overlapped parts will kept their original annotations, but their start or end position will be adjusted to the edge of overlapping. A new scene will be created for overlapped part and it inherits both scenes’ annotation. For example, in Figure 1(b), the new scene has annotation of both “E” and “P”. A sub-scene must have its start and end between its parent scene. When a sub-scene has been created, its parent physically keeps its original annotations but should have all annotations from its sub-scenes.

The main goal of this thesis is to make the database of videos queriable. Although the context of a query consists of a database containing multiple videos here we offer some examples assuming that our database {M} consists of a single movie M. The video level metadata (e.g. movie title and directory) and user defined annotations form the main ingredients for queries. Queries will be expressed in XQuery language enabling a user to ask sophisticated queries with the objective of narrowing the scenes as mush as possible, here we confine to simple examples with simple informal syntax.

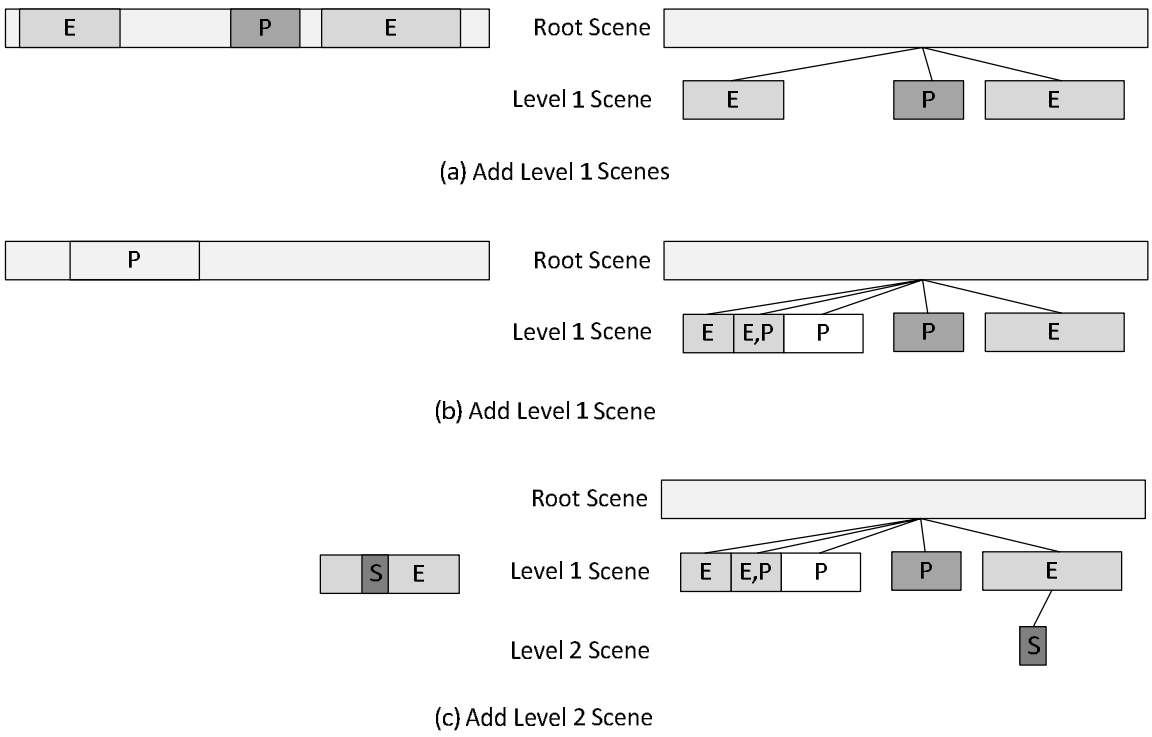

Figure 1. Different situations of scene annotation (P represents "playing", E represents "eating", and S represents "Singing").

Query {"Fun"} will result in the whole video. The query {"Misery"} will return nothing. The query {"S"} will return a scene consisting of a single clip from level 2. Query {"L"} will return nothing, but if there were other videos in the database where annotation "L" appears all corresponding clips from all videos will be displayed. Query {"E"} will return all clips where annotation "E" appears. The query {"E", "P"} will return all clips where "E", or "P", or both appear. Note that {"E"} will absorbs {"S"} completely. In the current state of the database the results of queries {"E"} and {"E", "S"} are indistinguishable.

CHAPTER 2. DESIGN OF STORAGE MODEL

The structure of the XML file has been described in corresponding schema file which was developed with commercial software XMLSpy. The contents of the whole XML file have been categorized into four major parts:

- Video Information
- Person Information
- Reward Information
- Query Information

Each part of the information is a complex type so that we can treat them as a whole. The overall document structure is shown in Figure. 2.

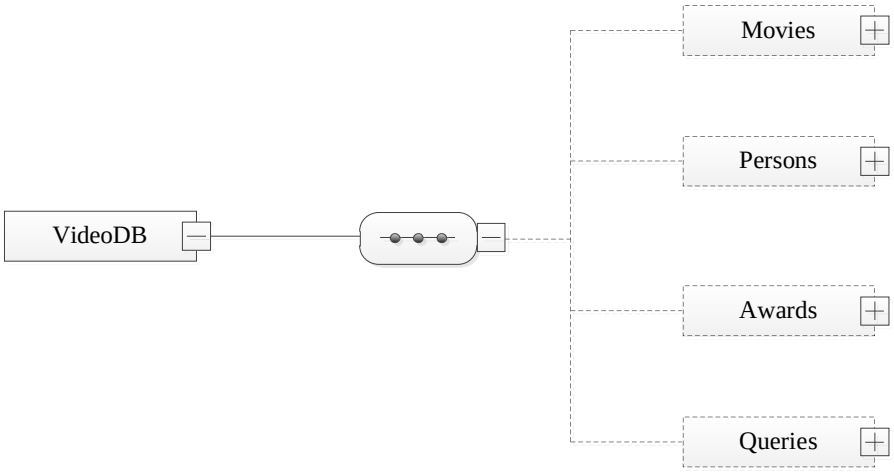

Figure 2. Overall data structure for XML-based video database

Solid line indicates that it is a required component, thus the root element *VideoDB* is required. Elements shown in dotted line are optional. We expect the video database to be a comprehensive one, but we cannot rule out the possibilities that in specific videos only part of the information is available. This approach leads to the great flexibility of using XML as information storage.

As clearly indicated by their names, these four types, realized as complex types following XML terminology, are used to separate information based on their nature and also to normalize the data to avoid unnecessary duplications. Although the order of these parts is not essentially important, we add the constraint on the order of these four elements to make the structure more organized.

2.1 Movie Type

This is the most complicated type as most of the information resides in this element. As shown in Figure. 3, inside the *Movie* type, the information has been divided again into three types based on its nature.

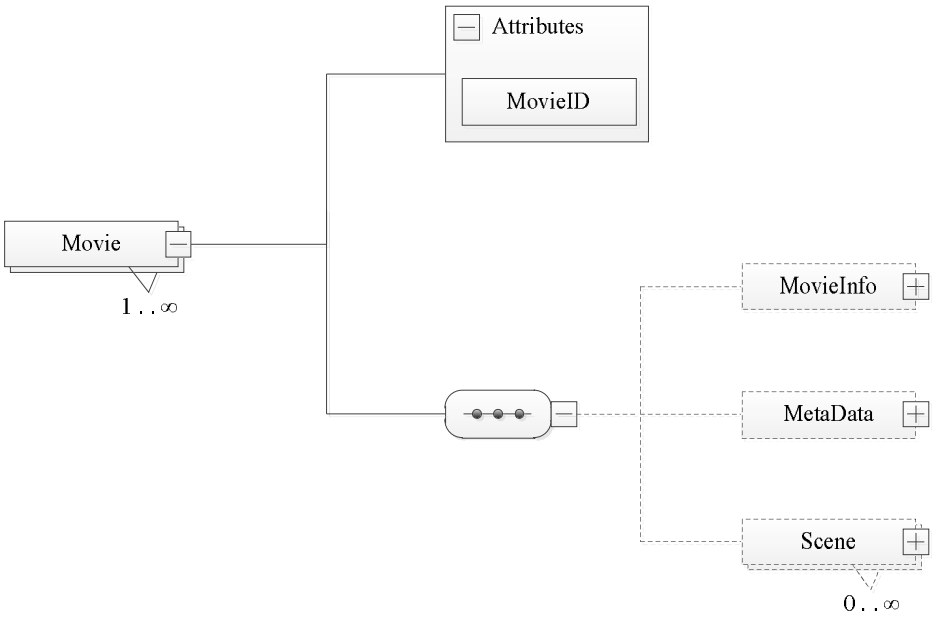

Figure 3. Data structure of complex type *Movies*

Each movie has a unique movie id as attribute, which is the primary key to distinguish the movie. This has been designed as a required attribute. Industrial information about the video such as title, release date, budget, rating, overview, actors etc.

is considered as movie info. We try to include as much information as possible so that there will be enough information for user to narrow down their searches based upon their interests. It is useful to note that most of the data inside *MovieInfo* are atomic, some of them are not. For example, Actors element has an attribute *PersonID* which points to *Person* element so that users can find the actor's personal information. In addition, the actor has a *Character* element as this is part of the movie information instead of personal information, as indicated in Figure. 4.

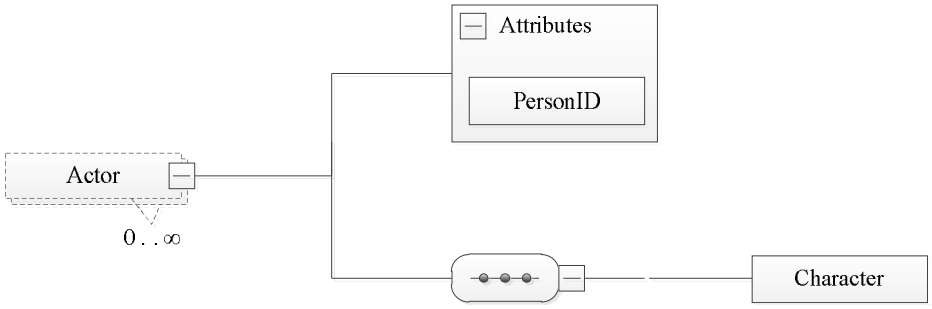

Figure 4. Date structure for *Actor*

MetaData is the video's technical information such as *FileName, BitRate, Codec, Resolution, AudioTrack* etc. Again, most of these elements are atomic values except for the *AudioTrack*. A video can have multiple audio tracks with different purpose or languages. In addition, audio track can has its own technical information including sample rate, language etc. Therefore, the *AudioTrack* is considered as another complex type.

The last but the most important child of *MovieType* is the *Scene* element. A scene is normally considered as an action in a single location and continuously. While there could exist other classifications. For example, a video might be split by topic of interest. Here, we do not limit how the scene is split. Users can split the video in any way they want

based on their own interests. To better mimic the real world example, we designed the scene structure as a hierarchy structure, as indicated in Figure 5.

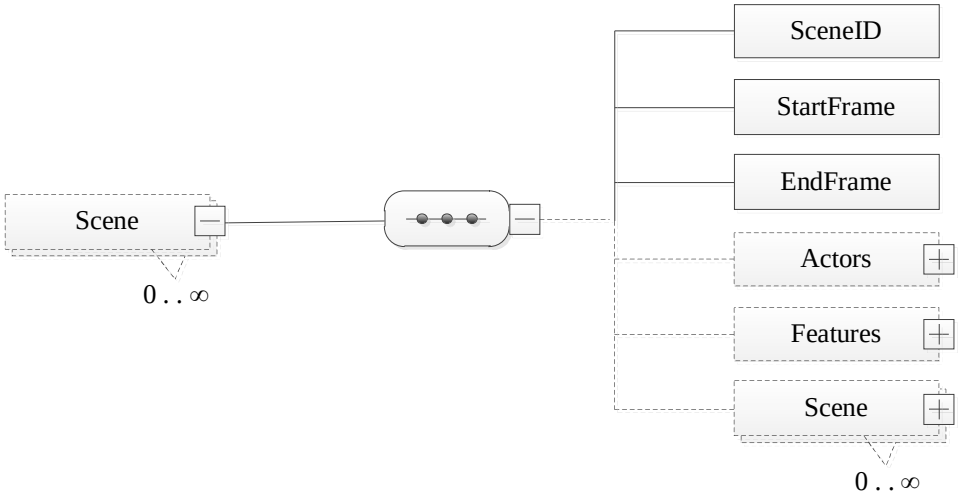

Figure 5. The hierarchy of scene structure

Three elements are defined as required, which are *SceneID*, *StartFrame,* and *EndFrame*. *SceneID* is the identifier of scene, but is only unique within a movie. The root scene of a video is considered as the whole video and has an id of "1". While the second layer of scenes have ids like "1.x", and so on. User can define as many as layer they want. The *StartFrame* and *EndFrame* define the boundary of a scene in the video. The rest of the elements are optional, as shown in Figure. 5. In order to remove information duplications, the *Actor* and *Feature* elements are only physically appear in leaf scene. Non-leaf scene does not have these two element filled, but shows a collect of actors and features from all its descendant scenes in the user interface.

2.2 Person Type

Person is a relatively simple complex type. It is used to store personal information for actors, directors, speakers etc. Similar as *Movie* element, a *PersonID* attribute is used as

unique identifier for *Person* element and it is an IDREF type so that *Movie* element can get personal information by *PersonID*. Child elements include *Name, DateOfBirth, Biography* etc.

One important element is *Award* element in *Person*, as indicated in following schema. This element contains the information about the awards the person received. It also has a *MovieID* attribute to point to the movie or video the person was rewarded for.

2.3 Award

The third element of *VideoDB* is *Award*. This is also a pretty simple complex type, which record details of all types of awards. A unique *AwardID* attribute is used as identifier. Award name, country and description are the only elements.

2.4 Query

A comprehensive video management system should have the capability to run user define queries as pre-defined categories might not be sufficient to narrow down the video searching. And users might want to provide their own criteria for video screening. The MDM integrates the XQuery so that users can write their own queries and also save the queries for later use or future development in the *Query* element.

Query element has a unique attribute *QueryID* just like *MovieID*, *PersonID*, and *AwardID*. The *Title* element stores a short name for the query and *Description* element describes the details of the query. The *String* element stores the query itself.

```
<xs:element name="Award" maxOccurs="unbounded">
   <xs:complexType>
      <xs:attribute name="AwardID" type="xs:IDREF"/>
      <xs:attribute name="Year"/>
      <xs:attribute name="Role"/>
      <xs:attribute name="MovieID" type="xs:IDREF"/>
   </xs:complexType>
</xs:element>
```

(a) XML schema for *Award* Element in *Person* Element

```
<xs:complexType name="AwardType">
   <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Country" type="xs:string"/>
      <xs:element name="Description" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="AwardID" type="xs:ID" use="required"/>
</xs:complexType>
```

(b) XML schema for complex type *AwardType*

```
<xs:complexType name="QueryType">
   <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
      <xs:element name="Description" type="xs:string"/>
      <xs:element name="String" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="QueryID" type="xs:ID" use="required"/>
</xs:complexType>
```

(c) XML schema for complex type *QueryType*

Figure 6. XML Schemas

2.5 Generate Testing Video Database

As mentioned in our assumptions, the work presented here is a proof of concept. Thus we create a sample XML video database to test our implementation. Considering the large amount of information, it is not realistic to manually input all the information. Thus the database was created step by step as described below.

The movie list is obtained from IMDB (Internet Movie Database), the most authoritative source for movies, tvs. It is not possible to include all movie information, so we choose the Top 250 movies as our movie list. An open source java project

themoviedbapi is then used to fetch *MovieInfo* and *Person* from another popular movie information source TMDB (The Movie Database) by movie titles. The information is inserted into the video database at right position with duplicated information removed. The *MovieID* and *PersonID* are generated and mapped automatically during this process.

The movie trailers are used as our video file sources. Xuggler, a third-party java package handling multimedia, is used to extract the metadata from the video files. The metadata information is then appended as *MetaData* element right after the *MovieInfo* element for each movie.

Scene annotations are generated randomly. Although the *Scene* element has been designed as a hierarchy structure, we only create two levels in the beginning: a root scene and random number of child scenes. For each movie, the number of scenes is randomly generated between 1 and 10. After the number of scenes has been determined, the percentage of scene length over movie length is randomized for each scene and their total is normalized to 100% and then followed by calculating the start frame and end frame. For meta data associated with the scene, actors are also randomly picked from the actors in *MovieInfo* element.

Scene feature is critical as it is one of the keys for users to find scene precisely using some logical (or physical) criteria. Again, randomization is used to populate scene features. We defined four types of features (denoted as *FeatureName* in the XML file) and various feature value (denoted as *FeatureValue* in the XML file), as indicated in Figure 7. Other types of scene feature can be added into the database through the user interface and will be discussed in next chapter. This makes searches highly customizable.

Moods	Locations	Times	Actions
Happy	Mountain	Morning	Eating
Sad	Sea	Noon	Sleeping
Angry	River	Evening	Talking
Anxious	Sky	Night	Playing
Bored	OutSpace	Dawn	Singing
Cheerful	House		Driving
Disappointed	School		Fighting
Frustrated	Park		Running
Lonely	Supermarket		
Peaceful	Street		

Figure 7. List of features and corresponding values

Award information was populated manually from IMDB as no suitable API available to retrieve the award information. Only part of the movies and persons have their award information filled.

The *Query* part is intentionally left empty at the beginning as it is programmatically added when users enter their own queries.

CHAPTER 3. USER INTERFACE DESIGN AND IMPLEMENTATION

A Java Swing based interface has been developed for user interaction with the XML-based video database. The overall interface is shown in Figure. 8. There are four tabs on top of the main screen providing four main functionalities which are described in details in following sections.

In addition to the standard Java SDK, a couple of third-party packages have been used to facilitate the implementation including:

- Saxon: used as our XQuery engine
- Vlcj: used for video playback
- Xuggler: used for video metadata extraction and scene materialization
- Themoviedbapi: used for commercial movie information retrieval from themoviedb.org

3.1 Movie Information Display

Movie Information tab is used to display video information. This panel has been divided into three sub-panels. Data have been read from XML file and re-organized for better information demonstration. Thus, the information layout on the screen is not necessarily identical to its physical structure in the XML file.

This is the default screen showing when program starts. The video list table displays all videos in the XML file and right bottom panel shows information of first video in the video list table by default.

Left panel is a tree structure that lists some basic categories that users can narrow down the video display. Currently *Country*, *Genre*, *Language*, *Release Year*, and *Studio* have been added into the list and potentially more criteria could be added. Each time

when user click on the root node (*Movies*), the movie list table on top right will be updated and show all movies in the file. Clicking on non-leaf node won't do anything. When a user clicks on any leaf-node under each category, videos fall into selected category will be listed in video list table. At the same time, right bottom panel changes to current first video in the video list table. For example, if user click the node "The United States" under *Country* node (not shown in Figure. 8), then all videos produced in the United States are listed. The details of the first movie in the table will be display in right bottom panel.

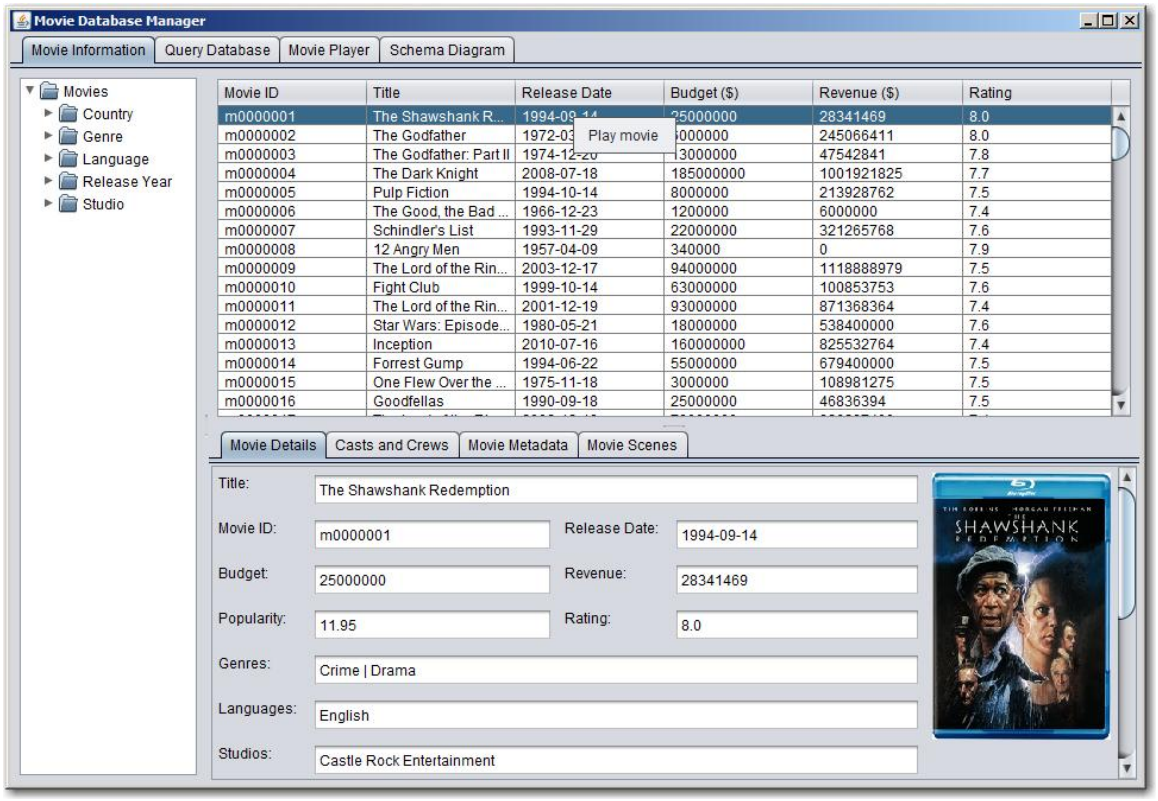

Figure 8. Overview of Movie Database Manager user interface

Right panel is split into two sub-panels. Top one is used to display filtered video results. As we indicated earlier, *Movie* element contains most of the information. A single

table won't be able to display them all in a manageable way, so only a couple of fields are included in the table such as id, title, etc. The information gives user a good overview of the video. Right click on the selected movie will pop up a *Play Movie* menu so that users can play the movie in *Movie Player* tab, as indicated in Figure. 8.

Details of the video are shown in right bottom panel. This tabbed panel has four panes including *Movie Details, Casts and Crews, Movie Metadata,* and *Movie Scenes*. The panel titles are self-explained. *Movie Details* displays industrial information of the video and *Casts and Crews* display all persons related to video such as actors, directors, writers, etc. These two tabs map to the *MovieInfo* element in the XML file. The third tab is metadata of the video including file name, file location, codec, audios, etc. As clearly indicated, it maps to the *MetaData* element in the XML file. The final part maps the *Scene* element in the XML file, as indicated in Figure. 9. However, unlike the other three tabs, it is not just displaying the scene information, but also is editable.

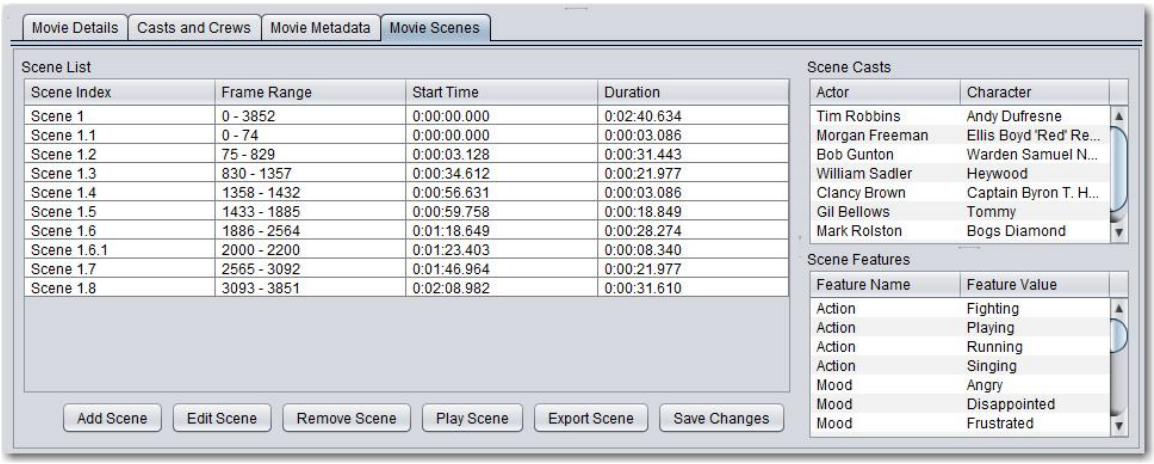

Figure 9. *Scene* tab in *Movie* Information

The *Scene* tab is split again into different zones. Left panel list all scenes in the selected video with their frame range, start time, and duration. As an example, Figure. 9

shows all scenes in movie “The Shawshank Redemption”. When users click on any scene, the corresponding actors and features are shown in the right panel. As we mentioned in Chapter 2, a non-leaf scene shows all actors and features from its sub-scenes. Thus XML is very conducive to such organization.

A number of actions can be performed by users regarding scenes. User can add sub-scenes to the existing scene. When user clicks *Add Scene* button, a new dialog pops up, as shown in Figure. 10.

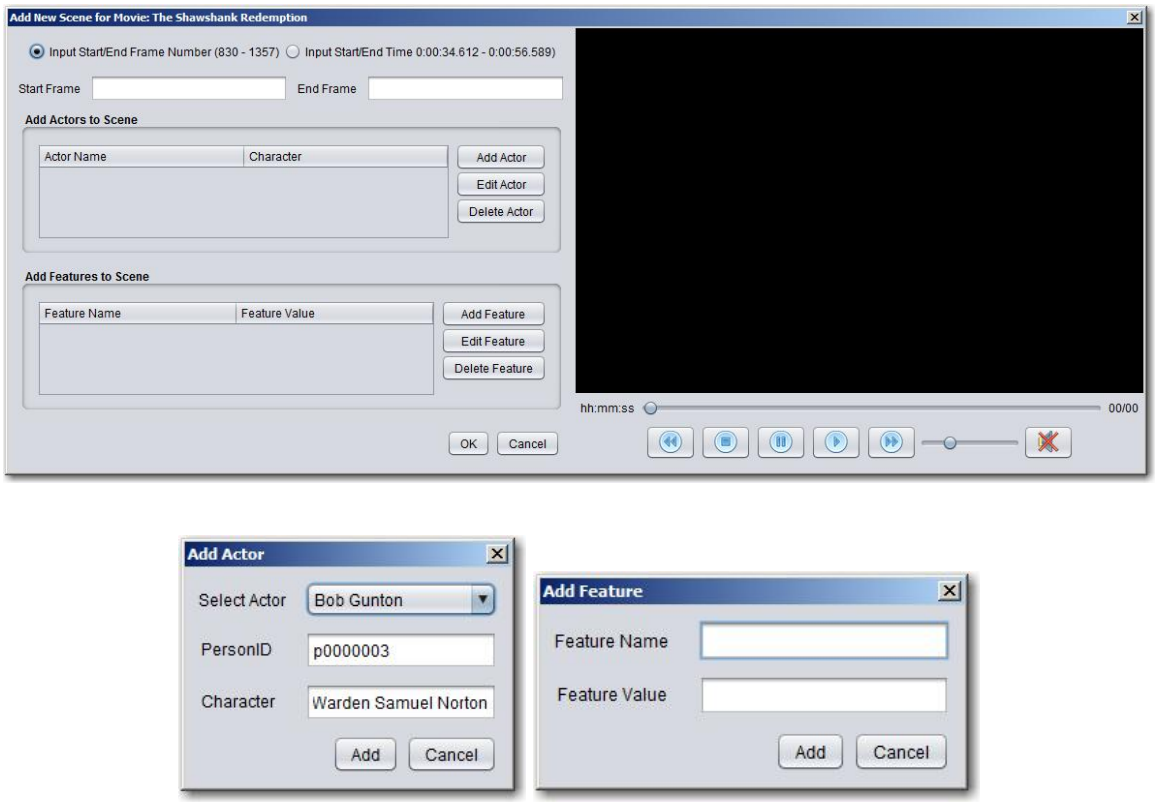

Figure 10. Add new scene dialog

When this dialog shows up, it automatically applies the start frame and end frame of current scene to limit the range of sub-scene. The *Start Frame* and *End Frame* are required fields as we discussed in our storage model chapter. Scene id will be

automatically assigned. If we adding a scene inside scene 1.6 (scene 1.6 has no sub-scene right now), then a scene id 1.6.1 will be assigned to the new scene. Users can also add optional element actors and features for the new scene. When adding actors into sub scene, we would like to limit the actors to be a subset of actors in parent scene. Thus when users click *Add Actor* button, an *Add Actor* dialog shows up as indicated in Figure. 10. The drop-down list lists all actors have played a role in parent scene. Each time when an actor is selected, the corresponding person id and character is shown in the text fields. Add feature dialog is similar to add actor. However, we don't limit the features. User can add whatever features they believe can describe the scene well they are adding. This is at the core of the ability of MDM to customization by users. A player is used to help user to better annotate the scenes. This player is similar to the player in Movie Player tab, but with much simpler controls.

There are a number of different situations when adding new scenes. If adding a scene to a leaf scene, the scene will be directly listed under the parent scene. The original actors and features of the parent scene are cleared as it is not a leaf scene any more. (As we indicated in our storage model design, non-leaf scenes do not maintain their own actors and features.) If adding a scene to a non-leaf scene, any child scenes falls in the range of new scene will be deleted. Any child scenes that intersect with new scene will adjust their start frame and end frame to give space to the new scene. If the new scene falls into the range of a child scene, the child scene will be split to accommodate the new scene.

Besides the add scene function, users can also edit selected scene. The *Edit Scene* dialog is similar as *Add Scene* dialog except that the actor table and feature table have been prefilled. We will not discuss it in detail here.

Delete scene will remove current scene from the hierarchy structure. It also has a number of different scenarios. If user deletes a leaf scene, it will be directly deleted. If a non-leaf scene being deleted, all its sub-scenes are also deleted at the same time. If users try to delete a root scene, then all non-root scenes are deleted. At the same time, the root scene obtain full actor list from *MovieInfo* element with an empty feature list.

Single scene can also be watched in the *Movie Player* tab by click the *Play Scene* button. Difference between play movie and play scene is user does not have to watch the video from the beginning to end. Player plays scene from their start frame. In such manner, users can quickly jump to the part they are interested.

It is beneficial that user can save or share scenes they are interested in. Therefore, we implemented another important feature so that user can materialize the selected scene to a real video file. A message windows pops up when the materialization finished, as indicated in Figure. 11.

Figure 11. A popup windows indicating the end of scene materialization.

All changes occur in the memory until users press the *Save Change* button to persist the changes. If the user forgets to save changes and tries to terminate the program, a popup window will remind user to save the changes, as shown in Figure 12. In such way, user won't lost their work by inadvertently closing the program.

Figure 12. Persist unsaved changes.

3.2 Query the Database

The categories in the *Movie Information* panel are quite broad and only work on the video level. To obtain more specific contents, users have to run pre-defined queries or build their own queries. Thus, MDM provides a query panel to fulfill this requirement.

The query interface has several main fields. On the left side is a query list. All queries stored in the XML file are shown here. As the query description could be long, it is separated from the table and displayed in the textbox below the table. In the middle of the query panel is the query builder and right side is the query result. To assist users in building the queries, a console is added to display error and success messages.

Query id is an ineditable field and is generated automatically. When users click on a query in the table, the query id, query name, description, and actual query show in their own textbox, as indicated in Figure. 13. When running the query, the query is parsed by the integrated Saxon XQuery engine. If it is a valid query, a success message will shown in the console and results outputs to the result windows. The program then parses the query output again. If it contains both *MovieID* and *SceneID*, the *Play* button is enabled. Thus, if users click the *Play* button, the list of scenes extracted from the query results will be sent to the *Movie Player* panel, which will be discussed in next section. The query result can also be export as text file, html file, or xml file.

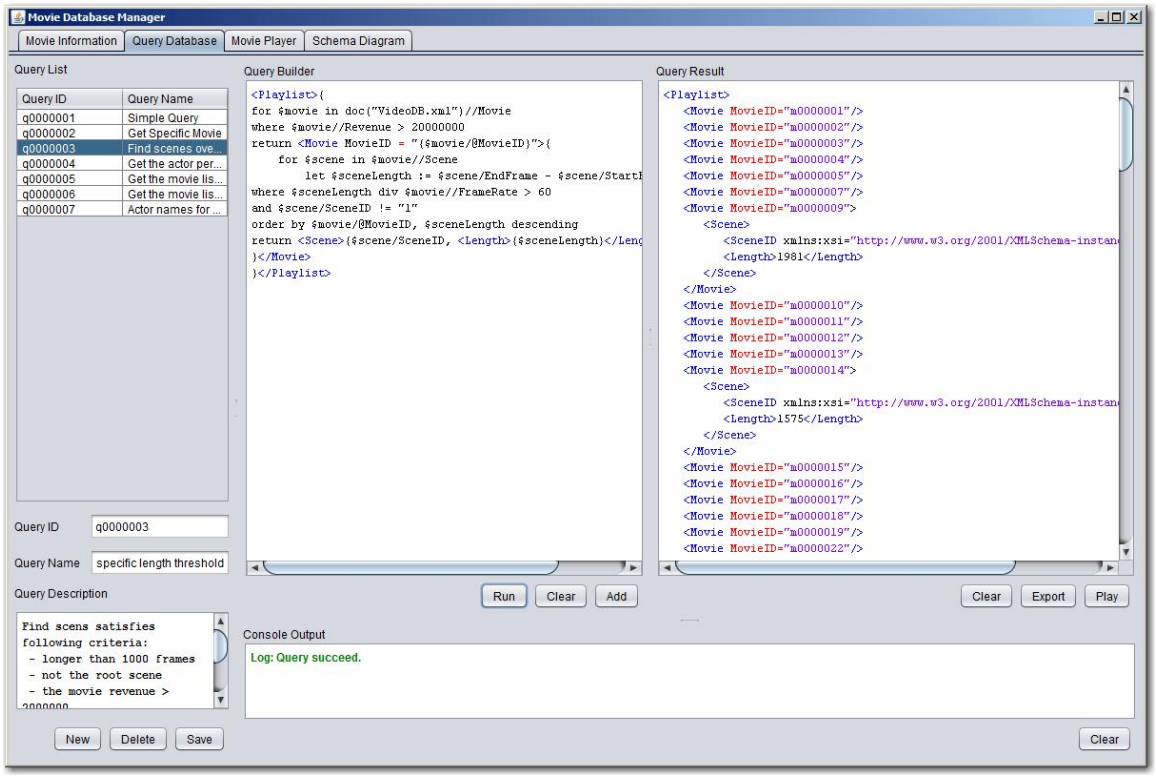

Figure 13. User query interface

Users can either create a query from scratch or modify an exist query to meet their requirement. When a new query is created by clicking the *New* button, a query id is assigned automatically. Users can then enter the query name, query description and start to build query in query builder. After its creation, the query can be added into the query list. If working on an existing query, adding it to the list will overwrite the original one. Again, all these actions are still recorded in the memory until users click on the *Save* button to persist them. Therefore, users can work on several queries at the same time and save them as a batch. Here too, if user forgets to save query changes and tries to close the program, the “Save Changes” windows pops up to remind user that there are unsaved chnages.

3.2.1 Query Examples

The query does not tie to the scene search. Any information in the database could be queried. For example, following is an simle query to generate a list of movies that have rating at least 8. This query only returns movie information.

```
<Movies>{
   for $movie in doc("VideoDB.xml")//Movie
   where $movie/MovieInfo/Rating >= 8
   order by $movie/MovieInfo/Rating descending
   return <MovieID>{
      $movie/@MovieID, $movie/MovieInfo/Title, $movie/MovieInfo/Rating
   }</MovieID>
}</Movies>
```

Another example that looks for actor information is shown below

```
List the actor names and their characters played in movie “The Godfather”
```

```
<Actors> {
for $e in doc("VideoDB.xml")//Movie
for $a in $e/MovieInfo/Actors/Actor
for $p in doc("VideoDB.xml")//Person
where $e/MovieInfo/Title = 'The Godfather'
and $a/@PersonID = $p/@PersonID
return <Actor> {
       <Name>{$p/Name/text()}</Name>,
       <Character>{$a/Character/text()}</Character>
   }</Actor>
} </Actors>
```

Part of the results are shown below. The result does not contain any movie or scene information, thus the *Play* button will not be activated.

```
<Actors>
  <Actor>
    <Name>Marlon Brando</Name>
    <Character>Don Vito Corleone</Character>
  </Actor>
  <Actor>
    <Name>Al Pacino</Name>
    <Character>Michael Corleone</Character>
  </Actor>
  ...
  <Actor>
    <Name>Gabriele Torrei</Name>
    <Character>Enzo, the baker</Character>
  </Actor>
</Actors>
```

Below are more queries that are not scene related by are also very helpful to retrieve information from the database:

- Extract scenes with famous dialogs
- List actors who have acted in 25 or more movies.
- List all queries currently in the database

Besides for general information query, the most valuable and also the core of this application is to query the scene information. Careful designed query can return user very accurate results.

Extract non-root scenes that are at least 60 seconds long and the corresponding movie has revenue greater than $2,000,000. The result is ordered by *MovieID* first and then by the length of the scenes.

```
<Playlist>{
   for $movie in doc("VideoDB.xml")//Movie
   where $movie//Revenue > 20000000
   return <Movie MovieID = "{$movie/@MovieID}">{
      for $scene in $movie//Scene
         let $sceneLength := $scene/EndFrame - $scene/StartFrame
   where $sceneLength div $movie/MetaData/FrameRate > 60
   and $scene/SceneID != "1"
   order by $movie/@MovieID, $sceneLength descending
   return <Scene>{$scene/SceneID, <Length>{$sceneLength}</Length>}</Scene>
   }</Movie>
}</Playlist>
```

The result lists the information satisfies above query and since both movie information and scene information are included, the *Play* button will be activated and user can send the result to player to review.

```
<Playlist>
  <Movie MovieID="m0000001"/>
  <Movie MovieID="m0000002"/>
  <Movie MovieID="m0000003"/>
  <Movie MovieID="m0000004"/>
  <Movie MovieID="m0000005"/>
  <Movie MovieID="m0000007"/>
  <Movie MovieID="m0000009">
    <Scene>
      <SceneID xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">1.2</SceneID>
      <Length>1981</Length>
    </Scene>
  </Movie>
  ...
  <Movie MovieID="m0000212">
```

```
    <Scene>
      <SceneID xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">1.7</SceneID>
      <Length>1967</Length>
    </Scene>
  </Movie>
  <Movie MovieID="m0000215"/>
  <Movie MovieID="m0000216"/>
  ...
</Playlist>
```

As user can ask any questions to obtain the movies or scenes they are interested, this integrated query capability provides a very good opportunity for searching the database. Some examples are listed below

- Find the scenes with famous dialogs. This assumes that annotations for famous dialogs have been used.
- Show stunt scenes from James Bond and Jackie Chan movies.
- Extract action scenes involving chase from movies involving while collar crime where both Tom Cruise and Gene Hackman have acted.
- Extract the climax scenes from movies with happy ending where the prosecutor wins the case. Here, we envision that scene level annotations have been used to identify scenes with climax and where prosecutor wins and movie level annotations indicating have happy ending and court case.
- List the scenes in show “Colombo” where he talks about his wife. (In this serial it was amusing that the detective Colombo would randomly mention his wife who remained unseen for several seasons. Here it is assumed that such scenes have been so annoted.
- Obtain scenes with theme music of movies that won Oscars for the best music.
- Show scenes where the actor Dev Anand is shown as singing sad songs.
- Show my favorite scenes from my favorite movies.
- Show the whole scene surrounding Dandi March from the Gandhi movie.
- Show suspense scenes from our family favorite movies.

3.3 Movie and Scene Playback

The third important function of Movie Database Manager is the integrated video playback. When searching or querying the database, users will be interested in watching the selected videos or scenes without leaving MDM. This is also helpful for making

scene annotations as it is very difficult to add or edit a scene without visual assistance. As indicated in Sections 3.1 and 3.2, video and scene can be sent to this panel for playback. That requires the player can handle both full video and part of the video.

The *Movie Player* panel has a table holding the playlist. To play a scene, it requires both *MovieID* and *SceneID* to be present at the same time since a scene can't be located by *SceneID* alone. Therefore, both *MovieID* and *SceneID* are listed in the table. In addition, video title is also present as reference. Figure. 14 shows the list of scenes extracted from query result in Section 3.2. The playlist table accepts multiple selection and generate a playlist for continuous play. Just like the scene materilization in scene information section, user can materilization multiple scenes from the playlist.

A full functional player similar to the commercial player has been integrated, as shown in Figure. 14. Bottom figure is the full control panel designed for the player. The implementation utilizes an open source framework vlcj and allows an instance of a native vlc media player to be embedded in the window. To make the embedded player work, vlc player needs to be installed in the operation system.

The player has the basic functions such as play, stop, pause etc. It also has the exclusive scene navigation functions. The leftmost button use to jump to previous scene when playing a list of scenes. Similarily, the right button near the volumn adjustor jumps to next scene in the playlist. These two functions help user to quickly go through or skip the scene without waiting.

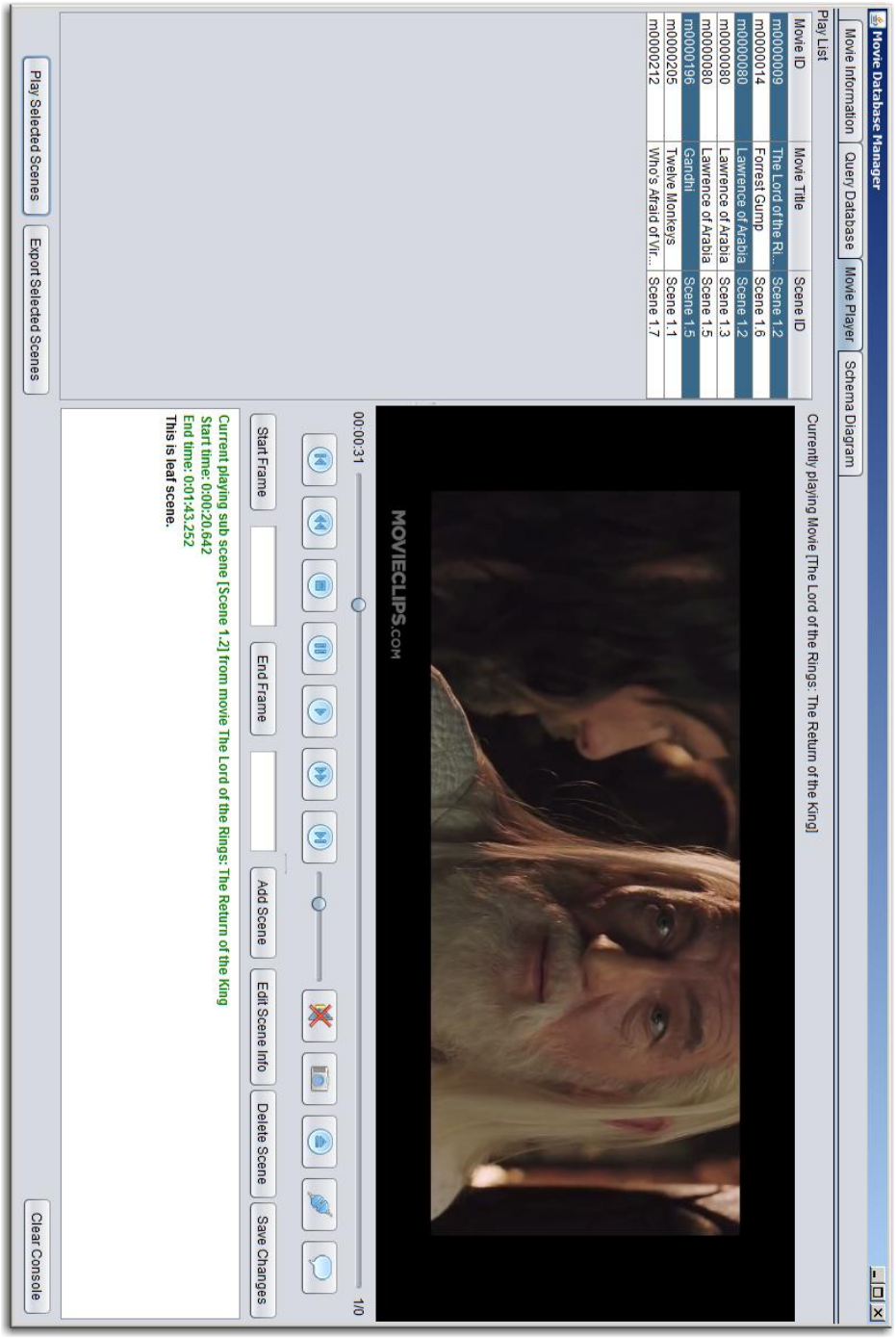

Figure 14. Video playback interface

Information about the playing scene are displayed in several places. On top of the player, it always shows the video title of current playing scene. While in the console

window, more detailed infromation is displayed including *SceneID*, video title , start and end. Sub-scenes are also listed in the console window if it is not a leaf scene.

Scene annotation is greatly enhanced here. When playing a scene, if a new scene is desired, user can either enter the frame number or by click the Start Frame and End Frame buttons to capture the start frame and end frame. Add Scene and Edit Scene Info buttons invoke the same Add Scene and Edit Scene dialogs described in Section 2.1 and will not be discussed here.

3.4 Schema Diagram

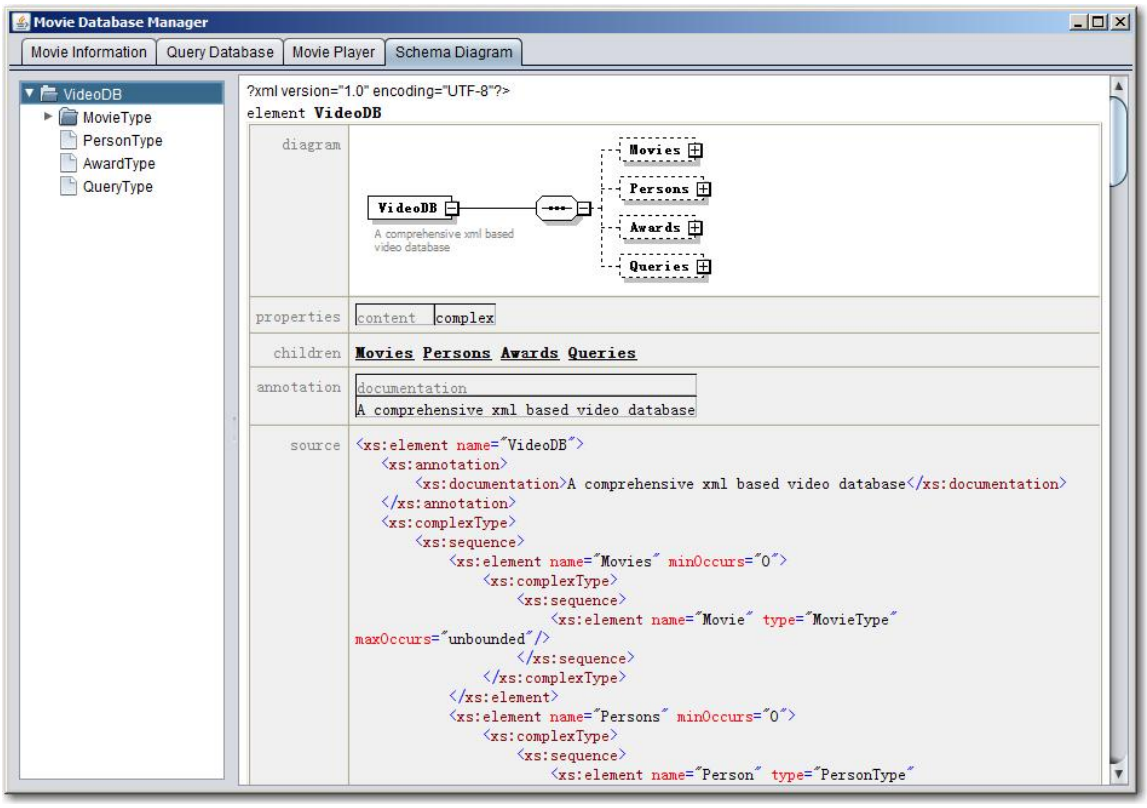

Figure 15. Schema diagram of Movie Database Manager

The schema diagram is mainly used to illustrate the structure of XML file. Left panel is a tree structure demonstrating the main elements. Each node corresponds to the

complex type in the schema. Clicking on tree node will display schema information for the node including diagram, properties, etc. in the right panel. This part of information is used to assist query development.

CHAPTER 4. PRIOR AND RELATED WORK

The idea of using XML to store video and scene information for easy managing, browsing, and searching was first initiated in an ISU course COM S 661 as a course project.

Yu Liu and Sourajit Ghosh Dastidar developed a simple storage model of the VMS (Video Management System) as a plugin to the Cyclone Database Implementation Workbench (CyDIW) . [8] The system uses two separate XML files to store video information and clip information separately. They also implemented a simple media player based on Java Media Framework (JMF).

The work done by Liu et. al have proved the feasibility of using XML as storage for video management. Although the integration with CyDIW gives it lot flexibilities, however, it also has limitations such as interface, project management, etc. Thus the Video Management System was separated from CyDIW and turned into a stand-alone program with its own user interface.

The storage model has been greatly improved to fit our needs. First of all, we combined the two separate XML files into one single XML file for easier management. The original model for video only contains video id, video name, source, start time, and end time and the model for video clip contains clip id, video id, start time, end time, and features. To turn the idea into a real workable program, the model structure has been expanded tremendously. Industrial information, video metadata, refined scene information, person information, award information, query information have been added into the storage model and make the XML file richer.

The query engine has been integrated to remove the dependency from CyDIW. The video playback has also been greatly enhanced. JMF is an old java framework and has stopped developing for a long time. We integrated the VLC player into our program to get better control and user experience.

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

It is an information explosion era. How to manage information becomes a great concern. Video is one of the important information sources and become part of people's life nowadays. How to find useful information efficiently has become a critical task. Traditional documents such as books, newspapers are normally searchable when digitized, while videos are not. In addition, most the videos are lengthy and might contain useless information that may be difficult to bypass. There is strong desire to make video searchable so that users can easily find what they are interested.

In this thesis, we have designed and implemented an XML-based video database management system meets the requirements. All video information except the video and image files is stored in one XML file. Natural structure of the information has been retained. Based on the nature of information, there are four complex types representing four types of information: *Movie*, *Person*, *Award*, and *Query*. Among these four types, *Movie* is the most complicated one and contains the most information. Inside *Movie*, three sub-elements, *MovieInfo*, *MetaData*, and *Scene*, are created to better organize the information.

Scene is part of the video and normally happen in one place or has one single topic. Several scenes might be combined to form another topic. Theoretically it is not a scene anymore since it can't be split into smaller pieces; however, in this thesis, we override the meaning of scene and consider this big chunk as a scene too. Thus, the scene forms a

hierarchy structure with unlimited levels. Annotation normally occurs at leaf level. Non-leaf scenes obtain the annotation from their descendants.

A user interface has been implemented to help user visually working on the XML file. The interface displays the information in a different but more user friendly way from its storage in XML file. Users can browse the database on almost everything such as movie/scene information, metadata, and scene information etc. The interface also integrated a query engine so that user can refine their searching. The output is parsed again to generate the scene playlist if exists.

A full functional video playback has also been integrated into the interface. Thus users can watch the videos, scenes, or the playlist created by the query directly. Integrated with the scene creating, editing, and annotating function, the interface can make scene annotation easier.

5.2 Future Work

Although the Movie Database Manager is fully functional on video managing, querying, and playing, there are still some directions that could further improve or expand its functionalities.

The user interface still can be improved. As we indicated in Section 2.5, an open source project *themoviedbapi* is used to collect commercial movie information from The Movie Database (http://www.themoviedb.org). This function can be integrated into the user interface so that users can make their own contributes. This function might introduce adverse effects, but it gives the user more controls. A statistical module could be very useful to obtain an overview of the whole database. It could be pure text or visualized. Improvement could also be made by giving the user ability to operate their playlist.

Similar as queries, another *PlayList* element can be created to host the playlists. User then can maintain the playlist without run query each time. Also results returned from different queries could be integrated into one play list. The management of the playlist needs to thoughts as the scenes may go through changes.

So far the Movie Database Manager is still a stand-alone program that can only be access by one user. It is desired to turn it into a multi-user program. Lot of concerns need to be covered such as security, threads, etc. Also how to store user information would also be challenge. Swing based interface could also be turn into web-based, thus user does not need to worry about environment setup, software installation etc.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Shashi K. Gadia, for the many insights and ideas he has given me. His expertise in databases and XML improved my knowledge and skills and prepared me for future challenges. Without his supervision and help, this thesis would not have been possible. The amount of time he spent helping me was invaluable and deeply appreciated.

I gratefully acknowledge and thank my committee member, Dr. Leslie Miller and Dr. Manimaran Govindarasu for their concerns, suggestions, and comments during this process.

I also would like to thank Dr. Wallapak Tavanapong for her inspiring talk on image and video processing techniques.

Finally, during my time at Iowa State University, I have received a great deal of support and understanding from my family especially my wife, Xiaohong Deng and my son, Kaden M. Li. I would very much like to thank them for everything they've done for me.

REFERENCES

[1] *Video file format*, http://en.wikipedia.org/wiki/Video_file_format

[2] *Video codec*, http://en.wikipedia.org/wiki/Video_codec

[3] *Scene (film)*, http://en.wikipedia.org/wiki/Scene_%28film%29

[4] *An XML-based framework for management of a course catalog system with zero information loss*, Xiaofeng Wang, Iowa State University Thesis

[5] *Document Object Model*, http://en.wikipedia.org/wiki/Document_Object_Model

[6] *XPath*, http://en.wikipedia.org/wiki/XPath

[7] *661 Project report: Video Management System*, Yu Liu and Sourajit Ghosh Dastidar, Iowa State University Course COM S 661 project.

[8] Shashi K. Gadia and Jay H. Vaishnav. *A Query Language for a Homogeneous Temporal Database*. Proc. Fourth Annual ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1985, pp 51-56.

[9] Shashi K. Gadia. *A Homogeneous Relational Model and Query Languages for Temporal Databases.* ACM Transactions on Database Systems, Vol. 14, 1988, pp 418-448.