Querying Relational MPEG-7 Image Database with MPEG Query Format

Mohammad Al-zoubi, Alaa' Al-zoubi

Dept. of Computer Graphics and Animation,
Princess Sumaya University for Technology, Jordan
mzoube@psut.edu.jo

Dept. of CIS, Irbid University, Irbid, Jordan
alzou3bi@gmail.com

Abstract

The growth of multimedia is increasing the need for standards to access and search distributed repositories. Recently, the Moving Picture Experts Group (MPEG) has developed the MPEG Query Format (MPQF) to make multimedia access and search easier and interoperable across search engines and repositories. Since MPQF is not supported by any database products including native XML databases and relational databases, a translation of MPQF to the multimedia database query language is required. This paper presents a system for querying relational MPEG-7 image database with MPQF. A translation of selected query types from MPQF to SQL is proposed based on an especially designed relational schema that preserves typed data of the image descriptions.

Keywords: *MPQF, MPEG-7, Image database*

1. Introduction

The use of multimedia has grown massively in recent years. Thus, the need for description, management and retrieval solutions has become ever more imperative for users to be able to access these multimedia resources effectively. The Moving Picture Experts Group (MPEG) addressed the description challenge by developing the MPEG-7 standard which provides a rich set of tools for completely describing multimedia content [1]. MPEG-7 offers a comprehensive set of multimedia description tools to create so-called *descriptions*, which can be used by applications that enable quality access to content. MPEG-7 distinguishes itself from other relevant metadata standards in its support for a range of abstraction levels, from low-level signal characteristics to high-level semantic information. The user can combine low-level descriptors and high level semantic information in a single description in a structured way, laying explicit links between these types of data features, and can choose among different descriptors and combine of them into meaningful sets of description units for each application.

Basically, MPEG-7 descriptions are XML documents that rely on an extension of XML Schema, called *Description Definition Language*. Thus, it is an obvious idea to employ XML database solutions for an efficient management of MPEG-7 descriptions. There are many different XML database approaches on the market with different capabilities, commercial, as well as open source or research prototypes. This includes native XML database solutions that develop XML storage solutions from scratch [2], and XML extensions that use the extensibility services of modern databases such as those given in [3].

However, a detailed analysis of the mentioned approaches has shown that neither native XML databases nor XML database extensions provide full support for managing MPEG-7 descriptions with respect to their requirements [4]. This is mainly because these solutions store and treat simple element content and the content of attribute values of MPEG-7 descriptions largely as text, regardless of the actual content type. This is inappropriate because in MPEG-7 many description schemes consist of non-textual data like numbers, vectors, and matrices. It is desirable that applications can access and process these schemes according to their real type and not as text. The problem of the inspected solutions is that they do not sufficiently make use of schema and type information offered within MPEG-7 descriptions. The majority of these approaches ignores schema definitions for the storage of XML documents, and uses them for validating XML documents only. To overcome these problems, a second approach concentrates on how to map XML schema to an equivalent relational database schemes specifically designed for that content [5, 6].

Recently, a new query language was developed by MPEG standardization committee to query MPEG-7 Data or other multimedia metadata. This query language is called the *MPEG Query format* (MPQF), which aims to provide a standardized interface for multimedia content retrieval systems, and to facilitate and unify access to distributed multimedia repositories [7]. To achieve this goal, the MPQF standard specifies precise input and output parameters to express multimedia requests and to allow clients easy interpretation and processing of result sets. Moreover, the management component of the MPQF covers searching and the choice of the desired multimedia services for retrieval. For this purpose, the standard provides a means to describe service capabilities and to undertake service discovery.

MPQF is based on XML, and therefore, is platform independence. So, developers can write their applications involving multimedia queries independently of the system used, which fosters software reusability and maintainability. However, this requires a translation of MPQF to the multimedia DBMS query language, i.e., mapping MPQF to SQL in the case of relational database.

In this context, this article presents a method to translate an MPQF query to SQL and an implementation of selected query types of MPQF within Oracle's relational database management system. The reset of the paper is organized as follows: Section 2 presents the MPEG-7 relational database schema. Section 3 presents the details of the mapping algorithm from MPQF to SQL, while Section 4 gives an example. Section 5 presents related work, and finally, Section 6 concludes the paper.

2. Design of Relational MPEG-7 Database

In this section, we present a design method of relational database schema that makes use of MPEG-7 schema, to manage the MPEG-7 descriptions into RDBMS. The method depends on the fact that MPEG-7 document can be viewed as tree graph. In this graph, actual values are stored in the leaf nodes (nodes at the end of the tree). A detailed description of this method can be found in [8], and here we present an overview of the main steps.

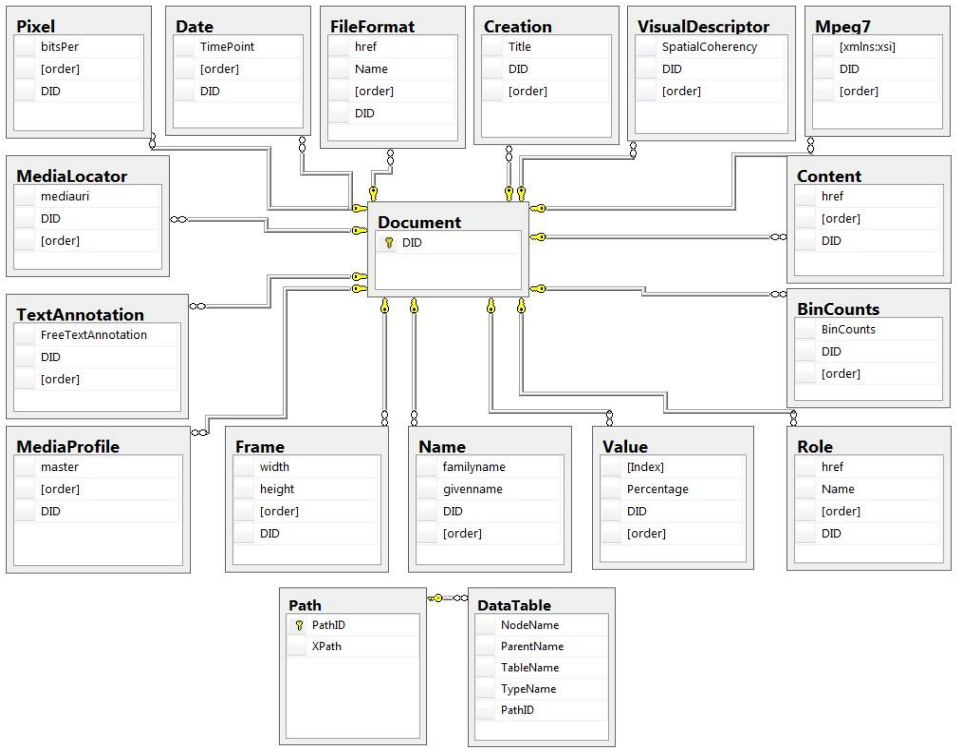


Figure 1: Relational database schema used to manage MPEQ-7 image descriptions.

First, an *XPath* expression is generated for every leaf node (element/attribute) defined in the XML schema. Second, a relational table is created for all items that have the same direct parent element. The names of tables and columns are specified as defined in the MPEG-7 schema. Third, columns data types (and constraints) are selected from Oracle DBMS types that are equivalent to the original data types defined in the schema (constraints are added when needed). Forth, an extra table is created to store unique IDs for the MPEG-7 documents, this table is named *Document*. And finally, an extra two columns are added for every table (generated up to this point); the *order* column which is used to preserve the order of elements as they appear in the original document, and the *DID* column, which is a foreign key references to the primary key in the *Document* table.

In addition, a second set of tables is created to store further information about the elements and attributes in the MPEG-7 description. This set consists of two tables; the *Path* table and the *Data* table. The *Path* table stores the *XPath* expressions generated in the first step above. Each expression is identified by a unique ID, while the *Data* table stores the node name, the parent node name, the name of the table that stores the node, the name of a complex data type defined in the metadata schema, and a reference to the *PathID* in the *Path* table.

Figure1 shows the relational schema generated to manage the MPEG-7 image descriptions which are created manually using *Caliph & Emir* image annotation tool [9]. Each description contains Media Information, Creation Information, Text Annotation, and Visual descriptors (dominant color and edge histogram descriptors) description tools. These description tools will enable us to perform various types of query conditions as will be illustrated in rest of the paper.

3. Mapping MPQF to SQL

MPQF is an XML-based multimedia query language that defines the format of queries requests and responses. The key elements of the MPQF are shown in Figure 2. A detailed explanation of these elements can be found in [10]. In the following subsections, we present a detailed description of our mapping approach for the Input Query Format, which represents the query request, to SQL that a RDBMS can execute based on the relational schema presented in Section 2.

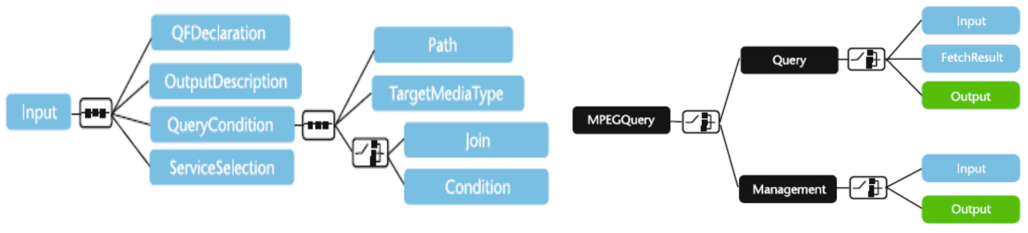


Figure 2: The main elements of MPQF and Input query

3.1. Output Description

The *OutputDescription* element enables the user to select information and its structure that the result set must contain. It also allows limiting the maximum number of items per output page and the overall items number. Besides, it allows the users to use aggregation and sorting processes. The *OutputDescription* element consists of the four major elements: *ReqField*, *ReqAggregateID*, *GroupBy*, and *SortBy*. In order to explain how to map these elements to SQL format, consider the following query sample:

```
<MpegQuery  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Query>
         <Input>
        <OutputDescription >
        <ReqField typeName =" PersonType">Name/GivenName</ReqField>
        <ReqAggregateID>MaxPercentage</ReqAggregateID>
        <GroupBy>
        <Aggregate xsi:type="MAX" aggregateID="MaxPercentage" >
                    <Field>Value/Percentage</Field>
       </Aggregate>
      </GroupBy>
      <SortBy xsi:type="SortByAggregateType" order="descending">
       <AggregateID>MaxPercentage</AggregateID>
      </SortBy>
     </OutputDescription>
    </Input>
  </Query>
</MpegQuery>
```

The *ReqField* element describes a data path within the item's metadata, which a user asks to be returned. Paths are specified using absolute XPath expressions, which refer to the root of the item's metadata. *ReqField* element has an optional attribute named *typeName* which is used to specify the name of the complex data type defined in the schema. Mapping *ReqField* element to SQL is performed by the following steps:

1) Extract the name of the element (or attribute) and the name of its relational table by matching the absolute XPath expressions and the value *typeName* attribute (if present) in the *ReqField* element with values of the *Path* and *Data* tables.
2) Append the name of the element (or attribute) in the SELECT statement, and the corresponding relational tables in the FROM statement.

The *ReqAggregateID* element describes the ID of the aggregate operation, the requester asks to be returned. When one or more *ReqAggregateID*s are used, the aggregate ID should be in the *GroupBy* element. Mapping this element to SQL is achieved by following steps:

1) Get the name of the aggregate function from the *GroupBy* element that has the same value in the *aggregateID* attribute of the *Aggregate* element. The name of the aggregate function is specified in the *xsi:type* attribute. (From the example above, the name of the aggregate function in the example is MAX.)
2) Get the element that will be used as attribute of this function from the *Field* element. (From example above the field name is *Percentage*.)
3) Apend the aggregate function in the SELECT statement, and use the value of the *aggregateID* attribute as an alias for that column.
4) Append the name relational table of the element in the aggregate function into the FROM statement, then insert all the elements in the SELECT statement that are not listed in the aggregate function into the GROUP BY statement.

The *SortBy* element describes the sort operation the user wants to apply on the query results. In MPQF, the *SortBy* is performed by either using *SortByFieldType* or by using *SortByAggregateType*. Mapping this element to SQL format is performed by the following steps:

1) When the sorting is performed using S*ortByFieldType*, the element of that field is appended to the ORDER BY statement. While when using *SortByAggregateType* the name and the element of the aggregate function are specified by the value of the *AggregateID* element, then the aggregate function is appended to the ORDER BY statement.
2) The type of sorting, whether ascending or descending, is determined by the *order* attribute of the *SortBy* element. The appropriate SQL format for sorting order (ASC or DESC) is appended to the ORDER BY statement.

After applying the steps above, all the tables in the FROM statement are *joined* in the WHERE statement on the *DID* (Document ID) field, and the complete SQL statement for the MPQF above will be:

```
SELECT [Name].[GivenName], MAX([Value].[Percentage]) AS MaxPercentage
FROM [Name], [Value]
WHERE ([Name].DID = [Value].DID)
GROUP BY [Name].[GivenName]
ORDER BY MAX ([Value].[Percentage]) DESC
```

3.2. Query Condition

The *QueryCondition* element is the part of the Input Query Format where the user specifies the properties of the media or the metadata to be retrieved. *QueryCondition* element has an optional *Path* element, T*argetMediaType* element, and a choice between *Condition* or a *Join* element. The following subsections will illustrate how to translate these elements to an equivalent SQL.

3.2.1. Condition element: The *Condition* element is a placeholder for a Boolean expression type and may result in a filter tree. The filter tree can be constructed by three main constructs, namely comparison expressions, Boolean operators, and query types (Figure 3).

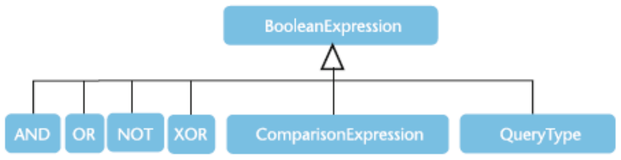


Figure 3: Component for construction condition element.

Usually, a comparison expression is defined by an operation and two operands. The operations defined include: *GreaterThan*, *GreaterThanEqual*, *LessThan*, *LessThanEqual*, *Equal*, *NotEqual* and *Contains*. The two operands should belong to the same OperandClass within a comparison expression. OperandClass denotes a representation of a specific data type, such as Boolean, String, Arithmetic, DateTime or Duration. Every element of an OperandClass can be described by a value of the data type, an XPath expression pointing to a value of the specific data type, or a corresponding expression resulting to a value of the specific datatype.

Arithmetic expressions allow performing arithmetic operations on arithmetic operands. These operations are: *Add, Subtract, Multiply, Divide, Modulus, Abs, Ceiling, Floor* and *Round*. Arithmetic Operands allow specifying one element which has arithmetic value content. The name of such element is specified by the *ArithmeticField* element. The value to be compared with is set by either *DoubleValue*, *LongValue* elements, or another Arithmetic expression. Aggregate expressions are subclass of arithmetic expressions that perform aggregate operations such as: *AVG, StdDev, Variance, SUM, Count, MAX* and *MIN*.

Boolean expressions allow building expressions, using Boolean operators (*AND, OR, NOT, XOR*), which evaluate to a Boolean value. The Boolean Operands allow specifying one element which has Boolean value content (true or false). The name of such element is specified by the *BooleanField* element. The value to be compared with is specified by the *BooleanValue* element, or by another Boolean Expression. Finally, the purpose of string expression type is to facilitate case insensitive processing of String Operands. These types include: *UpperCase* and *LowerCase*. The String Operands allow specifying one element which has String value content. The name of such a field is specified by the *StringField* element. The value to be compared with is specified by the *StringValue* element or another String Expression.

To map the *Condition* element to SQL, a recursive algorithm is implemented to traverse the condition tree and translate it to WHERE clause of the SQL statement as illustrated in the following pseudo code:

```
Read the attributes of the QueryCondition Element
Call ReadNext()

ReadNext()
{
    Read next element
    Case operandField:  // gets the names or values of the operands
        Call OperandsFunction()
    Case ConditionElement:
        Read the value of the xsi:type attribute
        If the value is one of ComparisonExpressionTypes  (GreaterThan,
        GreaterThanEqual, LessThan, LessThanEqual, Equal, NotEqual, Contains)
        then
            Call ComparisonExpressionFunction()
        Else if the value is one of the BooleanExpressionTypes (AND, OR, NOT,
        XOR) then
            Call BooleanExpressionFunction()
    Case ArithmeticExpression:
        Call ArithmeticExpressionFunction()
    Case StringExpression:
        Call StringExpressionFuntion()
    Case End of QueryCondition element:
        Append the condition to the WHERE statement and tables to the FROM
        statement
}

ComparisonExpressionFunction()
{
    Read next element
    If element is ArithmeticExpression then
        Apply the ComparisonExpression between the value returned by
        ArithmeticExpressionFunction () and ReadNext()
    Else if element is operandField then
        Apply the ComparisonExpression between the value returned by
        OperandsFunction() and ReadNext()
    Else if element is StringExpression then
        Apply the ComparisonExpression between the value returned by
        StringExpressionFunction () and ReadNext()
}

ArithmeticExpressionFunction()
{
    Read the value of the xsi:type attribute
    If type is one of (Add, Subtract, Multiply, Divide, Modulus, Abs, Ceiling, Floor,
    Round) then
        Read next element
        If element is operandField then
            return  the expression as
            ArithmeticExpressionOperation(OperandsFunction())
```

```
                    Else if element is ArithmeticExpression then
                        return the expression as
                        ArithmeticExpressionOperation(ArithmeticExpressionFunction())
                    Else if the type is (AVG, StdDev, Variance, SUM, Count, MAX, MIN) then
                            call AggregateExpressionFunction()
}

AggregateExpressionFunction()
{
        Read the value of the field element
        Find the relational table of this field from the dataTable and path table
        Return the expression in the form of
        AggregateExpressionOperation([Table  Name].[Element Name])
}

StringExpressionFunction()
{
        Read the value of the type attribute to specify the StringExpression operation
        Read next element
        If element is StringValue or StringField then
             Return the expression in the form of
             StringExpressionOperation(OperandsFunction())
        Else If element is StringExpression then /
             Return the expression as
             StringExpressionOperation(StringExpressionFunction())
}

OperandsFunction()
{
        Case operands that specifies a field element in the description (ArithmeticField,
        BooleanField, StringField, TimeField, DurationField)
        Get the relational element name and the relational table name from the dataTable and
        path table
        Return the operand in the form of [Table Name].[Element Name]
        Case operands that specifies a value (DoubleValue, BooleanValue …)
        Return the value of the element
}

BooleanExpressionFunction()
{
        Case AND, OR:
                Apply the operator between all the Conditions belong to this expression
        Case XOR:
                Apply the operator between the two conditions belong to this expression
        Case NOT:
                Apply the operator to the condition that belongs to this expression
}
```

As an example, consider the following query portion:

```
<Condition xsi:type="OR">
    <Condition xsi:type="Equal">
        <BooleanField>MediaProfile/@master</BooleanField>
        <BooleanValue>true</BooleanValue>
   </Condition>
   <Condition xsi:type="Equal">
        <StringField>Name/GivenName</StringField>
        <StringValue>Salma</StringValue>
   </Condition>
</Condition>
```

Mapping this query will result in the following SQL WHERE statement:

WHERE ([MediaProfile].[master] = 'true') **OR** ([Name].[GivenName] = 'Salma')

As another example:

```
<Condition xsi:type="Equal">
   <ArithmeticExpression xsi:type="Abs">
    <ArithmeticField>width</ArithmeticField>
   </ArithmeticExpression>
  <LongValue>500</LongValue>
</Condition>
```

will map to: *WHERE (**ABS**([Frame].[width])=500)).*

And Finally,

```
<Condition xsi:type="GreaterThan ">
  <ArithmeticField>Frame/@width</ArithmeticField>
  <LongValue>500</LongValue>
</Condition>
```

will map to: WHERE ([Frame].[width]>500).

3.3. Query Types

MPQF provides the following query types: QueryByMedia, QueryByDescription, QueryByFeatureRange, SpatialQuery, TemporalQuery, QueryByXQuery, QueryByFreeText, QueryByROI, and QueryByRelevanceFeedback. All these types inherit from the *QueryType* type which returns a Boolean denoting whether an evaluation item satisfies the condition specified by the operation or not. In our system, we implemented the most two popular image query types, namely, QueryByFreeText and QueryByMedia. The following subsections will present the implementation details.

3.3.1. Query by Free Text: The QueryByFreeText type enables the user to perform a search based on the use of free-text. It contains a *FreeText* element containing text description as a condition, and an optional choice of fields (*SearchField, IgnoreField*), which allow the user to if the search should be performed in specific elements only or if specific elements should be ignored. Mapping this query to SQL is performed by the following steps:

1) Get the *value* from the *FreeText* element.
2) Get the column name where the free text is stored from the *SearchField* element (if the *SearchField* is not present, we select the *FreeTextAnnotation* as the *SearchField*).
3) Apply the LIKE operator between the values in the first and second steps.

The accumulated condition will be appended to the WHERE statement as:
WHERE *Column_Name LIKE Free_Text.*

As an example, consider the following query:

```
<MpegQuery  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Query>
    <Input>
      <OutputDescription outputNameSpace="urn:mpeg:mpeg7:2004">
        <ReqField typeName="ImageType">MediaUri</ReqField>
      </OutputDescription>
      <QueryCondition>
        <Condition xsi:type="QueryByFreeText">
          <FreeText>rose</FreeText>
          <SearchField>TextAnnotation/FreeTextAnnotation</SearchField>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

From this example, the query text is 'rose', and the relational table and column names are *TextAnnotation* and *FreeTextAnnotation*, respectively. The SQL statement representing this query is:

```
SELECT   [MediaLocator].[MediaUri]
FROM  [MediaLocator], [TextAnnotation]
WHERE  ([MediaLocator].DID = [TextAnnotation].DID )
AND ( [TextAnnotation].[FreeTextAnnotation] LIKE 'rose' )
```

3.3.2. QueryByMedia: The QueryByMedia type enables the user to perform a search based on a given example of media resource. It provides an attribute (*matchType*) to set the search criteria whether similar-match or exact-match. To illustrate how this query type is translated to SQL, consider the following query sample:

```
<MpegQuery  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Query>
    <Input>
```

```
      <OutputDescription outputNameSpace="urn:mpeg:mpeg7:2004">
        <ReqField typeName="ImageType">MediaUri</ReqField>
      </OutputDescription>
      <QueryCondition>
        <Condition xsi:type="QueryByMedia" matchType="similar">
          <MediaResource resourceID="Image001" >
            <MediaResource>
              <MediaUri>file:/D:/Pictures/600.jpg</MediaUri>
            </MediaResource>
          </MediaResource>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

As indicated by the *matchType* attribute of the *condition* element, the value of this attribute is *similar*. To implement this query, a set of functions are implemented to extract low level descriptors, such as dominant color descriptor or edge histogram descriptor, of the input image and match it with all images in the database.

To map and execute this query, the steps below are followed:

1) The descriptor value for the query image is retrieved using a simple function.
2) The distance between the query image and all images in the database is calculated and the results are inserted into a temporary table called *Similarity* (consists of *sim* and *DID* columns).
3) The similarity values are retrieved in *ascending* order.

The Final SQL statement for the MPQF sample above will be:

```
SELECT   [MediaLocator].[MediaUri]
FROM  [MediaLocator], [Similarity]
WHERE  ([MediaLocator].DID = [Similarity].DID )
ORDER BY  [Similarity].[sim] ASC
```

However, when the value of the *matchType* attribute is *exact*, we define a function called *GetDescription* to retrieve the Descriptor value from the database of the query image, then a simple SQL statement is appended to retrieve the descriptions that match (exactly) the query Description. The SQL statement for the query above will be as the following code:

```
SELECT   [MediaLocator].[MediaUri]
FROM  [MediaLocator], [BinCounts]
WHERE [BinCounts].[BinCounts] = GetDescription("file:/D:/Pictures/600.jpg")
ORDER BY [Similarity].[sim] ASC
```

where the descriptor values are stored in the BinCounts column of the *BinCounts* table.

3.4. Target Media Type

The *TargetMediaType* element is part of the *QuertCodition* element that contains MIME type descriptions of media formats, which are the targets for retrieval. A MIME type is composed of (at least) two parts: a *type* and a *subtype* separated by "/". For instance, the MIME type audio/mp3 would filter all results for audio files depending on the MP3 format. In our implementation, mapping this type is performed by the following steps:

1) Extract the MIME type from specified in the *TargetMediaType* element.
2) Identify the *type* and subtype and determine the relational table that store each value. (Note: in our image database, we assumed that the main type is in the *href* of the Content element and the subtype is in the *Name* element of in the *FileFormat* parent element).
3) Append tables names to the FROM statement, and append the appropriate condition(s) in the WHERE statement.

To illustrate these steps, consider the following query sample:

```
<MpegQuery  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Query>
    <Input>
      <OutputDescription outputNameSpace="urn:mpeg:mpeg7:2004">
        <ReqField>MediaUri</ReqField>
      </OutputDescription>
      <QueryCondition>
        <TargetMediaType>image/JPEG</TargetMediaType>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

Based on the steps above, the resultant SQL statement will be:

```
SELECT   [MediaLocator].[MediaUri]
FROM  [MediaLocator], [Content], [FileFormat]
WHERE  ([MediaLocator].DID = [Content].DID ) AND ([MediaLocator].DID =
[FileFormat].DID )
AND ([Content].DID = [FileFormat].DID)  AND (( [Content].[href] = 'image' )
AND   ( [FileFormat].[Name] = 'JPEG' ))
```

4. Experimental Results

To test our system, a database based on Oracle 10g is created as specified in Section 2, and a set of images were annotated and stored in this database. We aim here to test the mapping algorithm not the efficiency of retrieval process. The Example below presents a query that combines the use of QueryByMedia, QueryByFreeText, and the TargetMediaType, to retrieve images URIs of all types that have either "rose" or "yellow" as a free text and have been created after 2002-05-30 and *similar* to the following query image:

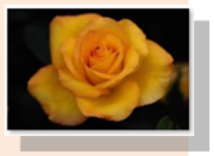
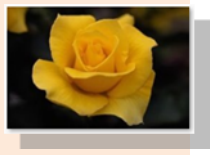
```
<MpegQuery  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <Query>
  <Input>
   <OutputDescription outputNameSpace="urn:mpeg:mpeg7:2004">
    <ReqField>MediaUri</ReqField>
   </OutputDescription>
   <QueryCondition>
    <TargetMediaType>image/*</TargetMediaType>
    <Condition xsi:type="AND">
     <Condition xsi:type="GreaterThan">
      <DateTimeField>Date/TimePoint</DateTimeField>
      <DateTimeValue>2002-05-30</DateTimeValue>
     </Condition>
     <Condition xsi:type="QueryByMedia" matchType="similar">
      <MediaResource resourceID="Image001">
       <MediaResource>
        <MediaUri>file:/D:/Pictures/600.jpg</MediaUri>
       </MediaResource>
      </MediaResource>
     </Condition>
     <Condition xsi:type="OR">
      <Condition xsi:type="QueryByFreeText">
       <FreeText>rose</FreeText>
       <SearchField>TextAnnotation/FreeTextAnnotation</SearchField>
      </Condition>
      <Condition xsi:type="QueryByFreeText">
       <FreeText>yellow</FreeText>
       <SearchField>TextAnnotation/FreeTextAnnotation</SearchField>
      </Condition>
     </Condition>
    </Condition>
   </QueryCondition>
  </Input>
 </Query>
</MpegQuery>
```

Based on the mapping technique discussed in Section 3, the SQL statement is:

```
SELECT   [MediaLocator].[MediaUri]
FROM  [MediaLocator], [Content], [Date], [Similarity], [TextAnnotation]
WHERE  ([MediaLocator].DID = [Content].DID ) AND ([MediaLocator].DID = [Date].DID
) AND ([MediaLocator].DID = [Similarity].DID ) AND ([MediaLocator].DID =
[TextAnnotation].DID ) AND ([Content].DID = [Date].DID ) AND ([Content].DID =
[Similarity].DID ) AND ([Content].DID = [TextAnnotation].DID ) AND ([Date].DID =
[Similarity].DID ) AND ([Date].DID = [TextAnnotation].DID ) AND ([Similarity].DID =
[TextAnnotation].DID )
AND
(([Date].[TimePoint] > '2002-05-30') AND (( [TextAnnotation].[FreeTextAnnotation] LIKE
'rose' )
OR ( [TextAnnotation].[FreeTextAnnotation] LIKE 'yellow')))  AND ([Content].[href] =
'image')
ORDER BY [Similarity].[sim] ASC
```

Executing this query resulted in the following output which is according to the Output query format of the MPQF specifications:

```
<MpegQuery xmlns:xsi="www.w3.org/2001/XMLSchema-instance">
 <Query>
  <Output CurrPage = "1" totalPages = "1">
   <ResultItem  recordNumber="1">
    <Description xmlns:mpeg7="urn:mpeg:mpeg7:2004">
     <mpeg7:Mpeg7>
      <mpeg7:DescriptionUnit >
       <mpeg7:MediaUri>file:/D:/Pictures/600.jpg</mpeg7:MediaUri>
      </mpeg7:DescriptionUnit>
     </mpeg7:Mpeg7>
    </Description>
   </ResultItem>
   <ResultItem  recordNumber="2">
    <Description xmlns:mpeg7="urn:mpeg:mpeg7:2004">
     <mpeg7:Mpeg7>
      <mpeg7:DescriptionUnit >
       <mpeg7:MediaUri>file:/D:/Pictures/633.jpg</mpeg7:MediaUri>
      </mpeg7:DescriptionUnit>
     </mpeg7:Mpeg7>
    </Description>
   </ResultItem>
   <ResultItem  recordNumber="3">
    <Description xmlns:mpeg7="urn:mpeg:mpeg7:2004">
     <mpeg7:Mpeg7>
      <mpeg7:DescriptionUnit >
       <mpeg7:MediaUri>file:/D:/Pictures/628.jpg</mpeg7:MediaUri>
      </mpeg7:DescriptionUnit>
     </mpeg7:Mpeg7>
    </Description>
   </ResultItem>
   <ResultItem  recordNumber="4">
    <Description xmlns:mpeg7="urn:mpeg:mpeg7:2004">
     <mpeg7:Mpeg7>
      <mpeg7:DescriptionUnit >
       <mpeg7:MediaUri>file:/D:/Pictures/609.jpg</mpeg7:MediaUri>
      </mpeg7:DescriptionUnit>
     </mpeg7:Mpeg7>
```

```
    </Description>
   </ResultItem>
   <ResultItem  recordNumber="5">
    <Description xmlns:mpeg7="urn:mpeg:mpeg7:2004">
     <mpeg7:Mpeg7>
      <mpeg7:DescriptionUnit >
       <mpeg7:MediaUri>file:/D:/Pictures/615.jpg</mpeg7:MediaUri>
      </mpeg7:DescriptionUnit>
     </mpeg7:Mpeg7>
    </Description>
   </ResultItem>
  </Output>
 </Query>
</MpegQuery>
```

The result above encloses five records, where each record is represented by the *ResultItem* element. The *ResultItem* element includes the image information that satisfies the query conditions, i.e., an image which is created after 2002-05-30 and similar to the query image. The result shows that our system is capable of searching similar images, contrary to other implementations, and this the main advantage of our system. However, implementing this query requires a preprocessing step to calculate the similarity vector between the query image and the database. This is not efficient, because the search is linear, and requires further investigation.

5. Related Work

To the best of our knowledge there are two implementations of MPQF over relational-based MPEG-7 databases. The first implementation [11] is based on Oracle's object-relational database management system, where the implemented query types operate on one table containing all inserted MPEG-7 documents as *XMLTypes*. The system implements the following query types: QueryByFreeText, QueryByXQuery, QueryByDescription, and QueryByMedia (*exact* match only), and it is possible to generate complex combinations of these query types by using *AND and OR* operators through assembly of the individual parts with SQL intersect and union operations.

The second implementation [12] relies on MS SQL Server 2005 which provides an automatically mapping process of XML schema to an equivalent database schema, and provides SQL-based extensions which allow the user to embed XPath/XQuery expressions in SQL statements. Two query types were implemented: QueryByFreeText and QueryByMedia (*exact* match only). QueryByFreeText translated into XQuery expressions embed in SQL statement. QueryByMedia implemented by using both XQuery expression and SQL functions.

The major limitation of both systems is that MPEG-7 document is stored in one table and one column of *XMLType*, and hence, typed data are stored as text. This is not useful because in MPEG-7 many description schemes consist of non textual data like numbers, vectors, and matrices, and hence, incurs limitations in processing query types which requires numerical calculations, such as QueryByMedia with *similar* attribute.

6. Conclusions

In this paper, we presented a system which provides a means for storing and querying MPEG-7 image descriptions with MPQF. A relational database schema is designed to store and manage these descriptions, which preserve the typed data of the visual descriptors. This method provides the flexibility to perform queries which requires processing of numerical data. However, it will make it difficult to implement queries which rely on XML structure such as QueryByDescription, since the XML structure of the description is lost by the mapping process.

Then, based on this schema, an MPQF query processing algorithm for mapping the MPQF input query and translate it to equivalent SQL statement is implemented. The system features implementations of the two query types: QueryByFreeText, and QueryByMedia (*exact* and *similar* match), and it is possible to combine them with intricate constraints on any element or attribute value using *AND, OR, NOT, and XOR* operators.

In a future work, we will try to improve the efficiency of the system, for example, splitting the query to speed up the retrieval process.

References

[1] Manjunath, T.S.B.S. and Salembier, P. Introduction to MPEG-7: Multimedia Content Description Interface, Wiley, Chichester, England, ISBN: 978-0-471-48678-7, 2002.

[2] Jagadish, H.V., Al-Khalifa, S., and Chapman, A. *et al.* TIMBER: a native XML database. *VLDB Journal*, **11** (4), 274–291. 2002.

[3] Kosch, H. Distributed Multimedia Database Technologies Supported by MPEG-7 and MPEG-21, CRC Press, Boca Raton, FL, 248, ISBN: 0-849-31854-8, 2003.

[4] Westermann, U. and Klas, W. An analysis of XML database solutions for the management of MPEG-7 media descriptions. *ACM Computer Surveys*, **35** (4), 331–373, 2003.

[5] Chu, Y., Chia, L.-T., and Bhowmick, S.S. Mapping, indexing and querying of MPEG-7 descriptors in RDBMS with IXMDB. *Journal of Data and Knowledge Engineering (DEK)*, **63** (2), 224–257, 2007.

[6] Westermann, U. and Klas, W. PTDOM: a schema-aware XML database system for MPEG-7 media descriptions. *Software: Practice and Experience*, **36** (8), 785–834, 2006.

[7] Doller, M., Tous, R., Gruhne, M. *et al.* The MPEG Query Format: on the way to unify the access to Multimedia Retrieval Systems. *IEEE Multimedia*, **15** (4), 82–95, 2008.

[8] Alaa' Alzoubi and Mohammad Alzoubi. Automatic Mapping of MPEG-7 Descriptions to Relational Database, will appear in *The International Arab Journal of Information Technology*.

[9] Mathias Lux, Caliph & Emir: MPEG-7 photo annotation and retrieval, Proceedings of the seventeen ACM international conference on Multimedia, October 19-24, Beijing, China [doi>10.1145/1631272.1631456] , 2009.

[10] ISO/IEC 15938-12 FCD Query Format, http://www.chiariglione.org/mpeg/working_documents.htm#MPEG-7

[11] M. Döller, R. Tous, M. Gruhne, M. Choi, T.-B. Lim, J. Delgado and A. N. Ndjafa Yakou; Semantic MPEG Query Format Validation and Processing; *IEEE Multimedia* 16(4):22-34, October-December 2009.

[12] Zhang Shaolong; Wu ZhiHua, An implementation of MPEG Query Format over relational-based multimedia database, 2nd International Conference on Future Computer and communication (ICFCC) , pp. V1-614 - V1-617, Wuhan ,China, 2010.