Semantic MPEG Query Format Validation and Processing

Mario Döller and Armelle Natacha Ndjafa Yakou
University of Passau, Germany

Ruben Tous and Jaime Delgado
Polytechnic University of Catalonia

Matthias Gruhne
Fraunhofer Institute for Digital Media Technology

Miran Choi
Electronics and Telecommunications Research Institute, South Korea

Tae-Beom Lim
Korea Electronics Technology Institute

This article presents a validation and processing architecture for the MPEG Query Format, which provides a standardized interface to multimedia document repositories.

The retrieval of multimedia content has experienced a tremendous boost in the research and industry sectors during the last couple of years. Due to the intensive work in this area, an unmanageable diversity of approaches and retrieval systems in the image,[1] video,[2] and audio[3] domains has emerged. In addition, computer scientists and industry professionals have fostered major developments in the area of multimedia databases.[4–6] While this diversity serves to stimulate the development of new technologies, it also prevents clients from relying on a universal, interoperable search-and-retrieval system. In this context, the MPEG standardization committee (ISO/IEC JTC1/SC29/WG11) has developed a new standard, the MPEG Query Format (ISO/IEC 15938-12, MPQF), which provides a standardized interface to multimedia document repositories, including multimedia databases, documental databases, digital libraries, and geographical information systems.[7]

Currently, the working groups are implementing the MPQF reference software, which consists of three different software modules, the MPQF validator, parser, and basic interpreter.[8] The MPQF validator first checks the XML form and validity of an MPQF input/output query according to the rules of XML 1.1 and the MPQF XML schema. Secondly, the validator checks if the input or output query is semantically compliant with the rules, described in the MPQF standard, that cannot be enforced by the XML schema. After successful validation, the MPQF parser translates the XML-based query instance into an internal representation of a Java object, complete with methods for accessing and modifying the different parts of the query. Finally, the MPQF basic interpreter serves to help understand the semantics of certain parts of the language, focusing on basic conditions and query types.

In this context, this article presents the definition and implementation of a semantic MPQF validator and a processing engine for an MPEG-7 multimedia database. This article also presents an implementation of selected query types of MPQF within Oracle's object-relational database management system.

Format and engine

The MPQF is an XML-based multimedia query language that defines the format of queries and replies that can be exchanged between clients and servers in a multimedia search-and-retrieval environment. The key parts of the MPQF define three main components:

- The Input Query Format provides means for describing query requests from a client to a multimedia retrieval system (MMRS).
- The Output Query Format specifies a message container for MMRS responses.
- The Query Management Tools provide means for functionalities such as service discovery, service aggregation, and service capability description (for example, which query types or multimedia formats are supported).

1070-986X/09/$26.00 © 2009 IEEE Published by the IEEE Computer Society

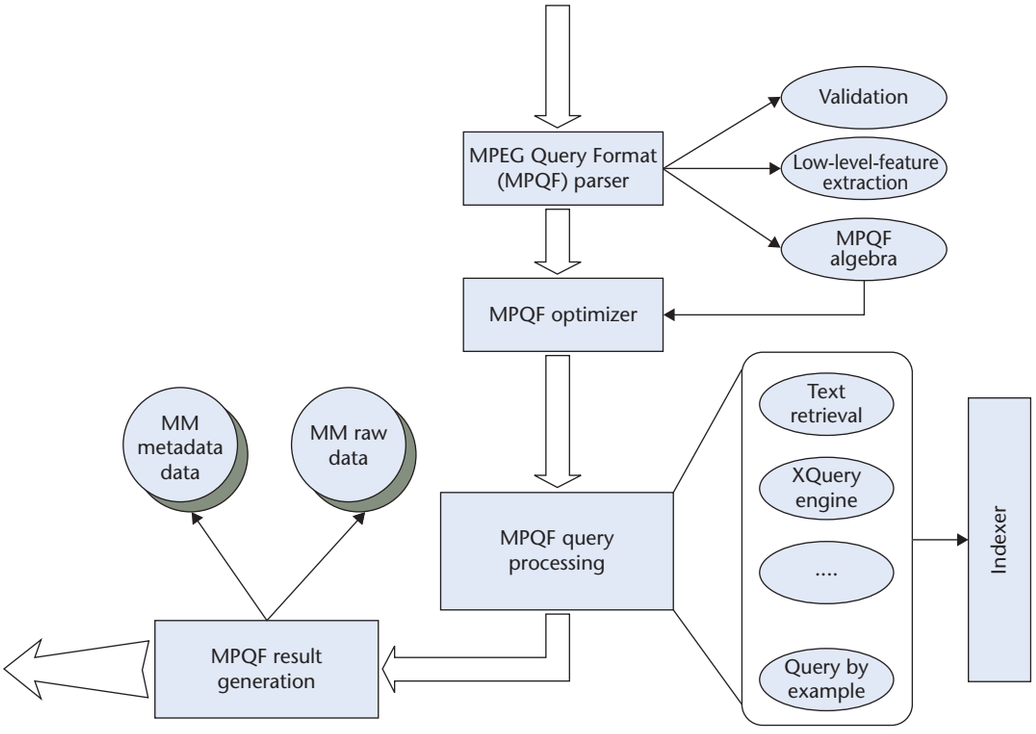

Figure 1. Architecture of MPEG Query Format-based retrieval engine.

The term *service* refers to all MMRSs, including single databases as well as service providers administrating a set of MMRSs.

A query request can consist of three different parts. The declaration part points to resources (for example, an image file or its metadata description) that are reused within the query condition or output description part. By using the respective MMRS metadata description, the output description part allows the definition of the structure as well as the content of the expected result set. Finally, the query condition part denotes the search criteria by providing a set of different query types (for example, QueryByMedia) and expressions (for example, GreaterThan), which can be combined by Boolean operators. To respond to MPQF query requests, the Output Query Format provides the ResultItem element and attributes signal paging and expiration dates. A detailed description about MPQF can be found elsewhere.[7]

Figure 1 presents the overall architecture of our MPQF-based retrieval engine. The engine receives MPQF queries, which are forwarded to an internal parser that transforms the XML-based MPQF types into a Java-based object tree representation. The transformation triggers a syntactic and semantic validation of the query, and low-level features are extracted for further processing. In the next step, the query tree is optimized according to rules (based on cost, selectivity, and other factors) to implement an efficient query-execution plan. This part of the process is still a work in progress and is therefore not considered here. Finally, the query plan is executed by our query-processing engine and the results are generated.

Query validation and rules

In general, the validity of an XML document can be evaluated by a syntactic check of the underlying XML schema. However, the MPQF schema specifies a format for expressing query requests to MPQF-aware retrieval systems. This format contains several internal rules that have to be considered to create a semantically valid request. In this context, the following sections introduce obstacles a user is confronted with in producing a valid query. Note that in most examples (except as noted otherwise), we use the MPEG-7 standard as the multimedia metadata format.[9]

AnyDescription semantic

MPQF can be used in combination with any XML-based multimedia metadata format. The standard provides the AnyDescription element, which allows the integration of XML-based descriptions. The possibility of including any

schema into query format messages introduces a large amount of flexibility, but becomes more complex in series validation. The following code snippet is an AnyDescription example:

```
<QFDeclaration>
  <Resource resourceID="DC_title"
  xsi:type="DescriptionResourceType">
   <AnyDescription xmlns:dc=
    "http://purl.org/dc/elements/1.1/"
    xsi:schemaLocation=
    "http://purl.org/dc/elements/1.1/dc.xsd">
      <dc:title>World Cup 2006</dc:title>
   </AnyDescription>
  </Resource>
</QFDeclaration>
```

As the code illustrates, the use of an AnyDescription instance requires the declaration of the used metadata format (for example, Dublin Core) and the information that expresses the search criteria (for example, the title should be ''World Cup 2006'').

Before validating semantic aspects of the AnyDescription element, the schema itself must be validated. Then the data to be searched for must be checked to see whether it validates according to the schema described in the AnyDescription field.

XPath semantic

XPath expressions are used to identify elements and attributes in the corresponding responder description in an input query or an output query. The instance of the xPathType must be compliant with the XML Path Language (XPath) 2.0 specification (see http://www.w3.org/TR/xpath20/). Furthermore, instantiated types referring to XPath expressions must be valid in terms of their base element. The following snippet is an XPath example:

```
<QueryCondition>
  <Condition xsi:type="Equal">
    <DateTimeField typeName="CreationType">
     /CreationCoordinates/Date/TimePoint
    </DateTimeField>
    ...
  </Condition>
</QueryCondition>
```

This code presents an XPath example integrated in a comparison expression. Note that the standard distinguishes between two different versions: absolute and relative addressing. If the typeName attribute is missing, then the XPath is absolute, meaning that resolving must start at the top-level element of the service's metadata schema. Otherwise, the attribute symbolizes the starting type and the XPath term points to an element relative to the starting type.

Validating the XPath expressions begins if they satisfy the rules of the World Wide Web Consortium (W3C) XPath 2.0 specification. Then, the content of the XPath expression is tested to see whether the selected path can be retrieved according to the schema.

GroupBy Semantic

Grouping of result sets is important even for multimedia requests to support the aggregation of description values (for example, file size). The snippet below is a GroupBy example:

```
<OutputDescription maxItemCount="30"
  maxPageEntries="10"
  outputNameSpace="urn:mpeg:mpeg7:2004">
 <ReqField typeName="Creator">
   /Character/FamilyName</ReqField>
 <ReqField typeName="CreationInformationType">
    /Creation/Title</ReqField>
 <ReqAggregateID>avgSize</ReqAggregateID>
 <GroupBy>
   <GroupByField typeName="Creator">
    /Character/FamilyName</GroupByField>
   <Aggregate xsi:type="AVG" aggregateID=
     "avgSize">
    <Field typeName="MediaFormat">
    /FileSize</Field>
   </Aggregate>
 </GroupBy>
</OutputDescription>
```

In this code, the resulting media items are grouped by the attached family name information and an average file size is calculated. To fulfill this operation, all used fields in the GroupBy clause must occur as fields in the above section of the OutputDescription element.

For validation, the XPath semantic rules for all involved XPath expressions are first applied. Then, for all used fields in the GroupBy clause, their occurrence in the OutputDescription section must be evaluated.

OutputDescription semantic

MPQF supports the OutputDescription type, which enables the requester to select the information that the result set must contain. This type allows limiting the maximum number of items per output page and the overall item number. The OutputDescription consists of the following elements:

- ReqField describes a data path, within the item's metadata, that a requester asks to be returned;
- ReqAggregateID describes the ID of the aggregate operation the requester asks to be returned;
- GroupBy describes the grouping operation the user wants to apply to the query results; and
- SortBy describes the sort operation the user wants to apply to the query results.

In addition to using these elements, several attributes can be selected to restrict the maximum number of pages or the use of free text.

The OutputDescription instance in the previous GroupBy example allows a maximum result set size of 30 items in MPEG-7 descriptions. This number is requested by the attribute outputNameSpace. The ReqField elements select the specific MPEG-7 descriptions that the requester wishes to be returned.

The validation of embedded XPath expressions in the ReqField starts checking if the embedded expressions satisfy the rules of the W3C XPath 2.0 specification. Then, the content of the XPath expression is tested to see whether the selected path can be retrieved from the related outputNameSpace. Special importance must be given to XPath expressions selected in the GroupBy element. Because the results are grouped by these types, the same types must also be selected in the ReqField. Finally, the system must evaluate whether maxPageEntries is less than or equal to maxItemCount.

QueryByFeatureRange semantic

The QueryByFeatureRange type enables the client to perform two different range searches. The first method allows a search on the basis of given descriptions (for example, a color histogram) denoting the start and the end range of the retrieval area. The second option allows a range search on the basis of a given description and a distance. The snippet below is a QueryByFeatureRange example:

```
<QFDeclaration>
 <Resource xsi:type="DescriptionResourceType"
  resourceID="startID">
   <AnyDescription xmlns:mpeg7
    ="urn:mpeg:mpeg7:schema:2004">
     <mpeg7:Mpeg7>
      <mpeg7:DescriptionUnit xsi:type=
       "mpeg7:AndioSpectrumEnvelopeType">
        1 2 3 4 5
      </mpeg7:DescriptionUnit>
     </mpeg7:Mpeg7>
   </AnyDescription>
  </Resource>
  <Resource xsi:type="DescriptionResourceType"
   resourceID="endID">
   ...
  </Resource>
 </QFDeclaration>
 <QueryCondition>
  <Condition xsi:type="QueryByFeatureRange">
   <Range RangeEnd="endID" RangeStart=
    "startID"/>
  </Condition>
 </QueryCondition>
```

Querying by a feature range assumes that the RangeStart and the RangeEnd elements belong to the same data type contained in the DescriptionResourceType. For example, both DescriptionResourceType types in this snipped must point to an AudioSpectrumEnvelopeType description.

In addition, the validation determines that RangeStart and RangeEnd addresses a resource of type DescriptionResourceType. Furthermore, the addressed descriptions (in the AnyDescription element) must use the same metadata format (for example, MPEG-7). And the descriptions must point to the same data type (for example, AudioSpectrumEnvelopeType).

SpatialQuery semantic

The SpatialQuery query type enables the search of spatial features within still image data. Depending on the underlying metadata model and the given parameters, the query type supports the retrieval of images or parts of them containing a specific spatial configuration of regions or content. The following snippet is a SpatialQuery example:

```
<QFDeclaration>
 <Resource resourceID="stillImage1"
  xsi:type="DescriptionResourceType">
   ...
 </Resource>
</QFDeclaration>
<QueryCondition>
  <Condition xsi:type="SpatialQuery">
    <SpatialRelation sourceResource=
      "stillImage1"
```

```
    relationType="urn:mpeg:mpqf:cs:
                  SpatialRelationCS:2008:
                   northwest-Of"/>
 </Condition>
</QueryCondition>
```

The main semantic rules that need to be applied in this example determine whether the content of the relationType attribute is valid. Furthermore, depending on the semantic of the spatial distribution (for example, every spatial direction supports three different semantic meanings: north, north-Of, north-In), the targetResource attribute must be omitted by the use of X-Of or X-In values, with X standing for a substitution of the respective spatial direction.

QueryByRelevanceFeedback semantic

The QueryByRelevanceFeedback element describes a query operation that uses the result from the previous retrieval method for post-processing. This query operation allows the requester to identify good or bad examples within the previous result set and indicates what should be retrieved. The following code snippet is a QueryByRelevanceFeedback example:

```
<QueryCondition>
 <Condition xsi:type=
     "QueryByRelevanceFeedback"
  answerID="IDofPreviousQuery">
  <ResultItem>4</ResultItem>
  <ResultItem>8</ResultItem>
 </Condition>
</QueryCondition>
```

This snippet shows a QueryByRelevanceFeedback request where two examples of a previous result set have been identified as good examples. Bad examples are represented by a negation of the condition.

To validate the QueryByRelevanceFeedback type semantically, the answerID must be evaluated and the previous result set obtained. On the basis of the previous results, the number in the ResultItem element must be verified.

Data type operand evaluation

The standard supports a set of operand data types (for example, long for arithmetic operands or date for temporal operands) for expressing the values of comparison expressions. The following code demonstrates a GreaterThan comparison expression that evaluates whether the value denoted by the XPath has a minimum of 340,000:

```
<QueryCondition>
 <Condition xsi:type="GreaterThan">
  <ArithmeticField
     typeName="MediaFormatType">/
       TargetChannelBitRate
  </ArithmeticField>
  <LongValue>340000</LongValue>
 </Condition>
</QueryCondition>
```

A semantic check of this query condition must verify whether the data types of the operands contain the correct value. For instance, the LongValue element must point to a long or integer type. The standard distinguishes only between floating and whole numbers. Furthermore, for all field elements, the XPath semantic rules must be applied.

Resolution of references semantic

To simplify MPQF queries, we can declare resources, such as the location of a video, that are referenced by individual query types (for example, by a QueryByMedia) in the condition clause. The following snippet is a QueryByMedia example:

```
<QFDeclaration>
 <Resource resourceID="Image001"
  xsi:type="MediaResourceType">
 ...
 </Resource>
</QFDeclaration>
<QueryCondition>
 <Condition xsi:type="QueryByMedia"
     matchType="similar">
   <MediaResourceREF>Image001
    </MediaResourceREF>
 </Condition>
</QueryCondition>
```

In this QueryByMedia example, the query type must reference only resources of MediaResourceType type. In addition, the QueryByDescription query type is only allowed to reference resources of DescriptionResourceType type.

QueryByFreeText semantic

The QueryByFreeText type enables the user to perform a free text search. It has an optional choice of fields that allow the user to state whether the search should be performed in specific elements (SearchField) or if specific elements should be ignored during the search

(IgnoreField). The following snippet is a QueryByFreetext condition example:

```
<QueryCondition>
 <Condition xsi:type="QueryByFreeText">
   <FreeText>Blob</FreeText>
   <SearchField>
    Mpeg7/Description/MultimediaContent//
      TextAnnotation
   </SearchField>
 </Condition>
</QueryCondition>
```

The expression *Blob* is retrieved from all TextAnnotation elements. The validation applies XPath semantic rules for all SearchField and IgnoreField elements, and verifies that no IgnoreField element is equal to the SearchField.

XQuery semantic

The current version of MPQF supports the QueryByXQuery query type, which allows embedding XQuery expressions in the form of conventional MPQF query types. However, the standard specifies that the XQuery expression is restricted to the use of constructs that produce a Boolean true or false decision on a single evaluation item in the target database. Furthermore, within the XQuery expression included in the QueryByXQuery type, no output description is allowed. This restriction poses a challenge to static validation of MPQF instances, because determining if the result of a given XQuery will be a singleton value (one item) just by examining the XQuery expression is not trivial. The following snippet is an XQuery example:

```
<QueryCondition>
 <Condition xsi:type="QueryByXQuery">
  <XQuery>
   <![CDATA[
     let $a := node()//Creator/Role/Name
     return
     $a/text()="Artist"]]>
  </XQuery>
 </Condition>
</QueryCondition>
```

This XQuery expression only consists of a Let sentence with an equality comparison between a local variable reference and a string. However, because an XQuery expression could result in more than one iteration, the result could be a singleton sequence with just one Boolean value inside, or a sequence consisting of more than one Boolean value.

Validating embedded XQuery expressions requires checking whether the embedded expressions satisfy the rules of the W3C's XQuery 1.0 specification.[10] The validator must check whether the result of the expressions will be a Boolean value. As shown previously, it's not always possible to determine whether the expected result of an XQuery expression will be a singleton Boolean. The standard provides the following recommendation for such cases: the return of a Boolean value might be guaranteed by the use of the fn:boolean function provided by XQuery.

The fn:Boolean, which is defined in the XQuery 1.0 specification, outlines some rules to determine the effective Boolean value of any XQuery sequence, whether it's a singleton Boolean or not:

- If it's an empty sequence: return false.
- If it's a sequence whose first item is a node: return true.
- If it's a singleton value of type xs:boolean or derived from xs:boolean: return the value of its operand unchanged.
- If it's a singleton value of type xs:string, xs:anyURI, or xs:untypedAtomic (or a derived type) and the value has zero length: return false.
- If it's a singleton value of any numeric type or derived from a numeric type: return false if the value is NaN or is numerically equal to zero; otherwise return true.
- In all other cases: raise an error.

When writing XQuery expressions for MPQF queries, it's important to know that Boolean values within XQuery 1.0 expressions can appear under the following circumstances:

- the fn:boolean, fn:true, and fn:false functions;
- logical expressions (and, or);
- the fn:not function;
- quantified expressions (some, every); and
- comparison expressions (=, ! =, and so on).

Regarding comparisons, XQuery defines three kinds of comparisons:

- value comparisons (“eq” | “ne” | “lt” | “le” | “gt” | “ge”),
- general comparisons (“=” | “! =” | “<” | “< =” | “>” | “> =”), and
- node comparisons (“is” | “≪” | “≫”).

The result of a comparison (of any kind) that doesn’t raise an error is always a single true or false. Value comparisons are used for comparing single values. General comparisons are existentially quantified comparisons that might be applied to operand sequences of any length. The result of a general comparison that doesn’t raise an error is always true or false. However, we envisage that implementers will always wrap the XQuery expression within a call to the XQuery fn:boolean function, which implements the previously mentioned rules.

Service description capabilities semantic

The management part of MPQF enables multimedia services to specify query capabilities because queries might be valid for one retrieval engine but invalid for another. The capability description format supports expressing what kind of query types, algebraic operations, and metadata formats are covered by the respective engine. The following snippet is a service description example:

```
<AvailableCapability
    serviceID="http://Service-1">
 <SupportedQFProfile href="urn:...:Full"/>
 <SupportedMetadata>urn:mpeg:mpeg7:2004
 </SupportedMetadata>
 <SupportedExampleMediaTypes>audio/mp3
 </SupportedExampleMediaTypes>
 <SupportedQueryTypes href=
     "urn:...:100.3.6.1"/>
 <SupportedExpressions href=
     "urn:...:100.3.1"/>
<AvailableCapability>
```

In this snippet, the supported metadata is based on the MPEG-7 schema and the supported media types are restricted to audio (MP3 format). Moreover, the multimedia engine supports the following functions: query types (100.3.6.1 stands for QueryByMedia) and expressions (100.3.1 means that all Boolean expression types are supported). For simplicity and space concerns, we don’t present the full classification scheme. For validation, all used components of the database, such as MediaResources, query types, algebraic operations, or metadata formats, must be processable by the target system.

Preference value semantic

The MPQF standard supports the assignment of preference values to individual conditions to highlight user priorities. Those preferences are represented in the range of 0 to 1. The following snippet is a preference value example

```
<QueryCondition>
 <Condition xsi:type="AND">
  <Condition xsi:type="QueryByMedia"
    preferenceValue="0.3">
    ...
  </Condition>
  <Condition xsi:type="QueryByMedia"
   preferenceValue="0.7">
   ...
  </Condition>
 </Condition>
</QueryCondition>
```

This snippet demonstrates the use of preference values to signal that one media type should be considered more important during searches. For validation, the sum of all preference values must be 1.

Framework and architecture

The MPQF validator provides an extensible, module-based framework that allows independent development and assembly of verification components. Verification components can be divided into two main groups: syntactic and semantic. Syntactic verification deals with the evaluation of XML documents according to the following two characteristics: well formed and valid. An XML document is well formed if it obeys the syntax of XML, and is valid if it obeys the syntax of the underlying XML schema. Related to the MPQF validator, an MPQF query is syntactically correct if it’s well formed (with respect to the XML) and valid according to the MPQF XML schema. Semantic verification deals with the evaluation of rules that aren’t expressed by syntactic means within the XML schema.

Figure 2 presents the internal workflow of the system. Whenever an instance of the validator is created, a corresponding validation chain is instantiated. A validation chain

consists of a set of validation modules that are selected for the individual validation process. The current implementation provides validation modules for evaluating XQuery, GroupBy, RelativeFields, and service capabilities.

The validation process evaluates the incoming MPQF query by traversing the validation chain, step by step. During this process, each validation module verifies the query according to its specific rules (syntactic or semantic). In case of an error, the validation stops and the respective error message is returned.

Figure 3 (next page) illustrates the class hierarchy of the MPQF validator. In general, we can define three different parts: the public classes (red), the validation modules (light blue), and the internal package (white). The validation modules represent the individual semantic validation routines that implement the ValidationPart interface. Every module receives as input the MPQF query as a document object model (DOM[11]) representation. The internal package provides some helper classes for DOM-tree handling, error processing, and syntactic validation. The public part supports the creation of an MPQF validator instance.

MPQF query processing

The MPQF query-processing module has been integrated in the MPEG-7 MultiMedia DataBase System (MMDB), which relies on Oracle's object-relational database management system. This system provides means for storing and querying MPEG-7 documents and has been chosen because of its extensibility and integrated indexing framework. However, the original proprietary query library has been replaced by the newly developed MPQFLib, which is capable of processing MPQF queries.

As noted, the query-execution plan is forwarded to the processing engine, where all query types and expressions are transformed into database functions. The current system features implementations for the following query types: QueryByFreeText, QueryByXQuery, QueryByDescription, and QueryByMedia. It's 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.

Most implemented query types operate on one table containing all inserted MPEG-7 documents as XMLTypes. An enhanced approach that considers the mapped database schema is in progress.

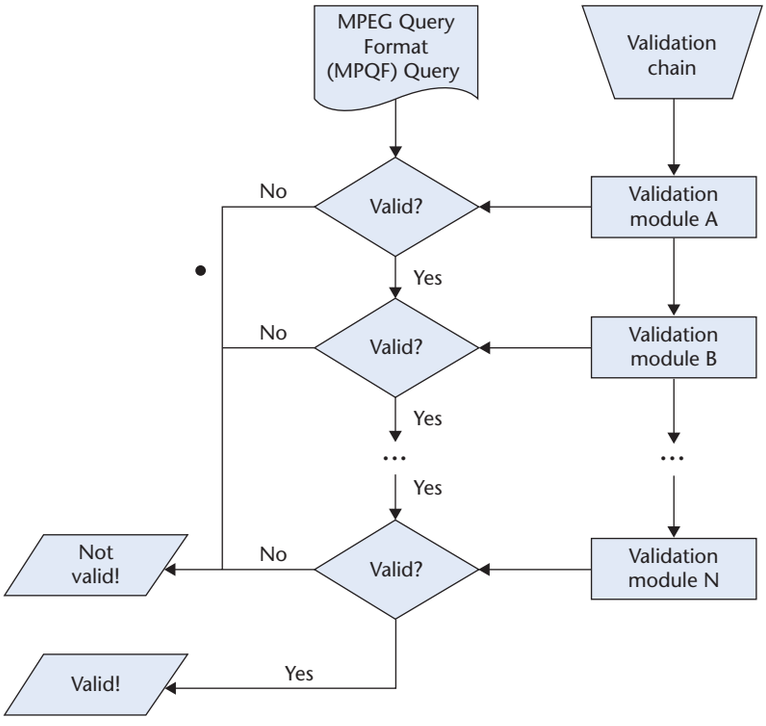

Figure 2. Validator workflow.

Query type implementation

The QueryByFreeText query type allows users to perform a free-text search with a given phrase or regular expression in combination with SearchField and IgnoreField information to narrow the query. Text retrieval in XML documents is supported in Oracle through the use of Oracle Text (see http://www.oracle.com/technology/products/text/index.html). Depending on the enabled functionality, these indexes enable users to search by specific field (with the InPath clause), by excluding specific fields (with the StopList clause), and by arbitrary text (with the contains clause). However, the regular expression feature of the query type is not supported by those indexes.

Figure 4 shows how MPQFLib maps a QueryByFreeText request to a SQL command. The phrase in question is combined with the SearchFields by the contains operation and the InPath clause. The Extract part uses the XPath values of the OutputDescription (not shown in the MPQF example) to retrieve the desired information. The individual result items are ordered by the extracted score, which is calculated by the occurrence ratio of the search phrase within the document.

The QueryByXQuery type enables the evaluation of XQuery expressions. Oracle provides

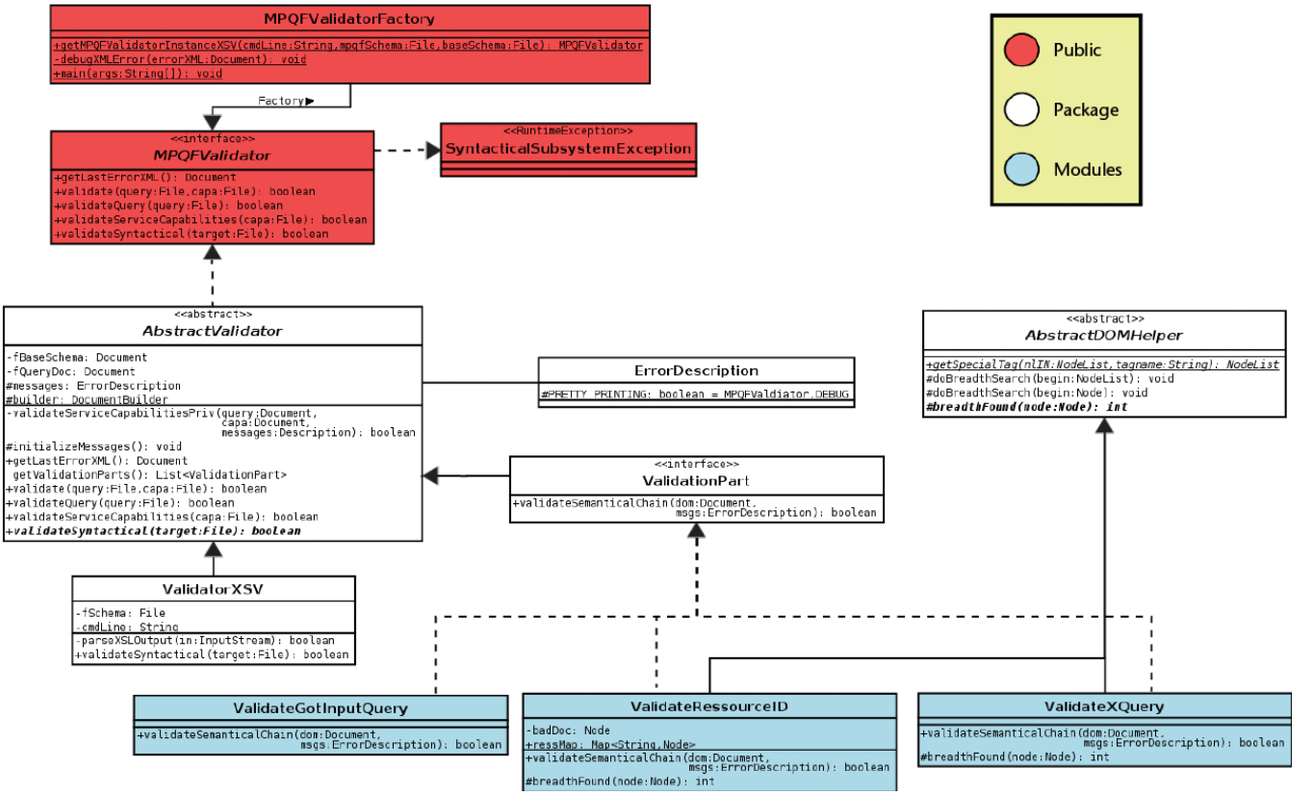

Figure 3. Class hierarchy.

```
SELECT extract(Document,'/*/*/*/*/MediaLocator[1]/MediaUri/text()',
              'xmlns="urn:mpeg:mpeg7:schema:2001"').getStringVal()as
                    imageLocation,
              score(1) as score_nbr
FROM Mpeg7_Document
WHERE contains(Document, 'Blob INPATH Mpeg7/Description/MultimediaContent//
     TextAnnotation', 1) > 0
ORDER BY score_nbr;
```

Figure 4. QueryByFreeText implementation.

```
 1 SELECT extract(Document,'/*/*/*/*/MediaLocator[1]/MediaUri/text()',
 2                        'xmlns="urn:mpeg:mpeg7:schema:2001"').getStringVal()
 3                        as imageLocation,
 4        XMLQuery('declare default element namespace "urn:mpeg:mpeg7:schema:2001";
 5                  (::)
 6             let $a := node()//Creator/Role/Name
 7             return if ($a/text() = "Artist") then 1 else 0 '
 8             PASSING Document
 9             RETURNING CONTENT).getNumberVal() as evaluation
10 FROM Mpeg7_Document
```

Figure 5. QueryByXQuery implementation.

the XQuery function in combination with optional context information. To fit to the MPQF evaluation model in which every operation is required to return a score value between 0 and 1 for every evaluation item, the result of the execution must be narrowed to 0 or 1. Therefore, the MPQFLib maps an incoming QueryByXQuery operation to a SQL command, as shown in Figure 5. In the figure, the XQuery command is extended to guarantee that the result is narrowed to a value between 0 and 1.

The QueryByDescription query type allows the retrieval for similar or exact descriptions (for example, ScalableColor in MPEG-7) of the target metadata format. Currently, our implementation is restricted to the evaluation of an exact match. For this purpose, we use the existNode function provided by the Oracle internal XMLType for harvesting the XML content. This member function receives an XPath expression and evaluates this expression on the stored XML data. If target node (including content) exists, 1 is returned; otherwise 0 is returned. Complex descriptions can be evaluated by a concatenation (by AND) of successive existNode functions holding the individual subpaths.

The QueryByMedia implementation requires preprocessing the provided data, from which features for comparison are extracted. The system currently supports the extraction of low-level image features (ScalableColor, HomogenousTexture, and so forth) through operations

Related Work

A recent overview paper offered an analysis of several multimedia architecture approaches.[1] This paper proposed an architecture classification that divided them into systems developed from scratch,[2] plug-in systems of existing databases,[3] systems tuned for improving query-processing techniques,[4] and systems focusing on semantic retrieval.[5] In addition, the authors introduced a generic multimedia architecture consisting of feature extractors, metadata schemas, user profiles, search logics, and a component to deal with ontologies.

To support interoperability among individual retrieval systems, the definition of a generic architecture must rely on standardized components. In this line of thinking, several research groups have considered standardized solutions for single components. For the query language, recent approaches include the MPEG Query Format (MPQF),[6] an XML-based multimedia query language for distributed multimedia retrieval. Like other approaches (for example, SQL/MM[7]), MPQF combines a data-centric model with fuzzy retrieval.

There are other systems that rely on standardized metadata formats (for example, MPEG-7[8]) for representing low- and high-level information. The selected format often defines the diversity and power of the retrieval system. We have chosen MPEG-7 as our description format for representing the content of our multimedia data because it is the richest standardized format available. Other XML-based formats—for example, P/Meta (see http://www.i3a.org/) and Dublin Core (see http://dublincore.org/documents/dces/)—can be used together with MPQF by supplying appropriate retrieval components.

Other research projects have introduced multimedia retrieval systems that support MPEG-7.[3,9–11] For instance, Persistent Type Document Object Model (PTDOM) is a schema-aware XML (and therefore MPEG-7) database system that supports document validation, typed storage of elements and attribute values, structural indexing facilities, and optimizations of query plans.[10] Nevertheless, this system focuses on data retrieval and neglects multimedia-retrieval features (such as query by example and spatiotemporal queries).

As another example, the IXMDB system represents an approach for the integration of the MPEG-7 standard into a relational database management system.[9] Within this system, the advantages of schema-conscious and schema-oblivious mapping methods are combined to support efficient retrieval. Like PTDOM, only XPath and XQuery evaluation is supported in IXMDB, which limits its use for content-based multimedia retrieval.

In addition to offering a mapping of the MPEG-7 standard to an object-relational database model, our MPEG-7 system[3] provides an integrated indexing framework based on the Berkeley GiST implementation.[12] It also provides a set of internal libraries featuring the insertion, retrieval, and deletion process. Specialized application libraries can run on top of this architecture to adapt the system to specific needs (such as for image or audio retrieval).

References

1. O.A. Hamid et al., ''Generic Multimedia Database Architecture Based upon Semantic Libraries,'' *Informatica,* vol. 18, no. 4, 2007, pp. 483-510.
2. T. Ojala et al., ''CMRS: Architecture for Content-Based Multimedia Retrieval,'' *Proc Infotech Oulu Int'l Workshop Information Retrieval,* 2001, pp. 179-190.
3. M. Döller and H. Kosch, ''The MPEG-7 Multimedia Database System (MPEG-7 MMDB),'' *J. Systems and Software,* vol. 81, no. 9, 2008, pp. 1559-1580.
4. A.P. de Vries, ''Mirror: Multimedia Query Processing in Extensible Databases, *Proc. 14th Twente Workshop Language Technology: Language Technology in Multimedia Information Retrieval,* Twente Univ. Press, 1998, pp. 37-48.
5. Y. Yildirim, T. Yilmaz, and A. Yazici, ''Ontology-Supported Object and Event Extraction with a Genetic Algorithms Approach for Object Classification,'' *Proc. 6th ACM Int'l Conf. Image and Video Retrieval,* ACM Press, 2007, pp. 202-209.
6. Mario Döller et al., ''The MPEG Query Format: On the Way to Unify the Access to Multimedia Retrieval Systems,'' *IEEE MultiMedia,* vol. 15, no. 4, 2008, pp. 82-95.
7. J. Melton and A. Eisenberg, ''SQL Multimedia Application Packages (SQL/MM),'' *ACM Sigmod Record,* vol. 30, no. 4, 2001, pp. 97-102.
8. J.M. Martinez, R. Koenen, and F. Pereira, ''MPEG-7, *IEEE MultiMedia,* vol. 9, no. 2, 2002, pp. 78-87.
9. Y. Chu, L.-T. Chia, and S.S. Bhowmick, ''Mapping, Indexing and Querying of MPEG-7 Descriptors in RDBMS with IXMDB,'' *J. Data and Knowledge Engineering* (DEK), vol. 63, no. 2, 2007, pp. 224-257.
10. U. Westermann and W. Klas, ''PTDOM: A Schema-Aware XML Database System for MPEG-7 Media Descriptions,'' *Software: Practice and Experience,* vol. 36, no. 8, 2006, pp. 785-834.
11. O. Wust and O. Celma, *An MPEG-7 Database System and Application for Content-Based Management and Retrieval of Music,* Univ. Pompeu Fabra, 2004.
12. J.M. Hellerstein, J.F. Naughton, and A. Pfeffer, ''Generalized Search Trees for Database Systems,'' *Proc. 21st Int'l Conf. Very Large Databases,* Morgan Kaufmann Publishers, 1995, pp. 562-573.

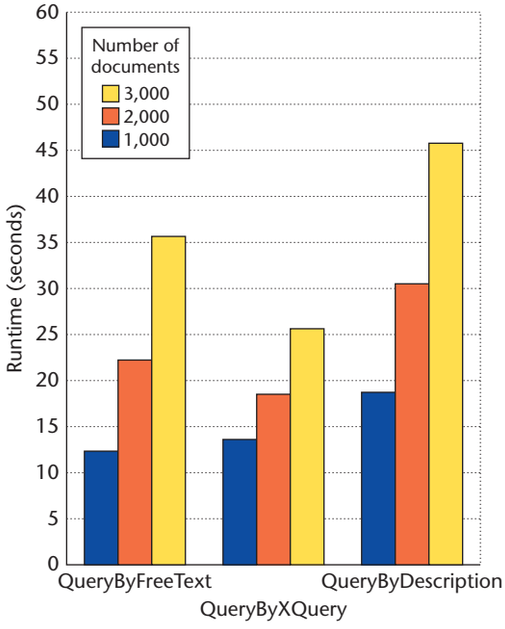

Figure 6. Runtime results of the MPEG Query Format evaluation.

developed in other research.[12] The extracted information is transformed into a QueryByDescription command and executed. The QueryByMedia evaluation currently supports only exact-match retrieval.

Query processing evaluation

We evaluated the performance of the query-processing module on a small test set of images. We performed the tests on a Windows PC with 4 Gbytes of RAM and a 3-GHz Pentium processor running Oracle 11g. Figure 6 presents the average results of five runs processing the query types mentioned here on a data set containing the MPEG-7 descriptions of 1,000, 2,000, and 3,000 images. The test measured the overall query response time for the execution of an individual query type. In further investigations, we will consider more detailed evaluations, including time and memory complexity, optimization capabilities, and query-type combinations.

The results show that the XQuery operation performs best, followed by the QueryByFreeText and the QueryByDescription. The QueryByDescription operation shows an acceptable runtime, considering that we used no indexing mechanism (in contrast to the QueryByFreeText operation). As mentioned, most query types operate on one table containing all MPEG-7 documents in XMLType instances. However, the database model of MPEG-7 MMDB also features an object-relational schema where the content of the individual MPEG-7 instances can be segmented in nested tables. Ongoing research in this area no doubt will explore the implementation of these query types in this alternate database schema.

Conclusion

The validator framework was approved at the 86th MPEG meeting in October 2008.[8] In our future work in this area, we plan to work on semantic rules for other standard committees. JPEG, for example, uses a subset of the MPQF in its JPSearch project.[13] We also plan to continue to focus on the query-processing engine for MPQF and its integration into the MPEG-7 MMDB. Currently, the processing engine supports the evaluation of QueryByFreeText, QueryByXQuery, QueryByDescription, and QueryByMedia. But there is more work to be done. **MM**

Acknowledgment

This research was supported in part by the Theseus Program, which is funded by the German Federal Ministry of Economics and Technology.

References

1. M. Cord, P.-H. Gosselin, and S. Philipp-Foliguet, ''Stochastic Exploration and Active Learning for Image Retrieval,'' *Image and Vision Computing,* vol. 25, no. 1, 2007, pp. 14-23.
2. A.G. Hauptmann, R. Jin, and T.D. Ng, ''Video Retrieval Using Speech and Image Information,'' *Proc. SPIE,* vol. 5021, 2003, pp. 148-159.
3. S. Pauws, ''CubyHum: A Fully Operational Query by Humming System,'' *Proc. 3rd Int'l Conf. Music Information Retrieval* (ISMIR), IRCAM-Centre Pompidou, 2002, pp. 187-196.
4. Y. Chu, L.-T. Chia, and S.S. Bhowmick, ''Mapping, Indexing and Querying of MPEG-7 Descriptors in RDBMS with IXMDB,'' *J. Data and Knowledge Engineering* (DEK), vol. 63, no. 2, 2007, pp. 224-257.
5. M. Döller and H. Kosch, ''The MPEG-7 Multimedia Database System (MPEG-7 MMDB),'' *J. Systems and Software,* vol. 81, no. 9, 2008, pp. 1559-1580.
6. U. Westermann and W. Klas, ''PTDOM: A Schema-Aware XML Database System for MPEG-7 Media Descriptions,'' *Software: Practice and Experience,* vol. 36, no. 8, 2006, pp. 785-834.

7. Mario Döller et al., ''The MPEG Query Format: On the Way to Unify the Access to Multimedia Retrieval Systems,'' *IEEE MultiMedia,* vol. 15, no. 4, 2008, pp. 82-95.
8. M. Döller et al., *WD1.0 of ISO/IEC 15938-12/Amd.1 MPEG Query Format Ref. Soft & Conf., N10257,* MPEG Requirements Group, 2008; http://www.chiariglione.org/mpeg/.
9. J.M. Martinez, R. Koenen, and F. Pereira, ''MPEG-7,'' *IEEE MultiMedia,* vol. 9, no. 2, 2002, pp. 78-87.
10. W3C, *XQuery 1.0: An XML Query Language,* 2007; http://www.w3.org/TR/2007/REC-xquery-20070123/.
11. World Wide Web Consortium (W3C), *RDF Primer,* 1999; http://www.w3.org/TR/REC-rdf-syntax/.
12. M. Lux, W. Klieber, and M. Granitzer, ''Caliph & Emir: Semantics in Multimedia Retrieval and Annotation,'' *Proc 19th Int'l Codata Conf.,* The Information Society: New Horizons for Science, 2004, pp. 64-75.
13. M.K. Leong et al., *JPSearch Part 1: System Framework and Components. ISO/IEC JTC1/SC29 WG1 TR 24800-1:2007,* JPEG, 2007.

Mario Döller is an assistant professor at the University of Passau, Germany. His research interests include multimedia search engines, mobile multimedia applications, and multimedia content description. Döller has a PhD in applied computer science from the University of Klagenfurt, Austria. He is an editor of MPEG (MPEG Query Format, MPEG-7 audio) and JPEG (JPSearch) standardization committees. Contact him Mario.Doeller@uni-passau.de.

Armelle Natacha Ndjafa Yakou is a PhD student in computer science jointly supervised by the INSA Lyon in France and the University of Passau in Germany. Her research interests include data spaces, medical information systems, information retrieval, and distributed multimedia systems. Yakou has a MSc in computer science from the University of Passau. Contact her at ndjafa@fim.uni-passau.de.

Ruben Tous is an assistant professor at the Department of Computer Architecture, Polytechnic University of Catalonia, Barcelona. His research interests include metadata interoperability, semantic-driven multimedia retrieval, and knowledge representation, and reasoning for multimedia understanding. Tous has a PhD in computer science from the technology department of the Universitat Pompeu Fabra, Barcelona. Contact him at rtous@ac.upc.edu.

Jaime Delgado is a full professor in the Computer Architecture Department of the Polytechnic University of Catalonia, Barcelona, and is head and founder of the DMAG research group. His research interests include electronic commerce, digital rights management, metadata, multimedia content, security, and distributed applications. Delgado has a PhD in telecommunication engineering from the Polytechnic University of Catalonia. Contact him at jaime.delgado@ac.upc.edu.

Matthias Gruhne is a senior research engineer at Fraunhofer Institute for Digital Media Technology, a research institute of applied technology in Ilmenau, Germany. His research interests include the development of audio identification techniques and music information retrieval. He works on the MPEG standardization committee, where he is as editor of the MPEG-7 metadata and the MPQF standards. Contact him at ghe@idmt.fhg.de.

Miran Choi is a researcher at Electronics Telecommunication Research Institute, Korea. Her research interests include natural language processing, information retrieval, and the Semantic Web. Choi has a PhD in computer science from Chungnam University, Korea. Contact her at miranc@etri.re.kr.

Tae-Beom Lim is a project manager in the ABCS Team, Digital Media Center, Korea Electronics Technology Institute, Korea. His research interests include digital broadcasting solutions, personalized digital TV solutions, embedded systems, and operating systems. Lim has a master of engineering in computer science from Sogang Graduate School, Korea. Contact him at tblim@keti.re.kr.