XQuery Framework for Interoperable Multimedia Retrieval

Mario Döller, Florian Stegmaier, Alexander Stockinger, Harald Kosch
Chair of Distributed Information Systems
University of Passau
94034 Passau, GERMANY
firstname.lastname@uni-passau.de

ABSTRACT

Multimedia retrieval relies on the underlying metadata format for effective querying of multimedia information. Most of the metadata formats are XML-based (for instance MPEG-7 or P/META). In this context, the XQuery query language is a natural choice for querying these data based on exact matches. However, XQuery lacks in expressing and evaluating multimedia specific requests (e.q., spatial, fuzzy requests). Therefore, the MPEG Query Format (MPQF), a novel XML based query language tuned for standardized multimedia requests, has been developed. Based on this, the paper introduces a MPQF aware XQuery framework which features a.) a plug-in architecture for external multimedia routines, b.) an automatic approach for MPQF to XQuery transformation and c.) an injection of information retrieval capabilities to XQuery (e.g., scoring, ranking). Besides, the framework can be adopted to any available XQuery repository and allows the retrieval in any XML based multimedia metadata format.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query Processing—*Multimedia Databases*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query Formulation*

Keywords

XQuery, MPEG Query Format, Multimedia retrieval

1. INTRODUCTION

Retrieving information in multimedia repositories is one of the major challenging tasks in the multimedia life cycle. Whenever, multimedia retrieval is discussed, one has to deal with the related metadata (formats) which are often XML based (e.g. MPEG-7[1]). In series, by investigating XML based retrieval techniques, one finally ends up by the well known and established XQuery[2] language. The XQuery language has its strengthens in expressing data centric and exact queries over XML data such as *Give me all images whose filesize > 100 kByte*, but lack the ability to express fuzzy requests common in multimedia retrieval (e.g., Query-By-Example). To fill this gap, a new multimedia query language, the MPEG Query Format [5] (MPQF) has been standardized in late 2008 by MPEG (formally known as ISO/IEC SC29 WG11). The query language addresses XML based metadata formats (e.g., MPEG-7) and combines data and information retrieval components as well as management functionalities.

In this context, the paper contributes with a XQuery framework for multimedia search that features a.) a plug-in architecture for external multimedia routines, b.) an automatic approach for MPQF to XQuery transformation and c.) an injection of information retrieval capabilities to XQuery (e.g., scoring, ranking). Besides, the framework can be adopted to any available XQuery repository and allows the retrieval in any XML based multimedia metadata format. Another benefit of adopting XQuery for multimedia retrieval is the broad diversity of available XQuery tools (databases, parsers, etc.).

The paper uses the MPEG Query Format (MPQF). The definition of the format is out of scope of this paper and has been published elsewhere. Readers not familiar with MPQF may look in [5] for detailed information. Further note, the specified transformation model presents only selected transformation rules and an extended version of this paper can be found at: `http://dimis.fim.uni-passau.de/iris/GI_Workshop_extended.pdf`.

The reminder of this paper is organized as follows: Section 2 introduces related work in the area of XQuery extensions for fuzzy retrieval. This is followed by Section 3 where the proposed MPQF to XQuery framework is described. In this context, Section 4 specifies our mapping approach for a MPQF to XQuery transformation. The specification is evaluated by a small example in Section 5. Performance analysis results are presented in Section 6. Finally, this article is concluded in Section 7.

2. RELATED WORK

As highlighted in the introduction, multimedia retrieval considers (to an high extend) multimedia metadata which is often XML-based (e.g., TV-Anytime, MPEG-7, etc.). In this context, in the past several query languages that are especially designed for XML data have been developed such as

[1]`http://mpeg.chiariglione.org/`

[2]`http://www.w3.org/TR/xquery/`

23rd GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 31.05.2011 - 03.06.2011, Obergurgl, Austria.

XIRQL [4], but the most well known representative approach is XQuery [8]. XML based query languages are strong in expressing data centric, but lack the ability to express fuzzy requests common in multimedia retrieval (e.g., Query-By-Example). There are already some approaches (e.g., VeX-Query [9]) aiming to extend XQuery in this direction. However, none of them is completely adequate for multimedia retrieval in terms of missing support for weighting of query terms (to reflect user preferences) or support for temporal or spatial retrieval etc. Further approaches that extend XQuery for fuzzy retrieval can be summarized as follows: Early works (e.g. [6]) introduced an XQuery rank operator for the evaluation of information retrieval request that target on an estimation of the *relative relevance of documents within document collections*. The integration of a vector space model and an associated *vscore* function has been presented in [7]. The *vscore* function returns the similarity degree between a query vector and a content element vector. Recently, in [3], the authors proposed a fuzzy XQuery processing technique which allows the users to use linguistic terms based user-defined functions in XQuery instances. The approach has been implemented in the native eXist XML database and provides better output than normal XQuery language execution.

3. MPQF BASED XQUERY FRAMEWORK

This section describes the overall structure and architecture of the proposed MPQF based XQuery framework.

3.1 Architecture

Figure 1 presents the overall architecture and workflow of the framework. First, the incoming MPQF request is parsed by an internal MPQF parser which establishes a visitor pattern syntax tree.

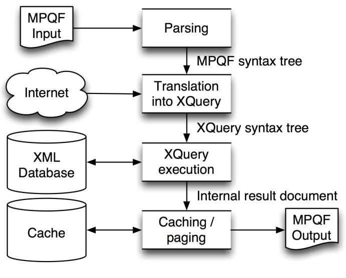

Figure 1: Overview of the system architecture

The MPQF syntax tree is forwarded to the transformation module which executes a mapping algorithm for producing an equivalent XQuery request. The plug-in system (see Subsection 3.2) supports the integration and evaluation of external information retrieval routines (e.g. query by example). After the finalization of the transformation process, a final XQuery instance is available. In series, this XQuery request is forwarded to the connected XQuery database for execution.

3.2 Plug-in system

In order to support a flexible system, the framework introduces a plug-in system. Figure 2 demonstrates where plug-ins take action in the transformation life cycle.

The framework identifies different categories of plug-ins: support for individual query types (e.g., QueryByMedia),

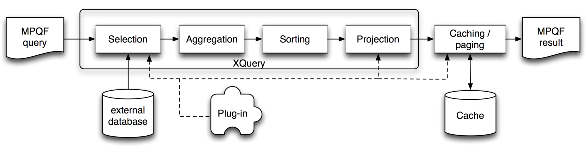

Figure 2: Overview of plug-in injection

extension of the XQuery language set (e.g., power function), data model dependent transformations (e.g., MIME type filtering) or functionality specific for the underlying XQuery database. In general, a plug-in is a Java module which receives a fixed set of input parameters and is able to manipulate the internal context of the transformation process. For instance, the plug-in module for our QueryByMedia implementation receives as input parameters the possible elements and attributes of the specified MPQF query type (e.g., *matchType* and *MediaResource*). Then, the module performs a similarity search at an external PostgreSQL[3] database where the ScalableColor features of MPEG-7 are stored. After finalization, the module responds with the results as presented by Transformation Rules 7 and 8.

4. MPQF TO XQUERY TRANSFORMATION RULES

The transformation formalism presented in this paper concentrates on the implementation of a global framework for fuzzy retrieval within XQuery based on the MPQF standard. In general, the rules distinguish between *data model depended* and *data model independent* transformation. That is, *data model depended* rules need to be adopted to the underlying metadata format as the required information can not be addressed by a given XPath[4] within the MPQF query request but need additional domain specific data.

4.1 Transformation Rules for Selection

The transformation rules for selection cover the mapping of the filtering step which is described by the *QueryCondition* element in an MPQF request. In this context, the skeleton for integrating a single MPQF condition (e.g., query type or comparison condition) into XQuery is implemented as follows:

TRANSFORMATION RULE 1 (QUERY CONDITION). *Let $\ll Score_{-i} \gg$ be a substitute for the transformation of an expression (e.g, comparison evaluation see Transformation Rule 6), a boolean operator (see Transformation Rule 5) or a query type which refers to external processing (see Transformation Rule 8) for e.g., information retrieval evaluation. Furthermore, let $\ll Threshold_{-i} \gg$ be the user given limit of the minimum similarity score for an operation (where $1 \leq i \leq n$ and $n \in \mathbb{N}$). Then the transformation of a MPQF query condition to XQuery is embedded as:*

```
let
   $scoreVar_i := <<Score_i>>,
   ...
where
   $scoreVar_i >= <<Threshold_i>>
```

Note, $scoreVar$_{-i}$ is a variable keeping the score value of the evaluation.

[3]http://www.postgresql.org/
[4]http://www.w3.org/TR/xpath20/

Constitutively on the abstract transformation of query conditions to score values, a closer look to its specific substitutions has to be performed. As defined in MPQF, an individual query condition can contain comparison, string and arithmetic operations. Exemplarily for this set of operations, the contains operation (see Transformation Rule 2) and comparison operation (see Transformation Rule 3) are defined.

Transformation Rule 2 (Contains Operator). *Let $\ll Op_i \gg$ be the respective operands of a MPQF contains condition where $1 \leq i \leq 2$. Then the corresponding transformation to XQuery is defined as:*

```
contains(<<Op1>>, <<Op2>>)
```

Transformation Rule 3 (Comparison Condition). *Let $\ll Operand\_i \gg$ be the respective operands of a MPQF comparison operation where $1 \leq i \leq 2$ and $\ll Operator \gg$ is the substitute of one of the defined MPQF operators (e.g., EQUAL to =). Then the transformation of a MPQF comparison condition to XQuery is defined as:*

```
<<Operand_1>> <<Operator>> <<Operand_2>>
```

A special role plays the fuzzy boolean operators as they combine the results of preceding evaluations by the means of scoring functions. In this context, the fuzzy boolean operators of MPQF (OR, AND, etc.) are transformed as follows:

Transformation Rule 4 (Boolean operator). *Let $\ll scoreVar\_i \gg$ be the score value as described in Rule 1. Further, $\ll prefVar\_i \gg$ specifies the given user preference value (where i in [1 .. n] and $n \in \mathbb{N}$) for the respective operation. Then, the mapping of fuzzy boolean operators (AND, OR, XOR) follows the following interface:*

```
<<OP>>
(<<scoreVar_1>>,<<prefVar_1>>,...,<<scoreVar_n>>,<<prefVar_n>>)
```

Based on the abstract definition for fuzzy boolean operators in Rule 4 one example for a fuzzy AND operator is specified in Rule 5.

Transformation Rule 5 (AND Boolean operator). *Let $\ll scoreVar\_i \gg$ and $\ll prefVar\_i \gg$ be specified as presented in Rule 4 and $\$scoreVar\_(N+1)$ the variable for holding the final result. Then, a mapping for the AND fuzzy boolean operator to a scoring function using the product t-norm is defined as follows:*

```
$scoreVar_(N+1) := math:pow (
                    math:pow(<<scoreVar_1>>, <<prefVar_1>>)*
                    ...
                    math:pow(<<scoreVar_n>>, <<prefVar_n>>),
                   <<N>>)
```

Note, as the MPQF boolean operators rely on the fuzzy set theory, the scoring functions should cope the t-norm and t-conorm fuzzy logics [2], respectively. Further note, this Rule assumes that the XML engine provides specific mathematic libraries or is extensible in this direction. Otherwise, lookup tables containing precalculated values in the interval [0.0 .. 1.0] can be provided (e.g., has been applied as plug-in for the Berkeley DB XML).

The integration of data centric evaluations (e.g., comparison operators) which base on a true/false basis is applied by Transformation Rule 6.

Transformation Rule 6 (Expression). *Let $\ll Expression \gg$ be any expression derived by the Rules 2 and 3. Then the score of those operations is gathered by the following transformation:*

```
if (<<Expression>>) then 1.0 else 0.0
```

For the evaluation of information retrieval related techniques (e.g., the QueryByMedia or SpatialQuery query type) separate external processes have to be applied. These processes filter the document set and produce document id and score value pairs which are integrated into the final XQuery request. For instance, as described in Subsection 3.2, the QueryByMedia query type can be implemented by forwarding this part of the MPQF request to a specialized similarity search engine for image retrieval by example. The result of such an evaluation is then integrated as specified by Rule 7.

Transformation Rule 7 (Plug-in Integration). *Let $qbmVar be the container variable for results of an external evaluation and $\ll anyURI\_i \gg$ an unique identifier of an XML instance document and $\ll anyScore\_i \gg$ its respective score value (where i in [1 .. n] and $n \in \mathbb{N}$). Then, the external result is integrated by the following format:*

```
$qbmVar := (
  <qbm>
    <doc id='<<anyURI_1>>' score='<<anyScore_1>>'/>
    ...
    <doc id='<<anyURI_n>>' score='<<anyScore_n>>'/>
  </qbm>
)
```

Note, this approach assumes that there is a unique identifier for every document. However, parts of a description (e.g. low level features) can be swapped to specialized retrieval stores but the unique identifier remains as link between those parts.

Besides, the intermediate result set (stored in a variable) is integrated into the overall evaluation by Rule 8 supporting the access to already existing score results.

Transformation Rule 8 (Plug-in Evaluation). *Let $\ll qbmVar \gg$ be the container as specified in Rule 7. Then, access to individual score values is accomplished as follows:*

```
 if (exists($qbmVar/doc[@id = base-uri($doc)]))
 then (number($qbmVar/doc[@id = base-uri($doc)]/@score))
 else (0.0)
```

The *TargetMediaType* element of MPQF restricts the multimedia data set according to their mime type. This filtering is data model depended as no additional information (e.g. XPath to the data) is provided within the MPQF query itself. Therefore, the evaluation is embedded as follows:

Transformation Rule 9 (MIME type). *Let $\ll MIME\_type\_i \gg$ be defined as a MIME type description where i in [1 .. n] and $n \in \mathbb{N}$. Then, the filter criterion extends the XQuery where clause as follows:*

```
where
  {<<MIME_type_1>> OR
   ....
   <<MIME_type_n>>} AND
```

Finally, the resulting documents are ordered by their score evaluation and stored in an internal format for further processing (see Rule 10).

Transformation Rule 10 (Ordering). *Let $scoreVarN contain the final score value after N calculation steps, then the resulting documents are ordered and preliminary stored as follows:*

```
where
   ...
order by
    $scoreVarN descending
return
  <Doc score='{$scoreVarN}' id='{$id}'>{$doc}</Doc>
```

4.2 Transformation Rules for Projection

In a final step, the desired information is extracted from the filtered documents and integrated in a valid MPQF output instance. As the wanted elements are addressed as XPath expressions within an MPQF query, they can be recycled in the transformation as well. Note, the final MPQF query result is embedded in an internal proprietary format in order to support enhanced functionalities such as caching, paging, relevance feedback, etc. by the framework.

4.3 Transformation process

The so far introduced Transformation Rules (TR) describe techniques for mapping parts of a MPQF request to its equivalents in XQuery. The overall transformation process creates a rule chain during the evaluation of the query in order to map the entire MPQF request. Input of the chain is the MPQF request. Then, a post order traversal is applied which responds with a list of nodes (MPQF conditions) of the *QueryCondition* element. By parsing this list, the type of the current node is identified and the respective (set of) Transformation Rule(s) is/are accomplished. Then, the algorithm applies optional Rules for existing aggregation or sorting parts. Finally, the rest of MPQF's *OutputDescription* element is evaluated by applying the Projection rule (not shown in this paper). Finally, the output of this mapping process is an equivalent XQuery instance.

5. EXAMPLE TRANSFORMATION

For a better understanding of the defined Transformation Rules, a simple example transformation is demonstrated by the following MPQF query request (see Figure 3)[5]. The example request addresses MPEG-7 based image descriptions and selects the *title* and the *creator information* of all *JPEG* images whose file size is *greater or equal* to *500000* Bytes and where the creators *family name contains* the string *Bob*. Besides, the *threshold* of the combined score result must exceed *0.5*. Furthermore, the final result set should contain not more then *30* elements.

The processing engine of the incoming query tree applies a post order traversal to extract the internal nodes. For our example, this results on the following sequence: *Contains, GreaterThanEqual, AND, TargetMediaType.* Then, by traversing this sequence, the assigned transformation rules are executed.

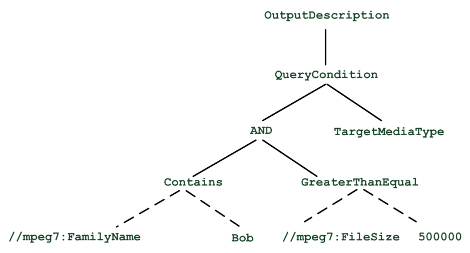

Figure 3: Example tree

In the following, Subsection 5.1 describes the used rules for the selection phase. Finally, in the project phase (see Subsection 5.2) the desired information is extracted and the entire XQuery is consolidated.

[5]Note, the request as MPQF language can be found in the extended version of this article.

5.1 Selection phase

The transformation mechanism starts by parsing all elements of the generated sequence. The first element is the *contains* condition which belongs to the set of *expressions*. and triggers the execution of Transformation Rule 2. This results in the following XQuery snippet (see Code 1).

Code 1 Transformation of the *contains* condition

```
contains($doc//mpeg7:FamilyName, 'Bob')
```

This is followed by Transformation Rule 6 which is used for the integration of expressions into XQuery (see Code 2)

Code 2 Integrating the *contains* condition

```
$scoreVar1 = if (contains($doc//mpeg7:FamilyName, 'Bob'))
           then 1.0 else 0.0
```

Similarly to the *contains* condition, the next element in the sequence is processed, namely the *GreaterThanEqual* condition. Here, in our example, the Transformation Rule 3 followed by Rule 6 are evaluated, which results in the XQuery snippet given in Code 3.

Code 3 Transformation of GreaterThanEqual

```
$scoreVar2 = if ($doc//mpeg7:FileSize >= 500000)
             then 1.0 else 0.0
```

After applying the Transformation Rules for the leaf nodes, our approach concentrates on the inner nodes (the boolean operators). The inner nodes regulate how the results of the leaf nodes are combined. For this purpose, scoring functions are assigned to the respective boolean operators according to t-norm and t-conorm rules. Our example uses the *Product* function for the AND operation in order to combine the individual score values. In this context, applying Transformation Rule 5 results in Code 4 for the *AND* condition.

The resulting score value and the threshold of an condition are integrated into the XQuery request by evaluating Transformation Rule 1. Due to space constraints in this paper, the final result for integrating the given thresholds is shown in Code 5. If no threshold is assigned, the minimum value is used (0).

As our example makes use of the *TargetMediaType* element for restricting the result set according to the file format, a data model depended filtering has to be found. In our example, we assume that the respective information is annotated in the *Content* and *FileFormat* elements of the MPEG-7 description. In assuming so, the following transformation for MIME-types (see Rule 9) has to be applied (see Code 6).

The final result of all combined transformations of the selection phase can be found in Code 7.

5.2 Projection phase

The last stage in the transformation process is the creation of a valid MPQF response and the integration of the requested information of the target data model. Our example only instantiates the *TextResult* and *Description* elements. As described beforehand, the final MPQF output description is wrapped in a proprietary format (*ResultDocu-*

Code 4 Applying Transformation Rule for scoring function

```
$scoreVar3 = math:pow(math:pow($scoreVar1, 0.5)*
          math:pow($scoreVar2, 0.5), 2)
```

Code 5 Integration of threshold values

```
$scoreVar1 >= 0.0 and
$scoreVar2 >= 0.0 and
$scoreVar3 >= 0.5
```

ment element) in order to support caching, paging, relevance feedback, etc.

6. EVALUATION

This section describes the series of experiments we performed in order to evaluate the effectiveness of our transformation approach. The tests were carried out on a subset of the CoPhiR[6] [1] data set containing MPEG-7 annotations of Flickr images. The sizes of our test data sets varied from 100 up to 10000 annotations. In order to demonstrate the transformation approach with various XML databases, the following solutions have been chosen: Saxon[7], Berkeley DB XML[8] and eXist DB[9].

The overall performance evaluation is divided into two main parts. First, the processing time of parsing and executing the Transformation Rules has been analyzed in Subsection 6.1. The final execution of the resulting XQuery instance at the mentioned databases is demonstrated in Subsection 6.2.

6.1 Transformation Evaluation

This subsection describes the set of experiments for evaluating the performance of applying the Transformation Rules. Input is a MPQF query request and output an equivalent XQuery query request. In order to receive clear differences between the used query classes a less powerful system configuration has been applied, namely an Intel Premium M 1.60 GHz CPU with 512 MB DDR2 400 main memory running Windows XP.

The complexity of the queries is divided into the following six classes in order to demonstrate the evaluation time for different compositions of Transformation Rules: queries where either aggregation or sorting is used (classes NoAgg/Sort, NoAgg/Sort Complex, Agg/NoSort), queries where both (class Agg/Sort) and queries where none (class NoAgg/NoSort) of these features have been used. Except the NoAgg/Sort Complex class, the example queries contain only one condition (e.g. EQUAL condition). The complex query class demonstrates the use of Boolean operators (AND/OR) and multiple other conditions (e.g., contains or comparison).

Finally, the sixth query class addresses the performance of the plug-in system by demonstrating a QueryByMedia query type which requires external processing. The external processing has been realized by the integration of a relational

[6]http://cophir.isti.cnr.it

[7]http://saxon.sourceforge.net

[8]http://www.oracle.com/database/berkeley-db/xml/index.html

[9]http://exist.sourceforge.net/

Code 6 Transformation of MIME type filtering

```
(
exists($doc//mpeg7:Content[@href = 'image'])
and
string($doc//mpeg7:FileFormat/mpeg7:Name/text()) = 'JPEG'
)
```

Code 7 Result of selection phase

```
$selected :=
 (for
   $doc in collection('db.dbxml')/*
  let
    $scoreVar1 = if (contains($doc//mpeg7:FamilyName, 'Bob'))
              then 1.0 else 0.0,
    $scoreVar2 = if ($doc//mpeg7:FileSize >= 500000)
              then 1.0 else 0.0,
    $scoreVar3 = math:pow(math:pow($scoreVar1, 0.5) *
              math:pow($scoreVar2, 0.5), 2),
    $id := base_uri($doc)
  where
   ((
    exists($doc//mpeg7:Content[@href = 'image'])
    and
     string($doc//mpeg7:FileFormat/mpeg7:Name/text()) = 'JPEG'
   )) and
   $scoreVar1 >= 0.0 and
   $scoreVar2 >= 0.0 and
   $scoreVar3 >= 0.5
  order by
    $scoreVar3 descending
  return
   <Doc score='{$scoreVar3}' id='{$id}'>{$doc}</Doc>),
```

PostgreSQL[10] database coping with low level features (color, texture, etc.) of images. Similarity calculation has been simplified on the basis of color features and the Euclidean distance (no index has been used). Figure 4 presents the average run time needed for the transformation of an MPQF query request to an appropriate XQuery request. The tests have been repeated 50 times. The evaluation shows that there is a slight increase of time consumption depending on the increase of query complexity. However, the maximum differences between query classes do not increase 17%.

6.2 Database comparison

The experiments have been executed on a Windows based stand-alone PC with Intel Core i7 1,6GHz (4 cores) CPU and 4 GB main memory. All databases have been tested out of the box without optimization. Similar to Subsection 6.1 six different query classes have been used during the evaluation, whereas exemplarily only two are demonstrated in this article. Note, the presented performance behavior is also valid for the rest of the tests. Figure 5 show the results of our experiments for a query where sorting has been enabled. The y-axis describes the average processing time per document (average processing time divided by the amount of documents in the database) and the x-axis shows the amount of MPEG-7 documents stored in the database. The measured overall processing time for one MPQF query consists of the transformation time applied by our module and the time needed for processing the resulting XQuery.

By evaluating the performance results, one can identify a linear scaling of the Saxon and Berkeley DB XML engines, which is stable over the increasing size of the test data set. The impact of the initial phase needed by the engines can

[10]http://www.postgresql.org/

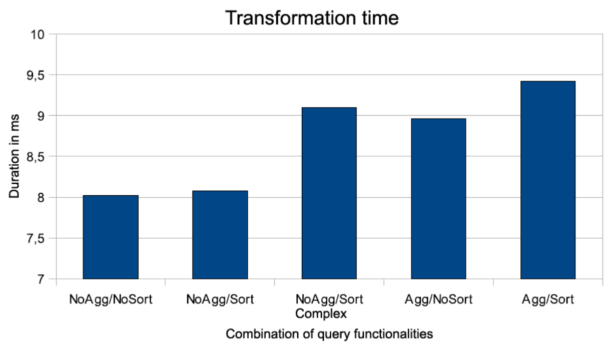

Figure 4: Performance of the transformation process

be observed by small test data sets (100 documents) and here the Berkeley DB XML is outperformed by the others. In contrast to Saxon and Berkeley DB, the eXist engine is outperformed clearly for larger test data sets as it shows a nearly quadratic scale factor.

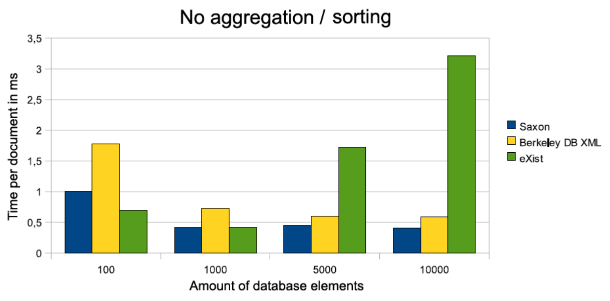

Figure 5: Comparison of database performance for XQuery with sorting

In general, the processing time for the MPQF-XQuery transformation is in average under 7% of the overall processing time for evaluating the final XQuery request and therefore negligible.

The last experiment targets on the evaluation of the plug-in injection process by the means of a QueryByMedia query type which realizes similarity search on multimedia data. As described beforehand, the test environment consists of a PostgresSQL database storing the low level features (ScalableColor of MPEG-7). As a proof of concept, similarity search is implemented by an SQL function in the target database. Of course, here is room for improvements (e.g., use of index structures or enhanced multimedia retrieval modules). Figure 6 show the results for the QueryByMedia evaluation.

7. CONCLUSIONS

This article proposed a MPQF based XQuery framework which provides a specification of a set of Transformation Rules for mapping a MPEG query format request to an equivalent XQuery request. Based on this, a framework has been developed featuring a plug-in system for external multimedia retrieval routines, a threading model for fast and scalable processing and an internal result set format enabling caching, paging and relevance feedback operations. The framework is able to connect to any available XQuery repository. By using the proposed framework, XQuery repositories can be enhanced for multimedia retrieval on any XML based multimedia metadata format in a standardized way.

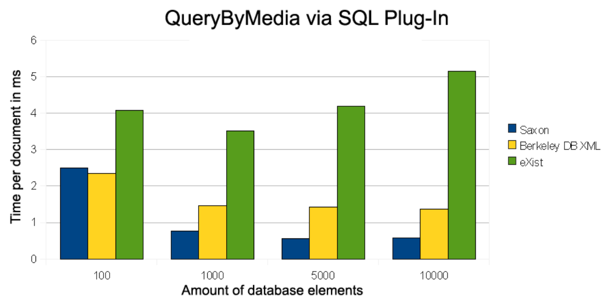

Figure 6: Comparison of database performance for XQuery with QueryByMedia plug-in

Future work will concentrate on further developments for the projection and output description part and the integration of additional plug-in elements coping for instance spatial and temporal retrieval.

8. ACKNOWLEDGMENTS

This work has been supported in part by the THESEUS Program, which is funded by the German Federal Ministry of Economics and Technology.

9. REFERENCES

[1] Paolo Bolettieri, Andrea Esuli, Fabrizio Falchi, Claudio Lucchese, Raffaele Perego, Tommaso Piccioli, and Fausto Rabitti. CoPhIR: a test collection for content-based image retrieval. *CoRR*, abs/0905.4627v2, 2009.

[2] Didier Dubois, Henri Prade, and Florence Sedes. Fuzzy Logic Techniques in Multimedia Database Querying: A Prelimiary Investigation of the Potentials. *IEEE Transaction on Knowledge and Data Engineering*, 13(3):383–392, 2001.

[3] E.J. Thomson Fredrick and G. Radhamani. Fuzzy Logic Based XQuery operations for Native XML Database Systems. *International Journal of Database Theory and Application*, 2(3):13–20, 2009.

[4] Norbert Furh and Kai Grossjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of the 24th ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 172–180, New Orleans, Louisiana, USA, 2001. ACM Press.

[5] ISO/IEC. Information technology - Multimedia content description interface - Part 12: Query format. *ISO/IEC 15938-12:2008*, 2008.

[6] Ji-Hoon Kang, Chul-Soo Kim, and Eun-Jeong Ko. An XQuery engine for digital library systems. In *Proceedings of the 3rd International ACM/IEEE-CS joint conference on Digital libraries*, pages 400–400, Houston Texas, 2003.

[7] Jacques Le Maitre. Indexing and Querying Content and Structure of XML Documents According to the Vector Space Model. In *Proceedings of the IADIS International Conference WWW/Internet*, pages 353–358, Lisbon, Portugal, 2005.

[8] Priscilla Walmsley. *XQuery*. O'Reilly Media, 2007. ISBN: 978-0596006341.

[9] Ling Xue, Chao Li, Yu Wu, and Zhang Xiong. VeXQuery: an XQuery extension for MPEG-7 vector-based feature query. In *Proceedings of the International Conference on Signal-Image Technology and Internet Based Systems (IEEE/ACM SITIS'2006)*, pages 176–185, Hammamet, Tunesia, 2006. Springer-Verlag.