An Extensible Query Language for Content Based Image Retrieval based on Lucene

Raoul Pascal Pein[1,2], Joan Lu [1], and Wolfgang Renz [2]

[1] Department of Informatics, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield HD1 3DH, United Kingdom , <R.P.Pein@hud.ac.uk> <j.lu@hud.ac.uk>
[2] Multimedia Systems Laboratory (MMLab), Faculty of Engineering and Computer Science, Hamburg University of Applied Sciences, Berliner Tor 7, 20099 Hamburg, Germany , <pascal.pein@haw-hamburg.de> <wr@informatik.haw-hamburg.de>

Abstract

One of the most important bits of every search engine is the query interface. Complex interfaces may cause users to struggle in learning the handling. An example is the query language SQL. It is really powerful, but usually remains hidden to the common user. On the other hand the usage of current languages for Internet search engines is very simple and straightforward. Even beginners are able to find relevant documents.

This paper presents a hybrid query language suitable for both image and text retrieval. It is very similar to those of a full text search engine but also includes some extensions required for content based image retrieval. The language is extensible to cover arbitrary feature vectors and handle fuzzy queries.

1. Introduction

After several years of research the idea of content based image retrieval (CBIR) [6, 17] is still not established in daily life. Currently most effort in CBIR is put into closing the semantic gap between simple visual features and the real image semantics. The work done in this area is very important to allow untrained users to work with image retrieval systems. A survey about such systems is available from Liu [9]. None of the systems analysed there is really capable of closing the gap completely. Either the solutions are far to specific or require much human attention. Liu concludes with the demand for "a CBIR framework providing a more balanced view of all the constituent components". The most important open tasks identified are "query-language design, integration of image retrieval with database management system, high-dimensional image feature indexing" and "integration of salient low-level feature extraction, effective learning of high-level semantics, friendly user interface, and efficient indexing tool" [9].

The cross-language image retrieval campaign ImageCLEF [5, 4] aims to evaluate different approaches of text and content based retrieval methods. The focus is set on a (natural) language independent solution for image retrieval exploiting both textual annotations as well as visual features. This effort also shows quite clearly the need for a powerful image retrieval system.

This paper introduces one approach to solve some of the claims stated above. It describes a *query language* which is designed to be reasonably *user friendly* and allows the integration of *high-level semantics* and *low-level feature extraction* in a single query.

Taking a look at current full text retrieval engines reveals the main differences to CBIR engines. Image retrieval inevitably contains fuzzy aspects. A search based on image features usually produces a list of results with decreasing similarity. In contrast a full text search can determine separate hit and miss lists, even if some fuzziness is added by language analysis (e.g. ignoring word endings).

Such a language must tackle the tasks of synthesizing simple result sets with fuzzy sets [7] as well as keeping the final result in a maintainable size. The latter requirement is important because every similarity above 0.0 is somehow part of the hits.

At the same time, query composing in CBIR environments is often much more difficult as there are no keywords for low-level features. The query language presented in this paper is rooted in the established area of text retrieval and is extended by essential CBIR related additions.

2. Related Work

Query Language The Lucene Query Language [10] is a full text retrieval language. The Lucene library comes with a parser which converts a query string into a query object. This object represents all query details and the search engine generates the result based on it. This language is not suitable to handle fuzzy results out of the box, but provides a simple and clear structure. It allows boolean and nested queries as well as the definition of document fields. These fields hold some meta information (i.e. title, content, author, ...) and can be used to compose reasonably complex queries.

With the development of object-oriented DBMS the ODMG-93 [3] standard emerged. The OQL query language [2] has been created. It combines SQL syntax with the OMG object model. An interesting extension to this language is called FOQL [12]. This language extension tries to capture fuzzy aspects which are required for CBIR applications. The FOQL approach is to attach a set of matching-methods to each stored objects. These methods are used to match any two objects of the same kind in a specific way. The resulting similarity is somewhere between 0.0 (no similarity) and 1.0 (identity). The newly introduced data type is the *Fuzzy-Boolean*. In addition the result can be limited by a threshold defining the minimum similarity.

Another query language is OQUEL [18, 19] which is designed to be user friendly. It is based on a simplified natural language and an extensible ontology. The system extracts a syntax tree from the query to retrieve images.

Data Description The feature vector paradigm states a plain list of several float values to create a vector. But looking at any random technique reveals that features may be composed in many different ways, containing probably complex data structures. These structures need to be mapped to the query language.

The language MPEG-7 [11] is rather a multimedia description than a query language. It is an emerging standard used in multimedia archives, often containing high-level semantic information. Using an XML based language for typed queries appears to be very unhandy and overly complex.

A possible alternative is the minimalistic approach in JSON. This sub set of JavaScript is an important part of the current Ajax technology. JSON is intended to be a simple data interchange format with minimal overhead.

3. Proposed Language Design

The proposed query language is based on the Lucene Query Parser [10] which defines a common language for full text search. It is intentionally chosen to provide beginners with a simple and familiar syntax. The language allows queries similar to those used in traditional search engines and the parser is generated by JavaCC.

This approach tries to merge the design principles of different languages. Some are like OQUEL [18] where queries are kept as simple and natural as possible. Others like SQL define a strict grammar to be highly machine readable.

There are two changes made to the Lucene grammar to fit the requirements of an extensible feature vector based query language: *fuzzy related operators* and a *nested two-layer grammar*.

The previous *boost* parameter for terms has been extended to multiple *TermParams* allowing additional control of fuzzy result sets.

To provide a high extensibility the grammar is split into two different layers.

The basic layer (see 3.1) is parsed and interpreted by the search engine directly. Here the grammar is predefined and fixed. Users may specify which meta information should be searched for by using fields. Images hold other fields than normal text documents, typically EXIF and IPTC information. Additionally a CBIR environment provides one or multiple feature vectors holding low-level information about the pixels. These feature vectors can be added by plug ins, each one having a unique identifier which is the field name for content based queries. The difficulty now lies in specifying how the query feature vector is entered. There are three different ways possible:

- ID of an image stored in the repository
- URI of a query image
- specification of the feature vector itself

The simplest way is to use an existing image for a query (*query-by-example*). Images already in the repository have the prepared feature vector available. Specifying the URI of an image requires the engine to load the image and to extract the feature vector. The most advanced and complicated way is to let the user specify a feature vector in detail.

As a custom feature vector may contain any kind of proprietary data, offering an all-embracing language is not possible. Thus a second layer is added to the query language. A *Term* may contain the string *<FEATURE_START> [<FEATURE_CONTENT>] <FEATURE_END>*. The parenthesized part *<FEATURE_CONTENT>* is extracted by the search engine and passed to the responsible plug in. The plug in is fully responsible for parsing and interpreting this string to return the object representation of the feature vector.

3.1. Grammar

```
Conjunction ::=  [ <AND> | <OR> ]

Modifiers ::= [ <PLUS> | <MINUS> | <NOT> ]

Query ::= ( Conjunction Modifiers Clause )*

Clause ::=  [ LOOKAHEAD(2)
  ( <TERM> <COLON> | <STAR> <COLON> ) ]
  ( Term | <LPAREN> Query <RPAREN> [TermParams] )

Term ::=
  (
     ( <TERM>  | <STAR> | <PREFIXTERM> |
       <WILDTERM> | <NUMBER>  | <URI> )
     [ <FUZZY_SLOP> ]
     [ TermParams [ <FUZZY_SLOP> ] ]
     | ( <RANGEIN_START>
         ( <RANGEIN_GOOP>|<RANGEIN_QUOTED> )
       [ <RANGEIN_TO> ]
         ( <RANGEIN_GOOP>|<RANGEIN_QUOTED> )
         <RANGEIN_END> )
       [ TermParams ]
     | ( <RANGEEX_START>
         ( <RANGEEX_GOOP>|<RANGEEX_QUOTED> )
       [ <RANGEEX_TO> ]
         ( <RANGEEX_GOOP>|<RANGEEX_QUOTED> )
         <RANGEEX_END> )
        [ TermParams ]
     |
       ( <FEATURE_START>
       [ <FEATURE_CONTENT> ]
        <FEATURE_END> )
        [ TermParams ]
     | <QUOTED>
       [<FUZZY_SLOP> ]
       [ TermParams ]
  )

TermParams ::=
    (
     <CARAT> boost (
      ([ <HASH> maxCount ]  [ <AT> threshold ])
    | ([ <AT> threshold ]  [ <HASH> maxCount ])
    )

    |   <HASH> maxCount (
      ([ <CARAT> boost ]  [ <AT> threshold ])
    | ([ <AT> threshold ]  [ <CARAT> boost ])
    )

    |   <AT> threshold (
      ([ <CARAT> boost ]  [ <HASH> maxCount ])
    | ([ <HASH> maxCount ]  [ <CARAT> boost ])
    )
)
```

3.2. Operators

The main difficulty of combining sub results from a CBIR system is the fuzzy nature of the results. Some simple features with filtering character (e.g. keywords) deliver a rather clean set of hits. But it is essential to have a a fuzzy model for merging these with highly similarity based features which results are usually a sorted list [7, 16].

The approach by Fagin [7] interprets results as *graded sets*, which are lists sorted by similarity and set characteristics. He uses the basic rules defined by Zadeh [20]:

- Conjunction:
  $\mu_{A \wedge B}(x) = min\{\mu_A(x), \mu_B(x)\}$ (AND)
- Disjunction:
  $\mu_{A \vee B}(x) = max\{\mu_A(x), \mu_B(x)\}$ (OR)
- Negation:
  $\mu_{\neg A}(x) = 1 - \mu_A(x)$ (NOT)

The text retrieval concept of *boosting* single terms by any float value is adapted to the extended engine. Before merging sub results, the similarities are boosted as specified to shift the importance into the desired direction.

An additional acknowledgement to the fuzzy nature is the use of additional set operators to keep the results at a reasonable size. The *minimum similarity* is a value between 0.0 and 1.0 and forces the engine to drop all results below this similarity threshold. As the efficiency of the threshold highly depends on the available images and features, a *maximum size* parameter limits the result to the specified size.

3.3. Plug-Ins

The plug in concept of the retrieval framework described in [15] allows the definition of any new feature. To make such a plug in available in this language, only a few requirements need to be met.

The plug in needs an identifier which is automatically used as a term field. With this information it is already possible to formulate queries containing an example image (either by internal id or URI).

The tricky part is to develop a syntax for user defined feature vector information embedded in a query. As features can be arbitrarily complex, it is intended to support a simple default language like JSON. Otherwise the embedded data string of a query is forwarded directly to the feature plug in where it needs to be converted into a valid feature object.

At this point, the plug in developer needs to decide on how to support wild cards. If a simple feature contains the RGB means of an image, the user could specify an array like *"[36, 255, *]"*. In this case the results should contain some red, dominant green and the rate of blue does not matter at all. Putting some more effort into the feature abstraction, a more convenient query like *"some red and very much green"*is also possible. This lies in the responsibility of the plug in developer.

3.4. Conversions/Alternative Representation

As the query language is based on the lucene toolkit, the parser automatically generates an object representation of the whole query. This query object could also be created by a suitable front end, but the stringified representation can be manipulated directly by users. Having an object representation of the query, converting it into XML or another standard is only a small step. Plug ins could also specify their feature conversion into XML and back to simplify the use of the MPEG-7 standard.

3.5. Examples

The following examples demonstrate the use of different language constructs, where the "keywords" field is the only text based one.

1. IPTC keyword:
   keywords:oystercatcher

2. external image, similarity at least 95%:
   histogram:"file://query.jpg"@0.95

3. wavelet of three images by internal ID:
   wavelet:(3960 3941 3948)

4. two histograms, maximum of 10 results each:
   histogram:3963#10^2.0 OR histogram:3960#10

5. spatial histogram without 50 images similar to image 190:
   spatial_histo:5456 -histogram:190#50

6. mean colour with embedded feature and filtering keyword:
   rgb_mean:($[200, 50, *]$) +keywords:car

Example query 1 is a simple text based query based on the IPTC meta information. It works exactly like every common full text retrieval. The field *keywords* is derived directly from the IPTC data and other fields such as *title*, *author* or *createdate* are also available.

More interesting are queries allowing CBIR relevant features. The fields are picked by the feature identifier and processed in the plug ins.

Number 2 searches for similarities based on a *histogram* plug in implementing a feature proposed by Al-Omari and Al-Jarrah [1]. An URI to an image is specified which is used for query-by-example. The engine loads the image and extracts the required query feature. The final result is limited to images with at least 95% similarity.

Query 3 calls the *wavelet* plug in which is an implementation of a feature by Jacobs et al. [8]. The query contains three internal image IDs. The engine performs three parallel sub retrievals and merges the three result lists by default with *OR*. Using the IDs shortens the query string itself and allows the engine to load the prepared feature vectors directly from the persistence.

Because of the fuzziness in CBIR it is not clear how many results are returned when giving a similarity threshold. Dependent on the quality of the feature implementation and the repository size, many thousands of images could have a similarity above a given threshold. This is usually a waste of resources because users want the result to appear in the first few hits, say the first result page. Query 4 presents the second way to keep the result size tight. Here the result set of each term is cut off after a maximum of 10 results. This restricts the maximum result size to $10 + 10 = 20$ images. Additionally the first term is boosted by factor 2, giving it a higher weight than the second term.

Having multiple feature plug ins opens an interesting new field to composing CBIR queries. Different features often mean very different result sets. The *NOT* modifier in query 5 shows an example how to remove unwanted content from the result. First the engine searches for the feature *spatial_histo*, which is a histogram with additional information about spatial colour distribution [14]. As this query might return several images which does not correspond to the wanted context, a *NOT* term filters out the 50 highest results similar to an unwanted result which are hopefully very similar in the simpler *histogram* space.

Finally the conjunction of the two different worlds is done by example 6. The first term searches for the content based *rgb_mean*. The embedded part within the brackets is interpreted by the simple *rgb_mean* plug in, where the three values stand for red, green and blue. The desired values for red and green are defined and the blue colour does not matter at all. Because this low-level feature is far too simple for efficient retrieval, a second term is specified. In this example the *keywords* field is mandatory (*AND*) and has a filtering effect. Only images containing the keyword "car" are allowed to be in the result.

4. Comparison

There are only a few query languages which try to tackle the task of merging aspects of full text and CBIR retrieval. Table 1 compares some of these languages and lists which important requirements are met.

The language presented in this paper represents the middleware of the previously described retrieval framework [15]. It is easily parseable and allows composition of any query that is supported by the framework. New feature plug ins extend the language automatically by adding a new field. Currently the language does not inherently support high-level concepts. A plug in for semantics could surely

Language	CBIR approach	fuzzy boolean	min threshold	user defined sorting	extensible (features)	AND-OR-NOT	weights	high-level concepts	simple structure	base language
this	feature driven	Y	Y	N	Y	Y	Y	N[1]	Y	Lucene
FOQL	object driven	Y	Y	Y	Y	Y	Y[2]	Y[3]	N	ODMG/OQL
OQUEL	ontology driven	Y	Y	N	N[4]	Y	Y	Y	Y	none/natural

[1] mapping only possible by nesting/meta features containing prepared low-level queries
[2] sum of partial weights must be 1.0
[3] keyword *define* to map low-level queries to high-level concepts
[4] ontology can be modified

Table 1. Languages Compared

be implemented with some effort by collecting pre-defined queries with low-level features.

FOQL appears to be too complex and thus unsuitable for untrained users. Nevertheless many concepts like *Fuzzy-Booleans* and *Fuzzy-Sets* are valuable. It is possible to add any kind of feature by defining an appropriate method for object comparison. Due to its complexity and sorting ability the language is adequate in SQL like environments.

A closer view to OQUEL reveals some interesting features. The language itself has been designed to be easy to use. Users only need to specify the desired features in simple words (e.g. "people in centre")[19]. It is very close to a natural language, however the ambiguity of these requires additional attention and a well designed ontology. Concepts of this language help creating a convenient user interface.

5. User Survey

A first small-scale user survey has been carried out to evaluate the language. The results are explained in detail in the related master thesis [13].

The test setup was a repository containing 6480 images with different levels of annotation, from no IPTC data at all up to a set of multiple descriptive keywords. The survey has been carried out with 5 testers with at least basic experience in computing sciences.

Tasks The tasks demand both CBIR and keyword based approaches. The basic tasks were: retrieving images based on a textual description or visual examples, tracking a given example image, optimizing queries (high Precision/Recall) for a specific content and ascertain the name of birds from given images.

Results After a short training time most testers were able to use both textual and content aspects in their queries. Mostly understandable features (colour mean, histogram) were used in combination with the query image IDs and the wavelet plugin was often ignored. The simple *rgb_mean* with its 3 values was a preferred feature. In some cases even the detailed histogram specification was tried out. As it was not allowed to draw query images, a popular approach was the use of random images to start with.

Most tasks were solved by the testers within less than 10 query iterations, but in some cases the available information and tools were not sufficient to ensure a quick success. The testers requested additional tools for *query-by-sketch* and complex feature composing.

Discussion In general the testers behaved as expected and solved the tasks. Additonal knowledge of certain image content (e.g. bird names) sometimes sped up the retrieval drastically. Where no or insufficient annotation was available, the search took significantly longer.

Ultimately the language still needs to be thoroughly tested in a full-grown usability study. It needs to be evaluated whether untrained and experienced users are both able to compose queries as intended, probably with additonal tool support.

6. Conclusion and Future Work

Achievements The proposed query language has a simple structure and is very similar to a full text search engine while also allowing fuzzy terms. Further it is easily extensible and allows arbitrary constructs for individual features. Complex queries are possible but not necessary, giving experts the chance to fine tune all parameters as required. Normal users could either enter simple queries or generate them with a graphical user interface.

Further the language can be easily mapped to machine readable formats like objects or XML.

Problems Remaining Providing a basic parser like JSON only simplifies the low-level query information. To support higher abstractions it is necessary to fully understand the feature itself, which is impossible for a generic language. For this reason, keeping the language simple is the task of the feature developers. They need to design appropriate sub languages which contain all feature specific information and remain as readable as possible.

Another issue is the naming of feature vector based fields. Currently the prototype compares each field name in the query with the list of available feature plug in identifiers. If the field name does not match a feature identifier, the term is handled by the underlying Lucene engine, executing a "classical" full text search on the field. Otherwise the term is forwarded to the corresponding feature plug in. Having overlapping feature identifiers, basic search fields could be hidden. It is necessary to formulate naming conventions like reserved words or a prefix for each feature identifier.

Future Work Unlike FOQL/SQL the language does not support user defined sorting like *ORDER BY* but sorts results by an overall similarity. It is to decide whether this extension is relevant for retrieval issues or not.

Depending on the combining functions and feature vectors used, query processing can be sped up significantly. A heuristic approach to query optimizing has been evaluated by Ramakrishna [16].

The support of high-level concepts is not realized yet. This could be a feature of the language itself by introducing constructs like *define* in FOQL which substitute certain terms by a pre-defined low level term. Alternatively the retrieval engine itself could be extended by high-level plug ins which map semantics to predefined low level requests. Developing such feature plug ins is a very complex task. A lot of testing is required to capture meaningful feature vectors information which represents semantics.

References

[1] F. A. Al-Omari and M. A. Al-Jarrah. Query by image and video content: a colored-based stochastic model approach. *Data Knowl. Eng.*, 52(3):313–332, 2005.

[2] A. M. Alashqur, S. Y. W. Su, and H. Lam. OQL: a query language for manipulating object-oriented databases. In *VLDB '89: Proceedings of the 15th international conference on Very large data bases*, pages 433–442, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[3] R. G. G. Cattell. ODMG-93: a standard for object-oriented DBMSs. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, page 480, New York, NY, USA, 1994. ACM.

[4] P. Clough, M. Grubinger, T. Deselaers, A. Hanbury, and H. Müller. *Evaluation of Multilingual and Multi-modal Information Retrieval*, volume 4730/2007, chapter Overview of the ImageCLEF 2006 Photographic Retrieval and Object Annotation Tasks, pages 579–594. Springer, 2007.

[5] P. Clough, H. Müller, T. Deselaers, M. Grubinger, T. M. Lehmann, J. Jensen, and W. Hersh. *Accessing Multilingual Information Repositories*, volume 4022/2006, chapter The CLEF 2005 CrossLanguage Image Retrieval Track, pages 535–557. Springer, 2006.

[6] J. Eakins and M. Graham. Content-based Image Retrieval. A Report to the JISC Technology Applications Programme. Technical report, University of Northumbria at Newcastle, Jan. 1999.

[7] R. Fagin. Combining fuzzy information from multiple systems (extended abstract). In *PODS '96: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 216–226, New York, NY, USA, 1996. ACM.

[8] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast Multiresolution Image Querying. *Computer Graphics*, 29(Annual Conference Series):277–286, 1995.

[9] Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma. A survey of content-based image retrieval with high-level semantics. *Pattern Recognition*, 40:262 282, 2007.

[10] Apache Lucene, 2006.

[11] J. Martinez, R. Koenen, and F. Pereira. MPEG-7: The Generic Multimedia Content Description Standard, Part 1. *IEEE MultiMedia*, 09(2):78–87, 2002.

[12] S. Nepal and M. Ramakrishna. Query Processing Issues in Image(Multimedia) Databases. *icde*, 00:22, 1999.

[13] R. P. Pein. Hot-Pluggable Multi-Feature Search Engine. Master's thesis, Hamburg University of Applied Sciences, 2008.

[14] R. P. Pein and Z. Lu. Content Based Image Retrieval by Combining Features and Query-By-Sketch. In H. R. Arabnia and R. R. Hashemi, editors, *IKE*, pages 49–55. CSREA Press, june 2006.

[15] R. P. Pein and Z. Lu. A Flexible Image Retrieval Framework. In Y. Shi, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *International Conference on Computational Science (3)*, volume 4489 of *Lecture Notes in Computer Science*, pages 754–761. Springer, may 2007.

[16] M. V. Ramakrishna, S. Nepal, and P. K. Srivastava. A heuristic for combining fuzzy results in multimedia databases. In *ADC '02: Proceedings of the 13th Australasian database conference*, pages 141–144, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[17] M. Renz and W. Renz. Neue Verfahren im Bildretrieval. Perspektiven für die Anwendung. In R. Schmidt, editor, *Proceedings der 22. Online-Tagung der DGI*, pages 102–128, May 2000.

[18] C. Town and D. Sinclair. Ontological query language for content based image retrieval. In *Content-Based Access of Image and Video Libraries, 2001. (CBAIVL 2001). IEEE Workshop on*, pages 75–80, 14 Dec. 2001.

[19] C. Town and D. Sinclair. Language-based querying of image collections on the basis of an extensible ontology. *Image and Vision Computing*, 22:251–267, 2004.

[20] L. A. Zadeh. *Fuzzy sets*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.