EXTENSIONS TO THE MULTIMEDIA RETRIEVAL MARKUP LANGUAGE – A COMMUNICATION PROTOCOL FOR CONTENT–BASED IMAGE RETRIEVAL

Henning Müller, Antoine Geissbuhler

University Hospitals of Geneva
Division of Medical Informatics
Rue Micheli-du-Crest 21
1211 Geneva 4, Switzerland
henning.mueller@dim.hcuge.ch

Stéphane Marchand–Maillet

University of Geneva
Computer Vision and Multimedia Lab
Rue du Général Dufour 24
1211 Geneva 4, Switzerland
marchand@cui.unige.ch

ABSTRACT

Content–based image retrieval (CBIR) or content–based visual information retrieval (CBVIR) has been one of the most active research fields in the computer vision domain during the last ten years. Many different tools and programs have been developed for the retrieval of images but there is still a lack of a standard to access retrieval systems in a unified manner. An accepted (and simple) protocol or query language such as SQL is in the database community or DICOM in the medical field has not yet been widely accepted. This is one of the main reasons that only rarely components are reused and it also hinders the inclusion of query engines into other programs such as document management systems. This lack of a communication protocol might also prevent commercial success of image retrieval applications.

One of the few open protocols for accessing and querying retrieval systems is the Multimedia Retrieval Markup Language (MRML). Descriptions are available and a set of clients as well as servers can be downloaded as open source free of charge. Although the protocol has a variety of interesting and important functions, there is still a lack in other parts such as simple database management functions and user/rights management.

This article gives recommendations of functions that are useful to implement and proposes some structural changes to build an MRML–server to handle the communication apart from the retrieval engine. When improving the functionality of MRML, some other necessities come up such as rights- and user management and an encryption of the communication because otherwise, the passwords are transmitted over the network in a human–readable manner (XML).

The GNU Image Finding Tool (GIFT) is taken as an example for implementing the changes but they are in the same way usable for other retrieval systems implementing MRML.

1. INTRODUCTION

A large number of commercial and academic retrieval systems exist for content–based access to visual data. Some of the earliest are QBIC [4] and Virage [1] for the commercial domain, Photobook [17] and Blobworld [3] as research prototypes. Although systems such as Photobook can be download from their web sides[1], not much attention has been spent on the reuse of components and the communication between the actual search engine and the user interface although this can make the reuse of components much easier.

A standardized and open communication between the search engine and interface has a number of advantages from the simple generation of meta search engines to completely automatic benchmarks [10]. Meta search engines have also been generated with varying access protocols [2] but with a single, open protocol, the effort can be reduced dramatically and new search engines can be added easily.

Several communication protocols have been proposed for the retrieval of multimedia and pictorial information. Most of them are extensions of database management systems and deal with extending them for the retrieval of multimedia data often still by exact matches of database fields. They frequently do not cover the communication between the user interface and the retrieval system. Only few deal with real visual similarity retrieval which is the main subject of this paper. Pictorial SQL (PSQL) [19] is one of the earliest protocols (1988) and mainly an extension of SQL allowing to query pictorial data for example by spatial properties. Pictorial Query By Example (PQBE) [16] is implementing the retrieval of direction relations of images that are represented in a symbolic way as relations among objects with an order in space. This system is mainly aimed at geographic applications where symbolic representations of images might be easier to achieve than with general stock

[1] `http://www-white.media.mit.edu/~tpminka/photobook/`

photography. MOQL (Multimedia Object Query Language) [9], based on OQL and has a large range of applications in mind including the description of varying media types and various applications. This includes the handling of spatial and temporal data. Still, it is basically an extension to include Multimedia data into database management systems.

Other query languages are MMQL (Multimedia Query Language) and CVQL (Content–based Video Query Language) An overview of several multimedia query languages can be found in [8]. Unfortunately, non of these protocols has gained widespread acceptance or defined a real standard. To our knowledge, there is no software available free of charge that implements these protocols in user interfaces and provides tools to help other research groups with such implementations.

With the Viper[2] project and its outcome, the GNU Image Finding Tool (GIFT[3], [21]), a standard retrieval language was defined, MRML[4]. As the software is free of charge under a GPL license and the sources are available, it is relatively easy to use the software to create clients and/or servers for MRML. Several projects are underway for the development of applications using MRML. Thus, MRML is not simply a language to formulate queries for multimedia objects but covers a broader and different range than other available protocols. It is not meant to be a query language like SQL where the user can formulate queries but it is meant for machine–machine communication between a query engine and a user interface, for example.

For sure, MRML has a number of very practical and interesting features such as the use of XML as a markup language and the possibility that a server can configure the interface depending on its needs (discussed in Section 2). Still, there is a need to not only connect to a query engine and execute queries via a standardized protocol but also to offer more functionalities, similar to those of database management systems.

The necessary security features are discussed in Section 3. The inclusion of database management functions into MRML is then the subject of Section 4. Section 5 finally gives our concluding remarks on the subject.

2. MRML

MRML was originally developed to separate the retrieval engine from the user interface. Such a standardized access proved to be of importance for a number of other applications as well. Detailed information on MRML can be found in [14] and a description of applications using MRML is given in [11, 13].

[2] http://viper.unige.ch/
[3] http://www.gnu.org/software/gift/
[4] http://www.mrml.net/

MRML is based on XML so that standard, freely–available parsers such as expat[5] or Apache's Xerces[6] can be used. MRML is a multi–paradigm protocol, offering features such as query by example (QBE), choice of databases, features or algorithms to use, and property sheets for specifying algorithm–specific parameters. It is extensible, so that private tags for special features of a system can easily be specified (even SQL can be embedded, if desired).

So far, a number of features are supported by MRML:

- connection of a client to a server;
- response of the server to the client with its capabilities; configuration of the interface according to the server's needs;
- query with image(s) by example, image browsing, or query by free text;
- return of query results as URLs with relevance scores;
- closing of a connection.

These features work in practice and have shown their utility in client applications using Java, PHP and CGI.

2.1. Example code for MRML

This section is not meant to replace the MRML definition but it gives a couple of simplified code examples to get an idea of MRML and its syntax. It also helps to group the proposed changes to MRML into the context to the existing definitions.

2.1.1. Connection to a server

When connecting to a server, the simplest command is to ask for certain properties of the server:

```
<mrml>
 <get-server-properties />
</mrml>
```

The server then informs the client of its capabilities.

```
<mrml>
 <server-properties />
</mrml>
```

With a connection the user can configure a session and transmit a user name and eventually a password. A session opening in reality where the user asks for the available collections on the server could look like this:

```
<mrml session-id="dummy_session_identifier">
 <open-session user-name="test"
  session-name="default_session" />
 <get-collections/>
</mrml>
```

[5] http://www.jclark.com/xml/expat.html
[6] http://xml.apache.org/

A reply to a request for all collections can look like this, where the server has one database with images (collections) available for the user:

```
<mrml>
 <collection-list  >
  <collection collection-id="c-tsr500"
   collection-name="TSR500"
   cui-algorithm-id-list-id="ail-inverted-file"
   cui-base-dir="/databases/TSR500/"
   cui-number-of-images="500"
   <query-paradigm-list  >
    <query-paradigm  type="inverted-file" />
   </query-paradigm-list>
  </collection>
 </collection-list>
</mrml>
```

Using similar, simple messages, the client can request a list of collections available on the server, together with descriptions of the ways in which they can be queried (query paradigms).

The client can open a session on the server, and configure it according to the needs of its user (interactive client) or its own needs (*eg.* benchmarking server). The client can also request the algorithms which can be used with a given collection:

```
<mrml>
 <get-algorithms
  collection-id="collection-1" />
</mrml>
```

This request is answered by sending the corresponding list of algorithms available for `collection-1`. This handshaking mechanism allows both interactive clients and programs (such as meta query agents or automatic benchmarkers) to obtain information describing the server.

2.1.2. Interface configuration

The client can request property sheet descriptions from the server. Varying algorithms will have different relevant parameters which should be user–configurable (*eg.* feature sets, speed vs. quality). *Viper*, for example, offers several weighting functions [20] and a variety of methods for pruning [12]. All these parameters might be irrelevant for other search engines. Thanks to MRML property sheets, the interface can adapt itself to these specific parameters. At the same time, MRML specifies the way the interface will turn these data into XML to send them back to the server.

Here is short example of an interface configuration:

```
<property-sheet
 property-sheet-id="sheet-1"
 type="numeric"
 numeric-from="1"
 numeric-to="100"
 numeric-step="1"
 caption="\% features evaluated"
 send-type="attribute"
 send-name="cui-percentage-features" />
```

This specifies a display element which will allow the user to enter an attribute with the caption "% of features evaluated". The values the user will be able to enter are integers between 1 and 100 inclusive. The value will be sent as an attribute *eg.* `cui-percentage-features="33"`. This mechanism allows the use of complex property sheets, which can send XML text containing multiple elements.

2.1.3. Query formulation

The query step is dependent on the query paradigms offered by the interface and the search engine. Here, QBE is taken as the essential example, but MRML is also being used for browsing queries and tests are being done with region–queries in MRML.

A basic QBE query consists of a list of images and the corresponding relevance levels assigned to them by the user. In the following example, the user has marked two images, the image `1.jpg` relevant (`user-relevance="1"`) and the image `2.jpg` non–relevant (`user-relevance="-1"`). All query images are referred to by their URLs.

```
<mrml session-id="1" transaction-id="44">
<query-step session-id="1"
 resultsize="30"
 algorithm-id="algorithm-default">
 <user-relevance-list>
  <user-relevance-element
   image-location="http://viper/1.jpg"
   user-relevance="1"/>
  <user-relevance-element
   image-location="http://viper/2.jpg"
   user-relevance="-1"/>
 </user-relevance-list>
</query-step>
</mrml>
```

The server will then return the retrieval result as a list of image URLs along with their similarity scores. A thumbnail URL can be transmitted as well to show the results on screen in a faster way.

```
<mrml session-id="1" >
<acknowledge-session-op  session-id="1" />
<query-result>
 <query-result-element-list>
  <query-result-element calculated-similarity=".9"
   image-location="http://viper/3.jpg"
   thumbnail-location="http://viper/3_thumb.jpg"/>
  <query-result-element calculated-similarity=".7"
   image-location="http://viper/4.jpg"
   thumbnail-location="http://viper/4_thumb.jpg"/>
 </query-result-element-list>
</query-result>
</mrml>
```

Queries can be grouped into transactions. This allows the formulation and logging of complex queries. This may be applied to systems which process a single query using a variety of algorithms, such as the split–screen version of *TrackingViper* [15] or the system described in [7]. It is important in these cases to preserve in the logs the knowledge that two queries are logically related one to another.

2.2. Version control in MRML

An essential point in MRML is that new elements can be added to the protocol without affecting old programs that do not know these attributes. Unknown attributes are simply ignored. Still, for this it is important to add elements in a way that no old functionalities are affected when the new elements are not known.

With new versions of MRML, it is essential to stay compatible with the older versions, so all clients and servers can still interact although new features might thus not be available in all applications. Especially when a big change in the structure is done, this might not always be simple. A suggestion is therefore to have an explicit version control for a certain session. The session can then completely be done in the version defined in the beginning. Such a version control allows to a client and server to stay compatible with a number of eventually different versions just by using a different DTD (Document Type Definition) or XML Schema.

To implement this in an easy way, we have to think how an MRML server can read in generic DTDs for different versions and then control the elements based on the version that is chosen in the session definition. This will also make it possible to automatically generate error messages for missing elements or badly formed queries or requests.

An element for a version number when opening a session is proposed as follows:

```
<mrml mrml-version="0.0.1" session-id="1">
 <open-session user-name="test"
  session-name="default_session" />
 <get-collections/>
</mrml>
```

The version number only needs to be transmitted in the first communication phase. Then, a version number is linked to the session number. If a system wishes to change MRML versions a new session needs to opened.

3. PROPOSED SECURITY FEATURES

When thinking about adding database management functions into MRML, two things become apparent: It is (1) necessary to identify the user and to grant him certain access rights, through the use of a login and a password. (2) It is important to encrypt the communication in MRML that is, at the moment, completely open via a socket in human–readable XML.

Care also needs to be taken with respect to log files or other visible parts that can contain important information such as passwords of users. Eventually, steps towards a more secure architecture need to be taken [6].

3.1. User management and access rights

So far, it is already possible to send passwords via MRML although this feature is not used all the time. As the data stream is human–readable, this does currently not make extremely much sense, either, from a security standpoint unless the communication is in a completely safe environment.

User management should contain global rights granted by the root user of the GIFT system through a configuration file. Besides users and groups that can be arranged by the superuser, it should be possible to add certain rights to users that are not known to the system or where the password did not match. This is necessary to grant simple query access to a database at a web demonstration, for example.

Two levels of access can be seen. The first one is with respect to the creation and deletion of entire databases (document collections) and the second one is with respect to operations allowed for one particular database (collection).

3.1.1. Creating/deleting databases

The right to create a database needs to be granted by the system administrator when creating a certain user. The creator can then decide whether he wants to give the right to delete the database to other users as well or if he wants to be the only to keep that right.

Other rights that the owner can grant are with respect to querying and adding/deleting images to a database as detailed in the following section.

3.1.2. Operations on existing databases

Some of the operations on created databases exist already in MRML such as the querying of a database. The MRML server only needs to make a selection of databases to transmit to the user based on the access right that he has and the security settings of the databases.

The following actions need to have separated access rights for each database:

- Execute queries with multimedia objects (ie. images) from the database,
- submit new, private objects as queries,
- add objects to the database,
- delete objects from the database,
- start the generation of a new index based on the currently contained objects.

The submission of new multimedia objects as queries is by default disabled in the MRML interfaces and in the GIFT query engine because of a risk of misuse. A configuration depending on the user and his/her access rights would be much more effective. A simple element of the server could grant this right to a user and if a client still tries to use this feature, an error message could be send by the server.

3.2. Encryption

There are two big parts that are important for encryption. First, all the *files* that contain important information such as log files of the communication and the password files for the user log on and user access rights needs to be protected so they can only be read by the system's super user.

Then, there is a need to encrypt the entire communication that takes place in MRML. It would be possible to only encrypt parts of the communication but it would of course be better to have the entire flow of communication via a secure channel (SSL).

The same server can, for example, serve encrypted and not encrypted clients depending on the operations that are needed. This will also help to improve downwards compatibility. Projects like the dataGrid [5] might also be able to help with such a security infrastructure or middleware for distributed data storage and access.

4. DATABASE MANAGEMENT

Database management function are necessary wherever not only queries are made from distant clients but also the addition or deletion of images is done from different computers and interfaces. This is the case for most commercial applications of content–based image retrieval systems or wherever applications are hosted by application service providers.

In our applications this is the case for the integration of GIFT into OpenMDV[7] (the document management system of the CERN) and into CasImage[8], a medical case database for images [18]. All these applications need the image server to be on one machine but the clients possibly to be on other machines. Still, the clients need to be able to submit new images to the database regularly and update the feature file accordingly.

4.1. Creating/deleting a database of visual documents

A command to create a new database can look like the following example. This takes into account that the information for the creation of a database can be extracted from default values (directories for files, for example).

```
<mrml session-id="1" >
 <create-collection collection-id="c-t1"
  collection-name="test"
  cui-algorithm-id-list-id="ail-inverted-file">
  <query-paradigm-list>
   <query-paradigm type="inverted-file"/>
  </query-paradigm-list>
  <document list>
   <document location="http://viper/pics/1.jpg/">
  </document list>
 </create-collection>
</mrml>
```

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

[8] http://www.casimage.com/

In this example, the document list is optional. In the case of an empty list, a database is created and only the empty structures are set up. If a list of images is transmitted, the generation of the inverted file can be started in the same message or manually afterwards.

The deletion of a database in the MRML syntax is even simpler:

```
<mrml session-id="1" >
 <delete-collection collection-id="c-t1"
  collection-name="test"/>
</mrml>
```

Normally, the ID for the database is sufficient but we add the collection name as another security feature to avoid deleting something by accident.

4.2. Adding/removing images from databases

The addition of new images can be done via:

```
<mrml session-id="1" >
 <add-document collection-id="c-t1">
  <document list>
   <document location="http://viper/pics/1.jpg/>
  </document list>
 </add-document>
</mrml>
```

Such an addition of documents does not necessarily have a generation of the index file as a consequence. Generations of index files can be done in the same step but have to be invoked with another command. This allows to add images regularly but only generates the index every once in a while when there are enough new images. It can of course be started in the same MRML message if needed.

Deleting images is done in a very similar way:

```
<mrml session-id="1" >
 <delete-document collection-id="c-t1">
  <document list>
   <document location="http://viper/pics/1.jpg/">
  </document list>
 </delete-document>
</mrml>
```

The generation of the feature file is also here not automatic but needs to be started with the corresponding MRML command. Thus, we have to take care not to delete the images or thumbnails before the re–generation of the inverted file. Otherwise invalid URLs can be transmitted for images and thumbnails.

4.3. Generation of the inverted file

The generation of the index file for a certain database can be invoked with the following, easy command. The system can then test whether a new generation is necessary and in this case start the scripts to build the inverted file.

```
<mrml session-id="1" >
 <generate-inverted-file collection-id="c-t1"/>
</mrml>
```

On the server side this can, in our case, simply use the executables of GIFT that exist for generating an inverted file. Care needs to be taken with respect to updating the configuration file so there are no inconsistencies.

4.4. Demand list of images

When the entire communication, including the database functionalities, is done via MRML, it is necessary to have some other commands that are useful. It is, for example, important to be able to ask for all the URLs in a certain database. Such a request is currently possible in GIFT by starting a random query with a large number of images but it makes more sense to transmit only the URLs and no additional information such as the thumbnails etc. In this case, the format can be very close to those of adding and deleting images.

Asking for a list of URLs of a certain collection:

```
<mrml session-id="1" >
 <demand-document-list collection-id="c-t1"/>
</mrml>
```

The response of the server, with respecting the access rights, might be similar to the following:

```
<mrml session-id="1" >
 <response-document-list collection-id="c-t1">
  <document-list>
   <document location="http://viper/pics/1.jpg"/>
  </document-list>
 </response-document-list>
</mrml>
```

Other commands might become necessary as well when using such an infrastructure. Another possible command is to ask for more information on a specific database, such as the name of the owner and access rights for specific users.

4.5. A stable MRML server

When taking a look at the current structure of GIFT/Viper, it becomes clear that the structure with using MRML (see Figure 1) has a number of advantages. For example, the search engine and the user interface are well separated so the integration into different programs and the reuse of components is easy.

Still, the query engine is completely coupled with the MRML server receiving the incoming requests of all kinds, at the moment. There is no real test for MRML validity, either. This means that not correctly formed MRML is treated by the server directly which also does the query processing. Unfortunately, this can lead to instabilities of the GIFT server at the moment. Incorrectly formed queries can make the server crash and the reinitialization takes time before the server is accessible, again.

A separated server (that can be like a proxy) will also make it easier for other research groups to start creating

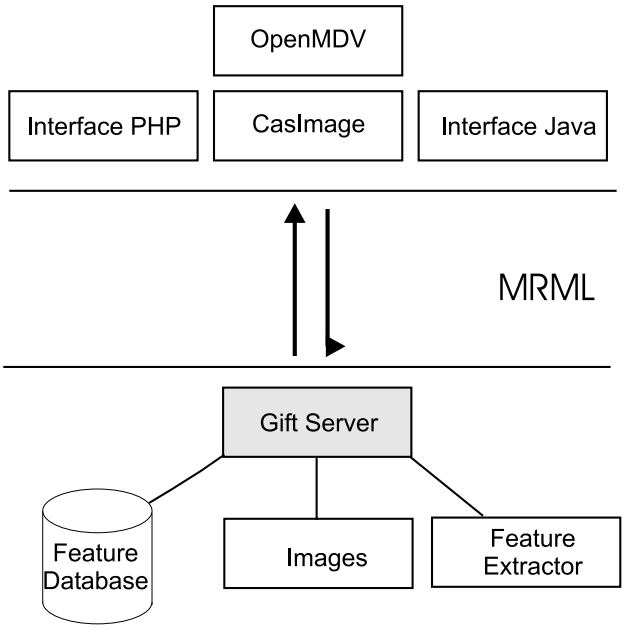

Fig. 1. The current structure of the MRML retrieval engine in GIFT.

MRML–based projects because the server can be an MRML interface and call various client applications such as command–line tools. This also means that the MRML code needs to be decoupled from the query engine code.

To avoid such instabilities, there are two propositions. The MRML server should be separated from the functionality of the query engine as much as possible and the server should control the correctness of incoming MRML commands by verifying it with a DTD or an XML schema that describes MRML.

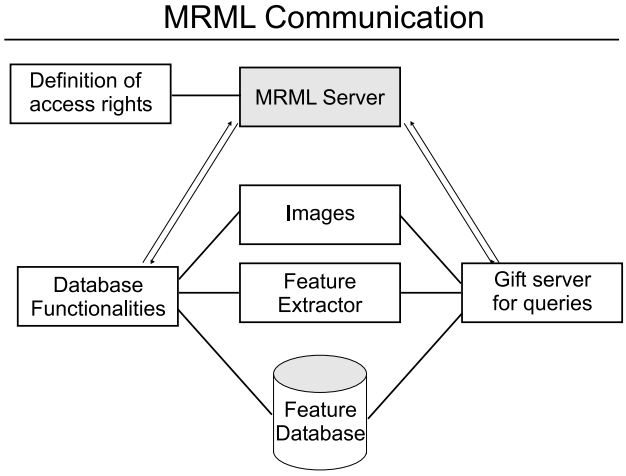

Fig. 2. A proposed structure to better separate the MRML communication from functionalities of the retrieval engine.

Figure 2 shows a structure of an MRML server that is separated from the query engine and that permits to integrate the database functions as they are described in the previous chapters by simply putting a layer in between the MRML communication and the query engine. This allows to treat commands for database actions separately, simply by using the existing scripts for these actions.

The MRML server also verifies the content of the MRML elements automatically based on a DTD or XML schema. It can then auto correct badly formed MRML or send an error message to the sender. In case of well formed queries, they are transmitted to the query engine that can have the same functionality as it currently does in the GIFT system or other retrieval systems can be incorporated via command line executables or other communication protocols.

5. CONCLUSION

MRML has proven to be useful and necessary for the field of content–based image retrieval for applications such as meta–search engines, automatic benchmarks or simple to share the same user interface with several query engines.

Still, with respect to manageability of image databases, MRML does not have any security features or encryption included, yet. When operating on databases with restricted access or when using sensible data such as medical images, these security features seem absolutely necessary. This is as well the case when the system is supposed to be used on multiple platforms. At the moment, the clients work on several platforms but it is not possible to start any database command via MRML. Functions such as adding images or adding and removing databases based on URL lists need to be possible.

This article proposes a structures that implements a stable MRML server that separates the incoming MRML commands into database commands and queries/search engine commands. As a by–product such a server can also make an automatic check of validity of incoming MRML commands before any actions are executed. This can lead to a much more stable version. It is important to also watch the closely related database field for available research and solutions as many of the basic problems are fairly similar.

Especially for the inclusion of image retrieval systems into various applications such an infrastructure is absolutely necessary to be able to manage the data from a distance. Such a security infrastructure with database functionalities is also important for the commercial success of content-based image retrieval systems. Standardized interfaces to manage multimedia data can most likely help more than the best retrieval techniques to have commercial success. The ease in reusing existing components is another important aspect in favor of a standard communication protocol for image retrieval.

6. REFERENCES

[1] J. R. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horowitz, R. Humphrey, R. Jain, and C.-F. Shu. The Virage image search engine: An open framework for image management. In I. K. Sethi and R. C. Jain, editors, *Storage & Retrieval for Image and Video Databases IV*, volume 2670 of *IS&T/SPIE Proceedings*, pages 76–87, San Jose, CA, USA, March 1996.

[2] M. Beigi, A. B. Benitez, and S.-F. Chang. MetaSEEk: A content–based meta-search engine for images. In *Symposium on Electronic Imaging: Multimedia Processing and Applications - Storage and Retrieval for Image and Video Databases VI, IST/SPIE'98, San Jose, CA*, pages 118–128, 1998.

[3] C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein, and J. Malik. Blobworld: A system for region–based image indexing and retrieval. In D. P. Huijsmans and A. W. M. Smeulders, editors, *Third International Conference On Visual Information Systems (VISUAL'99)*, number 1614 in Lecture Notes in Computer Science, pages 509–516, Amsterdam, The Netherlands, June 2–4 1999. Springer–Verlag.

[4] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by Image and Video Content: The QBIC system. *IEEE Computer*, 28(9):23–32, September 1995.

[5] F. Gagliardi, B. Jones, M. Reale, and S. Burke. European datagrid project: Experiences of deploying a large scale testbed for e-science applications. In M. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002*, Lecture Notes in Computer Science, pages 480–500. Springer–Verlag, 2002.

[6] C. D. Jensen. Secure software architectures. In *Proceedings of the Eigth Nordic Workshop on Programming Environment Research*, pages 239–246, Ronneby, 1998.

[7] C. S. Lee, W.-Y. Ma, and H. Zhang. Information embedding based on user's relevance feedback in image retrieval. In S. Panchanathan, S.-F. Chang, and C.-C. J. Kuo, editors, *Multimedia Storage and Archiving Systems IV (VV02)*, volume 3846 of *SPIE Proceedings*, pages 294–304, Boston, Massachusetts, USA, September 20–22 1999. (SPIE Symposium on Voice, Video and Data Communications).

[8] J. Li, M. Ozsu, and D. Szafron. Query languages in multimedia database systems. Technical report, De-

partment of Computing Science, The University of Alberta, Canada, 1995., 1995.

[9] J. Z. Li, M. T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. Technical report, Department of Computing Science, University of Alberta, January 1997., 1997.

[10] H. Müller, W. Müller, S. Marchand-Maillet, D. M. Squire, and T. Pun. A web–based evaluation system for content–based image retrieval. In *Proceedings of the 9th ACM International Conference on Multimedia (ACM MM 2001)*, pages 50–54, Ottawa, Canada, October 2001. The Association for Computing Machinery.

[11] H. Müller, W. Müller, D. M. Squire, Z. Pečenović, S. Marchand-Maillet, and T. Pun. An open framework for distributed multimedia retrieval. In *Recherche d'Informations Assistée par Ordinateur (RIAO'2000) Computer–Assisted Information Retrieval*, volume 1, pages 701–712., Paris, France, April12–14 2000.

[12] H. Müller, D. M. Squire, W. Müller, and T. Pun. Efficient access methods for content–based image retrieval with inverted files. In S. Panchanathan, S.-F. Chang, and C.-C. J. Kuo, editors, *Multimedia Storage and Archiving Systems IV (VV02)*, volume 3846 of *SPIE Proceedings*, pages 461–472, Boston, Massachusetts, USA, September 20–22 1999.

[13] W. Müller, H. Müller, S. Marchand-Maillet, T. Pun, D. M. Squire, Z. Pečenović, C. Giess, and A. P. de Vries. MRML: A communication protocol for content–based image retrieval. In *International Conference on Visual Information Systems (Visual 2000)*, pages 300–311, Lyon, France, November 2–4 2000.

[14] W. Müller, Z. Pečenović, H. Müller, S. Marchand-Maillet, T. Pun, D. M. Squire, A. P. D. Vries, and C. Giess. MRML: An extensible communication protocol for interoperability and benchmarking of multimedia information retrieval systems. In *SPIE Photonics East - Voice, Video, and Data Communications*, pages 961–968, Boston, MA, USA, November 5–8 2000.

[15] W. Müller, D. M. Squire, H. Müller, and T. Pun. Hunting moving targets: an extension to Bayesian methods in multimedia databases. In S. Panchanathan, S.-F. Chang, and C.-C. J. Kuo, editors, *Multimedia Storage and Archiving Systems IV (VV02)*, volume 3846 of *SPIE Proceedings*, pages 328–337, Boston, Massachusetts, USA, September 20–22 1999. (SPIE Symposium on Voice, Video and Data Communications).

[16] D. Papadias and T. Sellis. A pictorial query-by-example language. *Journal on Visual Languages and Computing*, 6(1):53–72, 1995.

[17] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Tools for content–based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254, June 1996.

[18] A. Rosset, O. Ratib, A. Geissbuhler, and J.-P. Vallée. Integration of a multimedia teaching and reference database in a PACS environment. *RadioGraphics*, 22(6):1567–1577, 2002.

[19] N. Roussopoulos, C. Faloutsos, and T. Sellis. An efficient pictorial database system for psql. *IEEE Transactions on Software Engineering*, 14(5):639–650, 1988.

[20] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[21] D. M. Squire, H. Müller, W. Müller, S. Marchand-Maillet, and T. Pun. Design & management of multimedia information systems: Opportunities & challenges. In *Design & Management of Multimedia Information Systems: Opportunities & Challenges*, chapter 7, pages 125–151. Idea Group Publishing, London, 2001.