

# A distributed architecture for content-based image retrieval in medical applications

M.O. Güld<sup>1</sup>, B.B. Wein<sup>2</sup>, D. Keyzers<sup>3</sup>, C. Thies<sup>1</sup>, M. Kohnen<sup>2</sup>, H. Schubert<sup>2</sup>,  
and T.M. Lehmann<sup>1</sup>

<sup>1</sup> Institute of Medical Informatics, Pauwelsstraße 30, 52057 Aachen, Germany  
mguelld@mi.rwth-aachen.de,

WWW home page: <http://www.irma-project.org>

<sup>2</sup> Department of Diagnostic Radiology, Pauwelsstraße 30, 52057 Aachen, Germany

<sup>3</sup> Chair of Computer Science VI, Ahornstrasse 55, 52056 Aachen, Germany

**Abstract.** Image retrieval in medical applications (IRMA) incorporates knowledge from the fields of medicine, image analysis for diagnostic purposes and system engineering. Its implementation as a distributed development platform is fundamental for an efficient interdisciplinary knowledge transfer. The distributed IRMA architecture provides location and access transparency for its resources, i.e. images, feature vectors and methods, resulting in automatic distribution to all participating work groups, including automated replication functionality. The necessary administration is done via a central database with special attention to automated replication functionality. Concurrency transparency and automatic distribution of tasks for image processing, feature extraction, feature evaluation and classification allow the utilization of the computational power of all IRMA integrated hosts regardless of their operating system or hardware configuration. Via extensive system transparency, IRMA drastically simplifies the cooperation of the interdisciplinary development team, allowing all partners to focus on their expert field. In particular, this vastly improves communication and evaluation processes, resulting in much shorter development cycles for new medico-diagnostic methods.

## 1 Introduction

The emerging fields of medico-diagnostic support via digital imaging and picture archiving and communication systems (PACS) demand efficient techniques to access medical image data online. Image retrieval in medical applications (IRMA) aims to provide a broad-purpose image retrieval solution for the clinical routine [1]. The project involves interdisciplinary teams from computer science, medical informatics and radiology at different development sites. To simplify interdisciplinary knowledge transfer between these teams, IRMA is implemented as a distributed system, allowing fast evaluation of image processing methods on image data from clinical routine. In this paper we present a detailed overview of the concepts and implementation of the distributed IRMA architecture.

**General CBIR systems** Early content-based image retrieval (CBIR) approaches extract global features derived from

- color histograms,
- texture, and
- shape,

which describe the entire image, resulting in a significant data reduction. Beside the raw-data layer, a feature layer is added. This enables online query functionality at the cost of precise results as queries are not performed on a level of identified objects, i.e. image content understanding in common systems is rather low. Later systems introduce a third abstract layer of interpretation by mapping a-priori information to image regions that result from a segmentation step. CORE [2] introduces “concepts” reflecting the query context. Blobworld [3] executes queries at a “blob” level, assuming that blobs correspond to objects or ROIs. A recent work combines the Blobworld concept with textual data [4].

**Medical Image Retrieval** Global features are insufficient for describing medical image data, resulting in poor performance of general-purpose CBIR systems applied to medical image data. Present medical CBIR systems mostly focus on a specific topic, thus offering only support for a restricted variety of image types and feature sets, e.g. a system for tumor shapes in mammogram X-rays [5] or the ASSERT system dealing with lung HRCTs [6]. TAGARE et al. made fundamental observations regarding the demands for medical image retrieval systems and their user interfaces [7]:

- Since medical knowledge and medical image content cannot be described precisely using human language, a content-based access is needed for medical image retrieval.
- Image retrieval systems in medicine require a high level of image understanding in order to provide satisfying query results.
- Relevant medical image information depends on local features and their mutual relationships in orientation, geometry or time. This implies a local approach instead of the global approach often used in systems for “stock house” image data.
- As the diagnostic context may vary for each examination or even during a single examination course, a flexible way of storing extracted features and the option to select query-relevant features is needed.
- With imprecise medical knowledge, diagnosis is often based on comparisons of image data with a prototype representing a “model of normality”. Feature extraction and feature selection are expected to evolve during the lifetime of the image retrieval system. Therefore, a medical image retrieval system must provide a flexible way to store images and derived features.
- The aim to integrate the CBIR system into clinical routine demands low data entry costs. Hence, sufficient automated content understanding must be provided by the system to ensure query completion.

However, neither a concept nor an implementation satisfying these requirements is given in [7].

## 2 The IRMA Approach

To provide a general system for image retrieval in medical applications, at least three additional semantic levels of abstraction are needed to cope with the complex medical knowledge, as compared to any standard CBIR system: A low-level of medical knowledge is determined by the imaging modality, including technical parameters, the body region examined, and the functional system under investigation. A mid-level of knowledge is described by the regions of interest within the image, and a high-level is obtained from information regarding the spatial or temporal relationships of relevant objects.

Consequently, IRMA splits the retrieval process into seven consecutive steps. Each step represents a higher level of image abstraction, reflecting an increasing level of image content understanding [1].

**Categorization** The categorization step aims to determine imaging modality, orientation, examined body region and functional system for each image entry. Although categorization-relevant information can be generated by DICOM 3.0 conforming modalities, surveys from clinical routine proved that this information might be not reliable [8]. Furthermore, the system also has to process secondary digitized image data (lacking modality-generated information). In IRMA, categorization is performed using global image features, i.e. a feature value describes the entire image. Note that in IRMA, categorization is not sharp: further steps can be applied for each likely category.

**Registration** Registration in geometry and contrast is done for each likely category and generates a set of transformation parameters that is stored for the corresponding image and each of its likely categories.

**Feature extraction** The feature extraction step derives local image descriptions, i.e. a feature value (or a set of values) is obtained for each pixel. These can be category-free (e.g. edge detection or regional texture analysis) or category-specific, like the application of shape models (incorporating category-specific a-priori knowledge).

**Feature selection** Decoupling feature extraction and feature selection allows the incorporation of the query context into the abstraction process. For instance, the same radiograph image might be subject to fracture or cancer examination, resulting in a contour-based or texture-based combination of features (contour-set or texture-set, respectively).

**Indexing** Indexing provides an abstraction of the previously generated and selected image features, resulting in a compact image description. This can be done via clustering similar image parts (according to the selected feature set) into regions (“blobs”), resulting in a segmentation of the image. In contrast to the Blobworld approach, this is done at multiple resolutions and results in a multi-scale blob-representation of the image.

**Identification** According to TAGARE, an essential requirement for satisfying medical queries is a high level of image understanding offering *object-oriented* retrieval. The identification step provides linking of medical a-priori knowledge to certain blobs generated during the indexing step and is the fundamental basis to introduce high-level image-understanding by analyzing regional or temporal relationships between blobs.

**Retrieval** In IRMA, the retrieval itself is processed on the abstract blob level. Note that all of the steps above can be performed at entry time of an image into the database. This of course requires offline computation of all likely branches. Branches are generated by the categorization and the feature selection step. Only the retrieval step requires online computations.

### 3 The IRMA Architecture

The core of the IRMA system consists of a collection of database driven applications supporting the steps described in Section 2. In the IRMA system, processing images is split into a network of *methods*. Each method is applied to one feature vector (or a set of feature vectors) representing the output from previous method invocations. Methods are executables embedding a user-programmed function (implementing the core feature processing functionality) into a pre-compiled framework provided by the IRMA system.

**Distribution** With respect to the interdisciplinary knowledge transfer, support for distributed development and querying has to be provided. Thus, distributed organization of algorithms, data structures, processed data and their persistence and consistency [9] is a key challenge of IRMA. Furthermore, it is desirable to access the computational power of a whole network cluster, especially for tasks like image processing. Therefore, the IRMA system provides:

- Automated method transfer to keep all sites up-to-date.
- Automated job distribution to balance the load of computations to be executed.

This enables the rapid development, implementation and evaluation of new methods on a large real-life medical image corpus.

**Transparency** The IRMA system has to support several aspects of transparency. From the viewpoint of a distributed system [10], we need:

- Location and access transparency for images, methods, and feature vectors.
- Replication transparency for methods, images and feature vectors.
- Concurrency transparency for job processing and feature generation.

From the user's viewpoint, IRMA aims for maximum system transparency comprising:

- System transparency at method implementation time, allowing the programmer to concentrate on the medico-diagnostic aspect of the algorithm.
- System transparency and job distribution transparency when performing a query.

This vastly simplifies the communication between the method programmer and the physician. It also minimizes the technical overhead when developing methods for image processing.

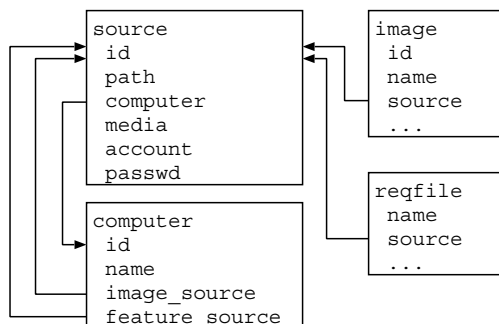
## 4 The IRMA Database

All IRMA development sites access a central relational database (DB), which stores administrative information for all distributable resources (images, methods, feature vectors, query views, connected hosts, and jobs). Each resource is identified by a unique system-wide ID and for each type of resource there exists a command-line executable to perform the insert operation.

## 5 Distributable Resources

A resource is distributable within IRMA if access and location transparency have been implemented. Once a resource is inserted on a host that is registered in the IRMA system, it becomes automatically accessible for all IRMA hosts. While certain elements can be easily distributed via the IRMA-DB, complex or large data objects like images, methods and parts of the feature vectors are stored outside the IRMA-DB. Thus, transparent access and automatic transfer must be provided by the system. Administrative information for this task is organized by the IRMA location management.

**Location Management** To access distributed resources, the IRMA-DB contains information about all hosts that are part of the IRMA system. It is stored in the DB table *computer*, mapping the name of the host (i.e. the IP-address) to its IRMA-ID. For each data object that is kept outside the IRMA-DB, its name (usually the filename) and its location (or locations, if the object has been replicated) are stored inside the DB. This location information is held in a pair consisting of the IRMA-ID of the host and the file system path (which must be



**Fig. 1.** DB tables that contain resource-administrative data: Table *source* stores information about a certain directory on a specific host, table *computer* holds information about a host integrated into the IRMA system. DB entries for e.g. images and source files used by the IRMA system refer to a source entry. References to objects stored in other tables are indicated as arrows connecting the corresponding attributes. Context-irrelevant table attributes are substituted by dots.

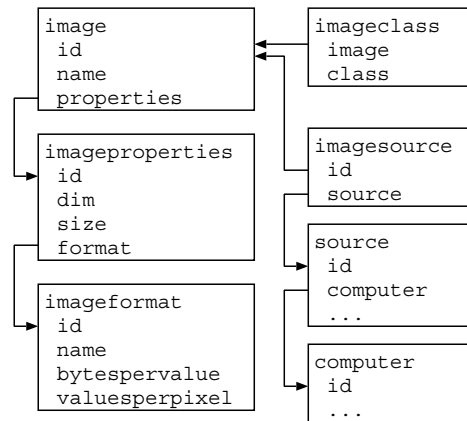
accessible via NFS or FTP for all other IRMA hosts). The pair is stored in the DB table *source* and each entry in the IRMA-DB referring to a file object links to a source entry. Figure 1 illustrates these DB tables and their relationships.

When a host accesses a file stored at a location (identified by its name and its source ID), it checks whether the host providing the exported directory is local (i.e. connected to the same local network) by comparing the respective domain names. Thus, the transfer method is determined: NFS for local hosts, FTP for remote hosts. NFS-accessible directories must use the same mount point on all hosts connected to the respective local network. The FTP transfer uses the account data stored in the DB to access the remote host. Furthermore, each IRMA host is configured where to store replications of images and feature vectors, which are generated e.g. during a feature extraction process running on the respective host.

## 5.1 Images

Besides DICOM data, secondary digitized images have to be processed. Using the IRMA location management, the image data can be stored on multiple file servers and one image can be replicated at several locations. Replication is performed on demand, i.e. a file transferred from a remote host (i.e. outside the local network) is automatically copied to the location specified by the configuration of the computer (see DB table *computer*). Figure 2 shows the image specific parts of the IRMA-DB. The name attribute of an image is identical to the image file name. Additional administrative data regarding the image format is stored in separate table entries to perform checks (e.g. whether a certain method is applicable to the image) without the need to access the image file itself.

In addition to the transparent file access, the image access is provided via an image class, hiding all file-format specific I/O from the method programmer.



**Fig. 2.** DB tables that contain image-related data: *image* stores the image name and one link to a data entry in *imageproperties*. An *imagesource* entry links the image to each location that holds a copy of the image file (table *source*). Entries in *imageclass* allow to store e.g. category information for the image they are linked to.

## 5.2 Methods

In IRMA, a method can either transform feature vectors (derived from images) into a set of new feature vectors (feature extraction) or evaluate one or a set of feature vectors for classification or query purposes. The programmer using the IRMA system only needs to implement the transformation or evaluation algorithm himself. All other parts of the method are system-generated and linked to the algorithm automatically. At the present state, five method types have been defined:

- *Local image processing methods* typically produce image features on a pixel or regional basis. A local method can be applied to each image (or feature vector, respectively) isolated from its application to other images.
- *Global image processing methods* produce a constant number of features per image. Like local methods, they are applied to each image independently.
- *Universal methods* extract features calculated from a set of images. Statistically driven feature reduction like the principal component analysis (PCA) or Fisher’s linear discriminant analysis (LDA) are examples for this type of method. Parameter training for a classifier is another method in this category, e.g. an expectation maximization (EM)-based training of mixture densities used by a statistical classifier.

- *Distance measures* calculate a float type value representing the distance between two feature vectors.
- *Classifier methods* apply a decision rule that is based on trained parameters to a sample feature vector.

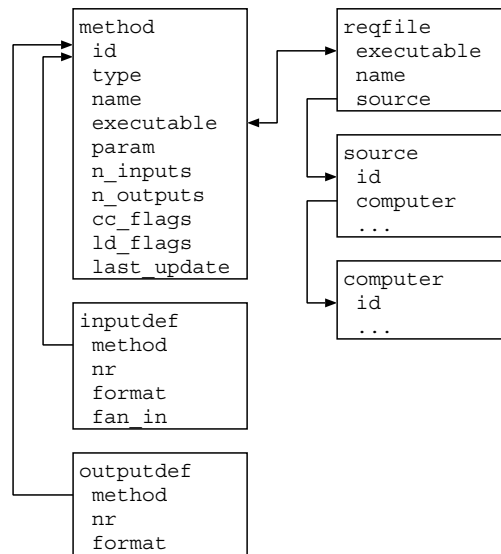
Regarding the nature of the method types, it is easy to define a suitable and fixed programming interface for each of them. Local and global methods have the same I/O structure, and have therefore the same programming interface. They are only listed separately here due to their difference in terms of image processing. For each method, a variable number of inputs and outputs can be specified to allow a flexible interconnection of methods. Since universal methods and distance measures (as they are wrapped into a nearest neighbor (NN) classifier) operate with feature vectors derived from all reference images, inputs can be declared as *fan-in*. Using a fan-in-input results in an implicit loop over all references to provide the required feature vectors. The consecutive access to the input feature vectors is provided by the IRMA method framework. Local methods and global methods can be executed in parallel for multiple images. Furthermore, the IRMA system offers classes for simple access to the method parameters, images (hiding all file I/O) and feature vectors (via a container). Consistent DB access (e.g. fetching and storing of feature vectors) and net-wide job distribution are handled automatically and fully transparent to the method programmer.

Since the IRMA environment is heterogeneous, methods are distributed as source code. When a new method needs to be installed, the IRMA daemon on the local host (see section 6) fetches the required source files and automatically initializes the make process to create the binary executable. Figure 3 gives an overview of this part of the IRMA-DB. Note that an executable can be used by more than one method, e.g. by specifying different parameters for each method definition. The parameter string also allows variable place-holders (via a C-like format string syntax). By using a time-stamp mechanism, method updates can be propagated via the IRMA-DB, resulting all hosts to update their copy of the method source code. Also note that the source location for the method transfer always remains the one hosted by the workgroup who developed the method, giving them exclusive maintenance responsibility.

### 5.3 Query views

When a user issues a query to the system, the query incorporates several aspects about how the data stored in the IRMA system has to be interpreted:

- The method to calculate the query result. This is based on the evaluation or comparison of feature values. Also, the user has to specify how the respective features used for the decision are to be calculated.
- The method parameters. While methods stored within the system may have variable parameters, the parameters have to be fixed for a specific query. They also reflect the user's viewpoint for the current query.

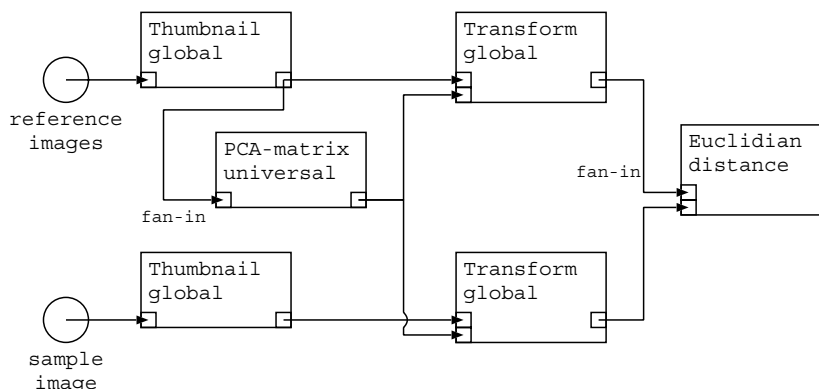


**Fig. 3.** DB tables that contain method-related data: *method* stores information about generic method parameters and the executable. The source files for the executable are stored in *reqfile* and handled via the location management. The tables *inputdef* and *outputdef* allow to specify the data interface for a method.

- The set of images the query is based on. This set is called *reference images*.

These elements form a *query view*. If the *query-by-example* paradigm is used, a *sample image* needs to be specified. In this case, either a distance measure (which will be embedded into a nearest neighbor classifier by the system) or a pair consisting of a universal (training) method and a classifier method can be used to calculate the query result. A query view defines the data flow between single methods, putting them into a specific context. This can be illustrated by a directed graph with each vertex representing a method and the edges modeling the data flow. Each query view possesses two symbolic data sources: one for the reference images and one for the sample image. Usually, each query view has one destination, i.e. the method calculating the query result. The static method definitions specify constraints for the interconnection to other methods. Note that the query view does not specify the sample image but only defines the query *context* (Figure 4).

The usage of query views enables easy experiment verification, and concurrent use of the system in production while other parts are being tested or evaluated. The representation as a directed graph also provides a way to construct algorithms by visual programming. When a new query view is inserted into the IRMA system, the user has to enter values for all variable parameters of all required method calls. This is done interactively via the *insert\_queryview-*



**Fig. 4.** Example for a queryview definition: NN-classification of PCA-reduced image thumbnails. Each method is displayed as a box. Their data interface is represented by small rectangles (inputs on the left, outputs on the right side of the respective box). Edges illustrate the data flow.

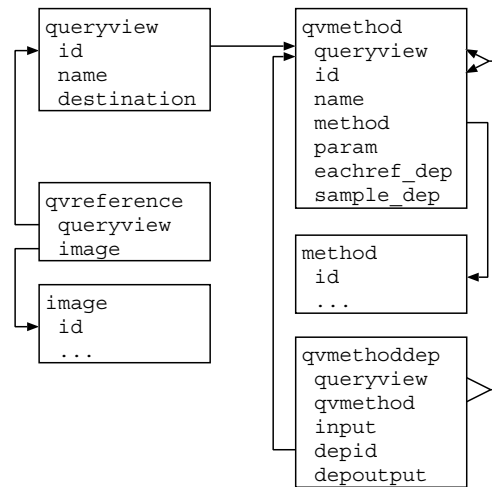
executable. Figure 5 illustrates the corresponding parts of the DB design. Within a query view, each method used is addressed via its “local” ID (i.e. query view-wide ID). Query views are intended to offer a pre-defined query context by fixing the interactive feature selection part (reflecting the medical background of the query) of the retrieval process. This allows all steps of the retrieval to be processed offline in advance, enabling fast online query processing.

#### 5.4 Feature vectors

Feature vectors stored inside the IRMA system are uniquely identified by a 5-tuple consisting of:

- The query view ID, which represents the context of the feature.
- The method used to calculate the feature value. The method is addressed via its query view-wide ID.
- The number of the respective method output.
- The image ID of a reference image (if the method calculation depends on an individual reference image, zero otherwise)
- The image ID of a sample image (if the method calculation depends on the sample image, zero otherwise)

Note that the dependency of a method on an individual reference image or the sample image can be determined automatically by analyzing the query view-graph. For example, in the query view displayed in Figure 4, the upper “Thumbnail” method depends on an individual reference image. The “PCA-matrix” method depends neither on individual reference images nor the sample image, since it is unreachable from the sample image source and receives reference



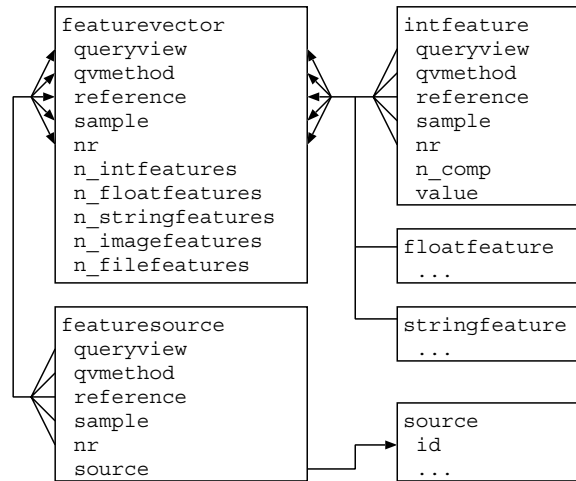
**Fig. 5.** DB tables used to store query view-related parameters: *queryview* stores the ID of the destination method, while *qvmethod* fixes the parameters for each method used. *qvmethoddep* is used to store the interconnection between methods used within the query view. *queryreference* holds the reference set.

images only via an input defined as a fan-in. Thus, feature vectors that are independent from the sample image can be reused by other queries which refer to the same query view, but use a different sample image.

IRMA offers three basic data types for feature values: *int*, *float* and *string*. Values for these types are stored directly inside the DB. Since the access to a large amount of basic feature values is inefficient, the *filefeature* data type provides the functionality to access files within the IRMA system. The definition of the file format is left to the programmer for a maximum of flexibility. The physical locations of images and file features are handled the same way as for images or source files, although file naming is done internally by the system to keep the amount of DB entries low. The *imagefeature* data type can be used for user feedback or browsing purposes (Figure 6). The replication of feature files is performed in the same fashion as for images, using analogous parts of the host configuration in DB table *computer*, but, in contrast, file naming is done internally by the system to save DB space. Each IRMA host uses the location specified by the *feature\_source* attribute of its entry in the DB table *computer* to store the generated file features.

## 5.5 Jobs

A job represents a single method invocation, processing one or multiple feature vectors, with the purpose of feature extraction or feature evaluation (depending on the method type). Pending jobs are stored within the DB table *job* that

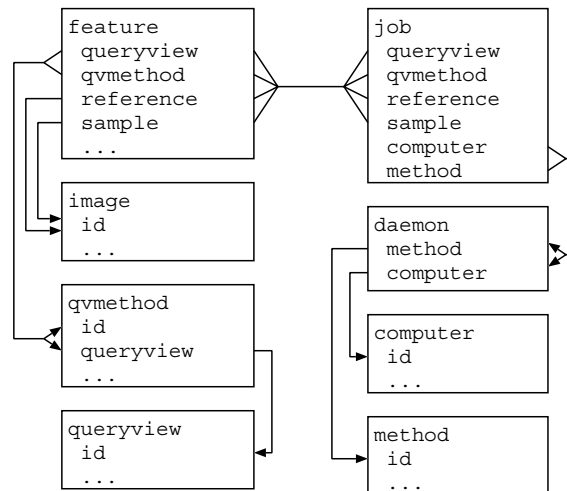


**Fig. 6.** DB tables that store feature vector-related data: a *featurevector*-entry corresponds to a vector generated by a method invocation. The resulting vector for each method output is handled separately, thus a feature vector is addressed by a 5-tuple (see text). The values for the three basic data types *int*, *float* and *string* are stored in the respective tables (only the *intfeature* table is shown here completely). Like for images, the physical locations of a feature vector are stored in a separate table, *featuresource*.

is accessed by all daemons (see below) of the distributed IRMA architecture. Figure 7 displays the DB tables involved. Note the relationship between the tables *job* and *feature* (which holds an entry for a 4-tuple once the corresponding job was processed).

## 6 The IRMA Daemon

Each IRMA daemon integrates its host into the job processing cluster and polls the table *job* of the IRMA-DB for new jobs. Jobs can be generated explicitly (via a manual insertion into the DB) or implicitly (e.g. when a user initiates a query). Through the method interconnection defined by the query view, a job may spawn other jobs, e.g. if a required feature has not been computed yet. For each accepted job, the IRMA daemon checks whether the corresponding method is already running on this host (via the DB table *daemon*). If this is not the case, it determines whether the executable for the method is present, and eventually initiates the method transfer to install the method locally. Then, a child process running the method is spawned. The child takes over the job, fetches the required feature vectors (or initiates their generation by adding jobs to the job-list) and builds the method parameters according to the query view context. Upon job completion, the child will poll the DB table *job* for jobs requesting the same method. To avoid DB load caused by busy waiting, the daemon and

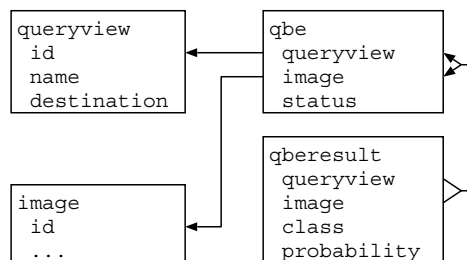


**Fig. 7.** DB tables used for job processing. The table *daemon* contains all started methods (and their respective hosts), table *job* holds pending jobs. For each job, a daemon first checks whether the feature is already present (table *feature*). If not, the method is started with the context defined by the corresponding *queryview* entry (see Figure 5).

the method executables sleep for a certain amount of time, if the IRMA-DB contains no further jobs to be processed. This is also done while waiting for the generation of features that are required for the method execution. To simplify system administration, daemons can be killed via DB entries.

**QBE processing** The start of a query-by-example (QBE) results implicitly in the insertion of a job into the IRMA job-list: It is the application of the corresponding destination method of the query view. This job is automatically expanded using internal standard rules, i.e. the system performs checks, whether the required feature vectors are present, e.g. the feature for evaluation has to be calculated for query image and all reference images. Jobs demanding the execution of a classifier method imply system checks whether the required feature vectors containing the trained parameters exist, and so on.

All QBEs are kept in the DB table *qbe*. Each QBE is identified by the pair consisting of the query view to be used and the ID of the sample image. A status field allows the supervision of the progress of the query. The query result is stored via the DB table *qberesult*: Each entry stores one response for a certain class along with the corresponding probability. (Figure 8).



**Fig. 8.** DB tables that store QBE-related data. As the query view fixes method parameters and the set of reference images for the query process, a *qbe*-entry represents an instantiated query: its *image*-attribute stores the ID of the sample image, the *status*-field is polled to check whether the query is completed. QBE results are stored by using *qberesult*-entries.

## 7 Current Status of Development

**Implementation** The implementation of the IRMA environment was done in C++, using the PostgreSQL DB system. Image file I/O is done via several free image libraries like libjpeg, libtiff and DICOM tools. Libcurl, which is also a free library, is used for FTP transfers. This provides easy portability across a wide range of UNIX-like operating system. The previous version of the system had reached production stage [11] and several experiments for other works were conducted using this system, e.g. [12]. With the back-end of the new system at beta stage, further system development now focuses on the implementation of a web-based front-end to provide easy-to-use query functionality.

**Image data** Currently, 4884 secondary digitized radiographs from clinical routine are stored in the DB. Their distribution among anatomic regions reflects the average occurrence in the routine at the Aachen University Hospital at Aachen, Germany, and therefore forms a data corpus, which allows a meaningful evaluation of medical image processing methods.

**Methods** In the past, IRMA-related research mainly focused on the categorization step. Several results have been published, using appearance based distance measures [13] or active shape models [14]. Further research concentrates on local feature extraction and feature evaluation as a basis for indexing.

## 8 Discussion

The implementation of the system models the IRMA concept using a processing chain composed of methods from five elementary method types. Each method type has a strictly defined programming interface to the IRMA system, while

the data interface is kept flexible. Note that this greatly improves lucidity in the development of complex retrieval processes and also enables the re-use of basic methods in several contexts. Furthermore, the system is flexible enough to emulate the architecture of other CBIR systems for comparison, e.g. Blobworld [3] or QBIC [15].

The extensive transparency provided by the IRMA implementation affects both users (e.g. physicians, radiologists) and programmers, allowing them to focus on their respective field of work. It also offers efficient and simple interdisciplinary communication and knowledge transfer:

- The programmer can take advantage of the minimum programming overhead of the system at method implementation time. The system not only hides all DB I/O, but also provides easy image handling and a container-like storage of feature vectors.
- The user (who is rather unfamiliar with technical details) can concentrate on the medical aspect of the retrieval system. The details of the query execution (including method dependency resolution and job distribution) are fully transparent.
- The automated method transfer offers an easy way to rapidly evaluate new methods on image data from clinical routine. Combined with the system transparency described above, a basis for efficient communication between physicians and programmers is provided. This drastically speeds up development and testing cycles. Also, maintenance is simplified, as method updates can be performed automatically.
- Concurrency transparency and transparent resource access keep most system parts and the workgroups independent from each other. This minimizes the communication required for coordination and maximizes time for interdisciplinary collaboration.

In future, these IRMA key features will provide a fast transfer of medical image processing methods into the clinical routine.

## References

1. T.M. Lehmann, B. Wein, J. Dahmen, J. Bredno, F. Vogelsang, and M. Kohnen. Content-based image retrieval in medical applications: A novel multi-step approach. In *Proc. SPIE*, volume 3972, pages 312–320, 2000.
2. J.K. Wu, A.D. Narasimhalu, B.M. Mehtre, J.P. Lam, and Y.J. Gao. CORE: a content-based retrieval engine for multimedia information systems. *Multimedia Systems*, 3(1):25–41, Feb 1995.
3. C. Carson, M. Thomas, S. Belongie, J.M. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. In *Visual Information and Information Systems. Third International Conference, VISUAL'99. Proceedings*, volume 1614 of *Lecture Notes in Computer Science*, pages 509–516. Springer, Germany, 1999.
4. K. Barnard and D. Forsyth. Learning the semantics of words and pictures. In *Proc. 8th Int. Conference on Computer Vision*, volume 2, pages 408–415, 2001.

5. P. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast and effective retrieval of medical tumor shapes. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):889–904, 1998.
6. C.R. Shyu, C.E. Brodley, A.C. Kak, A. Kosaka, A. Aisen, and L.S. Broderick. ASSERT: a physician-in-the-loop content-based retrieval system for HRCT image databases. *Computer Vision and Image Understanding*, 75(1-2):111–132, 1999.
7. H.D. Tagare, C.C. Jaffe, and J. Dungan. Medical image databases: A content-based retrieval approach. *JAMIA*, 4(3):184–198, 1997.
8. M.O. Güld, M. Kohnen, D. Keysers, H. Schubert, B.B. Wein, J. Bredno, and T.M. Lehmann. Quality of DICOM header information for image categorization. In *Proc. SPIE*, volume 4685 of *Medical Imaging*. SPIE, 2002. in press.
9. U. Asklund, B. Magnusson, and A. Persson. Experiences: distributed development and software configuration management. In *Systems Configuration Management. 9th International Symposium, SCM-9, Proceedings*, volume 1675 of *Lectures Notes in Computer Science*, pages 17–33. Springer, Germany, 1999.
10. George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 3rd edition, 2000.
11. J. Bredno, M. Kohnen, J. Dahmen, F. Vogelsang, B.B. Wein, and T.M. Lehmann. Synergetic impact obtained by a distributed development platform for image retrieval in medical applications (IRMA). In *Proc. SPIE*, volume 3972, pages 321–331, 2000.
12. T.M. Lehmann, S. Goudarzi, N. Linnenbrügger, D. Keysers, and B.B. Wein. Automatic localization and de-lineation of collimation fields in digital and film-based radiographs. In *Proc. SPIE*, volume 4684 of *Medical Imaging*, 2002. in press.
13. J. Dahmen, T. Theiner, D. Keysers, H. Ney, T.M. Lehmann, and B.B. Wein. Classification of radiographs in the 'image retrieval in medical applications' - system (IRMA). In *Proc. 6th International RIAO Conference on Content-Based Multimedia Information Access*, pages 356–360, Paris, France, 2000.
14. J. Bredno, S. Brandt, J. Dahmen, B.B. Wein, and T.M. Lehmann. Kategorisierung von Röntgenbildern mit aktiven Konturmodellen. In *Bildverarbeitung für die Medizin 2000*, pages 356–360. Springer, Germany, 2000. (in german).
15. W. Niblack, Xiaoming Zhu, J.L. Hafner, T. Breuel, D. Ponceleón, D. Petkovic, M.D. Flickner, E. Upfal, S.I. Nin, S. Sull, B. Dom, Boon-Lock Yeo, A. Srinivasan, D. Zivkovic, and M. Penner. Updates to the QBIC system. In *Proc. SPIE*, volume 3312, pages 150–161, 1997.