

A platform for distributed image processing and image retrieval

Mark O. Güld^a, Christian Thies^a, Benedikt Fischer^a, Daniel Keysers^b, Berthold B. Wein^c, and Thomas M. Lehmann^a

^aDepartment of Medical Informatics

^bChair of Computer Science VI

^cDepartment of Diagnostic Radiology

Aachen University of Technology (RWTH), Aachen, Germany

ABSTRACT

We describe a platform for the implementation of a system for content-based image retrieval in medical applications (IRMA). To cope with the constantly evolving medical knowledge, the platform offers a flexible feature model to store and uniformly access all feature types required within a multi-step retrieval approach. A structured generation history for each feature allows the automatic identification and re-use of already computed features. The platform uses directed acyclic graphs composed of processing steps and control elements to model arbitrary retrieval algorithms. This visually intuitive, data-flow oriented representation vastly improves the interdisciplinary communication between computer scientists and physicians during the development of new retrieval algorithms. The execution of the graphs is fully automated within the platform. Each processing step is modeled as a feature transformation. Due to a high degree of system transparency, both the implementation and the evaluation of retrieval algorithms are accelerated significantly. The platform uses a client-server architecture consisting of a central database, a central job scheduler, instances of a daemon service, and clients which embed user-implemented feature transformations. Automatically distributed batch processing and distributed feature storage enable the cost-efficient use of an existing workstation cluster.

Keywords: distributed system, image processing, image retrieval, medical image database

1. INTRODUCTION

The growing establishment of digital picture archiving and communication systems (PACS) in modern hospitals offers the chance to directly integrate various applications of computer-assisted diagnosis into a single platform, which interfaces the image archive. Undoubtedly, the most valuable impact is expected from the implementation of content-based image retrieval, allowing access to images by a consistent, reproducible and human-independent feature-based content description. Since satisfying precision and recall can only be achieved by a high level of image understanding,¹ a wide range of features must be extracted, combined and evaluated within a multi-level content abstraction process. This requires a large amount of storage space, processing power, and a data management concept. Thus, a platform for a medical image database with content-based access spans the fields of both (distributed) image processing and image retrieval.

Image Processing Environments. Most general-purpose image processing environments focus on interactive aspects. They allow easy generation of graphical user interfaces (GUI), rapid prototyping and on-line processing but omit structured feature storage or automatically distributed computing.

The Khoros system by Khoral Research Inc.² is a commercially available product, which offers fast development of image processing algorithms by providing a vast set of existing functions. These functions can be easily combined using the Cantata visual programming environment.³ For the user-implementation of processing routines and applications, Khoros provides a data model to store image information and greatly supports

Further author information send correspondence to:

Mark Oliver Güld, Department of Medical Informatics, Pauwelsstraße 30, 52057 Aachen, Germany, E-mail: mguelde@mi.rwth-aachen.de

the development of GUIs for interactive applications. Typically, each algorithm in Khoros is run only for a single tuple of inputs, which requires manual batch-processing (or manual encoding by using the time-axis of the Khoros data model) to extract features for a set of images and makes the automated use of features derived from an image set somewhat uncomfortable. Although distributed processing is supported, this feature is not fully automated and is not built on a unified communication interface. Instead, the programmer can utilize one of the standard communication types (files, shared memory, sockets, streams). Distributed processing is performed by starting a daemon service on each cluster computer. This service can accept and execute jobs, either via manual invocation or automated via the Cantata GUI.

Several PACS vendors offer an application programming interface (API) for embedding of image processing in their viewing stations, e.g. the clinical application interface (CAI) by Sectra or similar products by their competitors. Using the protocol for digital imaging and communication in medicine (DICOM), the extracted image information can be stored along with the image inside the PACS. However, most applications focus on interactive aspects (e.g. semi-automatic measuring, image enhancement) and can only be applied to the image that is currently viewed.

The distributed image processing environment (DIPE)⁴ offers to run algorithms (*macros*) which can be modeled as directed acyclic graphs. It allows to integrate legacy image processing software. However, it does not offer the persistent storage of feature data, e.g. for retrieval applications.

Image Retrieval Systems. Most image retrieval systems aim for general purpose applications and focus on fast query execution.⁵ Since the image domain is usually not limited, these systems offer a low level of content understanding and use only global features (i.e. very few features describing the whole image) to coarsely encode the properties of an image. Additionally, most systems focus on one specific algorithm, which is hard-wired into the implementation.

The Blobworld⁶ approach extracts local features (i.e. per-pixel features) for color and texture and uses clustering in feature space to perform a partitioning of an image into regions. These regions are then represented by *blobs*, which describe the region by its mean feature vector and its area moments. The retrieval query is executed by comparing the blob structures. The algorithm and the used features are hard-wired.

The PICSearch platform⁷ provides a framework to implement content-based retrieval systems. The user can implement C++ functions to extract *signatures* (i.e. global features) from an image. They can be stored in a database and used for retrieval. Thus, only one isolated processing step for each image can be performed.

Medical Image Retrieval. In common PACS environments, the access to a specific image is performed using the patient's name. Medical diagnoses can be stored as text along with the image. Although content-relevant information can be generated by DICOM 3.0 conforming modalities, surveys from clinical routine showed that this information is not always reliable.⁸ Furthermore, a medical system also has to process secondary digitized image data, which lacks image content related information. TAGARE et al. stressed out the specific needs regarding content-based image retrieval in the field of medicine¹:

- The textual description of image content is problematic due to its inter-person variance and the unsharp terminology when quantifying content properties. Furthermore, the description is context-related, i.e. it is focused on an actual examination whereas the image in general might also contain information relevant in other contexts.
- The medical knowledge constantly evolves. This requires a flexible storage concept and an extensible pool of methods for content extraction and evaluation.

Many existing content-based approaches focus on a specific domain, e.g. ASSERT⁹ for lung high-resolution computed tomographies (HRCTs). HSU uses a three-layered model for image content abstraction¹⁰ and models spatial relationships at the object level. The system uses shape based features and is applied to brain CTs.

The IRMA platform. This work continues the development of the IRMA platform. The first implementation¹¹ introduced the computation and storage of global features for an image database in a heterogeneous distributed system allowing one processing step for each image. The second version¹² used directed acyclic graphs to allow the definition of algorithms as a composition of computation steps. However, it did not offer subroutines (the inclusion of other graphs) or control elements. The stored features were assigned logically to the complete algorithm, making the re-use of features impossible. There were several shortcomings due to the database bottleneck when using distributed coordination of database access for each running process.

In this paper, we present the extensions and conceptual changes made to provide a more efficient and flexible platform.

2. THE IRMA CONCEPT FOR IMAGE RETRIEVAL

Compared to any standard CBIR system, at least three additional semantic levels of abstraction are needed to cope with the complex medical knowledge to provide a general system for image retrieval in medical applications: 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.¹³

Categorization. The categorization step aims to determine imaging modality, orientation, examined body region and functional system¹⁴ for each image entry. In IRMA, categorization is performed using global image features, i.e. features describing 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 generates a set of transformation parameters that is stored for the corresponding image and each of its likely categories. Note that the transformation is not performed explicitly: instead, later abstraction steps utilize the parameter set.

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 image category and the query context into the abstraction process. For instance, the same radiograph 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 (blob-tree).

Identification. According to TAGARE et al. , 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. Thus, it 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 either on the abstract blob level or referring to identified objects. 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 computation branches. Branches are generated by the categorization and the feature selection step. Only the retrieval step requires online computations.

The IRMA platform must provide support for the implementation of all required processing steps described above. Different types of features are used to model the image content extracted: global features (categorization step), local features (input for the indexing step) and the blob-tree (result of the indexing step). The aggregation and evaluation of these features (in parts done for each likely category) are computationally expensive.

3. REQUIREMENTS OF THE SYSTEM

In addition to the requirements originating from the retrieval concept, the design of a medical image retrieval system requires attention to several other aspects and domain specific properties. All requirements can be summarized as follows:

Flexibility. The platform must support the modular design of arbitrary retrieval algorithms. Modularity enables easy isolated verification of processing steps and further allows the re-use of an implemented processing step for other algorithms. All kinds of features (global, local, blob-tree) must be accessible in a uniform way. Hence, the feature model must be general and flexible. Furthermore, several algorithms do not work on an isolated image, but use a set of images (more general: a set of features) to extract knowledge. Additionally to the modularity at the algorithm level, the programmer of a processing step should be supported by providing simple ways to access the feature data and the context for the processing step. Easy handling of image data is also an option, which significantly shortens development time. New algorithms should be able to quickly access the image database for short cycles between development and testing. Therefore, the system must support the automatic transfer of new and updated processing components into the pool of retrieval algorithms available to the physicians.

Storage Space. Many features will be accumulated during the process of image content abstraction, which requires a considerable amount of storage space. Also, a structured organization of the feature storage is mandatory to re-use already computed features in order to save time.

Computation Power. With the same argument, the extraction and transformation of features requires a lot of processing power. Since no special computer hardware is available, the existing heterogeneous infrastructure of workstations must be utilized efficiently.

Interdisciplinarity. The field of medical image retrieval requires a means to model algorithms which is both understandable for physicians and detailed enough for technicians to implement the algorithm.

Transparency. All technical details of the platform should be as transparent as possible for all participants. The programmer should not have to concern platform-related communications regarding the query context, feature access, and feature storage. In particular, the storage location of features must be transparent. Furthermore, the programmer should not be concerned with synchronization aspects between consecutive processing steps, i.e. concurrency of processing steps should be transparent. For the physician, the execution of a query should not require additional technical knowledge about the underlying implementation of the retrieval process, i.e. all steps of the process must run fully automatic without further user interaction.

To cope with these requirements, we propose a distributed platform using a client-server architecture. The platform provides mechanisms for location transparent access to and transparent replication of stored images and features. Algorithms are modeled as directed acyclic graphs composed of feature transformation steps, making them intuitively understandable. All transformation steps run transparently for all other steps and no synchronization has to be implemented by the programmer of a feature transformation. We will describe the platform in the following two sections, first its logical elements and then its physical elements.

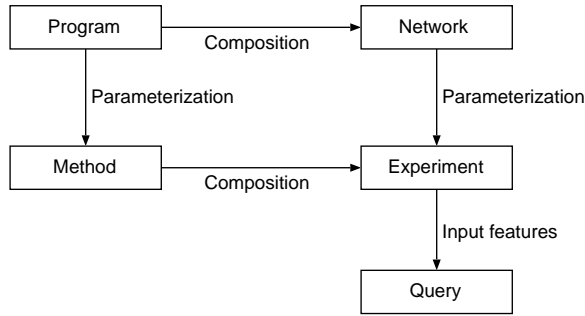


Figure 1. Entities used to model algorithms in IRMA.

4. LOGICAL ELEMENTS

This section describes the entities used to model all parts of the distributed computing environment. Each logical entity has a unique ID.

4.1. Locations

Administrative information about the physical network structure is required to implement access and replication mechanisms for large data objects. These objects (e.g. images) are stored outside the database. Additionally, this information is needed to manage client processes within the network cluster. Therefore, the system stores information about all *computers* and all storage points (*locations*) in the cluster. Each computer has a default location, where generated data will be stored.

4.2. Data

A *feature* is the smallest content-information entity. The generic feature model offers five basic data types to store image content information: integer, float, string, image, and file. The file feature is primarily used to store the blob-tree representation of an image in the extensible markup language (XML) format. But since the system does not concern the file content, this data type can be used by the programmer to define other data structures, taking responsibility for encoding and decoding the file content. Using the location management, features can be automatically replicated and provided on-demand by the system. A *feature set* is used to group semantically identical features into one entity.

4.3. Algorithms

Image processing algorithms are deconstructed into consecutive steps operating on features. Graphical representations of algorithms as directed acyclic graphs composed of interconnected transformation steps are intuitively readable and, furthermore, provide an elegant way to incorporate concurrent and parallel execution, which can be automated by analyzing the network. This concept¹⁵ is widely used, e.g. KAHN defined *process networks*, where concurrent processes communicate through unidirectional first-in, first-out (FIFO) channels. Furthermore, we use a hierarchical approach to model an algorithm, as illustrated in Figure 1. The approach is presented bottom-up in the following paragraphs.

Programs. A *program* encapsulates a user-implemented feature computation. Each program interfaces the system via a set of *inputs* and *outputs* that allow the exchange of features between consecutive programs during processing. For some basic type checking, each input and output is declared to accept exactly one type of feature. The feature interface is flexible, allowing an arbitrary number of inputs and outputs. The system stores information about the program parameters, allowing a program to be used in various contexts.

Both feature extraction and evaluation are regarded as a transformation of features. Considering the transformation, three program types exist:

- the transformation of a feature set into one feature (T:1, e.g. the calculation of a transformation matrix for statistical feature reduction by principal component analysis, PCA),
- the transformation of one feature into another (1:1, e.g. binary thresholding of an image), and
- the transformation of one feature into a set of features (1:T, e.g. the generation of multiple representations of one image via small transformations).

Note that the T:1 and the 1:T program types condense multiple features to one or expand one feature into many, respectively. This demands some attention when using these program types inside a data-flow oriented network. In order for a T:1 program to access each input feature sequentially, one or more of its input can be defined to be a *fan-in*. Analogously, some outputs can be defined to be a *fan-out* for 1:T-programs. The sequential access is provided to the programmer in a simple way. Information about all source code files required to build the program executable is stored using the location management. Since a program might be revised, the system stores a time-stamp for each program to detect the necessity of an update. Such an update, however also outdates all features (and all derived features) previously generated by this program.

Networks. The definition of an algorithm is done via a *network*, which combines a set of *nodes* to form a directed acyclic graph. Each node has a unique identifier inside the network it is used in. The data flow is defined by arcs, which connect two *ports* of two nodes. Each node has a set of input and a set of output ports. Every input port is reached by at most one output port. Output ports can be connected to an arbitrary number of input ports. A node has one of the following types:

- Program node. The feature interface of the program is mapped to the ports of the node.
- Interface node. These nodes allow to import and export features or feature sets. A *source* has no input ports and one output port. A *destination* has one input port and no output ports. For each source, it must be specified, whether a single feature (standard source) or a feature set (fan source) is expected. For destinations, this information can be derived from the type of the output port it is connected to.
- Network node. This allows the use of other networks as subroutines inside the network. The feature interface of the network is mapped to ports of the node.
- IF element. This control element allows the conditional execution of network subparts. It has two input ports, one for a feature to be tested for a condition and the other port for a feature to be propagated to one of the output ports (TRUE-branch or FALSE-branch) according to the condition. The condition can access integer, float and string components of the test feature and evaluates them.
- ENDIF element. This control element merges two branches of network subparts. Therefore, this element has two input ports and one output port.
- SUBSET element. The SUBSET element allows to selectively extract a set of features from its input features. This is done using two input nodes, the first for the features to extract from and the second for a feature of type `id_list` to define the subset to be extracted.
- The INTERSECT and the UNION elements perform the respective operations for their feature input. Each such node has two input ports and one output port.

Since the algorithm might introduce dependencies between parameters for several programs, a network itself can be parameterized. Therefore, a mapping of network parameters to program parameters must be defined along with each network.

The feature transfer between consecutive nodes inside a network is transparent. The IRMA platform keeps all generated features, which makes it possible to detect whether a feature is already present and can be re-used in another context.

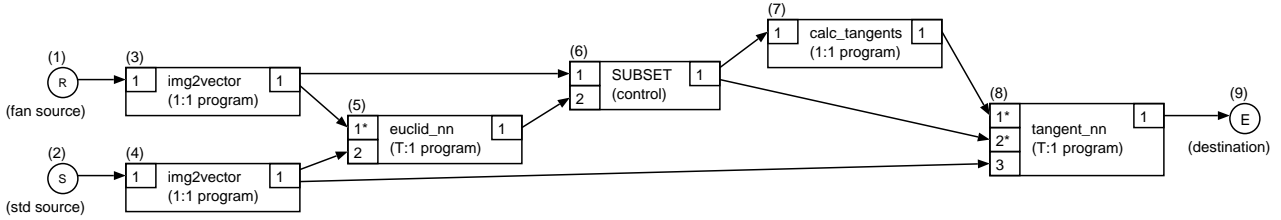


Figure 2. Example network 1: Two-step classification. Each Fan-In of a program is marked with (*).

Methods. A combination of a program and its parameters is called a *method*. By re-use, one program can result in several methods.

Experiments. Before a network can be executed, all variable parameters need to be specified. This concerns programs, IF-elements, and networks used inside the network. We refer to the resulting, fully parameterized network as an *experiment*.

Queries. A *query* defines all imports for a specific experiment, i.e. for each source node of the experiment’s underlying network, either a feature or a feature set (depending on the source type) is specified.

Example. Figure 2 illustrates a network for an automated categorization application: The two-step categorization is based on the Euclidean distance k-nearest neighbor (k-NN) for downscaled images (in vector representation), followed by a tangent-distance based k-NN using only the most promising neighbors of a sample image.

The network has two sources: one to import a feature set, consisting of the reference images, the second to import the sample image to be categorized. The downscaled representation of an image is generated by the program named *img2vector*, which expects two parameters, i.e. target width and height. The resulting feature is a vector. The Euclidean NN sequentially reads all reference vectors, compares them with the sample vector and outputs a list containing the IDs of the k nearest neighbors. This feature is used by the SUBSET element to extract only the remaining candidates as reference the tangent distance-based classifier. Next, for each candidate, the tangent vectors are calculated, which are finally passed to the classifier itself. Several things should be noted:

- The program *tangent_nn* in node (8) of the network has two fan-ins, but it only depends on one feature set.
- The 1:1 programs *img2vector* and *calc_tangents* in nodes (3) and (7) can be executed in parallel for each feature of their input feature sets.
- For consistency, the programs in nodes (3) and (4) must have the same parameterization. This can be ensured by the network to node parameter mapping of the network definition.

Also note that the data flow inside an IRMA network differs from common data-flow networks, since features can be multiplied implicitly. Figure 3 illustrates a network for a categorization task using PCA for feature reduction and a simple k-NN for classification. It calculates the PCA transformation matrix using all references and then applies it to all vectors (references and sample) before feeding them to the classifier. Node (6) using the program *transform* (which implements the matrix-vector multiplication) repeatedly uses the transformation matrix to generate all PCA-reduced vectors. The scheduling algorithm to run the network has to ensure the correct execution.

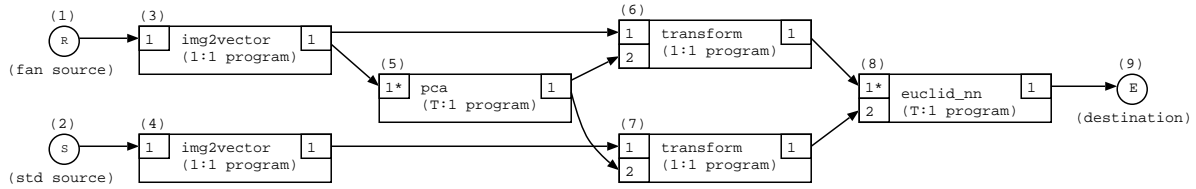


Figure 3. Example network 2: PCA-based classification. Each Fan-In of a program is marked with (*).

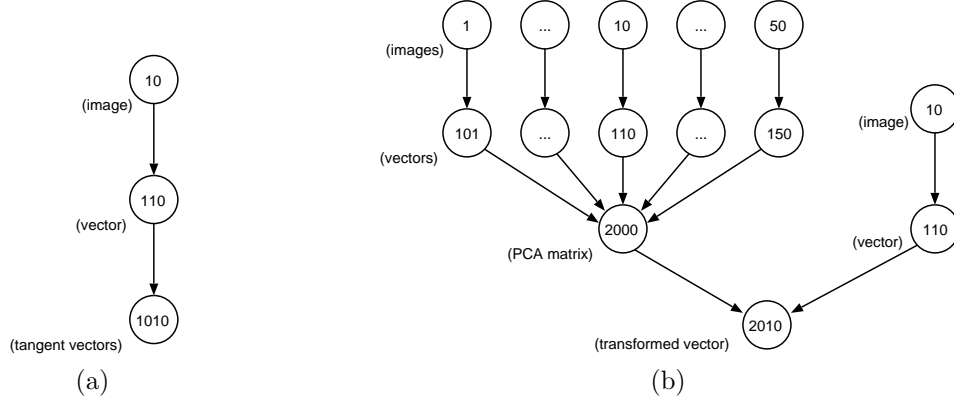


Figure 4. Feature generation history referring to the example networks in Figures 2 and 3.

4.4. History

The generation of each feature can be uniquely identified by a tuple consisting of

- The method ID, i.e. the program that was used to generate the feature along with its parameterization.
- The ID of the program output where the feature was generated.
- To uniquely identify the output of a 1:T program, an additional serial number is required.
- The IDs of all input features used by the program to perform its computation must be stored. For T:1 programs, serial numbers must be used to distinguish the input.

For each generated feature, this information allows the identification of already computed features during the processing of a query. Recursive unrolling of the results in a tree-like structure feature history with initial features in the leaf nodes.

Example. Referring to the example network illustrated in Figure 2, imagine a query that operates with features ID 1, ..., ID 50 (images) as references and feature ID 51 (image) as a sample. Furthermore, the downscaled version (ID 110) of feature ID 10 is among the k nearest neighbors for the sample. The feature generation history for its calculated tangents (ID 1010) is illustrated in Figure 4 (a). Note that the SUBSET control element along the feature generation path is irrelevant, since it only modifies feature sets, but not the features themselves.

Figure 4 (b) illustrates the feature history for the PCA-transformed vector for the same image (feature ID 10), assuming that the network in Figure 3 uses the same assignment for its sources and that *img2vector* uses the same parameterization as the first example network. The system is able to detect that the features ID 101, ..., 150 are already present. Thus, their calculation can be skipped.

5. SYSTEM COMPONENTS

The IRMA platform uses a client-server-architecture to perform all steps necessary to support the implementation of the previously described retrieval concept.

5.1. IRMA Database

A central relational database is used to store administrative information about

1. physical entities, i.e. the feature data (images, global and local features), tree data (a hierarchical image representation resulting from the segmentation step), and source code (user-implemented programs), and
2. logical entities, i.e. network infrastructure, definitions of algorithms for image processing and retrieval, and feature sets.

The physical entities are stored outside the database and can be hosted by any computer within the distributed system. Using the information about the network infrastructure, transparent access to and automatic replication of all physical entities are implemented.¹¹

5.2. IRMA Scheduler

The IRMA scheduler, a central service, manages the execution of all queries or feature extraction tasks. It is split into two functional parts:

- For each invoked query, the scheduler creates a data structure to log the progress during the execution of the corresponding method network. For each node of the network ready to be executed, a *job* is generated, which includes the method ID (which defines the program and its parameterization), the IDs of the input features and IDs of their location, and allocated IDs for the output features.
- The communication subpart will assign jobs to programs running in the network cluster. If the program needed by a job is not running, the scheduler selects an appropriate host and issues the IRMA daemon on this machine to start the program. Additionally, the scheduler can order idle programs to terminate in case this program will not be required within a look-ahead interval. The scheduler uses transmission control protocol (TCP) socket based communication to communicate with all daemons and the programs running inside the network cluster. Upon the completion of a job, the scheduler receives a notification from the program and updates the feature information in the database.

5.3. IRMA Daemons

Each computer within the system runs the IRMA daemon, a background process that automatically installs new programs on its host and starts them on demand. The daemon service is also used to inform the IRMA scheduler about the current load of its host. Furthermore, it can detect possible abnormal terminations of IRMA programs and report them to the IRMA scheduler for fault handling.

The IRMA daemon also performs the automated program transfer: By evaluating the versioning information from the database, the daemon can determine if a program needs to be installed or updated. In this case, all source code files are fetched using the location management. Due to the heterogeneous network infrastructure, programs are transferred as source code. Once the transfer is complete, a makefile is generated and executed, resulting in the program executable.

5.4. IRMA Clients

The physical executable of a program entity consists of a user-implemented subroutine linked to a generic main routine which handles all system communication: receiving a job from the IRMA scheduler, loading the input features, starting the subroutine, and storing the resulting output features at a designated location. The programmer only has to implement the problem-related part.

6. RESULTS

The IRMA platform is in the development phase. It is implemented using C++ and the PostgreSQL DBMS plus several freely available libraries for image formats, XML and network communications. Current platforms are IA32/Linux and Sparc/Solaris, but due to the portability of all libraries and the platform implementation itself, a wide variety of system architectures running under the Unix operating system can be targeted. Previous versions of the platform were used to carry several experiments. The distributed platform supports the implementation of a content-based image retrieval system by addressing the requirements stated in Section 3.

Flexibility. The concept of modeling an algorithm as a *network* composed of *programs* and control elements for set manipulation, conditional execution, and subroutines allows to break any retrieval algorithm into smaller parts. Via parameterization, *programs* can be re-used in different contexts. The feature model allows to store arbitrary image-content information, which enables the evolution of medical knowledge.

Distributed storage. The image database currently contains 6,514 medical images from various imaging modalities. Non-medical data (about 16,000 color images) is also used for testing purposes. Experiments running on previous versions of the platform mainly focused on the categorization step. Thus, programs for about ten types of global features (based on texture and shape) along with several classifiers have been implemented and can be easily ported this version of the platform.

Distributed computing. The IRMA scheduler provides the automatic control of the distributed computation by communicating with the IRMA daemons (one running on each host of the platform) and the IRMA clients, which are executables of a *program*. At the moment, the platform can utilize about a dozen workstations (IA32/Linux, Sparc/Solaris) for feature computations.

Interdisciplinarity. When discussing an new retrieval algorithm, physicians and programmers can utilize the graphical representation of the algorithm (i.e. the *network*) to communicate technical issues on a level that is both understandable for the non-technical participants and detailed enough to specify all relevant algorithm properties. The graphical representation allows to easily identify elementary feature processing steps, map the parameters of the algorithm to program parameters, specify the information which is passed along the network, and determine the type of each program in the network. Each transformation step can be matched to a program of either type T:1, 1:1, or 1:T.

Transparency. The IRMA platform allows to implement a transformation routine without requiring the programmer to handle platform relevant communication. Concurrent execution of programs and the storage of features are transparent to both minimize pitfalls and maximize the focus on the problem. Since the IRMA platform stores only information about the parameter types and does not know their semantic meaning, the programmer must also provide a function to check the parameterization for correctness. This function is very simple and identical for all program types. The function is called by the main routine before the feature transformation is started.

```
int check_method_args(  
    param        & params,  
    logging      & log);
```

In IRMA, feature transformation routines use a C++ interface. There are classes for access to the input and output features, access to the parameterization, a handle which offers some additional functionality, and images (which are feature components). The handle offers logging functionality (via a logging object) and information about where to find file features (the system only provides automated transfer and replication for them, i/o is left to the programmer). Furthermore, the program might use own data files which need to be accessed. The handle also provides information where these are stored (again, the platform provides them locally on the host machine, i.e. they can be accessed like files). Each feature is stored inside a **feature**-object with basic

access functionality to any component. All program inputs and outputs are mapped to `vector` containers for `feature` objects. Images are encapsulated into `image`-objects, freeing the user to manually load and store them. The `image` object also provides basic image processing functionality. For the feature transformation itself, the programmer has to implement one C++ function, which is slightly different for each program type. For a 1:1 program, the programmer has to implement the following function:

```
int transformation_1_1(  
    program_handle      & handle,  
    const vector<feature> & features_in,  
    param               & params,  
    vector<feature>     & features_out);
```

The inputs and outputs of the program are accessed by using their respective ID as an index for the vectors `features_in` and `features_out`. A T:1 sequentially reads a set of input feature tuples and calculates tuple of features (one feature for each of its outputs). Thus, the transformation subroutine receives a handle which offers a `read_infeatures()` function to fetch a certain input feature tuple. A 1:T program can output an arbitrary set of feature tuples. Again, a special handle is used which offers a `write_outfeatures()` function for this purpose.

Via the automatic distributed computing, the physician does not have to concern the system's internal job management. The execution is fully transparent and the final result of the query can be accessed using the location management.

7. CONCLUSION AND DISCUSSION

We have introduced a distributed platform for image processing, which focuses on maximum transparency for all human participants in the interdisciplinary research on content-based image retrieval in medical applications. The platform combines the easy modeling of graphical image processing systems with a database for automatically organized feature storage. It provides an efficient system to implement, maintain, and run arbitrary algorithms for image content description and image retrieval.

The proposed architecture implements the platform using four physical components: a relational database, a scheduling service and a daemon for each host in the network cluster, enabling distributed computation and storage. Based on this small number of core elements, it offers the following key benefits:

1. Since the graphical modeling of image processing algorithms allows easy visualization and interpretation, the interdisciplinary communication between programmers and medical experts during development is simplified significantly.
2. All trivial aspects of the system are transparent for users. This encompasses the programmer (feature i/o, system communication), the system administrator (automated method transfer, load balancing), and the physician (query computation).

Currently, the system is not integrated into the clinical routine and requires manual data transfer from the PACS at the Aachen University hospital. Utilizing the PACS vendor's API, data import directly from the PACS is planned.¹⁶ This procedure must of course comply to legal issues, i.e. separating all patient-related textual data from the image information. To further support the physician, a WWW-based graphical front-end for query interaction is in development.

8. ACKNOWLEDGMENT

This work was performed within the image retrieval in medical applications (IRMA) project, which is funded by the German Research Community (Deutsche Forschungsgemeinschaft, DFG) grant Le 1108/4.

REFERENCES

1. H. Tagare, C. Jaffe, and J. Dungan, "Medical image databases: A content-based retrieval approach," *Journal of the American Medical Informatics Association (JAMIA)* **4**(3), pp. 184–198, 1997.
2. D. Argiro, S. Kubica, M. Young, and S. Jorgensen, "Khoros: An integrated development environment for scientific computing visualization," tech. rep., 1999. available online at http://www.khoros.com/downloads/papers/khoros_IDE.pdf.
3. D. Argiro, K. Farrar, and S. Kubica, "Cantata: the visual programming environment for the Khoros system," in *Visualization, Imaging, and Image Processing. Proceedings of the IASTED International Conference*, pp. 697–702, 2001.
4. M. Zikos, E. Kaldoudi, and S. Orphanoudakis, "DIPE: A distributed environment for medical image processing," in *Medical Informatics Europe '97*, **2**, pp. 465–469, 1997.
5. A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(12), pp. 1349–1380, 2000.
6. C. Carson, S. Belongie, H. Greenspan, and J. Malik, "Blobworld: Image segmentation using expectation-maximization and its application to image querying," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(8), pp. 1026–38, 2002.
7. K. Lemström, J. Korte, P. Kuusi, P. Kyheröinen, and P. Päiväkumpu, "PICSearch - a platform for image content-based searching algorithms," *The 6th International Conference in Central Europe on Computer Graphics and Visualization* **2**, pp. 222–229, 1998.
8. M. Güld, M. Kohonen, D. Keysers, H. Schubert, B. Wein, J. Bredno, and T. Lehmann, "Quality of DICOM header information for image categorization," in *Procs. SPIE*, **4685**, pp. 280–287, 2002.
9. C. Shyu, C. Brodley, A. Kak, A. Kosaka, A. Aisen, and L. Broderick, "ASSERT: a physician-in-the-loop content-based retrieval system for HRCT image databases," *Computer Vision and Image Understanding* **75**(1-2), pp. 111–132, 1999.
10. C.-C. Hsu, W. Chu, and R. Taira, "A knowledge-based approach for retrieving images by content," *Knowledge and Data Engineering* **8**(4), pp. 522–532, 1996.
11. J. Bredno, M. Kohonen, J. Dahmen, F. Vogelsang, B. Wein, and T. Lehmann, "Synergetic impact obtained by a distributed development platform for image retrieval in medical applications (IRMA)," in *Procs. SPIE*, **3972**, pp. 321–331, 2000.
12. M. Güld, B. Wein, D. Keysers, C. Thies, M. Kohonen, H. Schubert, and T. Lehmann, "A distributed architecture for content-based image retrieval in medical applications," in *Proceedings of the 2nd International Workshop on Pattern Recognition in Information Systems*, pp. 299–314, 2002.
13. T. Lehmann, B. Wein, J. Dahmen, J. Bredno, F. Vogelsang, and M. Kohonen, "Content-based image retrieval in medical applications: A novel multi-step approach," in *Procs. SPIE*, **3972**, pp. 312–320, 2000.
14. T. Lehmann, H. Schubert, D. Keysers, M. Kohonen, and B. Wein, "The irma code for unique classification of medical images," in *Procs. SPIE*, **5033**, 2003. in press.
15. E. A. Lee and T. M. Parks, "Dataflow process networks," *Procs. of the IEEE* **83**(5), pp. 773–799, 1995.
16. T. Lehmann, M. Güld, C. Thies, B. Fischer, D. Keysers, M. Kohonen, H. Schubert, and B. Wein, "Content-based image retrieval in medical applications for picture archiving and communication systems," in *Procs. SPIE*, **5033**, 2003. in press.