An Architecture for Large Scale and High Performance Medical Imaging Applications

H. Duque[12], J. Montagnat[1], J.M. Pierson[2], L. Seitz[2], L. Brunie[2], and I. E. Magnin[1]

1. CREATIS, CNRS UMR 5515, INSA, 69621 Villeurbanne, France
   http://www.creatis.insa-lyon.fr/
   {*Isabelle.Magnin, Johan.Montagnat, Hector.Duque*}*@creatis.insa-lyon.fr*

2. LIRIS, CNRS FRE 2672, INSA, 69621 Villeurbanne, France
   http://liris.insa-lyon.fr/
   {*Lionel.Brunie, Jean-Marc.Pierson, Ludwig.Seitz*}*@insa-lyon.fr*

Corresponding author:
Hector Duque
CREATIS, INSA, Bât. B. Pascal
20 av. A. Einstein
69621 Villeurbanne Cedex
France

email: duque@creatis.insa-lyon.fr
http://www.creatis.insa-lyon.fr/~duque
phone: +33 472 43 82 26
fax: +33 472 43 85 26

Keywords: Medical imaging, Grid computing, Medical databases, High performance, Mass Storage Systems

An Architecture for Large Scale and High Performance Medical Imaging Applications

H. Duque[12], J. Montagnat[1], J.M. Pierson[2],
L. Seitz[2], L. Brunie[2], I. E. Magnin[1]
[1] CREATIS, CNRS UMR 5515, INSA, 69621 Villeurbanne, France,
http://www.creatis.insa-lyon.fr/
[2] LIRIS, CNRS FRE 2672, INSA, 69621 Villeurbanne, France,
http://liris.insa-lyon.fr/

June 9, 2003

Abstract

Medical data represent tremendous amount of data for which automatic analysis is increasingly needed. Grids are very promising to face today challenging health issues such as epidemiological studies through large image data sets. However, the sensitive nature of medical data makes it difficult to widely distribute medical applications over computational grids. In this paper, we review fundamental medical data manipulation requirements and then propose an architecture (Distributed Systems Engines - DSE) for building high performance Distributed Systems dedicated to medical image processing. We show how the DSE were used in developing a Distributed Medical Data Manager that addresses the medical data and security challenges.

1 Introduction

Recently, computational grids [11, 12] encountered a large success among the distributed computing community. Many technical developments aim at providing a middleware layer for submitting remote jobs [2, 6, 13], storing data [5], and monitoring a distributed system [24]. But most importantly, from the user point of view, grids should provide transparent access to distributed resources and ease data and algorithms sharing

Medical data analysis is a domain where grid technologies are very promising. Health centers are using an increasing number of digital 3D sensors for medical data acquisition, representing tremendous amount of data for which automatic analysis is needed. Grid technologies offer: (i) an increased computing power for complex modeling and simulation algorithms, (ii) a distributed platform with shared resources for different medical partners with similar data processing needs, (iii) a common architecture to access heterogeneous data and algorithms for processing, and (iv) the ability to process very large amounts of data, *e.g.* for epidemiological studies. To adapt the middleware layer to medical application needs, it should take into account the particular constraints associated to medical data. Although weakly structured, medical data have a strong semantic and metadata are very important to describe data (*e.g.* images) content. Furthermore, medical data are sensitive and should only be accessible by accredited users, which makes data manipulation over a wide area network difficult.

Managing large databases of medical images in a Grid environment is a challenge. Users (patient, medical staff, researchers) need to query medical databases using complex access patterns involving both metadata related to each patient record and image content analysis. These kind of queries imply:

- To preprocess the images in order to generate indices useful for image retrieval. Image processing allows to extract information such as histograms, texture parameters, etc, considered as metadata here. Other metadata include patient related information (name, age, ...), diagnostic-related, and therapy-related information.

- To analyze on-the-fly images when one query arrives, in order to extract features from the image. This implies performing computations before returning results to the user.

Other medical data management related challenges include data and processing distribution, high performance, structured semantics, massive storage, security and image processing techniques. While some of these problems are tackled easily by current Grid implementations, much of the work to interface securely the medical data and the Grid infrastructure remains to be done. In this paper we detail the motivation and the difficulty of distributing medical data on the grid, then we propose a novel modular distributed system for dealing with this kind of data and an architecture for building each component of this distributed system.

This paper is structured as follows. We first detail in section 2 medical data requirements and the related issues for Grid computing. In section 3 we propose a novel data management architecture. We show in section 4 how it can be used for developing a Distributed Medical Data Manager [10]. Our implementation is based on some of the *European DataGrid* project (EDG) middleware services [7]. Finally, we show results in section 5 and a conclusion gives the perspectives of the project.

2 Medical applications

2.1 Medical Data

Although there is no universal standard for storing medical images, the most established industrial standard is DICOM (Digital Image and COmmunication in Medicine) [9]. DICOM describes both an image format and a client/server protocol to store and retrieve images on/from a medical image server. Most recent medical imagers implement the DICOM protocol. They play the role of acquisition device and image server communicating with their clients over a TCP/IP hospital network. The DICOM file format is made of a header containing metadata followed by one or several image slice(s) in a single file. To the original metadata, users often want to associate additional metadata that are not originally part of the image acquisition such as notes taken by medical experts.

Today, medical data are often stored and archived inside each medical image producer (hospitals, clinics, radiology centers...). The medical record (image files and metadata) of one patient is distributed over the different medical centers that have been involved in his health care. Medical data are usually disconnected from the outside world to solve security issues. Several Picture Archiving, Communication (PACS) [14] and Radiology Information Systems have been developed to provide data management, visualization, and, to some extent, processing. However, they are restricted to data management inside each hospital and hardly address the problems arising when considering a global system with communication of sensitive data between sites.

2.2 Requirements for medical data on the Grid

Data management and replication mechanisms [22] proposed by grid middlewares mainly deal with flat files. Data access control is handled at a file level. In the DataGrid project for instance, user authentication relies on the asymmetric key-based Globus Grid Security Infrastructure layer (GSI) [3]. This infrastructure does not take into consideration metadata and does not address patient record distribution. Therefore, we investigate the creation of a *Distributed Medical Data Manager* unit (abbreviated as DM^2 later in this document) that interfaces with the grid middleware. It should provide:

- Reliable and scalable storage for images and metadata produced by medical imagers. This includes connection to the grid file replication mechanism and a metadata location service granting access to distributed medical records (see section 3.1 for details). To face scalability and reliability issues in a wide area environment, replication of metadata also appears necessary.

- Data and associated metadata should be synchronized by the information system as they are semantically connected (they should have the same lifetime, same access patterns...).

- Secure communications, encryption, integrity checking, authentication and a distributed access control mechanism are needed to secure the data.

Processing or querying data over a grid raise the problems of confidentiality and confidence that the user may have in the grid security infrastructure. Ideally, medical data should not be accessible by any unaccredited user, including system administrators of sites where the data are transported for computation. To ensure a reasonable confidentiality level, we plan to decouple the sensitive metadata from the image data. The metadata will only be stored on a limited number of trusted sites, where administrators are accredited, and the system will not send it to non-accredited user stations. The image data can be stored and replicated in encrypted form and will only be decrypted for manipulation, thus reducing vulnerability to content retrieval.

2.3 Illustration through usecases

To illustrate our proposal for a distributed medical application, we will consider a cardiologist making a diagnosis for a patient whose medical data is stored in the system. He needs to access its patient data and to query the system to find similar cases.

Accessing medical data. For accessing the data a software transparently communicates with the Grid storage facilities. A medical application first queries the data location service to get the location of the images since the records of a patient might be spread over multiple hospitals. Second, it gets the set of DICOM slices from the selected hospital (a 3D image is often stored has a set of files, each representing one slice in DICOM format). Slices are assembled in a single 3D image that is returned to the cardiologist for visualization.

Content-based image search. The cardiologist looks for heart images similar to the one he has just acceded in order to confirm his diagnosis. He wants to rank existing images through a similarity score resulting of a computation involving his patient image and an image database. Once the images are ranked, he needs to visualize the most similar cases and their attached diagnoses. The diagnosis he makes for his patient can be stored in the information system enriching the global knowledge as well as the patient record.

3 DM^2: A Distributed Medical Data Manager

We propose a *Distributed Medical Data Manager*. Such a distributed system is made of several interconnected computers and a shared state describing the cooperation of these computers [16]. To respond the requirements described in section 2.2 and the above usecases the DM^2 is designed as a complex system involving multiple grid service interfaces and several interacting processes geographically distributed over an heterogeneous environment. It is an access point to the grid services as well as an intermediary (proxy) between the grid and a set of trusted medical sites. Its complexity has motivated us to first propose an architecture describing the DM^2 components and second to implement our system as one possible instance of this architecture. To tackle the DM^2 complexity, we propose the multi-layers architecture outlined in section 3.2. We also need to interface the DM^2 with underlying grid services as detailed in section 3.1.

3.1 Interface between grid data storage services and medical servers

A grid middleware, such as the EDG (European Data Grid) middleware, proposes a standard storage interface to the underlying mass storage systems. Through this interface, the middleware may access files on distributed and heterogeneous storage pools. Grid-enabled files are handled by a *Replica Manager*

(RM): to ensure fault tolerance and to provide a high data accessibility service, files are registered into the RM and may be replicated by the middleware in several identical instances. The first file registered into the RM is a *master file*. Other instances are *replicas*. When a file is needed, the grid middleware will automatically choose which replica should be used for optimizing performances. Having multiple instances of a file also increase its availability since connection errors are likely to happen in a wide scale distributed system. To solve coherency problems, replicas are accessible in read only mode and modifying a master file invalidates all its replicas. To ease files manipulation, grid wide *Logical File Names* (LFN) are used to identify each logical data (*i.e.* a master and all its replicas).

In the hospital, each image can be made up of one or several DICOM files representing portions of the imaged body. The DM^2 plays a double role to interface DICOM servers with the grid middleware as illustrated in figure 1:

For each new DICOM image or set of DICOM images (depending on the semantic of the DICOM series) produced by an imager, a LFN is created and registered into the RM. The DICOM files thus becomes, from the grid side, a master file. There is not necessarily a physical file instance behind this LFN but rather a virtual file made up of a set of DICOM files, that can be reconstructed on the fly by the DM^2 if a request for this LFN comes in. For efficiency reasons, assembled files are cached on a scratch space before being sent outside. The DM^2 also stores metadata and establishes a link between an LFN and its patient- or image-related metadata.

The DM^2 storage interface ensures data security by anonymizing and encrypting on the fly images that are sent to the grid. Replicas of a medical image may exist on any grid storage node, given that encryption forbids data access without decryption keys. These keys are stored with the patient-related metadata on trusted sites only. In order to ensure data integrity, the grid storage interface does not allow the master files stored on the DICOM server to be deleted.

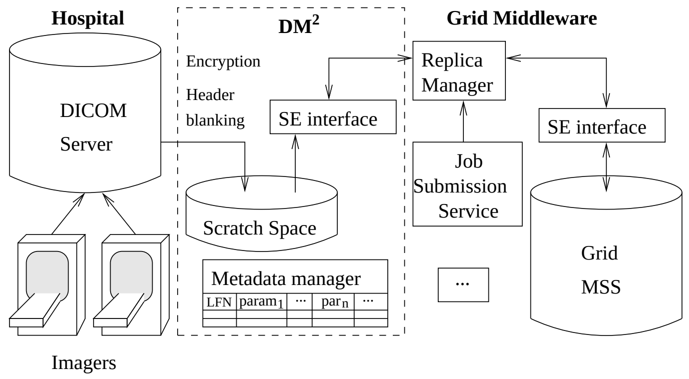


Figure 1: DM^2 interface between the medical imagers and the grid

3.2 Layered architecture

The DM^2 needs to interconnect with existing grid services on the internals of which we have no control [a]. We will refer to *engines* to designate the DM^2 services that we develop to avoid confusions. Each engine is composed of a set of independent processes which interact by exchanging messages. We design each *Distributed System Engine* (DSE) through a layered architecture that takes into account the requirements for high performance and data integrity. The architecture increases in semantic significance through five layers (see figure 2). At the lowest level, DSE^0, the system is made of processes that communicate through a message passing kernel. The DSE^1 level brings atomic operations (*transactions*) to process complex requests composed of many messages. The upper layer DSE^2 deals with distribution over several engines. On top of the distribution layer come the application (DSE^3) and the user interface (DSE^4). This section further describes each layer's role.

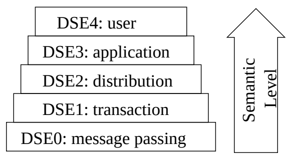


Figure 2: DSE layers

3.2.1 DSE^0: message passing

The core of the DSE^0 is a message passing kernel in charge of the efficient transmission of messages between the DM^2 processes. Our kernel implementation is based on Inter-Process Communication (IPC) services. The DSE^0 transmits messages between the IPC kernel and the external network for access to both local and remote services.

Well known message passing infrastructures such as PVM or MPI, implement a message passing for accelerating processing and reaching a high performance level. DSE uses message passing to build high performance distributed systems. PVM/MPI emphasize on processing power (parallelism) while DSE emphasize on the performance of distributed systems that are sharing information (concurrent messages exchanges). DSE^0 is a message passing mechanism designed to interconnect local processes inside an engine, not for doing remote operations. Using PVM/MPI at this level is unnecessarily costly.

3.2.2 DSE^1: transactions

On top of the simple message transportation layer, DSE^1 implements atomic operations (transactions) made of multiple sub-operations. A transaction succeeds if and only if all sub-operations succeed. If a failure occurs the system is

[a]applications which are not developed for us

left in a coherent state. It offers the ACID properties: Atomicity, Consistency, Isolation and Durability [16].

We deal with three types of transactions that we call as *Queries*, *Tasks* and *Requests*. This allows us to deal efficiently with the complexity of transactions which involves multiple calls to external services and engines. *Queries* are a set of *Tasks and Requests* which can be executed in sequence or in parallel. Similarly, *Tasks* are a set of Requests executing concurrently to shorten the processing time. *Requests* are a set of sequential messages to a service in the network side. *Queries, tasks, and requests* are implemented as specialized processes that we will call as *drivers*. A driver is a process handling different kinds of transactions instead of only messages:

- The QUeries Drivers (QUD) are processes managing a whole transaction (query) made up of a set of *Tasks and Requests. A query could imply concurrent or sequential access to different external services. A Query could be implemented as accessing a TKD but also having direct access to a RQD. This means that a query could be implemented without tasks but only with requests.*

- *The TasKs Drivers (TKD) are processes in charge of a specialized part of a query. This could imply getting parallel access to an external service throught a set of request drivers. TKD offers parallelism, distribution and transparency as a service which could be used by a QUD; e.g, a QUD requests a file to a TKD, but it does not check the localization of the file: the TKD can get it from cache, from a local hospital or from a remote hospital in a transparent way to the QUD. In order to find out the file, the TKDs uses the stuff of availables RQD in the engine.*

- *The ReQuest Drivers (RQD) are in charge of accessing the remote components such as other engines and external servers. They solve low level issues such as connection management. These drivers transmit messages and receive responses that they route to the calling processes* [b]*.*

- *The TOol Drivers (TOD) are processes performing internal operations that are independently implemented for reasons of performance and modularity. Examples of such processes are the caching of requests, files and results, logging, security checking, image processing and manipulation console operations. Tool drivers can be accessed from QUD, TKD and RQD.*

A DM^2 engine works as follows: (i) A message is received by a query driver and a query is initiated, (ii) The query starts different concurrent tasks, using independent processes (TKD), (iii) Each task get access to the requests drivers (RQD) so that it can reach the external services, (iv) The request drivers (RQD) open connections and send messages to the external services, (v) Each driver uses the tools it needs (TOD). See section 4.5 for a detailed example.

[b]In order to improve concurrency and to allows easy modular software development, there is a relationship 1 to N between the processes of QUDs and TKDs; similarly, a relationship 1 to M between the processes of TKDs and RQDs. This means that one query could be splited into N tasks, and each task could access in parallel an external service through a multiprocess RQD

3.2.3 Upper layers

DSE^2 brings additional distributed facilities on top of the two lowest levels. It is in charge of localizing data and services, transmitting requests to proper hosts, collaborating between DSE engines, etc. It may take advantage of completely decentralized Peer-to-Peer (P2P) techniques [15] or to semi-hierarchical tree structures such as LDAP for distributed data localization.

DSE^3 is the application layer, offering a programming interface (API) so that an application can be built on top of the underlying distributed system.

DSE^4 is the user layer. It offers high level access to data, metadata and algorithms registered in the system. It can be implemented as web portals or graphical applications. For instance, a user might have access to similarity algorithms published by the DM^2 architecture (through this portal) that he wants to apply to a subset of images (according to his access rights).

4 DM^2 implementation and usage

A prototype is currently under development to demonstrate the relevance of the proposed architecture in realistic medical applications. At the moment, we have implemented the interface to a DICOM server [c], metadata handling through SQL tables with a secured access interface (which is based on spitfire) [1], but we do not have a fully secured and distributed system yet. Our prototype is therefore an assembly of DSE^0, DSE^1 and DSE^3 processes (message passing, transactions [d] and application functions) following the above architecture. Nevertheless, it is still usable to evaluate these processes in realistic world as described below.

4.1 DM^2 software components

To illustrate our approach and the implementation being developed we chose the usecases described in section 2.3. In order to set up this application we implemented:

- A set of request drivers for sending requests to and getting results from the grid services. The *DICOM RQD* accesses the DICOM server where medical images are stored. The *metadata RQD* is a driver for the database service where image metadata are stored.
- Each of these services has an associated multiprocess task driver which is able to execute concurrent demands. The *DICOM TKD* can make parallel transfer of DICOM files for instance.
- In addition, a *communication daemon* QUD has been implemented in order to receive messages from the network side and to start the execution of queries. The secondary role of this daemon is to manage the load on the

[c]Central Test Node, http://www.erl.wustl.edu/DICOM/ctn.html

[d]three types: query, task and request

server by initiating as many parallel transactions as possible for efficiency while queuing requests when needed to avoid system overflow.

4.2 Extensibility and Interfaces

The architecture allows external applications to interact with the distributed system, and to execute locally on the same node or remotely. The system can access other servers offering additional services (*e.g.* grid services) or be accessed by clients trying to take advantage of the DM^2 services.

A low level API has been developed in order to offer message passing capabilities (layer 0) to an independent system which wants to connect to a DM^2. Those systems must be registered [e] into the DM^2 and they have to use the DM^2 messages passing API. The API offers functions to send and receive messages to from a DSE, using classical synchronous or asynchronous techniques.

It hides the complexity of the message passing mechanisms, makes transparent the IPC calls to the operating system and standardize the message exchanged.

For example, a function for receiving asynchronously a message into a standard list of arguments (*msg_argv* and *msg_argc*) or into a text (*buffer*), in a microseconds timeout interval, is the following:

```
retcode=DSEclient_ASYNCRcv(my_DM2_ID, buffer,
     &op_code, &msg_argc, msg_argv, DSE_API_TIMEOUT_USEC,
     &msg_fm, &snd_response);
```

in this function an operation code (*op_code*) was sent to the process which is waiting for the message (*my_DM2_ID*) and it is also informed whether response is expected by the caller or not (*snd_response*). In this way additional functions such as cache, security, files transfer and encryption, database accesses, image tools, etc, can be easily interfaced, and designed as independent modules for easing software development.

4.3 Security

As mentioned before, security is one of the main requirements for applications concerning medical data. Due to the highly confidential nature of medical data it needs to be protected against the following threats: *Unauthorized reading, modification, deletion and knowledge of existence.* We think that the advantages of using computational grids for medical image treatments outweigh those threats, therefore we need to implement countermeasures to protect the data. Our measures must be designed to provide trustworthy security to patients data and the usefulness of the grid should not be negated by applying them.

Fortunately, grid computing security can be based on existing solutions. Classical solutions for *secure transfer*, *authentication* and *integrity checking*

[e] to be registered into a DM^2 engine means that this engine was configured to accept connections from a specific external application, and this application makes a greeting process with it.

services exist and have been applied to grid computing without major modifications. Common solutions such as TLS/SSL, IPSec and SSH provide these services (although a secure public key distribution mechanism is needed for the authentication part). Grid framework architectures such as Globus-GSI, DataGrid-WP7 [f] and OGSA Security Architecture [g] implement these solutions for grid usage.

However, none of the existing implementations are fully satisfying medical data management constraints. The most important security components is access control. The reason that makes access control special in grid environment is that the data leave the sphere of influence of their owner (preferably this will even be transparent to the owner) and are stored on systems on which he has no control. In order to provide a flexible access control system, we have designed an architecture outlined in [21] that not only respects the limitations mentioned before, but also provides the following functionality:

- Rights granting based on semantic criterions, *i.e.* granting the right to access all *my dental data* to *all doctors of the X dental clinic*, where *my dental data* and *all doctors of the X dental clinic* are the semantic attributes that need not to be specified in the access granting process.
- Possibility of fine granular rights granting, i.e. granting the right to access one specific piece of data to one specific person.
- Storage location independence, *i.e.* the person owning an access right should be able to access any grid replica of the data.
- Offline delegation, *i.e.* data owners should be able to grant access rights even if they are not logged onto the system where the concerned data is stored.

The classical solutions for access control, such as RBAC or ACLs rely on centralized authorization granting, an architecture that does not provide the required scalability in grid environments. Solutions for distributed or grid access control such as Globus-CAS [17], DataGrid-VOMS, OASIS [25], AKENTI [23] or PERMIS [4] do not provide for all of the flexibility specified above. Therefore we have decided to design our own access control system, consisting of three cooperating structures:

1. Signed electronic documents called *semantic access certificates (sac).* Those SACs specify the person to which it is granted (owner), the person which issues it (issuer), the concerned data, allowed access modes and a validity period. Their integrity is ensured by a digital signature from the person who grants them. The person identifiers can be based on their public key, whereas the identifiers for specific files are made by concatenating the owners identifier with a hash of their content. The advantages of the SACs are that they can be granted offline and provide for fine granular

[f] http://eu-datagrid.web.cern.ch/eu-datagrid
[g] globus; http://www.globus.org

access control decisions. Through the mechanism of creating the file identifiers, they are not dependent of the storage location of the data and no further security measures are needed to protect SACs against theft, since to use them one would either have to authenticate as the specified owner or change them and forge a signature of the issuer.

The less fine granular access control decisions are made possible by adding semantic parts to the SACs. This means to replace unique user or file identifiers by a general specification of the group of users or type of data for which the SAC should apply.

2. On the server side an access control module must be running, that checks requests against SACs provided with them and evaluates their semantic parts, if any are used. This module can also enforce access logging, check certain perenity conditions if data is to be deleted or modified and provide special access modes such as anonymized access through the use of owner-specified access modules.

3. To provide the data needed by those systems, we will need to keep a certain extra amount of metadata with the data stored on the grid. We will interface with the metadata RQD and TKD through DSE to store and to retrieve those metadata.

   A user pseudonymization system could also be provided, by allowing using the public key of a data owner as identifier, without disclosing who owns this key. It should then be permitted to data owners to use several public/private key pairs, to limit the damage if the owners identity of one of them becomes publicly known.

   The system will be implemented as shown in step 1 of figure 3, as tool driver of the DSE system. It will be called first when a query arrives and provide a permission granted/denied decision to the rest of DSE.

4.4 Cache

The system performance is sensitive to the quality of the link to hospitals where the DICOM files are stored. This is the reason why the DM^2 uses several levels of cache in order to improve the latency of accesses:

- First, there is a *request caching* layer. The goal is to cache requests and results in order to have a high probability of finding precomputed responses [19] to incoming queries to the system.
- Second, the complete image is expected to be resident into the cache area (*image caching* layer).
- If not, the *file caching* layer works in order to find out some DICOM files (image slices) in the cache area.
- Finally, having no other option, DICOM files are transfered in parallel from the hospital, then registered into the cache. The image is assembled and also registered into the cache.

- All this process is started once somebody initiate a getDM2Image query to the DM^2. But before that, the *Grid Replica Manager*[8] looks for the required file anywhere into the Grid. This is really, the first level of cache for the DM^2.

4.5 Application Layer.

Starting from the detailed usecase described in section 2.3, we will show the use of DSE (see figures 3 and 4).

First, the cardiologist enters a query (*e.g.* to find the MRI of Mr X acquired yesterday in this hospital) through the DM^2 user interface. The DM^2 sends the request to the grid metadata interface through the *metadata RQD* and *TKD*. The user authorizations to access the data are checked by the security *TOD* (step 1 of figure 3a) and the patient file logical identifier and its associated parameters (imaging modality, region of interest, dynamic sequence, MR acquisition parameters, etc) are returned to the user interface.

A request is made to find all images (see step 3 of figure 3b) comparable to the image of interest (same body region, same acquisition modality...) and for which a medical diagnosis is known. The DM^2 layer 2 should be used here to distribute the requests on all hospitals with metadata services. In the current implementation one single metadata service is queried through the *metadata RQD* and *TKD* again. The logical identifier of all images matching the patient source file parameters are returned.

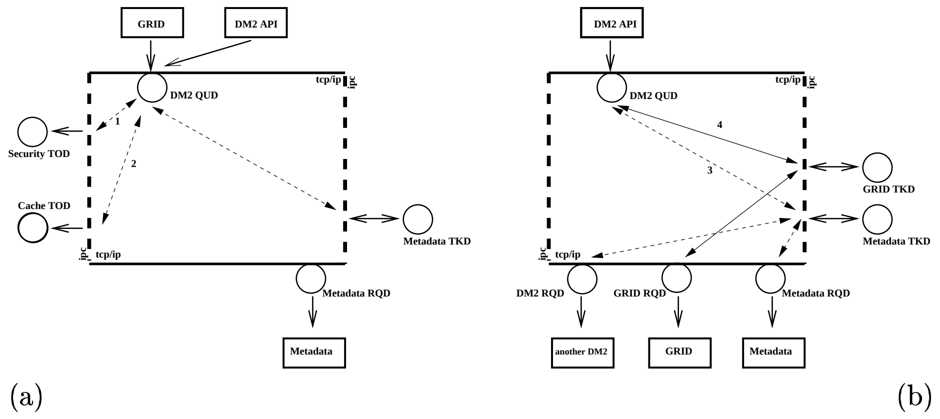


Figure 3: DSE^3 usage example: a hybrid query .
(a) 1-Security check 2- Cache query for stored results, (b) 3- A distributed request is issued to find all images comparable to the image of interest, 4- similarity measure process is issued to the Grid. See section 4.5)

A request is then made for the computation of similarity measures [18, 20] between the patient image and each image resulting from the query (see step 4 of figure 3b). The job submission service of the grid middleware is used to distribute computations over available working nodes. For each job started, the grid replica manager triggers a replication of the input files to process onto grid computation nodes. If the requested files are not registered into the Grid (as a replica), they are requested by the Grid to the DM^2 storage interface. The distant DM^2 asks the DICOM server, assembles MR images on the fly onto its scratch space and returns images to grid nodes.

Figure 4a details the operation; on top, the grid middleware triggers a DM^2 query for getting an image: (1) It first asks for the image to the *cache TOD*. (2) If that image is not available it then accesses the database (*metadata TKD*) to locate the DICOM files from which the image must be assembled. (3) The *cache TOD* is requested again in order to improve the DICOM file's latency access. (4) Assuming the cache does not contain the requested file, it should be copied from the DICOM server. The DM^2 requests the DICOM server through the *DICOM RQD* and retrieves in parallel a set of DICOM slices that are assembled onto scratch space to produce the 3D image requested. (5) The DICOM files are assembled into a 3D image using an *image TOD*. (6) Finally, the image is stored into the cache and returned to the grid - See step 2 in figure 3a.

some code at layer 3

DM^2 offers to the clients an API at application level (layer 3) allowing them to write a very simple C/C++ program to do the above process, as follows:

```
1   strcpy(requestToDM2, WHERE patientID="MR X" AND date="14May2003");
2   inqdm2image (&RequesttoDM2, &Imageslistforpatient);
3   strcpy(DM2imageid, Selectimagefrom(Imageslistforpatient));
4   inqDM2image (DM2imageid, &Algoparametersforimage);
5   inqDM2image (DM2imageid, &ComparablerequesttoDM2,
        &Algoparametersforimage, &Comparableimageslist);
6   inqDM2image (DM2imageid, &Algoparametersforimage,
        &Comparableimageslist, &Similarimageslist);
7   for (Img=0; Img<N; Img++){
8      Myprocess(Similarimageslist[Img]);
9   }
```

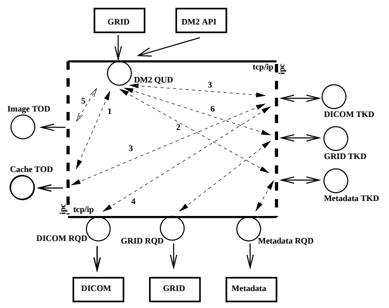
For simplicity we don't include variable definitions. At line 1 and 2 we build a query to the DM^2 system in order to get a list of images for the patient "Mr X" which were acquired in an specific day. One of those images is selected (line 3) considering a user's defined criterion, parameters of interest for a similarity algorithm are computed (line 4), and then an image database is queried (lines 5 and 6) to get similarity measures against the selected image [h]. A list of similar images is returned (line 6) and the user starts his process (lines 7 to 9) to do the image processing tasks.

5 Software evaluation

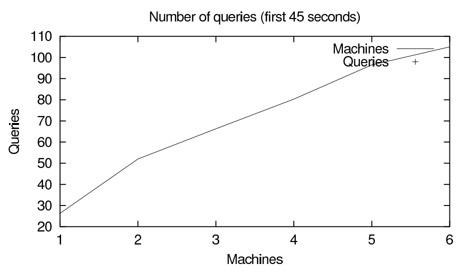
We have done a simulation of the system's behavior for the moments of highest traffic. So far, our test corresponds to the reception of N simultaneous queries for images, which means starting N *getDM2Image* queries (as it was discussed in section 4.5) in parallel .

We use a PC cluster made of 6 processors Intel Pentium 4 with 1 GB in memory. Each machine has a DICOM server in order to simulate 6 hospitals. Those machines

[h]this process must access the Grid in order to take advantage of its computing power because computing similarities issues are usually heavy to process



(a)



(b)

Figure 4: Usage Example and Experiment
(a) retrieving a medical image from the DICOM server (a set of 2D slices composing a 3D image for instance) , (b) solved image queries in the first 45 seconds when the number of hospitals grows up to 6.

are accessed from a remote server (Pentium 4, 1.7 Ghz, 512 MB RAM) where a DM^2 engine was installed. There is a network of 100 Mbits between them and a firewall controlling access. The engine was configured with 6 DICOM RQD, having each one 3 concurrent processes. We:

- Use images composed of 3 slices each
- Use slices (DICOM files) of 0.5 MB and then transfer in parallel
- Assemble an image of 1.5 MB (0.5MB x 3) each time.
- Deliver until 38 local messages per transaction (using the IPC resource *queue message* of the operating system). This is refereed to messages going between the different drivers (*QUD, TKD, RQD and TOD, inner the engine) without considering the remote messages*
- *Start one query transaction by image requested and 1 task by slice into the image, which means 3 tasks in parallel by requested image (each query started).*
- *Open 3 channels per DICOM server, which means 18 concurrent channels (18 DICOM file transfers in parallel).*

Figure 4b shows the engine's behavior in number of solved queries during the first 45 seconds, when we have different number of concurrent machines (hospitals). The engine processes 26.3 transactions in the first 45 seconds when images are recovered only from one hospital, but up to 105 transactions in the same period of time, when images are simultaneously recovered from 6 hospitals. This means an speed-up from 1 to 4 when available machines have increased from 1 to 6. We consider it very promising because of the asynchronous nature of DICOM transmissions.

6 Conclusions

Medical image processing over the grid opens new opportunities for applications involving large medical datasets analysis. However, full deployment of medical applications require medical sites to be interconnected and grid middlewares to provide secure and efficient access to the distributed and sensitive medical data. The semantic content of medical data should also be taken into account by developing grid-wide tools to manage associated metadata.

The architecture proposed in this paper allows us to build a complex distributed system, taking advantage of classical theory (transactions concept) and proposing solutions to implement a high performance data manager (decomposition of queries in concurrent tasks and requests). The DM^2 system allows the physicians to get secure access to their patients' images and to send hybrid requests over huge databases.

We implemented a first prototype to demonstrate the relevance of the DM^2 for realistic medical applications. The stressing tests and the interfacing experiences with other projects, show that results are promising about performance and extensibility.

While developed in the field of medical image management, our proposal could be generalized to many distributed systems where security has to be enforced.

On-going work concerns data security, caching, distribution, hybrid requests and content based queries. Many other aspects related to medical data management were not addressed in this paper including the need for tracking data origin and logging data processing.

Acknowledgments

This work is partly supported by the European DataGrid IST project, the French ministry ACI-GRID project, and the ECOS Nord Committee (action C03S02).

References

[1] William H. Bell, David G. Camero, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. Simulation of dynamic grid replication strategies in optorsim. *International Journal of High Performance Computing Applications*, 2003.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level Scheduling on Distributed Heterogeneous Networks. In *Supercomputing*, Pittsburgh, PA, USA, November 1996.

[3] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and Welch V. A National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12):60–66, 2000.

[4] D. Chadwick and A. Otenko. The PERMIS X.509 role based privilege management infrastructure. In *Proceedings of Seventh ACM Symposium on Access Control Models and Technologies, SACMAT*, pages 135–140, 2002.

[5] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23(3):187–200, July 2000.

[6] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. *Lecture Notes in Computer Science*, 1459:62, 1998.

[7] DataGrid project. http://eu-datagrid.web.cern.ch/eu-datagrid.

[8] DataGrid wp5. Architecture and design, decembre 2001. DataGrid WP5.

[9] DICOM: Digital Imaging and COmmunications in Medicine. http://medical.nema.org/.

[10] Hector Duque, Johan Montagnat, Jean-Marc Pierson, Isabelle Magnin, and Lionel Brunie. DM^2: A Distributed Medical Data Manager for Grids. In *Biogrid 03, Tokyo May 12th to 15th 2003, proceedings of the IEEE CCGrid03.*

[11] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[12] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, July 1998.

[13] Francesco Giacomini, Francesco Prelz, Massimo Sgaravatto, Igor Terekhov, Gabriele Garzoglio, and Todd Tannenbaum. Planning on the grid: A status report. ppdg-20, particle physics data grid collaboration., October 2002.

[14] H. K. Huang. *PACS: Picture Archiving and Communication Systems in Biomedical Imaging.* Hardcover, 1996.

[15] Ion. STOIKA and Robert. MORRIS and David. LIBEN-NOWELL and David. KARGER and M. Frans. KAASHOEK. Chord: A scalable peer-to-peer lookup service for internet applications. Technical report, MIT Laboratory for Computer Science, january 2002.

[16] Sape. Mullander, Michael. Schroeder, Fred. Scneider, and William Weihl. *Distributed Systems.* Addison Wesley, 1993.

[17] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proceedings of the 2002 IEEE Workshop on Policies for Distributed Systems and Networks*, 2002.

[18] G.P. Penney, J. Weese, J.A. Little, P. Desmedt, D.LG. Hill, and D.J. Hawkes. A Comparison of Similarity Measures for Use in 2D-3D Medical Image Registration. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'98)*, volume 1496 of *LNCS*, pages 1153–1161, Cambridge, USA, October 1998. Springer.

[19] Jean-Marc Pierson, Lionel Brunie, and David Coquil. Semantic collaborative web caching. In *WISE2002 : Web Information System Engineering*, pages 30–39, Singapour, December 2002. IEEE CS Press.

[20] A. Roche, G. Malandain, X. Pennec, and N. Ayache. The Correlation Ratio as a New Similarity Measure for Multimodal Image Registration. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'98)*, volume 1496 of *LNCS*, pages 1115–1124, Cambridge, USA, October 1998. Springer.

[21] L. Seitz, J. Pierson, and L. Brunie. Semantic access control for medical applications in grid environments. To appear in the proceedings of Euro-Par 2003.

[22] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and object replication in data grids. In *10th IEEE Symposium on High Performance and Distributed Computing (HPDC2001)*, August 2001.

[23] M. Thompson, S. Mudumbai, A. Essiari, and W. Chin. Authorization policy in a pki environment. In *Proceedings of the 1st Annual NIST workshop on PKI*, 2002.

[24] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolsky, and M. Swany. A grid monitoring architecture. technical report. gwd-perf-16-2, global grid forum, January 2002.

[25] W. Yao, K. Moody, and J. Bacon. A Model of OASIS Role-Based Access Control and its Support for Active Security. In *Proceedings of Sixth ACM Symposium on Access Control Models and Technologies, SACMAT*, pages 171–181, 2001.