Journal of Digital Imaging (2018) 31:628–639
https://doi.org/10.1007/s10278-018-0091-x

A Dependable Massive Storage Service for Medical Imaging

Marco Antonio Núñez-Gaona[1] · Ricardo Marcelín-Jiménez[2] · Josefina Gutiérrez-Martínez[3] · Heriberto Aguirre-Meneses[1] · José Luis Gonzalez-Compean[4]

Published online: 18 May 2018

Abstract
We present the construction of Babel, a distributed storage system that meets stringent requirements on dependability, availability, and scalability. Together with Babel, we developed an application that uses our system to store medical images. Accordingly, we show the feasibility of our proposal to provide an alternative solution for massive scientific storage and describe the software architecture style that manages the DICOM images life cycle, utilizing Babel like a virtual local storage component for a picture archiving and communication system (PACS-Babel Interface). Furthermore, we describe the communication interface in the Unified Modeling Language (UML) and show how it can be extended to manage the hard work associated with data migration processes on PACS in case of updates or disaster recovery.

Keywords Distributed storage system · PACS · DICOM · Massive information storage & retrieval · Fault-tolerance and scalability

Introduction

The amount of digital information that the world is accumulating, as well as the increasing rates under which information has been produced, have become the driving forces that move the scientific community to find new solutions, in order to face the so-called "deluge" of information [1, 2]. As storage has become a strategic asset, we are witnessing the evolution of storage technologies. Not only scientific computing but also the government, banking, and any IT-based organization will be the beneficiaries of this shift in massive storage [3–5]. In the short term, IT managers will be required to make an important decision concerning the storage capacities of their corresponding organizations: whether storage is supported based on their own resources, or as a service that is provided by outsourcing [6–9].

✉ Marco Antonio Núñez-Gaona
mnunez@inr.gob.mx

[1] Department of Technological Development, Instituto Nacional de Rehabilitación, Calz México Xochimilco No. 289 Col. Arenal de Guadalupe, 14389 CDMX, México

[2] Department of Electrical Engineering, UAM – Iztapalapa, Also collaborates with the research team at INFOTEC, CDMX, México

[3] Division of Research in Medical Engineering, Instituto Nacional de Rehabilitación, CDMX, México

[4] Cinvestav, Unidad-Tamaulipas, Tamaulipas, México

We understand that people need to store their private information and, for this, they can resort to either a physical device, or a virtual device. This is, in the first case, they choose a product and, in the second case, they choose a service. In either of the two situations, the user is the owner of the information and can independently choose the solution to save his or her information.

As for the storage of organizations, we have that the users depend on the access policies fixed by their organization and may be limited in the related decisions with the use, storage, and transport of information. IT managers have, among other responsibilities, to implement primary and secondary storage solutions, to mitigate risks, foreseeing contingencies and even disasters. In this context, they have to decide again whether, to build these capabilities, they buy a product, or they hire a service. However, the answer is not as simple as in the case of personal storage, because this time, the volume plays a key role to consider.

Whether it is the storage of a person or an organization, the solution chosen in both cases must take into account that access to information must be done under the control of its owner or the authorized persons. However, due to the volume, it is usually easier to guarantee the confidentiality of the number of documents of a person, if we compare them with the quantity of documents that an organization can classify as sensitive. If the managers of an organization are interested in buying a product to solve their storage needs, they would be assuming the cost of their management in exchange for maintaining control of access to their information. If, on the other hand, they were inclined towards the purchase of a service, they would be deciding to release the cost of the management, but relinquishing control of the storage details.

In the first instance, it would seem that an organization should not relinquish control of the details related to the storage of its information; however, there are two requirements to consider. Normally, the requirement of availability of information increases with the size of the organization. Besides, information grows even at exponential rates, and that means that the management of storage devices can become a task that absorbs more and more resources of the organization itself. The cost-benefit analysis must have a medium- and long-term horizon where the availability of information and the scalability of its storage are key elements in the decision-making process [10–14]. This scenario, related to the storage of organizations, is the context of our work.

The Babel Storage System (after "The Library of Babel," a short story by Jorge Luis Borges [15]) is a dependable, scalable, and flexible software-defined storage system, developed by INFOTEC. Among its main features, it can be underlined the availability of different types of data redundancy, a careful decoupling between data and metadata, a middleware that enforces metadata consistency, and its own load balance and allocation procedure which adapts to the number and capacities of the supporting storage devices. It can be deployed over different hardware platforms, i.e., fully hardware-agnostic. Enterprises and Service Providers are challenged to build scale-out storage infrastructures that support multiple application workloads and provide the highest data resiliency with the current technical limitations and costs of current traditional vendors. To stay flexible, efficient, and cost-effective, Babel provides different mechanisms for protecting data including replication and the information dispersal algorithm (IDA) [16]. Babel supports objects (HTTP/WebDAV/REST) [17, 18], easily integrating into many standard and custom applications.

Health institutions produce a huge number of digital images from clinical studies that may reach more than 100 thousand radiology procedures per year, which implies the preservation (also known as retention) of dozens of terabytes of uncompressed historical data. The minimum retention time of medical information depends on national regulations (for example: a study of x-rays in the USA has a retention period of 5 years from its generation [19], while in Australia, it is 7 years [20]). In Mexico, the institutions certified to provide health services must guarantee a retention period of 5 years [21] for all information concerning the health status of patients, the corresponding files weigh between 3.5 and 1500 MB and, according to national regulations, must be kept under very rigorous requirements of integrity, availability, and privacy.

We have developed an interface that connects Babel with the PACS (picture archiving and communication system) at the INR (Instituto Nacional de Rehabilitación) [22] which is a system to transmit, store, retrieve, and display medical imaging information, in strict conformance to DICOM (Digital Imaging and Communications in Medicine) [23] which is the international standard on this field. In this work, we present the design principles followed in the construction of this communication interface, we consider that the value of our proposal stands on the feasibility of an alternative approach, to meet the most stringent requirements on medical image storage.

The new trends in health sciences (such as genomics) will benefit from massive storage technology and indeed, our storage architecture may be exported to these scenarios too. Nevertheless, medical imaging will always be a key tool supporting health services. In this sense, we can grant that the PACSINR [22], being a system that is deployed and working in a National Institute, is fully complying with all the regulations required to provide support on such critical environments. Every day, at its most demanding moment, the PACS supports 300 concurrent users, including surgery operations, this shows that this is a highly dependable system. Medical image management procedures are continuously evolving to benefit not only from new image technologies, but also from the latest achievements on information management. This is the context that frames our contribution, we present a storage model that is not tied to any particular infrastructure and, as we already mentioned, can be exported to any situation where information is required to remain available over long terms, which implies the challenge of volume and scalability. We considered that the PACS at the INR has provided us with a very demanding testbed.

The rest of this paper includes the following: in "The Babel Storage System in a Nutshell," we introduce the main aspects of Babel itself. In "The Implementation Details of the Communication Interface," we show the main software elements and components than resolve the communications between a DICOM storage server application entity and the Babel Storage System. Finally, in the"Discussion" and "Conclusion" sections, we analyze the results and the directions for further work.

The Babel Storage System in a Nutshell

Storage systems can be classified in three main categories: file oriented, block oriented, and object oriented.

Files and blocks are based on the concept of file system. Most of the users are familiar with them, such as FAT, NFS, CIFS, and EXT. They organize data into files and folders in a tree-like hierarchy and give a path to the file while also retaining a small amount of metadata about this central entity. The key difference between file access (deployed on a NAS) and block access (deployed on a SAN) is that in the first case, the file system resides on the disk array. Meanwhile, in block access, the file system is external to the array and I/O calls are handled by the file system on the server, with only block-level information required to access data from the SAN. From this distinction can be explained the different scenarios where they are applied. NAS is best fitted to retention and access of entire files and has locking systems that prevent simultaneous changes and corruption to files. Meanwhile, SAN systems allow changes to blocks within entire files and so are extremely well suited to database and transactional processing.

In the other hand, as the name suggests, object-based storage allocates data in isolated containers known as objects. Each single object has a unique identifier and is stored in a flat memory model. This is important for two reasons: (i) an object can be rapidly retrieved by simply presenting its identifier (ID), thus making information much easier to find in a large pool of data. (ii) The data could be physically stored on a local server, or a remote server in the cloud. Object storage also lends much greater flexibility to metadata. Scalability is where object-based storage does its most impressive work. Scaling out an object architecture is as simple as adding additional nodes to the storage cluster.

As we have already pointed out, scalability has been a driving force in our design. Based on the fact that the DICOM standard enforces the compliance on the procedures, independently from technology, we consider that object storage (which is supported by Babel using REST-ful services) has been the right approach that will provide a long-lasting storage service.

We have built a distributed system that can be initially deployed on a small group of storage devices and offers a capacity of a few terabytes, but, if required, it may incorporate additional devices to achieve a massive scale, even of tens of petabytes (Fig. 1).

The key performance goals to consider in our construction are availability, fault-tolerance, and scalability. To meet these goals, we have considered two basic mechanisms: (a) two alternative information redundancy techniques and (b) a modular data placement algorithm. The former refers to the redundant information produced in order to attain availability and fault-tolerance. The latter refers to the (re)allocation of data within the available storage devices, in order to support changes and enforce load balance.

1. Redundant information can be obtained from simple replication that generates n copies of the original file. This is the principle followed by *GFS* [6] and *HDFS* [24]. Alternatively, redundancy can also be produced using error-correcting techniques such as the *IDA* [16, 25]. Traditional systems allocate several copies (replicas) of the object been uploaded, on the different devices that make up the cluster of disks. For instance, let us assume that a medical image of 1 GB is uploaded. Normally the system takes two additional copies, and finally, three instances are allocated on three different devices. The intention of data redundancy is to guarantee that even if various storage devices go out of service, with a good probability, there will be a device that remains available. Chances improve with a bigger number of copies of the same object. Nevertheless, the price to pay is the effective overall storage capacity. In our example, we say that each object occupies 300% of its original size, in storage capacity. In other words, it reduces the initial raw capacity to 1/3. In contrast, Babel uses a coding technique known as the IDA. Let O be an object to be stored, which occupies |O| bytes. Using IDA, O is transformed into n objects called dispersals, each of size |O|/m. O can be recovered provided that any m out of n dispersals remain available. In this case, the object occupies n/m × 100% of its original size and it is possible to tolerate the absence of any n-m storage devices. Compared with the initial example, we can transform the same object into $n = 5$ dispersals (for instance), each of size |O|/3; therefore, we produce an excess of information equal to (5/3) × 100% (vs 300% of the initial approach), and even if we lose any 2 (=5-3) of them (Fig. 2), it will be possible to recover the original object. Using IDA, the effective capacity is 3/5 of the raw capacity (vs 1/3 of the initial approach). Our current implementation of IDA achieves processing times that are similar to those observed in replica-based systems. Nevertheless, considering different scenarios, we decided to accommodate replication and IDA to offer alternatives to fit the requirements of different applications.
2. Data placement schemes (dps) must be designed considering the number of data units produced by the redundancy mechanism, herein referred to as "blocks." Besides, any pair of blocks resulting from the same redundancy operation must be allocated on two different storage devices. Otherwise, the failure of this very device could compromise the availability of redundant information and, therefore, the recovery of the initial object.

A data placement scheme is required to answer a very simple question: what is the storage device in charge of a given block? This question is issued at different moments during the block's lifetime, (i) the first time it is uploaded, (ii) every time it is retrieved. Nevertheless, a distributed system is a dynamic entity whose components may change over time. Indeed, there is a third condition to invoke the dps, (iii) when a new storage device has been incorporated either to replace a faulty one, or to scale up the overall system capacity. On this last circumstance, the dps helps the system to recover its load balance. A dps builds a link between each block and the storage device where it is allocated. In our solution, this link is calculated each time that it is required, based on a fixed attribute of the block, that we call its signature (such as its name, or its date of creation). This attribute is introduced to a given hash function that returns a position on a virtual address space. In turn, the virtual space is partitioned into parcels, and each parcel is assigned to a different device.

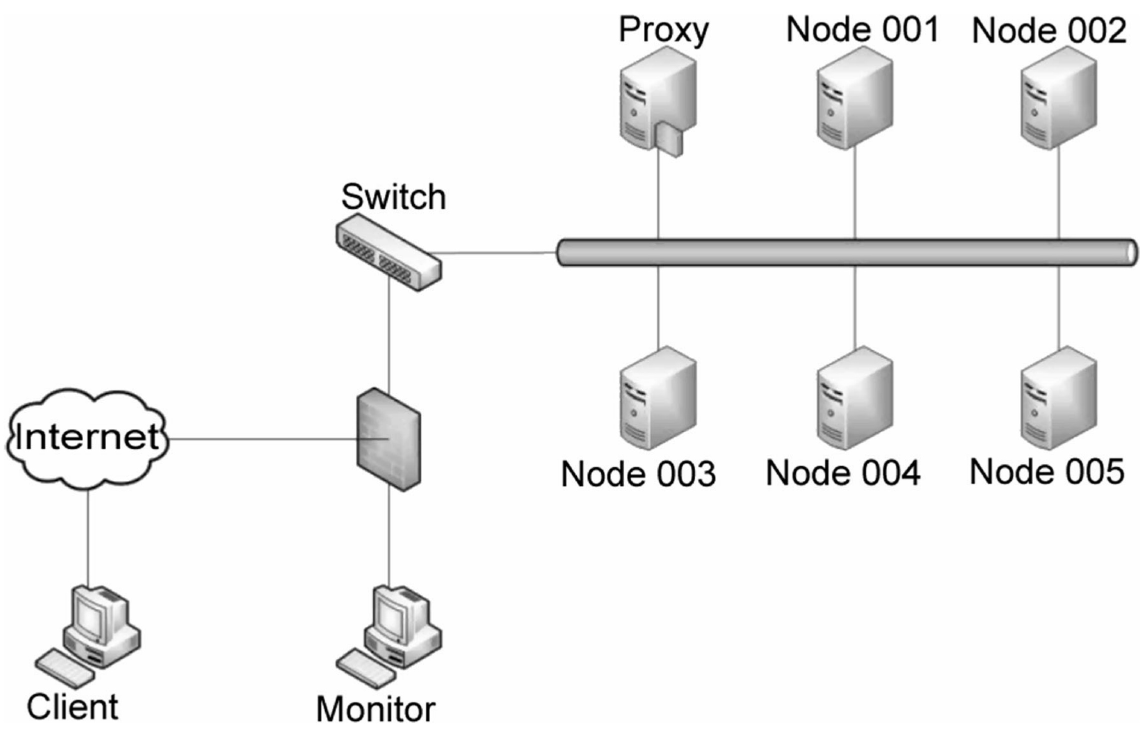


Fig. 1 The Babel Storage System consists of a set of machines with storage and processing capabilities, connected through a local network. Babel customers perceive only one machine, called a coordinator or proxy, which dispatches their requests for file storage and retrieval

One of the key design principles in object storage systems is the decoupling between data and metadata. The former are redundantly stored at the cluster of disks, while the latter correspond to a replicated database in charge of the system servers also known as proxies. This decision is the base on which scalability is achieved: data may be reallocated over time, but the information required to retrieve this data (i.e., the metadata) remains always at the same places. Notice that a redundant set of proxies prevents bottlenecks, but mainly introduce redundant functions that build fault-tolerance. Nevertheless, they also imply a potential inconsistency on the replicated information that each server has recorded. Also, as the load balance procedure is invoked, metadata may become out of date. The consistency of a replicated DB has been guaranteed by means of a middleware that enforces the updating of any DB replica, even if the corresponding server experiences temporal failures. This means that thanks to its redundant design, Babel is a dependable system that remains in full operation even when some of its components are degraded.

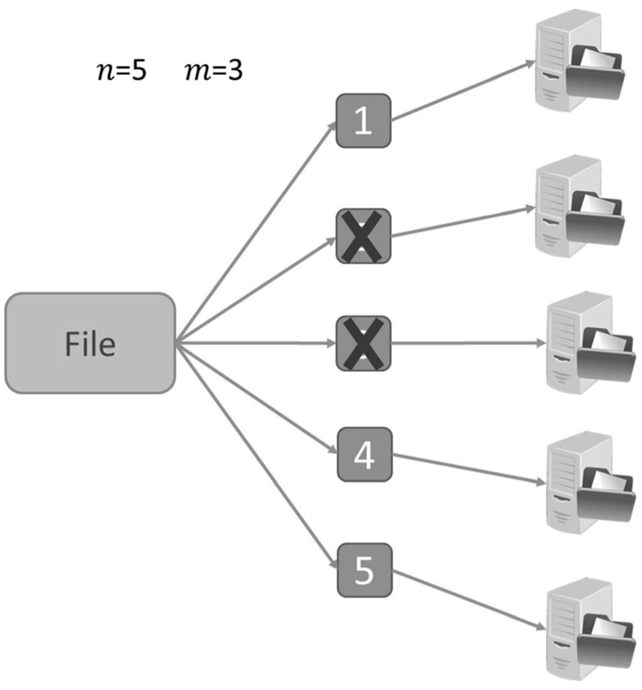


Fig. 2 Example of redundancy using IDA, a file is transformed into five dispersals and only three are needed to rebuild it again

The Implementation Details of the Communication Interface

Babel as a Storage Component of a PACS

A picture archiving and communication system (PACS) is an integrated medical image management system, designed to replace the film hardcopy by digital images. It consists of medical image data acquisition, storage, and display components, all integrated into a high-performance network [22].

The major components in PACS consist of an image and data acquisition gateway, a PACS server and archive, and several display workstations [26]. The PACS server is responsible for storage, retrieval, and clustering of multimodality images, as well as the access to the database through security mechanisms.

Digital Imaging and Communications in Medicine (DICOM) is the standard used in PACS systems for producing, storing, displaying, processing, retrieving, querying, and printing medical images and its associated information.

DICOM enables the integration of scanners, servers, workstations, printers, network hardware, and storage devices from multiple suppliers. It includes a file format definition, containing image and the patient's data, as well as communications rules that use the TCP/IP suite to exchange information between two systems or AE (Application Entities) [23]. The different devices that make part of a PACS must have DICOM conformance statements to describe the service classes that they support. The images, reports, and the patient's data represent data units which are called information object definitions (IODs).

A PACS should be able to support several simultaneous clients, either connected by means of a local area network or by a wide area network, in order to exchange digital images, regardless of the place where its services are required [26].

There are PACS schemes that do not offer a whole scheme of security (Manufacturer's Image Acquisition Device Network), database backups (field service engineers external task) and secondary storage servers [26], or any type of business continuity plans, so that under critical circumstances, information recovery may require considerable time-consuming processing, or even manual migration from the archived image data. Also to improve the efficiency, effectiveness, and safety of electronic health systems, it is necessary to incorporate the recommendations of the HIPAA (Health Insurance portability and Accountability Act of 1996) [27] associated with patient data privacy [28] taking into account that in the early PACS design, the HIPAA recommendations were not considered.

The DICOM standard is strongly oriented to the fulfillment of the image management procedures. Therefore, designers have sufficient leeway to decide the technologies that support each of the critical procedures. This implies that some recovery procedures, though available, might not be fully automatized (or optimized). Image preservation is the so-called core business of any PACS. Nevertheless, the amount of information to be preserved imposes a structure that divides the storage capabilities in two main categories: primary and secondary. Primary storage is where studies are initially allocated; it is required from these devices to offer small transfer latency. In contrast, secondary storage is the support of the long-term archiving. These devices must be able to accommodate a massive volume of studies.

Depending on the storage policies fixed by the organization, studies migrate from the primary to the secondary at some point of their lifetimes. This feature prevents the primary from being overloaded, limits its size, and allows the usage of solid state devices, for instance, which have very low latencies, but do not offer big capabilities. We propose the usage of Babel as an alternative to secondary storage. In our solution, each study that is received at the primary server is automatically backed up at Babel in a transparent way. Also, the catalog that describes the collection (logs and metadata from the PACS) is regularly backed up, enabling disaster recovery procedures. Should a study that has been eliminated from the primary server is required, it will be automatically recovered from Babel in a transparent way.

We call this new organization "the closed library model," because, as it happens in some libraries, users are not allowed to directly interact with the entire collection which remains closed for them (for security issues). Instead, there is an authorized clerk that stores and retrieves any document from the shelves. We built an automatic clerk, which is the only authorized to interact with the library of Babel.

We should emphasize that we built an interface that connects the PACS primary storage with Babel, as the secondary storage. At the PACS side, according to the DICOM standard, there must be a log that supports any possible audit and the same happens on the side of Babel (and the information recorded on each side is complementary). Although DICOM does not require it in a compulsory way, the architecture of Babel allows an encryption or ciphering stage before storing objects, but we did not consider it necessary because all the objects are fragmented and encoded before being saved.

On the other hand, we want to underline that, according to INR regulations, we developed the initial interface between a substitute PACS and Babel. In this stage, we also tested the upload and download speeds that support image transfers from both sides. Results show that we can fully migrate the secondary storage from its current server to Babel within a couple of months. In the second part of our project, both storage systems are working in parallel. We plan to shut down the commercial server, at the end of its lifetime, when Babel will have proved to be completely compatible with the INR requirements.

We built a test PACS prototype (storage server component) that supports a subset of the services described according to the DICOM specifications (Figs. 3 and 4). Our design is based on the *PixelMed* [29] Java DICOM toolkit, which is a set of free/libre and open source libraries implementing code for reading and creating data, network and file support, object database management, display of directories, and images.

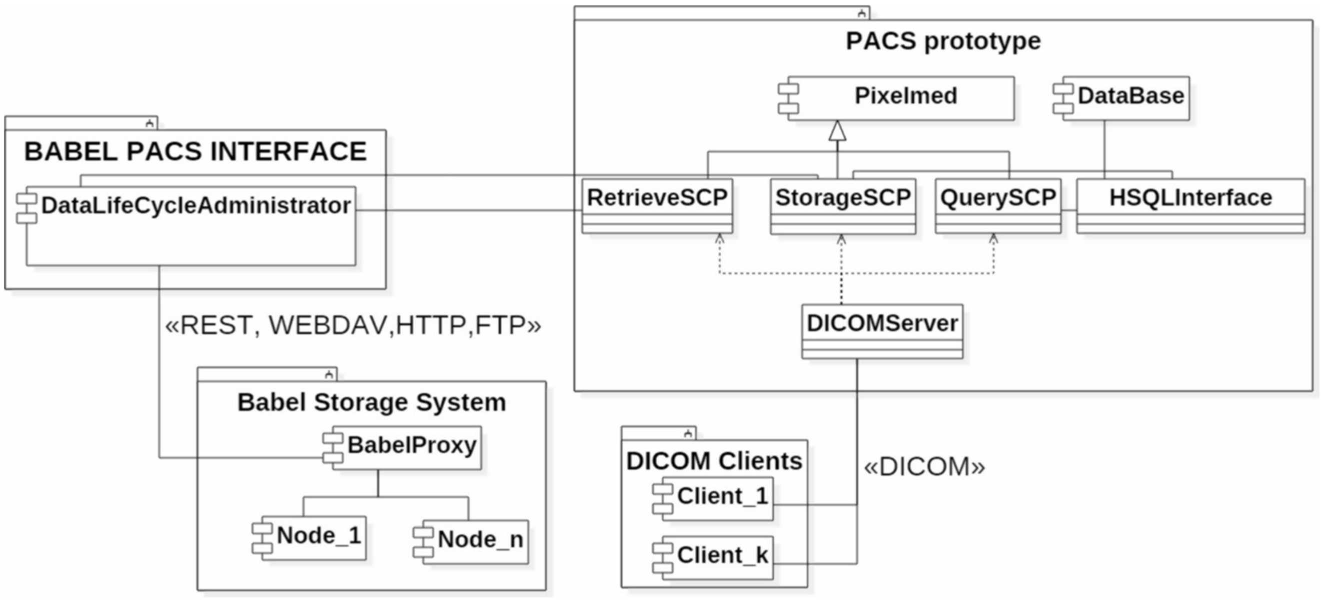


Fig. 3 Deployment diagram: the Babel Storage System is the main storage repository it supports REST, WEBDAV, and FTP communications protocols; the PACS prototype supports DICOM services (Storage, Query, Retrieval) to manage medical images; the PACS-Babel interface is responsible for medical images exchanging between the source PACS local file system and Babel, via REST-ful services

This prototype server is structured having the following layers:

1. DICOM communications: provides a normalized information exchange between AEs (PixelMed libraries).
2. DICOM services: inherits functionalities from the aforementioned layer and extends these capabilities, to support the hypertext terminal protocol (http); it also implements a database interface to support a connection with a database in order to control image identification.
3. DICOM storage: recognizes the normalized database scheme, including the patient's data, studies, series, and images. It is important to mention that this layer records a file-fingerprint granted by Babel, corresponding to each instance unique identifier (UID) of a SOP (Service Object Pair).

The storage server contains a database to store IOD datasets; it also offers four DICOM services: storage (StorageSCP), query (QuerySCU), retrieval (RetrieveSCP), and verification (EchoSCP), in order to support information exchange with an AE (DICOMClient) (Fig. 3).

When an AE (DICOMClient) requires storing an IOD, the storage server extracts the DICOM dataset from the IOD and records these properties in a database according to a normalized structure. Next, the server contacts Babel's proxy and submits the IOD. When this operation is successfully accomplished, the proxy replies with the ID that has granted to the IOD just stored. Then, the server updates its database attaching the ID to the information already recorded, concerning the IOD.

In the short term, the PACS prototype will be replaced by the PACSINR, which is a system in production. Nevertheless, according to the design principles that we settle, the interface obeys a well-defined pattern and a standard protocol that allows a smooth transition to the operational stage. The interface was developed in the Java language, based on the domain model (Fig. 5) which is a well-known resource from software engineering. Next, we defined a temporary buffer in which the PACSINR simultaneously stores a copy of the studies allocated at its primary server. Finally, we developed a component that verifies the status of the buffer and automatically backs up in Babel, each new study that is detected.

The PACSINR-Babel interface performs several tasks, including file exchange, as well as file system analysis to determine DICOM images migration (data life cycle management). It also offers a web GUI (graphical user interface) (Fig. 6). The figure shows the main functional requirements of the interface. In the first instance, the administrator is signed in and executes the use case DataLifeCycleManagement, which extends the three basic suboperations:

1. FileSystemManagement. Provides mechanisms to manage (create, monitor, and update) the file systems where the PACS server stores the DICOM images.
2. RulesManagement. Provides automated file system policies for file management (creation, updating, and deleting), including policies for image size, creation dates,

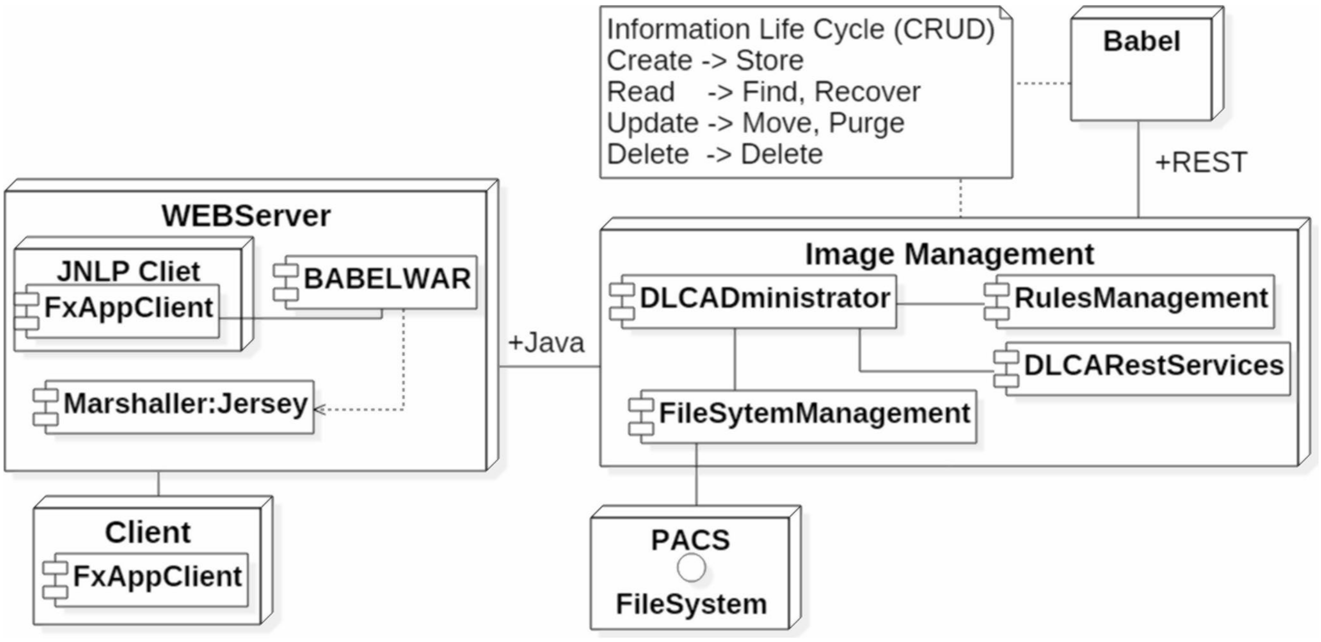


Fig. 4 PACSINR-Babel software architecture. The deployment diagram shows the general components of the proposal. The interface has access only to file systems where medical images are stored, and according to a set of rules, images are sent to Babel using REST web services

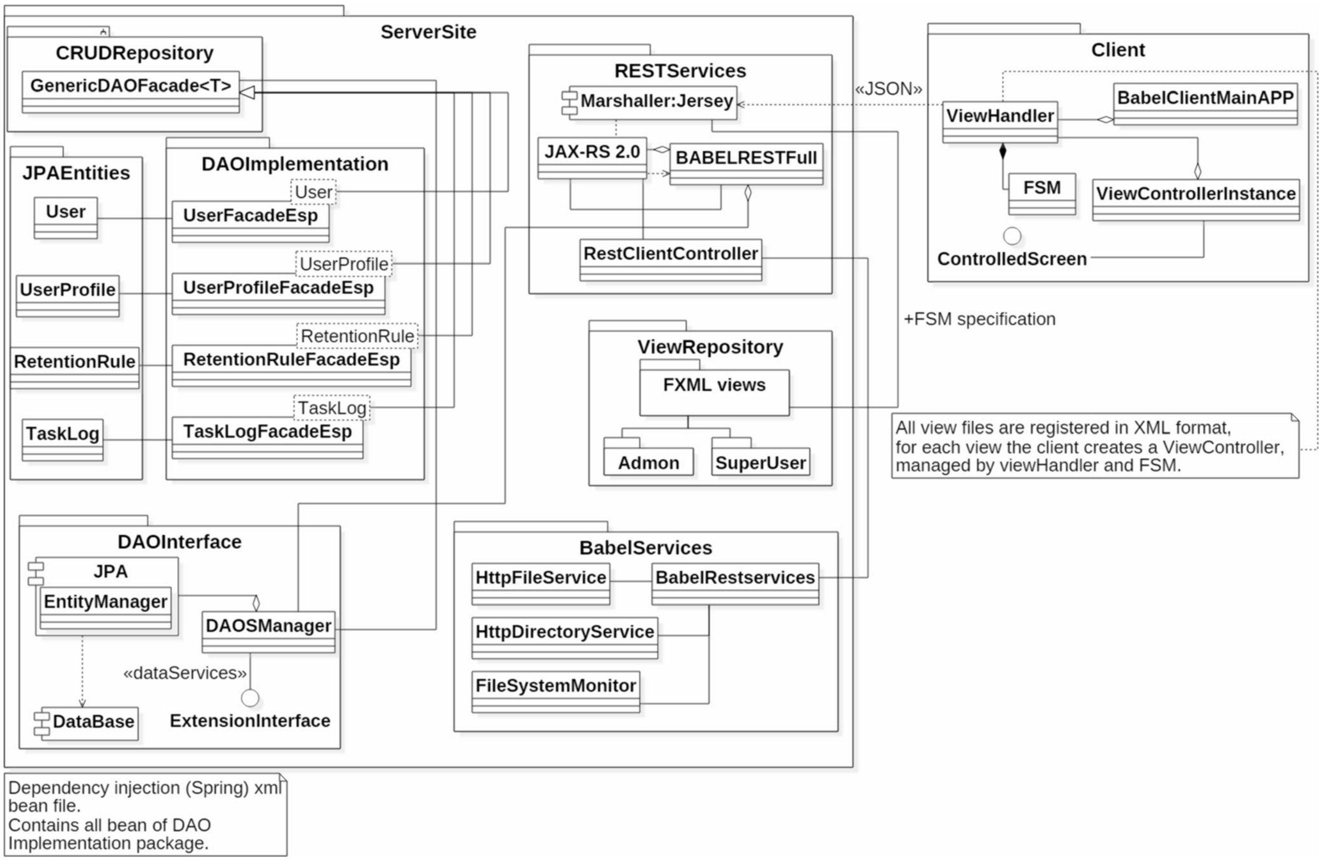


Fig. 5 Design model of the PACSINR-Babel communication interface, extends an MVC architectural style. The model is supported with dependency injection pattern and facade, the controller is supported with a REST-ful implementation, and the view is supported with a finite state machine (FSM). This proposal facilitates the maintenance of the architecture and decouples the dependencies between components

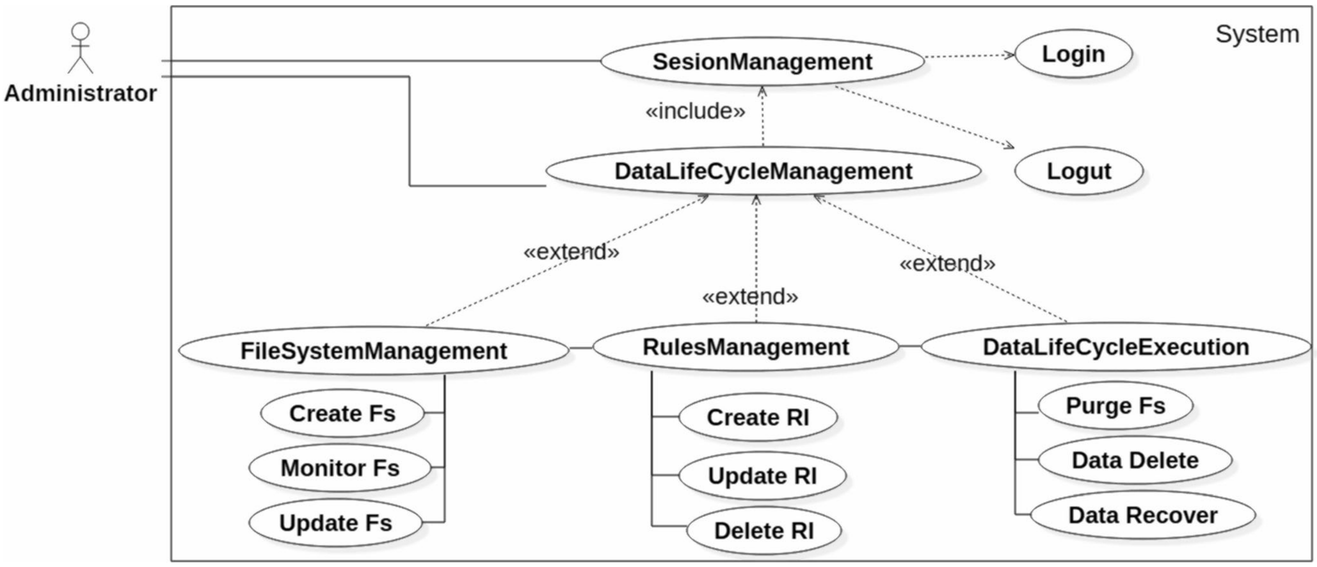


Fig. 6 Data life cycle management use case diagram. Only the administrator is allowed to execute the described operations for the file systems through the established rules. The diagram only shows this scenario, but the interface has the functionality to create users with different access profiles, provides REST-ful services, and manages the GUI through a finite state machine (FSM)

modalities (according to DICOM), as well as free space management.

3. DataLifeCycleExecution. This use case is responsible for automatically migrate target files, according to the defined migration policies. It also manages file systems utilization by automatically moving DICOM images from the primary file systems to Babel. Should the operation be successful, Babel returns a unique file identifier (UFID) for each DICOM image stored. This UFID is recorded in a binnacle (a database table), being the only way to access the image.

Table 1 shows the time it takes to store a study for each supported modality at the PACSINR. Notice that, depending on the type of study, it might include several frames; therefore, the total amount of information can be estimated multiplying the individual image size, times the number of frames included on the given modality. It is also worth considering that storage time does not only include communications and writing operations, but it might also perform transactions on a database using the DICOM Query/Retrieve service. This introduces a higher variability that explains the difference on service times between study 2 and study 3. This table can be understood as a sample (or snapshot) that was taken during the rush hour at INR network. In addition, the areas where the modalities come from do not necessarily belong to the same network segment where the storage server is connected. The last column shows the average throughput during the corresponding storage operation, which at the most demanding moment (in study 2) achieved 26.74 Mbps. In contrast, the link to Babel offers a speed of 185 Mbps that can easily accommodate the requirements of the INR. Our preliminary conclusion is that neither the architecture of our interface, nor the speed that supports the link, limit the adoption of Babel as an alternative for secondary storage.

The PACSINR-Babel Interface Architecture

The communication interface design is structured according to the Model-View-Controller (MVC) architectural style as it facilitates re-usability due to low coupling (separates application data, user interface, and control logic) between pattern elements. The *model* layer implementation is developed through a generic service class; this class enables the CRUD (Create, Read, Update, and Delete) database operations with the support of the dependency injection and facade patterns. In turn, the DAOSManager class handles the concrete classes through beans (configuration file in XML format) to declare the actual dependencies. The *controller* layer implementation is based on REST-ful services; it supports the client in charge of the interface management and uses an API that automatically sends to Babel all the DICOM images that are initially stored in a preconfigured file system. Finally, the client layer handles the mechanisms to show the *views* related to the role based access control. It is implemented using a finite state machine (FSM) that describes the navigation between views based on a descriptor relating the actions selected by a given user (Fig. 8).

Dynamic Behavior

To understand the file monitoring process, it is necessary to use another type of diagram where it is captured the way a

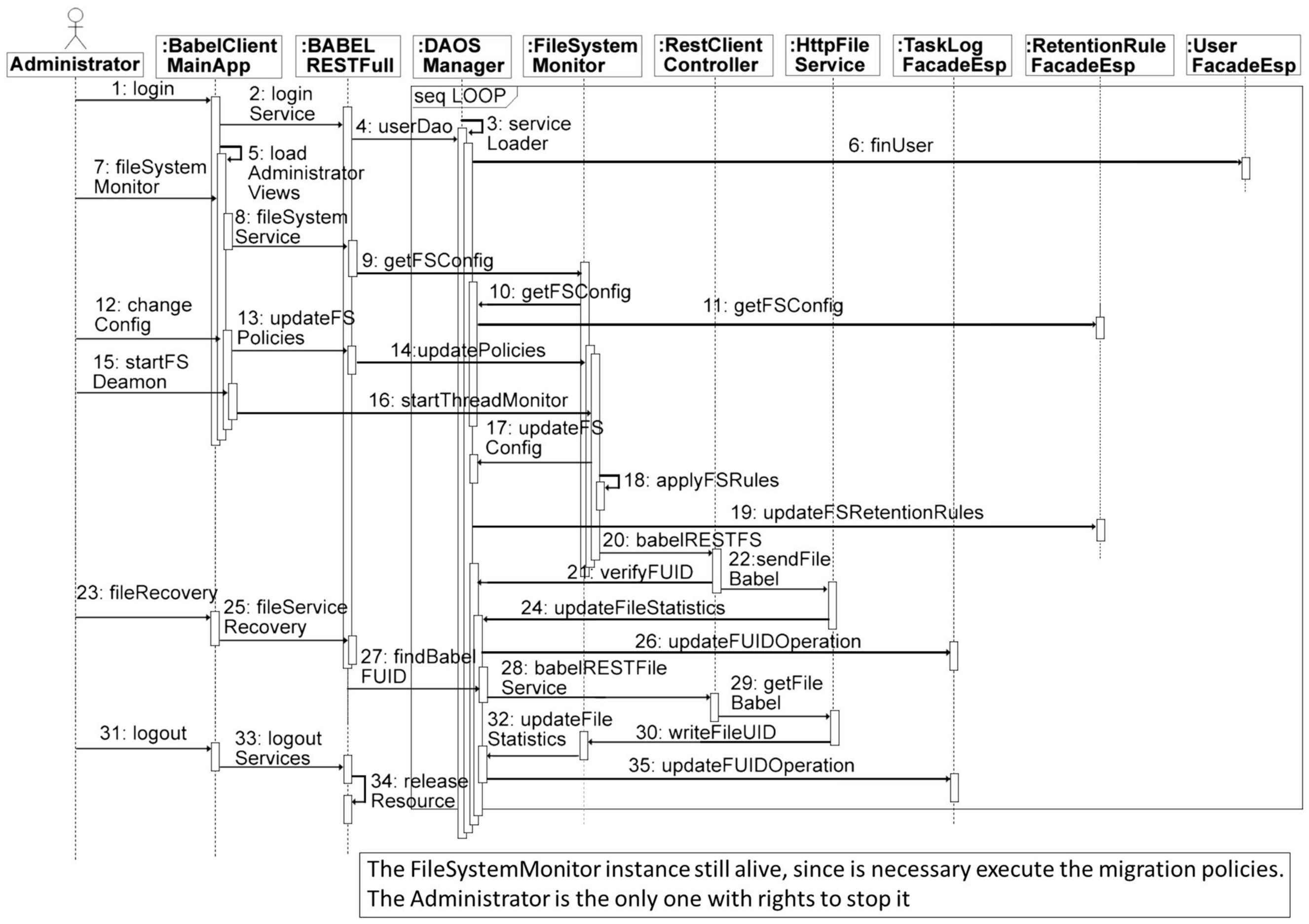
Table 1 Time it takes to store a study for each supported modality at the PACSINR

Study	Modality	Image size (KB)	Number of frames	Study size	Storage time	Average throughput
1	TC	512	3420	1.69 GB	10.13 min	2.84 Mbps
2	RX	22,423	7	153 MB	5.72 s	26.74 Mbps
3	MR	255–512	502	126 MB	2.19 min	0.96 Mbps
4	US	541	9	4.74 MB	4.84 s	0.97 Mbps
5	MN	3393	9	29.8 MB	4.86 s	6.13 Mbps

process evolves through time. Figure 7 introduces a sequence diagram. Initially, the administrator issues a login request to the system. Next, the system validates the request and identifies the user profile. With this information, the client settles the views allowed to the corresponding role, based on a FSM.

The most important operations that the administrator performs are file system configuration and updating, as well as the file exchanging policies that rule the image migration between the PACS server and Babel.

Based on the configured policies, the "fileSystemMonitor" class uses a thread to manage all the operations related to file migration. The coupled systems (i.e., PACS and Babel) exchange files invoking REST services which are implemented in the API "RestClientController." In turn, the "DAOSManager" class instantiates the specialized facade



The FileSystemMonitor instance still alive, since is necessary execute the migration policies. The Administrator is the only one with rights to stop it

Fig. 7 The sequence diagram shows the interaction between objects at runtime. The key elements of the architecture are described in the following way: the administrator issues a login request to the system, which identifies his/her user role and the corresponding finite state machine supporting the allowed navigation rules. When the administrator has been recognized, the system allows him/her to add, delete, or update the policies defining DICOM images exchange between the PACSINR and Babel. All operations are registered in a database for security, traceability, and file recovery

according to the requested service, thus hiding the operations that are performed on the database (preserving loose coupling).

User Interface Management, Client Side

The view repository is hosted on the server side (Fig. 8, deployment). When a user issues a login request, he\she also submits his\her credentials defining a profile and a given role supported by a set of navigation rules. In turn, these rules are described using a FSM. Upon starting, the system loads a default FSM specification, with a reduced set of supported instructions ("Log in" and "Sign Up" states only). Once a user selects a particular login option, the system validates the user role and configures the FSM according to the corresponding profile (root, super user, or AET). This design decision aims to facilitate the implementation of an independent GUI, regardless of platform or framework, since the file views and the FSM itself are specified in an XML format. Accordingly, each state contains a file view, the actions triggered by a user selection, the associated REST service, and the new state. The right side of Fig. 8 includes an excerpt of the partial FSM; it shows a state chart and the reactions upon a user login request.

Discussion

Health providers increasingly rely on medical imaging instruments, such as ultrasound (US), magnetic resonance (MR), computed tomography (CT), positron emission tomography (PET), endoscopy (ES), and computed radiography (CR), to provide quality care to patients. Within this context, a PACS is normally deployed within the hospitals' protected internal network. However, the lack of scalability and high-level disaster recovery provisions involve significant risks and costs for onsite systems. These systems usually generate a huge amount of data, which implies an increasing stress on the hospitals' computing, storage, and network infrastructures. Although many concerns exist about security, privacy, and liability issues involving highly sensitive medical information, external storage may represent a promising approach that can be regarded as the first step towards a private cloud, where all stakeholders may profit from a shared infrastructure.

As online PACS storage server requires more storage space, efficient storage strategies are becoming a key component in healthcare informatics. However, these strategies can be costly to manage and complex to access. The PACSINR-Babel interface (Fig. 6) simplifies management procedures for dependable archiving (redundant, highly available, and secure) which, from the PACS server perspective, perceive Babel as a part of a local file system.

In turn, Babel is a large-scale, highly dependable, software-defined, storage system that can be considered as a LEGO-type family of solutions, this means that it is a hardware-agnostic system that allows IT managers the possibility to settle a trade-off between price and performance, depending on their particular priorities. It has been designed to address three basic non-functional requirements that provide the foundations of a long-lasting system: (1) reliability, (2) scalability, and (3) service times. In addition, it supports a standardized set of communications protocols which guarantees interoperability.

Within this context, this paper shows the coupling between the PACSINR (a system in production) and an instance of Babel, by means of a standard web service. This interface immediately extends the value of an onsite central storage to external locations. The interface caches the active datasets, which means that application entities have immediate access to this information. Additionally, management is based on a web centralized model, which ensures full data consistency and solves data integrity issues on site. In other words, the administrators have the ability to control all file access policies and hierarchies that allow the growth of a secure access model.

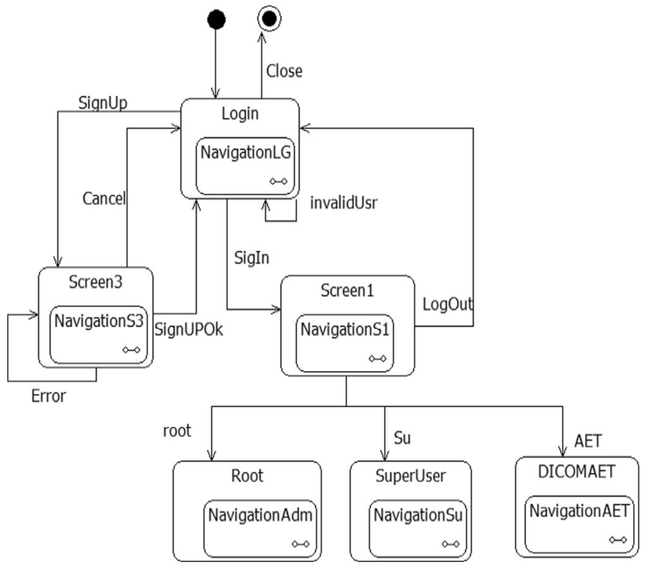


Fig. 8 Navigability rules definition and state chart diagram

Conclusion

In this paper, we described the design of a communication interface between a PACS and a massive storage system. In short, the main contributions of this work are

- The representation of the proposal is done in the UML (Unified Modeling Language), which facilitates its implementation in any programming language.
- The structure of the model (Fig. 6), based on multi-tier layers, supports the development of independent entities

each offering a well-defined service (JPAEntities, DAOInterfaces, DAOImplementation, CRUDRepository, RESTservices, BabelServices, etc.). This design principle, also known as "component loose coupling," prevents the propagation of errors when layers are being updated.

- The system is extensible, offering an interface for building new services to implement new functions (model and views are decoupled).
- Our proposal, on one hand, tries to provide an alternative to PACS storage self-management by the definition of storage automated mechanisms between both systems and, on the other hand, to implement the solution which allows creating an auxiliary system that facilitates the migration processes (PACS updates or disaster recovery) of medical images, regardless of libraries or third-party systems.

From a broad perspective, we have developed a new model that can be employed to manage any type of cataloged collection. This design is mainly focused on high availability and scalability. We have considered that the amount of information to be preserved imposes a structure that divides the storage capabilities in two main categories: primary and secondary servers. We propose the usage of Babel as an alternative to secondary storage. In our model, each document that is received at the primary server is automatically backed up at Babel in a transparent way. Also, the catalog that describes the collection (logs and metadata from the PACS) is regularly backed up, enabling disaster recovery procedures. Should a document that has been eliminated from the primary server is required, it will be automatically recovered from Babel in a transparent way. We call this organization "the closed library model." It is worth pointing that both storage subsystems may evolve independently from each other, without compromising availability, which offers IT managers a long-lasting profit from the involved infrastructure.

The future work is oriented along two main directions: (1) run stress tests to evaluate latency times to retrieve historical studies and compare them to a trading application such as EMC Centera. (2) Link of two health institutes, to share medical images based on Babel as a private storage cloud.

Acknowledgments The authors would like to thank the INR (Instituto Nacional de Rehabilitación, México) and INFOTEC (Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación) for their support and assistance with this project.

All the authors have contributed directly in the writing of this paper.

During the PACSINRBABEL project, all of the authors participated in the technical work in architecture and/or implementation, Marcelín-Jiménez was the project coordinator, while Núñez Gaona was the technical manager for the PACSINRBABEL project.

We wish to thank to the anonymous referees that have helped us to improve the quality of this document, with their valuable comments.

References

1. Yianilos P, Sobti S: The evolving field of distributed storage. IEEE Internet Computing 5(5):35–39, 2001. https://doi.org/10.1109/4236.957893
2. Bindel D, Chen Y, Eaton P, Geels D, Gummadi RP, Rhea S, Kubiatowicz J: Oceanstore: an architecture for global-scale persistent storage. Asplos 35(11):190–201, 2000. https://doi.org/10.1145/356989.357007
3. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., &Zagorodnov, D. The eucalyptus open-source cloud-computing system. In 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009, (2009).(pp. 124–131). doi:https://doi.org/10.1109/CCGRID.2009.93
4. Dean BYJ, Ghemawat S: MapReduce: a flexible data processing tool. Communications of the ACM 53(1):72–77, 2010. https://doi.org/10.1145/1629175.1629198
5. Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Gruber, R. E. Bigtable: a distributed storage system for structured data. 7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6–8, Seattle, WA, USA, 2006,205–218. doi:https://doi.org/10.1145/1365815.1365816
6. Ghemawat, S., Gobioff, H., &Leung, S.-T.:The Google file system. In Proceedings of the nineteenth ACM symposium on Operating systems principles—SOSP '03, 2003p. 29). doi:https://doi.org/10.1145/945445.945450
7. Palankar, M., Iamnitchi, A., Ripeanu, M., &Garfinkel, S.:Amazon S3 for Science Grids: a Viable Solution. In Proc. ACM Int. Workshop on Data-aware Distributed Computing, 2008, (pp. 55–64). doi: https://doi.org/10.1145/1383519.1383526
8. CloudNAS, Available at http://en.wikipedia.org/wiki/Nirvanix, Accessed 2017
9. Skydrive, Available at: https://es.wikipedia.org/wiki/OneDrive Accessed 2017
10. Chen, Y., Sion, R.:To cloud or not to cloud? Musings on costs and viability. In: Proceedings of the Second ACM Symposium on Cloud Computing, SOCC '11, New York, NY, USA, 2011, pp. 29:1–29:7.
11. Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., Molina, J., Controlling data in the cloud: outsourcing computation without outsourcing control. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCSW '09, 2009, pp. 85–90.
12. Ion, I., Sachdeva, N., Kumaraguru, P., Capkun, S.:Home is safer than the cloud! Privacy concerns for consumer cloud storage. In: Proceedings of the Symposium on Usable Privacy and Security (SOUPS 2011), 2009, Pittsburgh, PA, USA.
13. Singh S, Jangwal T: Cost breakdown of public cloud computing and private cloud computing and security issues. International Journal of Computer Science and Information Technology (IJCSIT) 4(April (2)):17–31, 2012
14. Walker E, Brisken W, Romney J: To lease or not to lease from storage clouds. Computer 43(4):44–50, 2010
15. J.LBorges. El jardín de los senderos que se bifurcan; Editorial Sur, 1941.
16. Rabin MO: Efficient dispersal of information for security, load balancing, and fault tolerance. Journal of the ACM (JACM) 36(2):335–348, 1989
17. Weber RO: Information technology—SCSI object-based storage device commands (OSD). Technical Council Proposal Document 10:2003–2031, 2004
18. R. J.Honicky and E.Miller, A fast algorithm for online placement and reorganization of replicated data, Parallel and Distributed

Processing Symposium,2003, pp., 22–26. doi: 10.1109/IPDPS.2003.1213151

19. Medical Center Records (UIC) Retention Schedule, Available at: https://www.uillinois.edu/cio/services/rims/retention_and_disposal/records_retention/medical_center_records_retention_schedule/#imaging
20. Patient information retention and disposal schedule version 4, 2014, Government of Western Australia, Department of Health,, Available at: http://www.health.wa.gov.au/circularsnew/attachments/985.pdf
21. NORMA Oficial Mexicana NOM-024-SSA3-2012, Sistemas de información de registro electrónico para la salud. Intercambio de información en salud.2012, Available at: http://www.dgis.salud.gob.mx/descargas/pdf/NOM-024-SSA3-2012.pdf
22. Gutiérrez-Martínez J, Núñez-Gaona MA, Aguirre-Meneses H, Delgado-Esquerra RE: Software and hardware architecture for a high-availability PACS. Journal of Digital Imaging 25(4):471–479, 2012. https://doi.org/10.1007/s10278-012-9494-2
23. Pianykh, O: Digital imaging and communications in medicine (DICOM) Cap 11. DICOM Media and Security, Springer 2nd Edition, 2012.
24. Shvachko, K., Kuang, H., Radia, S., &Chansler, R. (2010). The hadoop distributed file system. In 2010 I.E. 26th Symposium on Mass Storage Systems and Technologies, MSST2010. doi:https://doi.org/10.1109/MSST.2010.5496972
25. IBM Cloud Object Storage System features and benefits (Cleversafe), Available at: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=TSS03183USEN, Accessed 2017
26. Huang HK: PACS and imaging informatics. In: Basic Principles and Applications, 2nd edition. New Jersey: Wiley Blackwell, 2010
27. United States, Department of Health & Human Services, Summary of the HIPAA privacy rule, 2003, Available at: https://www.hhs.gov/sites/default/files/privacysummary.pdf
28. Josefina Gutiérrez-Martínez, Marco Antonio Núñez-Gaona, Heriberto Aguirre-Meneses, Business Model for the Security of a Large-Scale PACS, Compliance with ISO/27002:2013 Standard, J Digit Imaging. 2015, Aug; 28(4): 481–491. Published online2015 Jan30. doi: https://doi.org/10.1007/s10278-014-9746-4, 28, 491
29. PixelMed publishing, Available at http://www.pixelmed.com/dicomtoolkit.html, Accessed 2017.