Universidade de Aveiro Departamento de
2012 Electrónica, Telecomunicações e Informática,

Eriksson Jorge Melicio Monteiro

CloudMed – Plataforma de Comunicações para Medicina

CloudMed - Open Communications in Medicine

Universidade de Aveiro Departamento de
Electrónica, Telecomunicações e Informática,
2012

Eriksson Jorge Melicio Monteiro

CloudMed – Plataforma de Comunicações para Medicina
CloudMed - Open Communications in Medicine

"It has become appallingly obvious that our technology has exceeded our humanity."

— Albert Einstein

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2012

Eriksson Jorge Melicio Monteiro

CloudMed – Plataforma de Comunicações para Medicina
CloudMed - Open Communications in Medicine

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Carlos Manuel Azevedo Costa, Professor do Departamento Eletrónica e Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**Prof. Dr. Antonio Manuel Melo de Sousa Pereira** Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
vogais / examiners committee	**Prof. Dr. Carlos Manuel Azevedo Costa** Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
	Prof. Dr. Rui Pedro Sanches de Castro Lopes Professor Coordenador do Departamento de Informática e Comunicações do Instituto Politécnico de Bragança

agradecimentos / acknowledgements

Quero agradecer ao meu orientador, Professor Doutor Carlos Manuel Azevedo Costa, pelo interesse, apoio, incentivo e simpatia mostrados ao longo de todo o trabalho. O seu sentido crítico e sugestões/comentários contribuíram muito para a realização deste trabalho.

Quero agradecer ao Luís Bastião, pela disponibilidade, colaboração e apoio que me deu ao longo destes meses de trabalho.

Ao grupo de bioinformática da universidade de Aveiro gradeço por terem proporcionado a possibilidade de fazer parte desse grupo, que sempre me ajudou quando precisei.

A todos os meus colegas e amigos que encontrei ao longo deste percurso. Em especial, ao grupo de estudantes cabo-verdianos que esteve sempre comigo nesta fase da minha vida. Muito obrigado pela disponibilidade, apoio, incentivo e... amizade!!!

Por último, o agradecimento mais importante... a toda a minha família, que mesmo à distância, esteve sempre presente e muito apoiou para que eu pudesse superar esse percurso com sucesso!!! Especialmente, agradeço aos meus pais e irmãos por todo o carinho, amor, compreensão, apoio e incentivo!!!

Resumo

Os recentes avanços das tecnologias de informação e comunicação têm criado novos cenários aplicacionais na área da telemedicina, nomeadamente na forma como integramos diferentes fontes de dados, como acedemos e partilhamos estes recursos em ambientes móveis e como integramos ferramentas cooperativas inspiradas no paradigma das redes sociais.

Temos verificado nos últimos anos a terciarização de recursos computacionais, processo conhecido como Cloud Computing. Esta realidade cria novas oportunidades de exploração destes recursos para facilitar o acesso, partilha e integração de informação médica, em qualquer local e a qualquer hora. Mais ainda, a escalabilidade e fiabilidade oferecida por estas plataformas satisfazem os requisitos de serviço impostos a soluções telemáticas na área da saúde.

Esta dissertação teve como objetivo estudar o paradigma de software como serviço, suportado por uma estrutura em Cloud, tendo em mente a sua utilização em cenários de telemedicina e tele-trabalho. Muito concretamente, desenvolveu-se uma plataforma Web de serviços orientada às redes de imagem médica. Esta solução disponibiliza um ambiente cooperativo inovador onde os clínicos podem recolher dados, partilhar informação e aceder remotamente a recursos imagiológicos. Aspetos de segurança e interoperabilidade com os atuais sistemas e normas foram alvo de particular atenção.

Abstract

The recent technological developments in information and communications technologies are promoting new studies and research in telemedicine area, revolutionizing the access, integration and sharing of medical information. For instance, many systems have been focusing on ubiquity through the use of mobile computing and on enhance users cooperation through usage of social networking paradigms.

In this regard, the rise of new model of outsourcing computing resources, which is known as Cloud computing, creates new possibilities to explore their benefits to facilitate the sharing and remote access to medical information, anywhere and anytime. Moreover, the scalability and reliability offered by Cloud platforms fit well to the medical area requirements.

This dissertation aimed to analyze the current state of the art of Cloud Computing, namely studying their viability to support telemedicine and teleworking scenarios. The proposal was focused in the medical imaging field. The work resulted in a Cloud computing solution, following the software as a service model, to support cooperative tele-imagiology networks. It is a solution that allows users to setup collaborative environments in the field of imagiology, targeting the acceleration and improvement of decision-making processes. The proposal contemplates also other important issues like, for instance, security and interoperability with actual medical imaging systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The progress in medicine gained an important impulse with the evolution of informatics and telecommunications. As result of the adoption of the technology in the health care institution, emerged the telemedicine that is a way to provide health care services at a distance, which more has been leveraged by the evolution of informatics and telecommunications. Due to the fast evolution of informatics and telecommunications, and reduction of its price, the telemedicine scenario is being widely explored [1–3] by many investigators and health care service providers, with the aim to take the best advantage from the contemporary technology for improve services provided by health care institutions. This tremendous evolution of technology creates opportunities to improve and even create new telemedicine services. Moreover, the distinction of telemedicine will disappear by virtue of its pervasiveness. Technology applications will continue to evolve rapidly and will become more integrated and ubiquitous. This progress will be strongly impacted by wireless and broadband innovations. These changes promote portability and the seamless integration of technology into daily life. So, in time, telemedicine applications will even evolve beyond Internet-based opportunities. They will become wireless, portable and eventually, wearable.

The improving of telemedicine services' comprisement, using electronic information and communications technologies to leverage and enrich its quality of services when the distance barrier separating health care participants is a considered factor, has been successfully acquired with the embrace of the new technologies in this purpose. Thus, probably, these evolutions had and will continue to have a vast impact in the services provided by health care institutions. With the introduction of technologies at service of medicine became easier and efficient the management of all the data that is generated by health care institutions, taking advantage of specialized systems to manage these data and also it become easier and faster the way that the information is exchanged and transmitted. Focusing in the medical imaging area, it is possible to identify some informatic systems that were developed to increase the efficiency of how the medical images data are managed and transmitted inside a health care institution. To deal with the huge amount of data, which can reach Terabytes of information generated by health care institutions, was developed the PACS system. PACS systems are widely adopted by health care institutions to implement an informatic system that can be efficiently managed, allowing the standardization of the data acquisition, storage and transmission. Nevertheless, the investigation in medical informatics systems is active and investigators are focused in explore the technologies evolutions in health care area, thus it is common to appear new systems (e.g. RACS, cloud-based Virtual Radiologic-vRad etc) that are proposed to be applied in the broad area that is medicine.

The developments already done in the medical informatics' area show that the application of informatic technologies in the medical area is an added value, so it is important to continue to investigate the application of the new emergent technologies in this area. Moreover, evolution of the technology is being fast, thus new concepts are emerging and its application in medicine can be very important to improve actual services.

This thesis lays on the new technologies to create a new service based on Cloud computing to leverage telemedicine. Furthermore, this work will explore how the utilization of Cloud computing technologies can be efficient and an asset to telemedicine services, by taking advantage of its features like, for instance, outsourcing of computation and storage, information's access, availability, scalability, etc. It will be proposed a Cloud-based system to provide a service that can be used by radiologists to do telework through a cooperative environment. The service will allow radiologists to share medical images with colleagues and to interact with each other in examination of a given study, thus creating a channel which may be used to get a second opinion in an examination or to interact with an expert in a given area. Furthermore, the resulting service might be used to provide remote support to health care centres that does not have specialized radiologists, to integrate distributed medical information sources or in educational purpose.

1.2 Goals

To implement Cloud based software as a service is necessary to analyse the Cloud infrastructures and how they are deployed. Thus, just deploy an infrastructure in the Cloud does not mean that this infrastructure inherits Cloud characteristics like, for instance, scalability. Obviously, to implement a medical informatics system is necessary to take care of some issues related with the data security, because these kinds of systems deal with sensitive data, and with the availability of data any time a physician tries to access it. Also, to access data from inside a health care institution may bring some issues to solve related with firewalls. In general, the goal of this thesis is to create a fully integrated distributed system based on Cloud that tackles the presented problems. Moreover, it must be Web 2.0 compliant.

Accordingly, this work's main goals are the following:

- The study of Cloud computing fundamentals
- The study of teleimagiology services.
- The study and development of a Cloud based system for teleimagiology.
- The integration of Cloud based infrastructure with actual medical imaging data sources.

All the broad environment of new opportunities, which is being provided by new technologies, are ready to be explored, thus this is the challenge that this work aims to fulfil in order to provide better services, any-time and anywhere.

As result, it is expected to achieve a system well supported by technologies, with a lifecycle well defined, that should be provided as a service in the Cloud to serve radiologists, allowing the improvement of the workflow and dataflow in telework.

1.3 Thesis outline

This dissertation is organized according to the following structure:

Chapter 2 presents the medical scenario where this these is inserted. It is made a brief introduction to telemedicine, focusing on teleimagiology services. There is an overview of actual applications of teleimagiology services, the way new technologies are improving those services and empowering the appearance of better systems. Furthermore there is the presentation of Picture Archiving and Communication System (PACS), where is showed its importance to the revolution of imagiology. In addition, this chapter talks also about Cloud computing paradigm, depicting some deploying models and services available in the Cloud. Moreover, there is an explanation of how Extensible Messaging and Presence Protocol (XMPP) can be useful as message-oriented middleware for creation of collaborative systems.

Chapter 3 shows the system requirements and how it was modelled. First there is an identification of all functional and non-functional requirements defined for the system to be developed and, from those requirements, is made a proposal for implementation. There is an explanation of how this proposal designs the system's workflows and, also is presented a data model and architecture to support it.

Chapter 4 explains the implementation decisions. Meanwhile, it is presented a plug-in based system which empowers CloudMed Software-as-a-Service implementation. Furthermore, there is the presentation of the messages-oriented middleware that supports the system and, how was implemented CloudMed PACS gateway to enable the creation of an interface between the Cloud XMPP network and the PACS network. After, is explained the implementation process of CloudMed Web2.0 application. And finally, it is presented the results obtained with the implementation of CloudMed system.

Chapter 5 presents the conclusion of this thesis and, there is also an analyse of the system's strength, weakness and future works.

Chapter 2

State of the Art

In this chapter, will be presented the technologies that currently are supporting solutions in telemedicine, which enable the storage and access to medical images. Besides the analysis of contemporaneous telemedicine, it will be contemplated some Cloud service-oriented architecture, which will be a mainstay for the creation of the solution proposed in this work and it will be exposed a protocol that can be an asset to create collaborative systems.

2.1 Medical Environment

2.1.1 Telemedicine

Telemedicine [4] is based on the use of informatics and communications technologies to allow doctors to examine, monitor, investigate and treat patients who are physically apart, or do not have the availability to travel to the health center. This is an area of medicine that has captivated the interest of several researchers in recent years. The concept of telemedicine has become very important to medical science. In many countries where the population is highly dispersed, with a low population density in certain regions, telemedicine emerges as a vital medium that can save lives, because it is through those services that can get to anywhere the qualified health services. Also in countries where the demands on hospitals are increasing, telemedicine can be a solution that involves the use of current information technologies to create solutions that make it easier for hospitals to make available therapies and tele-consultations for patients, thus decreased the percentage for use of the physical space available in hospitals. Such means can also be used to assist doctors in diagnosis of medical images by remote diagnostics and / or teleconsulting.

With the advancement of information technology and telecommunications, telemedicine is no longer a forthcoming idea and the concept is becoming more mature and very important for institutions that provide health care. Telemedicine can be seen as a leverage of the quality and availability of services provided by health institutions and, it will be fundamental in the future to enable hospitals provide medical services to patients overcoming physical barriers, eliminating the need for patient travel, enabling hospitals to make available high quality services at distance.

Teleimagiology

Teleimagiology is a branch of telemedicine which has a wide use in medicine. It is based on medical images in digital format, such as CT (computed tomography), US (ultrasounds), MR (magnetic resonance), etc, which are sent from one location to another in order to be analysed and / or consulted by specialists. The traditional teleimagiology consists of three basic elements: the station that sends medical images, a transmission channel and receiving station.

This medical service is supported by currently available technologies and telecommunications, where images are obtained from digital medical equipment with capacity to produce images in digital format. They can be transmitted to any location via a network connection, like for instance, telephone lines POTS (Plain old telephone service), LAN (Local Area Network) or WAN (Wide Area Networks).

The first steps of teleimagiology dates back to 1929 when the first medical image was transmitted; it was transmitted a dental x-rays through a telegram to a distant location [5].

Currently teleimagiology is widely used for training new radiologists, for teleconsulting in cases where it is needed a second opinion in medical image analysis, for provide medical assistance to regions of difficult access, for assist and train radiologists in developing countries, etc.

The integration of radiological equipment from diverse manufacturers in the same teleimagiology network, is a task that is backwards some problems and challenges. The need for interoperability among all radiological equipment is very important in the hospital network, so was developed DICOM standard. This standard defines how medical images are acquired, transferred and stored. The format defined by the DICOM standard will be analysed in more detail in section 2.1.3.

Thanks to the DICOM is empowered the interoperability between equipment from different manufacturers, which facilitates the transfer of medical images from one place to another. The transmitter station, which is part of teleimagiology systems, is responsible for the conversion of medical images, for example, x-ray, in digital format and the compression of the images taking into account the desired resolution and bit rate available. The data compression is possible using the characteristics of the DICOM coding format, allowing lossy and lossless compression, where the degree of loss of information is variable which depends on the modality used. This compression reduces the density or number of bits per pixel and, consequently, there is degradation in image resolution. Ideally, in teleimagiology would not be necessary compression of images, if they are obtained by high-resolution equipment and high-speed transmission; however this is not possible due to technological limitations. In this sense the optimization of a parameter implies the degradation of other (e.g. increasing the speed of transmission often involves an increase in the level of compression and the reduction of the images' resolution).

As regards the receiving stations, they should also be connected to a high-speed network point. Besides, the quality of the monitors for display of medical images is also an important feature to facilitate diagnosis. The receiving stations are usually equipped with large monitors with good pixel resolution [6]. Another important requirement is the ability to support the monitor split screen so as to enable the radiologist to visualize more than one picture at a time in order to compare different images. The quality of the monitor's brightness is also very important for the radiologist because the monitors with greater vividness allow the radiologist to better identify regions of interest in the image that is analysed.

For the analysis of medical images is used specialized software with features that are an asset to the evaluation and decision making by the physician. Most of the software provide simple feature which let physicians to manipulate the gray level of the image, zoom, etc. But, there are expertise software with specialized functions available, such as allowing the physician to enhance edges, display histogram equalization, add notes to the relevant parts of the image, map a grayscale image, apply filters, etc. The value of the functionalities provided by software, which allow physicians to analyse medical images, in teleimagiology depends on the users' need and the type of images analysed.

The use of teleimagiology for enhance the quality of services provided by hospitals is an increasingly realistic scenario. Currently, there are several applications of teleimagiology, among which we can list the following:

- **Radiologist on call:**
  This is an application of teleradiology that allows a doctor to get at his residence through a portable device, patient's medical images that are transmitted from the hospital / clinic to be analysed remotely. This service allows the radiologist instant consultation with the attending physician, thus leading to the improvement of services provided by health centres.

- **Primary care physicians in rural areas:**
  The technologies allow doctors from remote clinics that are primary care provider in rural areas, to easily and quickly send medical images of patients, to be consulted and evaluated by experts who are in remote health centres.

- **The medical consultants requiring remote subspecialty of radiology:**
  Often it is necessary the evaluation of medical imaging by sub-specialists in certain areas of radiology. Through teleradiology this process is speed up because the doctor in a health center can easily send a set of a medical imaging study to be evaluated by a radiologist who is sub-specialist in a particular area (e.g., Paediatric Radiology).

The data's flow and its availability are very important in telemedicine to raise a solid and useful system [7]. In [8], a system called MIFAS (Medical Image File Accessing System) was created focusing their objectives to solve the problem of medical information exchange, store and share between different hospitals. It was used a Cloud based architecture, where they use Apache Hadoop and co-allocation mechanism to implement a distributed file system.

Also, in [9] authors analysed the advantage of using the new Cloud computing paradigm to improve health care systems. They implemented a PACS on Cloud where was promoted the Cloud's elasticity and scalability, providing universal access to the information and increase the data availability anywhere, any-time. The system uses the concept of "PACS-as-a-service" and the authors say that is possible to reach interoperability with DICOM devices through a PACS Cloud Gateway. Besides problems related with information access, availability and interoperability, the proposed architecture has also some concerns with security issues. All data stored on the Cloud are ciphered; furthermore the keys that are used to cipher the data are stored on a trustable provider or in-house, so the cloud providers are not able to decrypt the files and access the clear content.

There are some works that outlines others features like, for instance, user interactions, and what they can do with data available on Cloud. Many systems have been focusing on ubiquity through the use of mobile computing. For instance, there are systems that can be used by physicians to manage patients' health records and medical images using the emerging mobile computing [10, 11].

The authors on [11] were concerned with actual mobiles phones and tablets limitation like, for instance, the reduced computational power, limited storage and memory to implement a three-tier architecture. They have been working around these problems, aiming to enable users mobile access to DICOM medical images. Problems related with network management policies at healthcare network, that usually only allow protocols like HTTP or IMAP also influence telemedicine systems architecture. However, relays hosted in a Cloud service can be a work around to enable system connectivity [11], due to the fact that the relay is the responsible for the communications between Internet connected client device and a corporate PACS.

In the past, a system [12] created synchronous collaboration mechanisms among the medical staff, improving telemedicine services and real time collaboration. The developed system highlighted the importance of CSCW (Computer-Supported Cooperative Work) [13] on E-Health [4, 14] platforms. They provided synchronous and asynchronous collaborative capabilities through components like multimedia messaging, forum, video-conferencing tool, virtual shared space, on-line focus group, and so forth. Nevertheless, that system works only for a single institution.

The proposed on this dissertation work aims to analyse and find some path to improve the communication between multiple institutions and telework, providing some useful features like messaging and presence, collaborative and cooperative work in a Cloud-based platform.

2.1.2 PACS

Nowadays, the hospitals have recognized the benefits of computing and technology for the management of medical information. Three decades ago came the first PACS (picture archiving and communication system) [6], that have revolutionized radiology and somehow all of medicine. Due to the large amount of data generated in the imaging area of radiology [6, 15] by some medical exams as cardiovascular ultrasound (US), angiography (XA), digital mammography, computed tomography (CT), positron emission tomography (PET), etc, there was the need to create an infrastructure that provides a manner that clearly defines how it is stored and accessed medical images obtained from the hospital network [6]. The use of PACS in a hospital network facilitates and speeds up the way medical data is transmitted and accessed. PACS systems use the standard format defined by DICOM standard [6, 16] to store and transmit medical images in order to support interoperability between equipment from different manufacturers.

As an overview, it can be said that PACS encompasses technologies used for the acquisition, archive, distribution and visualization of a set of digital images using a computer network for diagnosis and revision in dedicated stations (Figure 2.1).

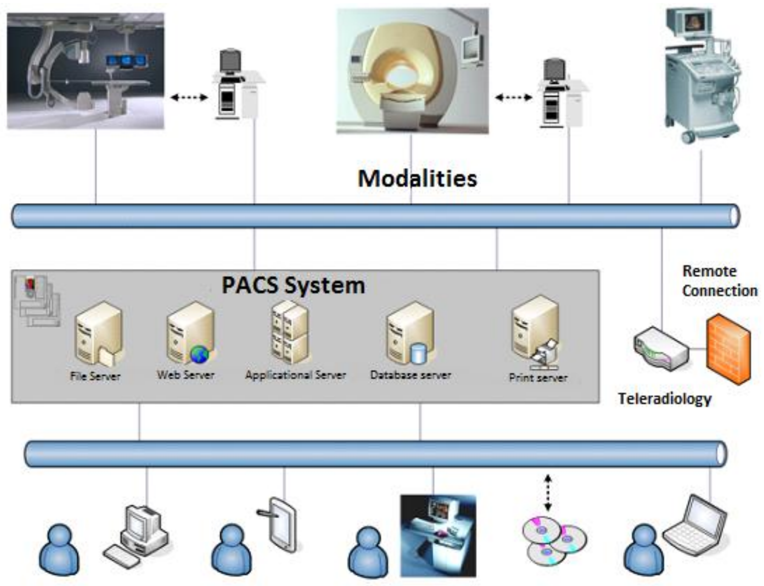


Figure 2.1: PACS system architecture

Components

Typically, a PACS is composed by four components [6]: gateway for medical imaging acquisition, PACS archive server, workstations for data visualization, application servers.

Beyond that components can be found some PACS application servers and access gateways which are used to connect to the information systems from other institutions that provide health care.

Gateway for data acquisition on medical imaging

The acquisition of medical imaging is one of the main tasks of a PACS system. Due to some problems related with equipment from different vendors is required the existence of gateways for data acquisition on medical imaging system. The existence of the gateway is a need because each vendor implements its own protocol in accordance with the statement made by the DICOM standard, and also because there is a need for communication between different PACS systems. Gateways consist of an important point of the system where is made the formatting of medical images from given radiological equipment to a standard format used to communicate in the PACS system, which is compliant with the format defined by the DICOM standard, and then these images are forwarded by the gateway to the PACS archive server or a workstation for viewing.

The gateways are placed between the units responsible for obtaining medical images of each modality and the rest of the PACS system. They interact directly with the image acquisition equipment that is connected to the medical imaging modalities.

PACS archive server

The PACS archive server is the engine of any PACS system. This server consists of high-performance computers. It is composed by a database where information about patients are indexed, from the HIS and RIS, and a file system where files are stored in the short / long term or permanently.

In a PACS, the PACS server is responsible for receiving the images of medical examinations and updating the management system database. In addition to the function of obtaining medical, the server can include others functions like, for instance, compression of images, check the integrity of files, extracting information that describe the examinations received through the header of DICOM images, communication interface with applicational PACS servers, etc.

Workstations

The workstations include communication with the PACS system network, a monitor for visualization of images and image processing software. Some workstations may also be equipped with a local database or not, depending on the needs. The workstations that have a local database are usually equipped with many features for medical image processing and need only to communicate with the PACS server sporadically as they can keep information in the local database. On the other hand, the workstations are not equipped with a local database need only some basic processing functions and are constantly being aided by the PACS server.

Radiologists use the workstations to make medical diagnoses of radiological images. They have available a set of features that can be used to interact with the PACS archive server or even other PACS components. These features include the possibility of select the images he wants access through DICOM query / retrieve, measurement tools that can aid him in image interpretation and diagnosis, the accumulation of all relevant information and images of a particular examination from a given patient, etc.

Application Servers

The application servers are connected to the PACS server and, for instance, they can be used to filter the data retrieved from PACS server, with the intention of satisfying a particular purpose, or to make any desired data processing. In the PACS, it is possible to find various types of application servers. For example, it may have application servers whose purpose is to create a web service for viewing medical images, creating an electronic patient record (EPR) based on images, create an application server for educational purposes, etc.

2.1.3 DICOM

DICOM [16] is an international standard that defines data formats, storage organization and communication protocols of digital medical imaging, which was approved by ACR (American College of Radiology) and NEMA (National Eletrical Manufactures Association) in October 1993. This standard has emerged as a result of the appearance of many equipments with capacity to acquire, transfer and store data imaging, and the consequent need to standardize the communication processes of these devices on the network.

The ACR and NEMA have developed a set of standards, recommendations and guidelines that allow the communication of information between different digital imaging equipments, which is the basis of the strong development and expansion of PACS.

The Digital Imaging and Communications in Medicine (DICOM) [6, 16] emerged in the early 90's, and was released in 1993 a new version consisting of 13 parts. But in its present DICOM version 3.0 [16] is divided into 18 parts that include 160 supplements on specific aspects of one or several parts of the standard. Thus the DICOM has become the international standard that defines formats and communication protocols of medical digital images. It defines the semantics of the commands and associated data so that different devices can interact. The DICOM standardizes the entire set of methods of storage and transmission of digital medical images. Thus, it enables the communication of medical information between digital equipment (from several different manufacturers), such as imaging modalities, workstations, printers, servers, etc.

DICOM Information Model

DICOM Information Model is the model used by DICOM standard to represent real world information like patients, studies, medical devices, and so on. All real-world data are represented in DICOM as objects with respective properties or attributes. DICOM objects and attributes are standardized according to DICOM Information Object Definitions (IODs). IODs are collections of attributes, used to describe each particular data object. For example a patient IOD can be described by name, medical record number (ID), sex, age, and so on.

DICOM operates guided by a service-rendering model where DICOM applications provide services to each other. Moreover, there is an association between particular service types with the data (IODs) that they process. DICOM calls these associations Service-Object Pairs (SOPs), and groups them into SOP Classes (Figure 2.2).

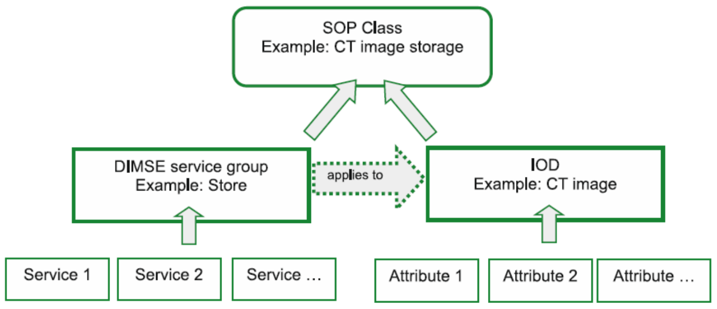


Figure 2.2: SOP Class structure: DIMSE services applied to IOD instances [16].

All DICOM commands and most DICOM data attributes are always bonded with the four-level information model represented by Patient-Study-Series-Image hierarchy (Figure 2.3), wherein:

1. One patient may have multiple studies.
2. Each study may include one or more image series.
3. Each series has one or more images.

DICOM hierarchy reflects what happens in the real world. For example, a patient goes to a hospital and can make several studies (for example, MR, CT, and ultrasound exams) and these studies may have multiple image series (coronal, axial, with or without contrast, with varying imaging protocols, and so on). And each series will have one or more images associated with it.

To identify patient, study, series, and images, the DICOM hierarchy defines UID (Unique Identifier) to each hierarchy level. The patient level is identified by “Patient ID” (all patients should have IDs that identify them uniquely). All DICOM element is identified by a tag. The definition of DICOM tag will be presented **DICOM Data Format** 2.1.3 subsection. The “Patient ID” element is stored with the (010,0020) tag, making this tag required for any type of imaging. The same principle applies to the other three levels of the Patient-Study-Series-Image hierarchy: at the Study level, each study has its unique “Study Instance UID”, (0020,000D); at the Series level, each Series has its unique “Series Instance UID” (0020,000E); and at the Image level, each Image has its own “SOP Instance UID” (0008,0018).

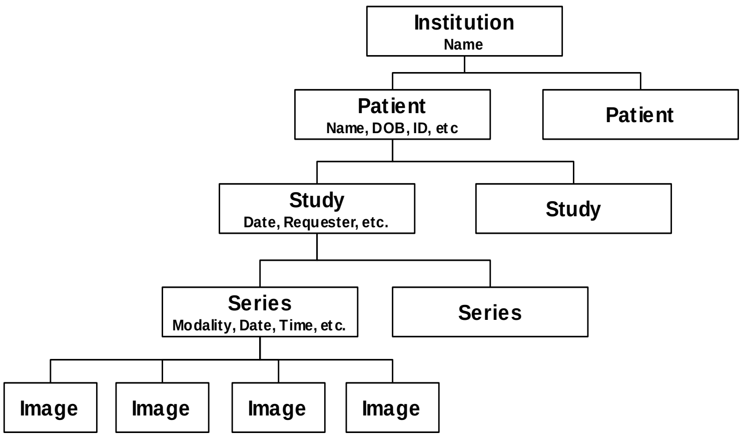


Figure 2.3: DICOM information hierarchy.

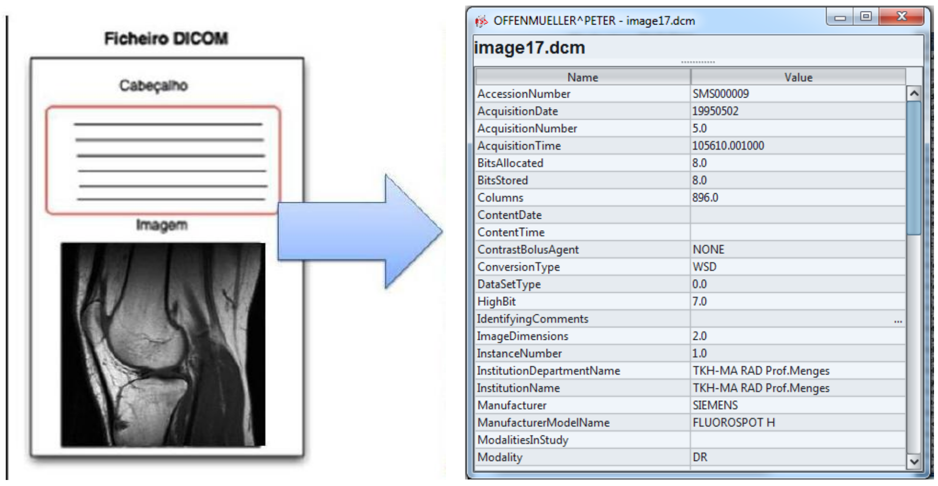
DICOM Data Format

DICOM files support multiple types of elements of medical information, such as several medical imaging modalities, waveforms, clinical structured reports, etc. Every DICOM file has a header that contains metadata, used to represent the DICOM Information Model [6], including information related with patients, clinic staff, institution, equipment and conditions of the examination, etc. The DICOM Information Model [6] contains the data that make correspondence with the real world information (Figure 2.4).

DICOM groups information into data sets. The each data set of a DICOM file contain some specific information, for example, within a DICOM file can be found the patient ID, institution name, modality type, digital medical image bytes. Moreover these data set can never be separated from by mistake.

A DICOM data object is formed by many of attributes, and one special attribute that contains the image pixel data. A DICOM object contains only one pixel data attribute. That can correspond to a single image. But in some modalities, this attribute may contain multiple frames, in order to storage cine loops or other multi-frame data. Moreover, DICOM object’s pixel data can be compressed using a variety of standards, including JPEG, JPEG Lossless, JPEG 2000, MPEG3, MPEG4, etc.

A DICOM object Data Element is structured as follows: GROUP (2 bytes) ELEMENT (2 bytes) VR (2 bytes) LengthInByte (2 bytes) Data (variable length). it can be encoded using different Data Element encoding schemes. With Explicit value representation (VR) type of Data Elements, where VR defines the element's value representation (that can be DA for date, TM or time, PN for patient name, etc). Also, DICOM objects can have Implicit Data Elements wherein VR is not present [17].



Name	Value
AccessionNumber	SMS000009
AcquisitionDate	19950502
AcquisitionNumber	5.0
AcquisitionTime	105610.001000
BitsAllocated	8.0
BitsStored	8.0
Columns	896.0
ContentDate	
ContentTime	
ContrastBolusAgent	NONE
ConversionType	WSD
DataSetType	0.0
HighBit	7.0
IdentifyingComments	...
ImageDimensions	2.0
InstanceNumber	1.0
InstitutionDepartmentName	TKH-MA RAD Prof.Menges
InstitutionName	TKH-MA RAD Prof.Menges
Manufacturer	SIEMENS
ManufacturerModelName	FLUOROSPOT H
ModalitiesInStudy	
Modality	DR

Figure 2.4: DICOM Object structure

DICOM data packing follows the TLV format (Tag, Length, Value) as shown in Figure 2.5. The tag has 4 bytes, where the first 2 bytes are for the group and the last 2 byte for the element, thus representing the "(group, element)" tag used to identify a given Data Element. For example, Patient Name tag has the group value 0x0010 and element value 0x0010, thus it is represented as (0010,0010) tag.

As expressed, the Value Representation (VR) field is used to define the encoded type for a given element and it depends on the element tag. For instance, VR fields can have the given value DA for Date, TM for time, OB for Object Binary, etc. Some elements can have implicit VR because it is possible, according with tag value, consult a DICOM Dictionary that defines what type an element stands for.

The Length field is used to define the length in bytes of element's value field. This value depends on the tag value. At last, the element has the value field, which contains the data that is stored in the element (e.g. image pixel data, patient birthday, patient name, patient ID, etc).

As can be verified in Figure 2.5, DICOM files are formed by a sequence of Data Element or TLV sequence. Each TLV represents an attribute. Also it is possible to create private tags and extend DICOM protocol capabilities to represent extra real world information.

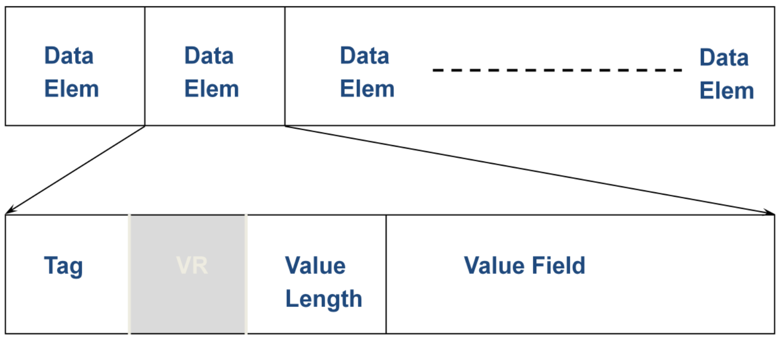


Figure 2.5: DICOM Object Data Element

DICOM Communications

DICOM provides of many different services, most of which involve transmission of data over a network. The protocol uses TCP/IP to establish a reliable connection between endpoints. As others protocols like, for instance, SMTP, HTTP, FTP, etc, DICOM only adds its own networking language (at application layer) over TPC/IP. This language consists of high-level services, DICOM Message Service Elements (DIMSE). DICOM communication follows the client/server model. In DICOM network, it can be found the Service Class Provider (SCP) and Service Client User (SCU). SCP or SCU is used to denominate a kind of device, according to its role in the network (Figure 2.7). An entity can play SCP or SCU application roles to communicate with each other (Figure 2.6). For instance, a modality or workstation

DIMSE
C-Store Request / Response
C-Find Request / Response
C-Get Request / Response
C-Move Request / Response
C-Cancel Request / Response
C-Echo Request / Response
N-Event Report Request / Response
N-Get Request / Response
N-Set Request / Response
N-Action Request / Response
N-Create Request / Response
N-Delete Request / Response

Table 2.1: Create DIMSE services associated with composite SOP classes (DIMSE-C) and services associated with normalized SOP Classes (DIMSE-N).

that produces or consumes images, has to interact with PACS Archive. In this case the modality or the workstation is SCU and the PACS server belongs to SCP.

In DICOM network, each device has an Application Entity Title (AETitle) that identifies it. AETitle is used to address entities into a DICOM network. To access to any DICOM service or to communicate with any DICOM device first must be created a DICOM association, in order to create a channel for information exchange. In the establishment of a DICOM association, there is negotiation of several parameters, such as, encoding (ex. little-endian, big-endian), image compression formats, what kind of information will be transferred, and the duration of the association. After the negotiation, the service commands are executed between SCU and SCP to perform the service goal.

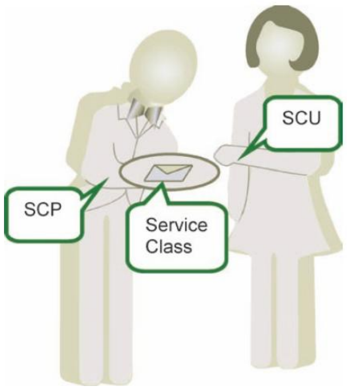


Figure 2.6: DICOM SCU-SCP model [16].

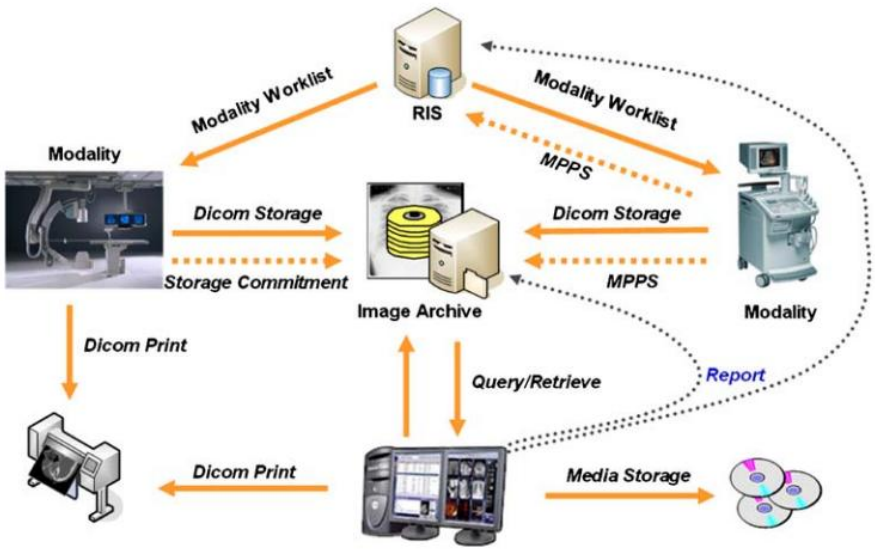


Figure 2.7: DICOM services and network entities.

Store

The DICOM Store service is used to send images or other persistent objects (structured reports, encapsulated PDFs, etc.) to a PACS archive or workstation. This service uses C-Store (Storage SOP) command to move DICOM images between the entities, over a DICOM network (Figure 2.8).

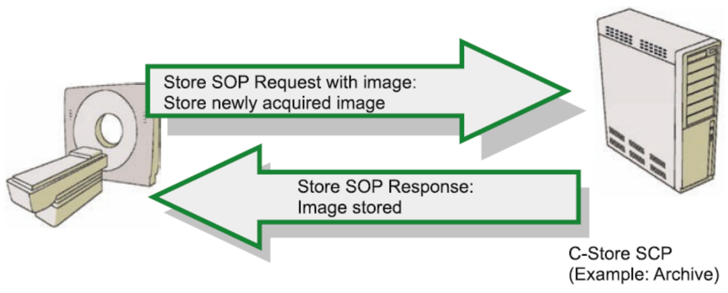


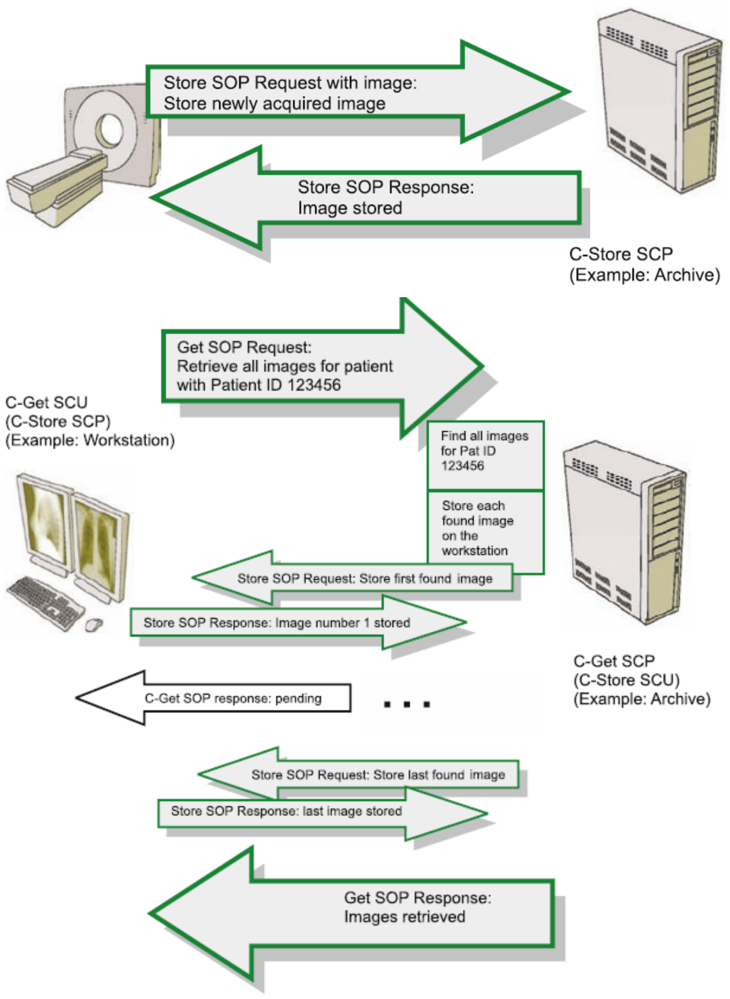
Figure 2.8: DICOM C-Store [16].

Storage Commitment

The DICOM storage commitment service is used to confirm that an image has been permanently stored by a device (either on redundant disks or on backup media, e.g. burnt to a CD). SCU uses the confirmation from the SCP to make sure that it is safe to delete the images locally.

Query/Retrieve

This enables a workstation to find lists of images or other such objects and then retrieve them from a PACS archive. Basic DICOM image retrieval is executed using C-Get SOP. C-Get wraps C-Find and C-Store into a single service class where C-Find is used to query the required images, followed by a C-Store to retrieve these images. When a C-Get is sent to the SCP, the SCP first uses the search parameters to find the images then invokes C-Store to return them to the SCU (Figure 2.9).



Store SOP Request with image:
Store newly acquired image
Store SOP Response:
Image stored
C-Store SCP
(Example: Archive)
Get SOP Request:
Retrieve all images for patient
with Patient ID 123456
C-Get SCU
(C-Store SCP)
(Example: Workstation)
Find all images for Pat ID 123456
Store each found image on the workstation
Store SOP Request: Store first found image
Store SOP Response: Image number 1 stored
C-Get SOP response: pending
...
C-Get SCP
(C-Store SCU)
(Example: Archive)
Store SOP Request: Store last found image
Store SOP Response: last image stored
Get SOP Response:
Images retrieved

Figure 2.9: Wrapping the C-Store service in the C-Get service [16].

Therefore, it is possible to use more advanced retrieval service that enables to move images to third parties (Figure 2.10). The C-Get is used to return images to the same SCU that makes the request, nevertheless with C-Move it is possible to send the images to any entity. Thus, C-Move needs to know where to return the images. Indeed, this question is never raised in C-Get; the C-Get SCP always returns the images to the image-requesting entity (C-Get SCU).

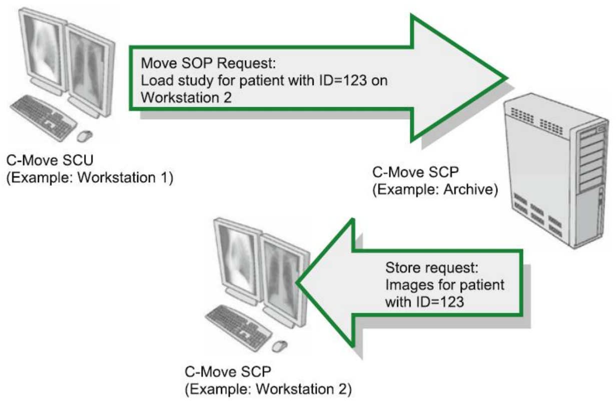


Figure 2.10: C-Move with three entities: Workstation 1 instructs the Archive to send an image to Workstation 2. The Archive sends images to Workstation 2 with C-Store suboperations [16].

Modality Worklist

This enables a piece of imaging equipment (a modality) to obtain details of patients and scheduled examinations electronically, avoiding the need to type such information multiple times (and the human mistakes caused by retyping) (Figure 2.11).

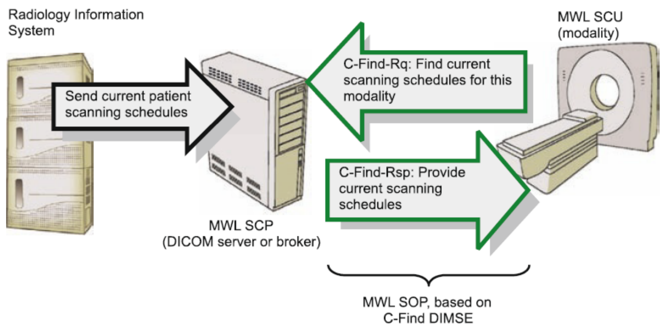


Figure 2.11: DICOM MWL example: populating imaging modalities with basic patient data [16].

Modality Performed Procedure Step

It is complementary service to Modality Worklist, this enables the modality to send a report about a performed examination including data about the images acquired, beginning time, end time, and duration of a study, dose delivered, etc.

It helps give the imagiology department a more precise handle on resource (acquisition station) use. Also known as MPPS, this service allows a modality to better coordinate with image storage servers by giving the server a list of objects to send before or while actually sending such objects.

Printing

The DICOM Printing service is used to send images to a DICOM Printer, normally to print an x-ray film. There is a standard calibration (defined in DICOM Part 14 [18]) to ensure consistency between various display devices, including hard copy printout.

2.2 Cloud Computing

2.2.1 Overview

Cloud computing is a concept that has spread around the world, but it is not a novel concept. In fact, historically it is not clear when this computer model has emerged. In the 60's, John McCarthy, an American computer scientist, foretold that "computation may someday be organized as a public utility" and that would happen to the computer the same as that happened with electricity. Instead people have generators in their homes, they would pay for the amount of electricity used. This is the concept that is currently used in Cloud computing, where people pay-as-they-go.

2.2.2 What is Cloud Computing?

Cloud computing is a computer model for provisioning of services that allow flexible use of virtual servers, massive scalability, and management services. In addition, the Cloud computing services are the result of a new concept that consists on delivery of computing as a service rather than a product [19]. To describe Cloud computing in terms of their characteristics and how it works along with other information technologies, can be said that it is a computer model that basis its functionalities over virtualized systems, were an aggregation of distributed resources is available and shared by virtual servers, enabling a massively scalability and the creation of a dynamic infrastructure [20].

Cloud Computing features

There are many characteristics that drive to the use of Cloud computing by many companies and suppliers of web-based services. Therefore, the Cloud services' users can see Cloud computing as a major combination of capabilities [19], such as, universal access, fine-grained usage controls and pricing, standardized platforms, management support services, etc, that together form an unique combination that improves computation and web-based services.

- **A Massively Scalable Infrastructure**

  The infrastructure scalability is the most characteristic feature of Cloud computing. This aspect distinguishes Cloud computing from others architectures that have some similar features like Grid computing.

  From the perspective of end users, Cloud computing massive scalability allows them to manage and control the amount of computation and store, as they need, allowing consumers to abstract themselves from IT (Information technology) management.

  Enterprises and web-based service providers that run their own servers usually need to expand their infrastructure to improve their computation power and storage, and with those expansions often came some problems related with buy additional hardware or to fit computation into existing servers and IT managers will run into issues related with incompatibilities with the operating system, conflicts in the scheduling of workloads, etc. Those problems are avoided when Cloud computing is used, because of three technology principles that it is based on: rapid allocation of virtual servers, standardized hardware and persistent Cloud storage.

- **Rapid Allocation of Virtual Servers**

  Cloud computing elasticity is one of the major features that characterizes this computing model, by enabling users to allocate the number and type of virtual machines needed to perform a given task.

  In a Cloud, all physical servers become shared resources, where the distribution of jobs and virtual servers running on a set of physical servers can change quickly among those physical servers. Elastic computing enables users' applications to allocate and release virtual machines instances on demand, so that the computation power allocation and release is completely transparent to users' applications.

  Amazon EC2 is one example of Cloud service that allows users' applications to scale horizontally if they need more virtual machine instances to run a task. Amazon EC2 monitors the virtual machines' CPU utilization, so that if it reaches a given limit of utilization, a new instance is automatically created. But the truth is that not all applications are enabled to take advantage of this scalability. The application must be designed taking into account the characteristic of elasticity of the Cloud. The scalability of the infrastructure does not automatically reflect the scalability of the application.

- **Persistent Storage in the Cloud**

  Cloud Computing scalability allows us to create new instances according to a given task necessity. Each instance allocates computing resources and temporary storage on physical servers but, once the virtual machines are released, all data locally generated and stored would be lost. Therefore, there is necessary persistent data storage over the Cloud, which can be accessible thought any server in the Cloud and subject to access control restrictions.

  It is usual to decouple of persistent storage from servers in the Cloud, enabling Cloud computing providers to have a fine-grained control over the resources allocated in the Cloud. In fact, the massive scalability is possible due to the combination of rapid provisioning of standard hardware and the use of persistent storage.

- **Universal Access**

  The users that use Cloud computing to deploy servers or web-based services have universal access to service from anywhere on the Internet. The access of information can be done anywhere through a web browser, a light weight desktop application or a mobile application, while the business model and data are stored on servers in the Cloud.

  Needless to say that universal access is not the same as open access, mainly when we are referring to private Cloud. To restrict the access to the resources in the Cloud, authorization and authentication mechanism are used. Even in public Cloud, regarding the accounting necessity, is required some identity mechanism to support the management and billing.

- **Fine-Grained Usage Controls and Pricing**

  Cloud is becoming a popular computing model especially because services consumers can rent what they need instead of building and running the entire infrastructure, which have a huge cost to set up and run, for instance, to implement a data center beside the cost related with hardware and software licensing, there are some others concerns like air conditioning, electricity, physical security, security systems anti-disasters (fire, floods, earthquakes, etc.), and so on.

  The economic benefits of Cloud Computing paradigm are key factors to its adoption; therefore users have fine-grained usage controls and pricing. It is common to have the problem to measure the computation necessity when it is needed to buy a new server to an infrastructure, where there is the risk to undersize the needs and do not meet the SLAs (Service-level agreement) or to oversize the capabilities needed spending an unnecessary amount of money. But with Cloud computing, the users will no more run into this problem, because they can allocate the compute power and storage as needed by their applications, for example, in periods of peak demand, is allocated resources from Cloud to meet the need and release them when they are no more necessary, so users pay only for what is used.

2.2.3 Deployment Models

There are 3 implementation model of Cloud Computing infrastructure, which are public Cloud, private Cloud and hybrid Cloud.

- **Public Clouds**

  This deployment model is based upon the outsourcing of computation and storage to third parties, which are located in data centres that operate outside of the companies that use them. In this model all the resources, processes and data are handled by a public Cloud service provider. Usually this public Cloud services are publicly provided through a pay-per-use model. This model can be confronted with some privacy and security issues.

  Public Cloud users see resources in the Cloud as an infinite resource, that the can allocate as they need, paying just for what they need and what they use. Cloud providers usually provide Web services or Web applications to access the infrastructure over the Internet. Public Clouds are provided by companies like Amazon, Hewlett-Packard, IBM, Google, Microsoft, Rackspace, Salesforce, etc.

- **Private Clouds**

  Private Clouds are operated by a company or a Cloud computing provider and the services provided are consumed internally by a single company, not being publicly available. Private Clouds use the same technology as public Clouds and they are often used when the infrastructure's compliance, security, and other risks are key factors. They also enable an individual company to maximize the use of its computing resources. Therefore, the utilization of private Clouds allows users to better manage policies and access control, and to define its own virtual machines to use in the Cloud.

  The utilization of private Cloud to in-house usage can be more efficient and responsive to a company needs compared with traditional IT operating model, because Cloud computing takes advantage of some important features to improve resource allocation which were discussed early. However, compared with the public Clouds, it requires a huge amount of capital expenditure with hardware and software licensing. The use of private Cloud is seen with some criticism, because some critics affirm that it is as like as the traditional servers' infrastructure maintained by a company, where a staff of IT professionals must be available to manage the infrastructure. Anyway private Cloud does not get rid of the issue of the capacity planning, and the expansion of its infrastructure would require capital and time expenditure.

- **Hybrid Clouds**

  Hybrid Clouds are a composition of private and public Clouds, where a company that have a private Cloud can use resources from public Clouds to extends its Cloud's capacity. The hybrid Clouds are commonly used when the business needs to deal with some security, privacy, and access control issues. There are a few ways to implement hybrid Clouds.

  This deploy model could use two Cloud as separately managed service platforms, where some policies are defined to designate what kind of task can be run in the public Cloud, and what can be run in the private Cloud. The consumers can choose freely which service they are going to use, taking into account some important variables like, for instance, the cost of use of public Cloud in a hybrid model can be less expensive or provide more capacity than the private Cloud.

  Another way to implement hybrid Cloud is to create a Cloud where the management of the private and public components is made within an unique service management platform. Using this approach the private Cloud and the public Cloud still as two independent services, which are managed using a single point of management.

  In some hybrid Cloud there is an implementation of a Virtual Private Network (VPN) in the public Cloud, in order to treat a portion of this public Cloud as an extension of the private Cloud.

  There are some vendors offering solutions that can be used to enable hybrid Cloud deployment. For instance can be found offerings like Amazon Virtual Private Cloud [21], Skytap Virtual Lab [22], and CohesiveFT VPN-Cubed [23]. These solutions use IPSec VPN tunneling to connect the public Cloud infrastructure to the on-premise Cloud resources.

There are many way that Cloud computing can be delivered. The choice of which deploying model to use highly depends on specific requirements, consequently seek to achieve maximum optimization of earnings and reduction of costs, i.e., capital expenses (CAPEX) and operating expenses (OPEX).

2.2.4 Services Layers in the Cloud

Cloud computing services can be divided into three layers [20], each implementing a different service model, which will be presented in this section. These models are very important to define an architectural pattern for solutions based on Cloud computing.

The Cloud computing services layers are organized in a manner that analysed from a top-down perspective can be concluded that each layer can be composed from the services of the layer underneath.

- **Software-as-a-Service (SaaS) Layer**

  This is the highest layer in the proposed model. Software-as-a-Service is characterized by it high level of abstraction, providing ready-to-run services that are deployed and configured to be used by end users. In this layer, users have no control over the underlying Cloud infrastructure, therefore it represents just an access point to end users to reach a given service like, for instance, portals or visualization tools.

  SaaS unlike traditional software it does not need a client side software installation, and all the data and business logic is held in the Cloud infrastructure. One of the SaaS important features is that the softwares provided as a services in a Cloud have universal access through Internet and it also has high availability accorded int the SLAs. The SaaS applications usually have a Web based interface accessed via Web Services [24,25] or Web2.0 [26,27].

  SaaS is massively used nowadays, one of its major utilization examples is the email service, which is highly used over the Internet, and has high availability. Email service like, for instance, Gmail is always available, and can be accessed through Internet using a web browser or a smartphone thin client. Besides email services, we can find others software provided as a service like, for example, Dropbox, Google Apps, social network applications (e.g. Facebook, MySpace), etc.

- **Platform-as-a-Service (PaaS) Layer**

  Platform-as-a-Service allows consumers to have abstraction of applications from traditional limits of hardware allowing developers to focus on application development and not worry about operating systems, infrastructure scaling, load balancing and system administration task. PaaS allows users implement applications to be deployed on Cloud providers infrastructures. These applications are developed using the programming languages and APIs defined by the Cloud provider. PaaS provides users limited control over the underlying cloud infrastructures. They can deploy and configure applications created using the vendor's programming environment. The process of implementing and deploying a cloud application becomes more accessible while allowing the programmer to focus on important issues.

  A well-known PaaS example is the Google App Engine [28], which enables developers to deploy applications using Python and Java API. Windows Azure [29] is Microsoft's PaaS platform and offers different types of runtime environments and storage services for applications.

- **Infrastructure-as-a-Service (IaaS) Layer** is the Cloud computing layer where low-level virtualized resources like computation, storage, network, are offered on demand, to be used in a self-service manner. Iaas allow users to instantiate virtual servers which can run several choices of operation systems and software stack. The Cloud Infrastructure-as-a-Services provides instant scalability and elasticity, so it is possible to dynamically expand applications' compute power through the capacity of instantiate new virtual machines rapidly; allowing users scale the computation power as they need.

  IaaS is the bottom layer of Cloud computing services' layers over which all others Cloud services layers are builded on. The resources are available in a virtualized system, where users have full administrative access to their virtual machines. Nowadays, users can find numerous Cloud providers that are offering IaaS. Furthermore some Cloud providers allow users choose which VM image they want to use from a variety of VM images, which can be based on a Windows platform or a Linux-based platform. Well-known example is Amazon EC2. Rackspace and GoGrid also provide similar services.

2.3 Extensible Messaging and Presence Protocol - XMPP

This work aims to use socialnetwork paradigm and technologies in the medical environment, where the connection between users is very important. Driven by the need of a connected environment, where users could interact with each other in order to improve telework, was adopted the XMPP instant messaging protocol. XMPP protocol empowers the creation of high scalable systems. XMPP and SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions) are the two dominant instant messaging and presence protocols, which has been competing over the years. SIMPLE protocol appeared as result of the necessity to support instant messaging and presence in SIP (Session Initiation Protocol). SIMPLE extensions define SIP signalling methods to handle the transport of data and presence. In these battle between these two instant messaging and presence protocol, XMPP is the one that is gaining ground in the Web. That is because XMPP benefits of being a XML-based protocol so, it can be extended with some new functionalities. XMPP has been gaining ground especially in the social networking and instant messaging applications domain.

The main reason that lead to the use of XMPP is because it has everything to become the future of Cloud computing services. It is a rich protocol that allows two-way communication (that is better than pooling), have rich features like pub-sub functionality and it is XML-based, which turn it in an easily extensible/flexible protocol that allows the creation of custom systems. For example, is commonly used to implement realtime systems' middleware [30,31], taking advantage of XMPP near real-time communication features.

2.3.1 Definition

The XMPP is an open standard instant messaging protocol, which is defined by the XMPP core RFC [32]. From analysis the RFC, is possible to figure out, how well defined is the protocol and how it can be an added value for near-real-time messaging, presence, and request-response services.

The application of XMPP is vary vast, consequently it is not only used in instant messaging but also in many application that focus on voice over IP, real-time collaboration, social networking, microblogging, lightweight middleware, Cloud computing, etc. Because XMPP uses XML to exchange data between client and server, it inherits the XML's extensibility. Consequently this flexibility/extensibility is put in great use on many of protocol extensions that are available for XMPP protocol.

2.3.2 Architecture

XMPP protocol is not wedded to any specific network architecture, but it is commonly used in a client-server architecture, wherein clients access the server using the XMPP protocol over a (TCP) connection. In addition, it is possible to connect servers with each other over a TCP connection. The XMPP defines three main entities that live in the XMPP network, which are the XMPP server, client and components.

Servers

The servers in a XMPP network entity that provides a channel to message exchange between clients. The server enables the exchange of information with others XMPP servers via a server-to-server protocol or with clients through a client-to-server protocol. Particularly the servers are the circulatory system for any XMPP network, hence they have the task to route stanzas from one user to another in the same domain or from a local user to an user in a remote server.

One of the strength of XMPP comes from its decentralization, which allows anyone to run a XMPP server. There are some servers that partially or fully supports XMPP definition, available for Windows, Mac OS X, or Linux systems, for instance, Ejabberd, Openfire and Tigase, which are the most popular open source XMPP servers.

Clients

Clients are the entities that connect with the server using the client-to-server protocol. Usually those entities are human-driven, like traditional Instant Messaging (IM), but they can also being automated clients running as bots that execute some specifics services. Moreover clients must authenticate with a XMPP server to access the XMPP network. Furthermore each client, after access the server of its domain, can manage some features like its roster, private content, etc. These elements are held by the server, which is also responsible to manage some aspects of the client's session like, for instance, presence exchange, roster manage, etc.

Components

Components are used to handle services as add-on modules for the XMPP server. They are third class of entities that can connect to an XMPP server, besides clients and other XMPP servers. Components, like Multi User Chat (MUC), connect to a server and get a specific subdomain of the server assigned to them (e.g., groupchat.ieeta.ua.pt). Whenever the XMPP server receives a stanza addressed to a JID within this domain, it directly routes the stanza to the component over the component's connection with the server. Setting up a stream between a component and the server is done using a simple handshake protocol, defined in Jabber Component Protocol [XEP-0114] [33].

Addressing

In XMPP network, every entity is addressed by one or more jabber identifiers, or JIDs. The JIDs are very similar with email addresses, for instance, an entity can be addressed using *entity@ua.pt*. In order to send any data to an entity in a XMPP network, the JID must be used to identity which entity will receive the XML and also the sender must be identified by its JID.

Domains

XMPP as others services in the Internet uses Domain Name System (DNS) to provide the underlying structure for addressing, instead of using Internet Protocol (IP) addresses. The domain is present in any entity's address.

Generally it is easier to user memorize that there is a service available on *ua.pt* rather then remember the server IP address. Furthermore, this format uses the complete DNS infrastructure as its address space. Because XMPP uses DNS to address different server in a network, it become more easily to manage than using IP addresses to identify each server.

Usually every JID is composed by three parts that are a local part, the domain, and the resource. However a JID that is composed just by the domain is also a valid JID, and represents the address of the XMPP server on that domain.

Users

The user identification in a XMPP network is made using the local part of the address, which is separated from the rest of the JID by the @ character, and by the domain. When an user registers on a XMPP server that is in a specific a domain, for instance *ua.pt*, he can choose the name that will be used to identify him in the server domain. For instance, the user can choose the name *john* and he will be identified as John at domain *ua.pt*, represented by the JID *john@ua.pt*. In some cases the user address in a XMPP network is automatically assigned by integrate an existing email address of a service as the user address in the network. One example of automatic assignment happened on GTalk, where users that have a Google email account can access the Gtalk's XMPP network using their Google's email address, without have to register on the Gtalk service.

Resource

A client can have several active connections on a XMPP network, thus to identify each connection there is the resource. Clearly, because an user can have multiple active connections, the resource will be used to route XML messages to the correct connection. For example the user John can have a client that is active at his home and another client that is active on his working place, thus these connection can be identified as *john@ua.pt/home* and *john@ua.pt/work* respectively. The resource allows the message routing to a specific connection, for example, if an user wants to contact John only when John reaches the client that is at *john@ua.pt/home*, the message is sent to that resource. Obviously, if there are many connections associated with an user, each connection have its own presence status, with different availability states, capabilities, etc.

2.3.3 Streaming XML

XMPP defines a technology for streaming XML. Moreover the streaming of XML, between endpoints, can occurs through an encrypted communication channel, using Transport Layer Security (TLS). In addition, before start streaming XML over the created stream the client must to authenticate using a mechanism that is done via Simple Authentication and Security Layers (SASL). After a client negotiate a stream with server and establish connection, he can exchange three type special XML, called XML stanzas, over the stream. These stanzas are <**message**\>, <**presence**\> and <**iq**\>.

The XMPP stream is basically a XML document that is build up incrementally by client and server. The document root element is <**stream:stream**> and the children consist of routable stanzas that are exchanged between client and server.

Unlike most traditional Internet technologies, like email or Web, the XMPP protocol opens a long-live TCP connection that is used by client and server to exchange XML stanzas in a XMPP stream. Traditional technologies like email and Web, use transactional connection, where the client makes a request, waits for the server response and, after receive it, the connection is closed. As a consequence, the transactional connection is not suitable for real-time applications, because there is no long-live connection between client and server that can be used by server to push information to client. In contrast, the XMPP creates a “always on” connection that is used by client to send several request without blocking while waits for server to reply, and the server will reply as soon as it has a response that satisfies the request using the long-live TCP connection create for stream the XML stanzas. XMPP brings new challenges to developers that are usually familiarized with to a different connection concept. On using XMPP, developers must be able to view the connection between client and server as a channel that is used to make asynchronous information flow, in form of XML stanza stream.

XMPP Stanzas

As was referenced before, the XML streaming on XMPP is based on three kind of XML stanza: <**message**\>, <**presence**\> and <**iq**\>. For each of these communication primitives, there are some attributes that define the semantic of the primitive. Moreover the stanza can have child element(s), which define its payload. The payload is presented to the user or processed according to the specification that defines its namespace.

There are some attributes that are common to all XML stanzas. These are used to determine how the stanza must be routed and what kind of stanza is being processed. The following attributes have the same mean for all of the XML stanzas:

- from

  It identifies the client that is sending the stanza.

- to

  It identifies the destination client of this stanza.

- type

  It can have several values, according to the stanza. It is used to determine the specific kind of stanza: <**message**\>, <**presence**\> or <**iq**\>.

- id

  It is used to create a way to identify responses. This attribute is required for <**iq**\> stanzas, because <**iq**\> is usually request-response, and this id can identify which is the reply of a given request. This identification is possible due to all reply stanzas have the same id as the request stanza, which they respond.

Message stanza

The <**message**\> stanza is used to send information from one place to another, and generally it uses a push method that does not need to be acknowledged. The <**message**\> stanza is based on a fired-and-forget concept, thus there is no built-in reliability. After send a message, the sender has no confirmation whether the message was delivered or when it was received. But, if it is necessary to receive the deliver acknowledge, the application protocol can be extended using, for instance, Message Receipts defined in XEP-0184 [34].

The application of message stanza is usually associated with instant messaging, but the scope of it usage can be vast, and there are some application of this kind of stanza in transport of any kind of structured information. For example, it can be used to make group chat, send notifications, alerts, to transport drawing instructions, communicate a game state and new game moves, etc.

The message stanza can be divided in five kinds, and they are distinguished by the **type** attributed:

- normal

  By type attribute omission the message stanza is considered as normal type message. This kind of message is sent outside the context of one-to-one chat, and they can be compared with an email, but, which can or cannot be replied.

- chat

  This kind of message is associated with an one-to-one chat communication, and it is usually used in instant messaging (IM) applications. This message is used in a real-time communication between two entities, which are primarily concerned with private one-to-one communication.

- groupchat

  The group chat is used in multi-users chat. This type of message is used to disambiguate messages that are sent to a single user in a chat room from messages that should be sent to everybody in the room, as broadcast.

- headline

  This kind of message is used by entities to push notifications or alerts, and clients that receive them are not expected at all to reply.

- error

  The error message is used to reply to a message that generated an error. For instance, if an entity tries to send a message to another entity that does not exists or to a domain that the server does not know, the server will generate a reply message with the type error.

Message stanzas are usually extended with some elements defined by the core XMPP specification. For instance, body and subject can be used as a message payload. The body element is used to hold the message text and, to identify the message subject, is used a subject element. Furthermore, it is possible to add some custom element as a message payload. These custom elements are applications specifics and only respective clients are able to process them.

Presence stanza

Presence stanzas are used to advertise, to the XMPP network, the entities' availability. The presence advertisement is catalyst for communication and collaboration in the Internet, because it allows people to know whether others are online and available to communicate. Unlike mail service, which users are not able to know if the recipient is checking the mails, with instant messaging and presence notifications, users are able to realise whether the recipient is available before send the message.

Nevertheless, in a XMPP network, it is necessary to be conceived and authorization to visualise an entity presence status. This authorization is gathered through a handshake between entities, and it is called as presence subscription. To make the subscription there are some specialized Presence types, represented in Table 2.2:

Type	Description
subscribe	The sender wishes to subscribe to the recipient's presence.
subscribed	The sender has allowed the recipient to receive their presence.
unsubscribe	The sender is unsubscribing from another entity's presence.
unsubscribed	The subscription request has been denied or a previously-granted subscription has been cancelled.

Table 2.2: XMPP Presence types used in subscription.

The XML stream in Listing 2.1, represents an example of mutual presence subscription between Bob and Alice.

Listing 2.1: Presence subscription

```
<presence from='bob@ua.pt/home' to='alice@ua.pt' type='subscribe'/>
<presence from='alice@ua.pt/mobile' to='bob@ua.pt/home' type='subscribed'/>
<presence from='alice@ua.pt/mobile' to='bob@ua.pt' type='subscribe'/>
<presence from='bob@ua.pt/home' to='alice@ua.pt/mobile' type='subscribed'/>
```

The XMPP presence subscription is a specialized publish-subscribe method, wherein the subscribed entity will receive updated presence information when one entity comes online, changes the status to "in a meeting" or "available to chat" and goes offline.

The basic presence is able to notify whether an user is online or offline. But core XMPP provides extended presences that introduce other states like "Away", "Do Not Disturb" and "eXtended Away". This extended presence is set using a <**show**\> element as presence stanza payload. The <**show**\> element can take the following values, represented in Table 2.3:

Show	Description
away	The entity or resource is temporarily away.
chat	The entity or resource is actively interested in chatting.
dnd	The entity or resource is busy (dnd = "Do Not Disturb").
xa	The entity or resource is away for an extended period (xa = "eXtended Away").

Table 2.3: XMPP extended presence

The XMPP Presence stanza can be enriched by customized status messages like, for example “I’m working, don’t bother me right now”.

To actualise the presence status the entity must send a <**presence**\> stanza containing information about his new status. The XML represented in Listing 2.2, shows an example of a Presence stanza.

Listing 2.2: XMPP Presence stanza

```
<presence from='bob@ua.pt/mobile'>
        <show>xa</show>
        <status>at the ball!</status>
</presence>
```

Beside those types of Presense stanzas that are used to make the user presence subscription or unsubscription there are others types of Presences stanzas with some specifics functions in the XMPP protocol (Table 2.4):

Type	Description
unavailable	Signals that the entity is no longer available for communication.
probe	A request for an entity’s current presence.
error	An error has occurred regarding processing or delivery of a previously-sent presence stanza.

Table 2.4: XMPP Presence stanza types

IQ stanza

The Info/Query stanza, or commonly known as IQ, are used to create a request-response interaction and implement simple workflows, similar to the GET, POST and PUT methods from HTTP. This stanza allows both get and set of information from a recipient or server. Every sent IQ stanza must receive a reply with the same id.

The <**iq**\> stanza can only contain an element as its payload. The payload element defines the request to be processed or the action that should be executed by the recipient. The <**iq**\> stanza comes in four types differentiated by the type attribute of the stanza. These four types of IQ stanzas can be divided in two groups (Table 2.5):

Throughout the document these different IQs are often abbreviated as IQ-get, IQ-set, IQ-result, and IQ-error.

	Type	Description
Request	get	Identifies the intention to get some information from the recipient.
	set	Identifies the intention to set some information to the recipient.
Response	result	Similar to HTTP status code 200, it is used to reply the success of and action performed by an IQ-set or IQ-get.
	error	This type is used to report an error generated by the execution of an action associated with an IQ-set or IQ-get.

Table 2.5: XMPP Info/Query stanza types

2.3.4 Extensibility

Any XMPP stanza can be extended with any payload defined by the developer. Because that extensibility capability, it is possible to create infinite extensions to the XMPP stanzas, by taking advantage of XML extensibility. An XML stanza can contain many other child elements, which can be used to carry information like SOAP data for Web services, geographical location, XML-RPC, forms to be fill out or submitted, XHTML-formatted message bodies, or any other type of information can be used as payload.

Because XMPP is an open standard it has been supported over the years by developer community, which has been defining many extensions to the XMPP core protocol. XMPP extensions development can be seen as an open and collaborative standards process. After the community acceptance of an extension for the core protocol, this will be published in the XSF's XMPP Extension Protocol (XEP) series at http://xmpp.org/. However, anyone can developed and extension to implement some particular functionalities, not being considered in the core XMPP extensions.

XMPP take advantage of XML namespaces to scope stanzas payloads, thus any extension is matched using the element name and its namespace. One example of XMPP extension is the Private XML Storage defined by XEP-0049 [35]. This extension is used to store private XML data on a XMPP server. It is identified by the namespace **jabber:iq:private** and payload child element **query** (<**query xmlns='jabber:iq:private'**>). Thus to create an extension to a XMPP stanza, developers must define and element name, for example <**cloudmed**> and a namespace to scope this extension as, for example, *custom:iq:cloudmed.*

Chapter 3

CloudMed Proposal Definition

The main goal of this thesis was to create a software that allows physicians to share medical images and enhance telemedicine and telework by exploring new approach for cooperative environments supported over the Cloud, optimizing the information share and allowing physicians to obtain a second opinion from specialized ones.

In this section, it will be presented the requirements that were considered to implement a SaaS that leads to the improvement of telemedicine's workflows and dataflow.

3.1 System Requirements Overview

At the beginning, were identified some features that could improve telework processes. These features were established to facilitate the medical images' share between radiologists, taking in account the users' needs and how a Cloud-based system could be useful to enrich users productivity by improving workflow in telework.

In telemedicine, there is the need to link people as well as machines, so from this necessity emerged the socialnetworks concept to connect colleagues. The creation of computer-supported social networks can be an important base for the creation of virtual communities in medicine. Thus, this work also had the aim to integrate the socialnetwork concept into a telemedicine service that can be used by radiologists. Therefore, users can take advantage of the proposed system to allow them to interact with each other in a synchronous or asynchronous way, thus creating an environment propitious to cooperative work.

The cooperative work is based on the medical images' share between colleagues, allowing the users to get second opinion about a given examination, share interesting clinical cases, implement reporting services, etc. With this service, a radiologist can annotate a given medical image, including text, sound and draw, as result of an examination, and interact with his colleagues through the use of social networking features like, for instance, messaging and presence.

Telework usually requires access to information that is stored in remote servers inside health care institutions. A system that aims to create services for telework has to take it in account. So the proposed system was defined in such a way that it enables users to remotely access to repositories. Thus, this feature was defined as an important requirement for the system that this work aims to create. It will allow radiologists access to medical images stored in a remote PACS repository, and also it will allow an user to share a repository with other colleagues, so improving the social networking and cooperation environment.

Obviously that must be considered some mechanism to control the access over the content when an user shares his PACS repositories with other colleagues. The access to those PACS repositories has to be controlled by an access list where only permitted users can access to the information. So, an user can have several PACS repositories available through the system; therefore he has to manage several access lists, where he manages who can access to a given PACS server. Besides the access list control, it should be possible to define if the information is anonymous shared or not.

The creation of a Cloud-based service for telemedicine would become an important service if it allows users to access their information from anywhere and any-time. This can be allowing the storage of information in the Cloud. So our proposed system was designed to allow users to store medical images in the Cloud, where they have their personal remote archive. The solution let users do more than just store their files in their personal archive, it is possible to share medical images with friends, thus improving telemedicine workflow.

Building the service in the Cloud enables files to be available through the web from everywhere users are, in their home, at the office, etc. Following the concept that led the creation of some services like Dropbox, Sky Drive and Google Drive, the proposed service also lets users to synchronize files from their remote archive with a given workstation.

With all the information gathered from the medical images repositories and personal files stored in the Cloud, the system allows the users to access and manipulate this information just using and Web browser, providing mechanisms to visualise medical images in a Web application, and attach information about the examination of a given study.

3.2 Functional Requirements

From the requirements overview defined in last section, were derived the functional requirements that are defined in the table 3.1.

Requirement	Description
PACS gateway	
Configure Web access to PACS server	The gateway should act as an interface that should allow web clients to interact with PACS infrastructure.
Enable query on PACS server	The gateway should receive web clients' messages and transform them in DICOM C-Find, allowing clients to search on PACS server.
Enable data retrieve from PACS server	It should be possible to a web client to order the gateway to retrieve given studies from the PACS server and store them in a repository in the Cloud.
Web application	
Add file to Cloud	User select a list of files on local computer and upload those files to his personal repository in the Cloud
Browse the personal repository	The system should allow the user browse through folders he has in his personal repository.
Browse shared repository	It should be possible to an user to browse on files shared by other colleague.
Remove file from repository	Users should be able to remove studies/series from their personal repository.
Synchronize files with workstation	It should be possible to users synchronize a given folder that they have in the Cloud with a given workstation.
Share folder with friends	Users should be able share all files from a given folder that represents an institution, patient, study or series with a friend.
View and manipulate medical images	The system should enable users to visualize and manipulate medical images using certain tools like, for instance, zoom, contrast, circumscribe areas, etc.
Visualize user presence	It should be possible to be aware of a given friend presence status using the service.
Chat with colleagues	The system should allow user to communicate with his colleagues.
Visualize his PACS gateways status	The user should be able to view if PACS gateway are on or off.
Share PACS gateways which friends	It should be possible to give access to a gateway to a colleague.
Visualize shared gateways status	Users should see if PACS gateways, which were shared by colleagues, are active or not.

Table 3.1: List of functional requirements.

The system main use cases are represented in the Figure 3.1, where we can see how user can interact with the Web application and PACS gateways.

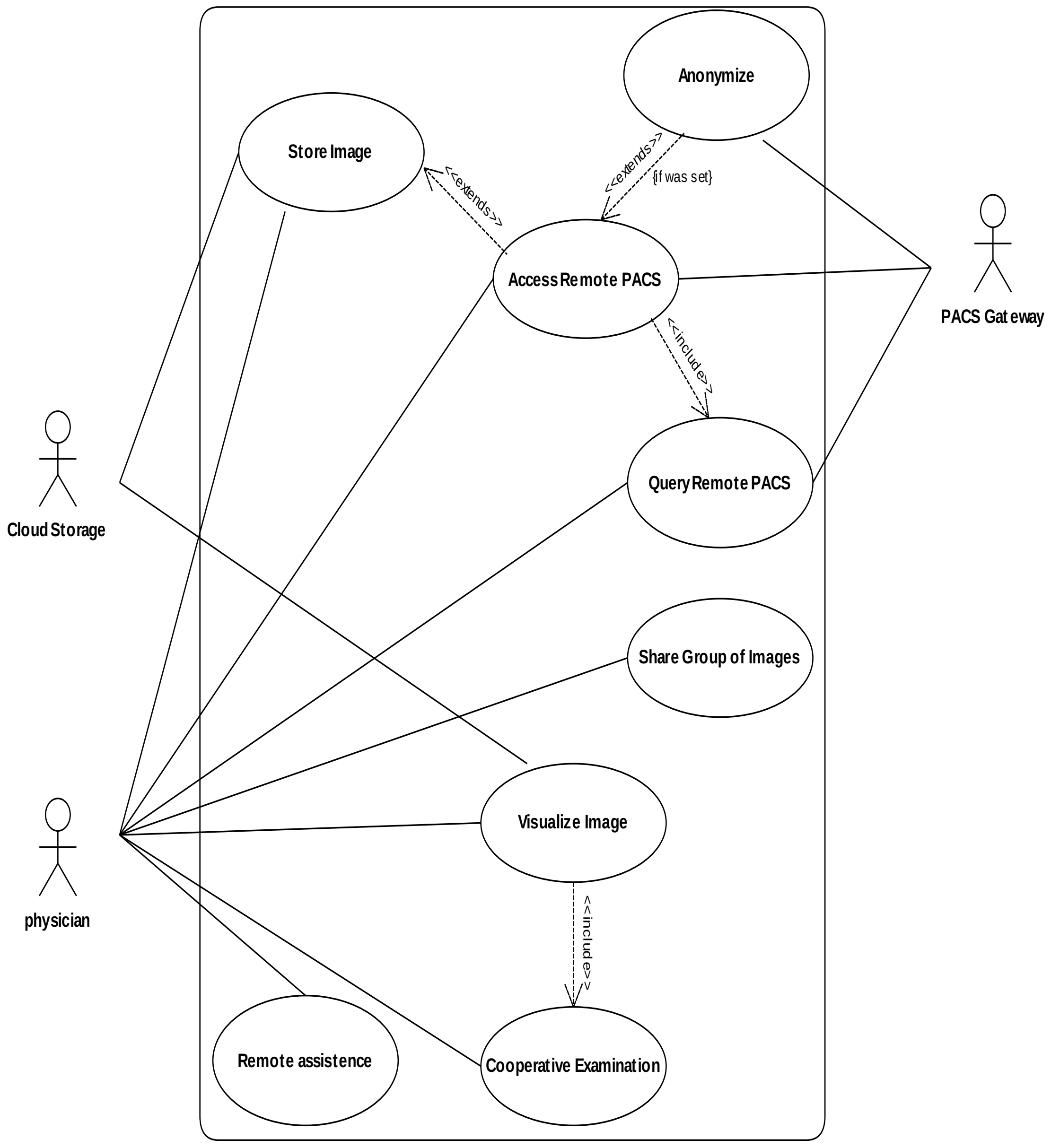


Figure 3.1: System use cases.

3.3 Non Functional Requirements

3.3.1 Information Accessibility and Availability

In the system requirements, was established that the service should be accessible from anywhere and it should have a high availability, any time that the radiologist tries to access it. This characteristic is very important in telemedicine scenarios where the information flow cannot be interrupted. Thus the availability is an important factor to create an efficient system, which takes care of a crucial service that should be always available to meet the needs of a system that is serving health care interests.

3.3.2 Information Security

Moreover, the information that flows in telemedicine services is very sensible, so the system needs to be implemented taking in account this issue related with security, namely users authentication, data integrity and confidentiality.

3.3.3 Service Reability

Also, when a system is leading with medical data, should be developed with some concerns about the data reliability, thus should be created mechanisms that prevent the data to be lost or stolen, because of its nature.

3.3.4 Service Scalability

In addition, it should be designed taking in account its scalability, in order to be easy to improve its computation and storage capacities to handle high demands that may exists in the future. The system should provide an architecture that allows it to scale horizontally, with introduction of new servers to handle the load and more storages capacity.

The four requirements later presented (data access and availability, security, reliability and scalability) can be achieved by taking advantage of Cloud computing concepts. When a system is developed based on Cloud computing it inherits directly from Cloud computing model, characteristics like the system access anywhere and anytime, and also the reliability. These features are usually defined in the SLA's terms of contract. Greater care should be taken with the information security. Anyway, a Cloud based system can ever improve the data security, ciphering the information that is stored in the Cloud.

3.4 System Modeling

In the following section will be conceptualized the system according to the system requirements identified in the previous sections. First, it will be presented the proposed functional model of the system, including all the functional blocks and the interaction between them. After, in the Data Model section, it will be presented a high-level data model of the system, which will underline the data need and created by business processes. Finally, it will be presented the resulting system architecture that integrates all components required to construct the desired system.

3.4.1 Functional Model

The functional model of the underling system was derived from the identification of all key components from a top-down analyse of the requirements. It was made a functional decomposition, which divide the system in such a way that each block identified was self-descriptive. Hence, it was possible to identify several global business processes that compose the system, which derive from the system requirements. The key processes of this system are:

medical image store in the cloud; access to those files for examination; share those images with colleagues; access files from remote PACS; share remote PACS with colleagues; enable cooperative collaboration.

Message-oriented middleware system

The system modelling was highly influenced by the main business processes, thus at modelling stage it was necessary to take in account some features related with each processes for the definition of the best implementation approach basing on these features and the way they are directly or indirectly influenced by each other.

System requirements define that is a must that the system should provide features to enable cooperative collaboration between users and, in addition, also allow the access to information from PACS infrastructure inside a health care institution. However, this access has to be done through a gateway, which the user needs to be aware of its current status (active or inactive). So at, modelling stage, it was necessary to define that the system provides mechanisms to allow firewalls bypass and notify gateway presence. Furthermore, associated with the need to enable user to chat with each other, notify colleagues presence, and create a flexible way to access remote PACS server, lead a system to model basing in a message-oriented middleware. Consequently, it was chosen to use Extensible Messaging and Presence Protocol (XMPP) because it features fit like a glove in the proposed Cloud-based service.

The XMPP allows us to implement some features like, for instance, chat and presence notification. Moreover, it is possible to bypass the issue related with gateways and clients behind a firewall, because the system will use a XMPP server in the cloud, which will act as a relay on messages exchange between XMPP clients. More generally, there are other characteristics that led to use of XMPP, such as, being based on open standards to implement a messaging and presence technology, which is widely used and supported by companies like Google, Apple, AOL, IBM, Jive, etc. In addition, XMPP is decentralized, so anyone can run his own XMPP server and it is possible to connect with other XMPP servers (Figure 3.2). Consequently, XMPP is a way to create highly scalable system. For example, Google's GTalk is scaled to millions of concurrent users on a single service.

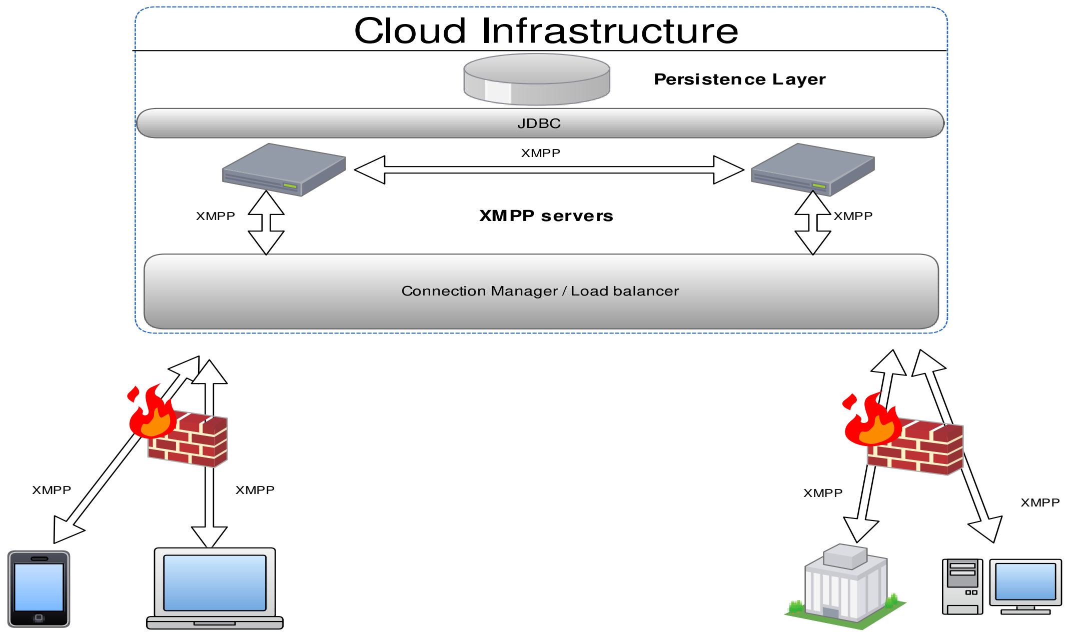


Figure 3.2: Scale XMPP infrastructure with multiple servers. The server-to-server communication in a XMPP network enables to balance the load between servers.

XMPP network architecture is highly scalable because it is possible to deploy several XMPP servers and make them communicate with each other using a server-to-server protocol. The use of more than one XMPP server in the XMPP network is an asset when the load of a single server the network becomes in certain manner unsupportable. Thus, in XMPP network where there are several servers to handle users' connections, it is possible to use a load balancer to distribute the load fairly by each server.

In concerns of security, XMPP provides a robust security protocol by using SASL (Simple Authentication and Security Layer) and TLS (Transport Layer Security). The security in the XMPP protocol is one concern that was taken in account since the beginning, and has been defined into the core XMPP specifications.

XMPP is XML-based, as consequence an easily extensible/flexible protocol, which empowers the creation custom cloud services. For example, Chesspark [36] is a service that integrates chess game in a social network environment, taking advantage of XMPP near real-time communication feature, that fits perfectly in a game that is often fast like chess.

The XMPP flexibility is very important for the underling system because it allows us implement a custom protocol (subsection 4.1.1) to interact with the PACS gateway, and also to implement some functionality like, for instance, store, get and search data in the service provided by this system in the Cloud.

Decouple Cloud storage and Cloud core infrastructure

Modelling a system to be deployed in the Cloud is a very important stage of its conceptualization. Because the vast number of Cloud providers that are appearing nowadays, is very important to design a system by taking an approach based on cloud provider decoupling.

Thus at this stage, the system design must be conceptualized taking in consideration the fast technology change and the appearance of new cloud providers offering better service at lower cost.

Undoubtedly, in the CloudMed system modelling is considered the Cloud provider decoupling, because it is designed to be deploy in any cloud provider. Furthermore, it separates the Cloud core infrastructure, which is deployed in an IaaS, from the cloud storage infrastructure. This decoupling of computation and storage allows the system to be implemented in several configurations seeking the better capital and operational expenditure balance. The decoupling is archived using well defined interface design patterns, where is possible to define an interface that is used to plug the core infrastructure with the storage infrastructure (Figure 3.3). By decouple the core computation infrastructure from the storage become possible to use different provider for storage and computation.

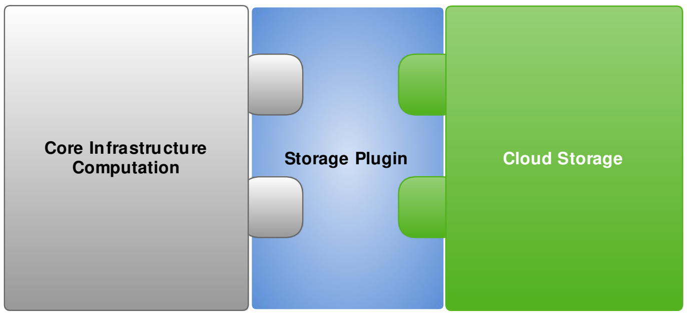


Figure 3.3: The diagram of how business logic layer interacts with the persistence layer. The core infrastructure requires a storage component that implements a given interface. The storage component can be implemented to use any Cloud storage provider.

In theory, if from the audit of a public Cloud storage services was determined that the levels of information security, privacy and reliability does not meet the system requirements, it can be possible to deploy the system as hybrid cloud, using public Cloud providers to deploy the core infrastructure and for meet the system data's security requirements, can be used a private cloud infrastructure as represented in Figure 3.4.

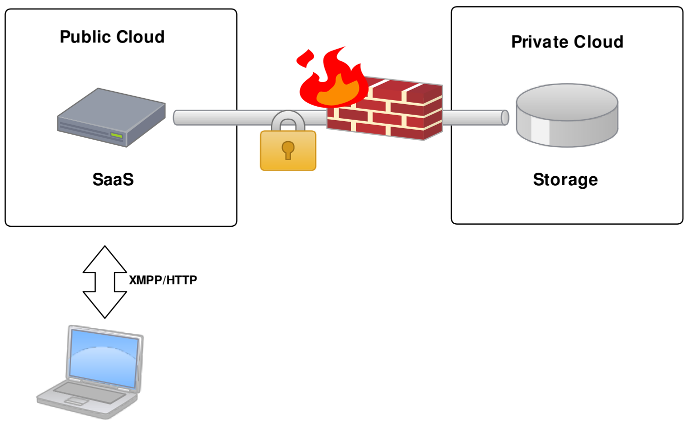


Figure 3.4: Holding medical images' data in a private Cloud. Nevertheless the service is available as a SaaS in a public Cloud.

Nevertheless, for prove of concept in this work, it was used a public Cloud infrastructure where both the core infrastructure and the storage infrastructure are deployed using public Cloud providers, as represented in Figure 3.5.

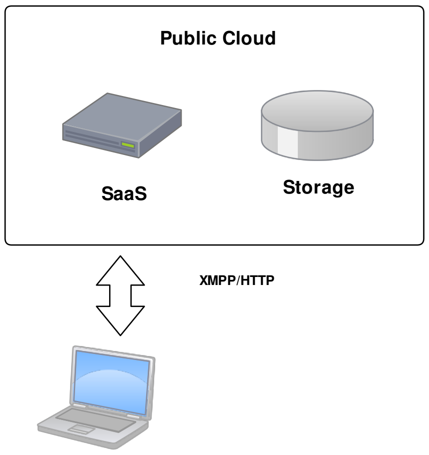


Figure 3.5: Deploying storage and service in a public Cloud.

Abstract private repository in the Cloud

The underling system, like Google Drive, Dropbox and Sky Drive, offers a private repository in the Cloud where users can store their data and access them from anywhere. As a consequence, it is important to model the basics business process that consists in the definition of how users store data in the Cloud and how they can manipulate these data. At the modelling process is considered that each user has a private repository in the provided service, thus all files he stores in the Cloud is linked with his private repository, which only he can access unless he share a specific study with some friends.

The creation of a Web application that allows user to interact with the system is a requirement; this Web application has to implement some of the abstraction used in desktop application to improve the user experience. Namely, when an user stores some medical images in the cloud, this action can be associated with the event triggered by the user when he drags and drops a list of files into the browser.

The way that the system manages the store action triggered in a specific private repository is represented by the diagram in Figure 3.6. Moreover, with the aim to reach better performance and reduction of processing load in the server, some pre-processing and modelling concerns are considered. Hence, there is a need to create a high quality JPEG (that is used in visualization) and a lower quality JPEG (used in web application thumbnails viewer) and, when the system is serving a given request that needs to access to an image, it is not necessary to do the conversion on-the-fly. Furthermore, to improve performance in medical images search, some information retrieved from DICOM image's header are indexed in a SQL database, allowing users to submit queries to select a given study or series.

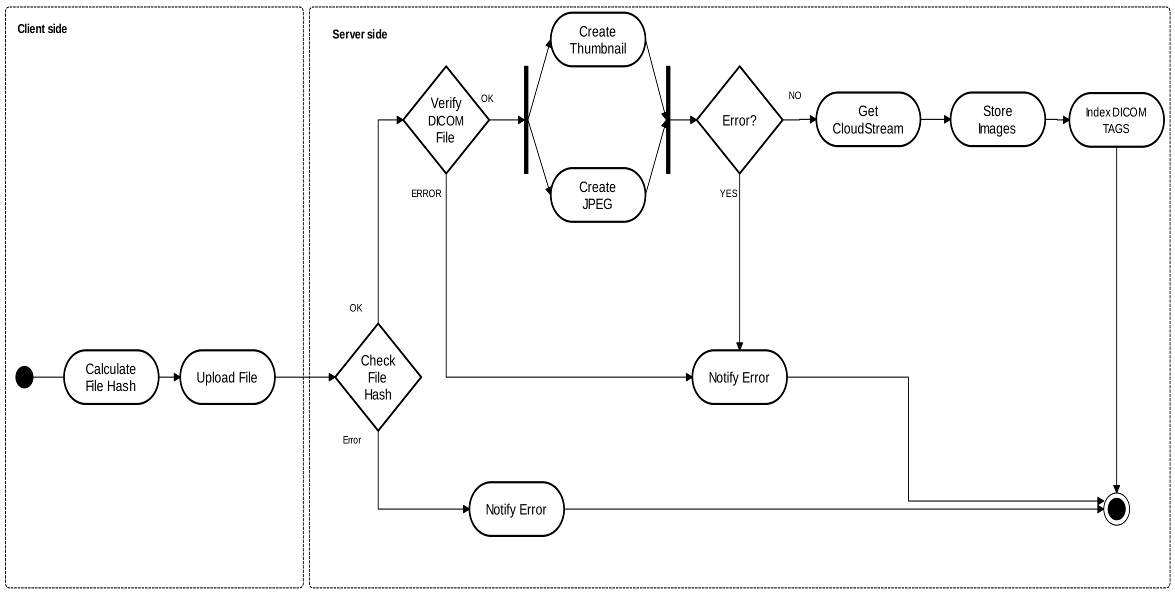


Figure 3.6: Flowchart of medical images' storage in the cloud.

DICOM images' header allows the system to get information of which institution, patient, study and series a given image belongs. This information identifies a hierarchical tree structured, which represents real world information (Figure 2.3). So, the system keeps this information indexed, aiming to enable the user to easily identify the relation between institutions, patients, studies, series, and medical images and, allowing him to execute queries to retrieve the desired data. If an user wants to share a given object that represents a series, a study or an institution, with a given colleague, he has to perform an action (Figure 3.7) that indexes this share into the system's database, and thus, his colleague becomes able to visualize and do an examination on images that belongs to this object.

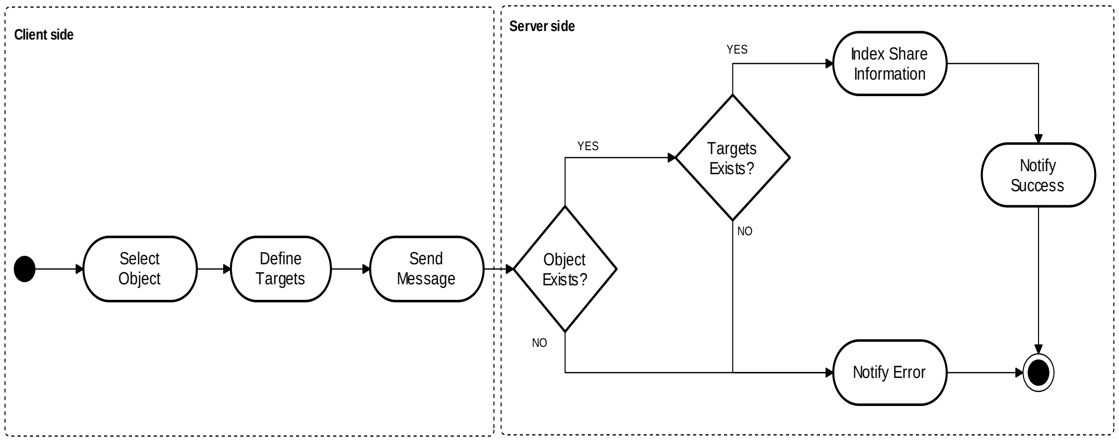


Figure 3.7: Folder sharing diagram. The object can be a given institution, study or series, and the share target represents an user that is registered into the system.

Accessing Remote PACS

The CloudMed system requirements define that the system must be modelled in order to make possible for users to retrieve series/studies from a PACS server and store those medical images into the users' repository in the Cloud.

The remote access to a PACS server is a functional requirement that is modelled using a gateway to establish the communication between CloudMed system and the given PACS archive server. This feature is an asset for the desired system. Hence, by allowing users to access remote PACS archive servers, users become able to search and retrieve information from remote repositories that are inside health care institution's information system.

Clearly, it can be said that the system is modelled with the aim to achieve the desired ubiquitous computing, which is becoming an important characteristic of contemporaneous information systems. Using the ubiquitous computing paradigm, CloudMed provides a thoroughly integrate information system into users' everyday activities. Thus, creating a pervasive computing system where the information is available everywhere, whether at home, work or on move.

To bypass issues related with firewalls that can block the access to PACS repositories from everywhere that is not from inside the institution, was modelled a Web 2.0 compliant relay based in the Cloud, using a message-oriented middleware to connect all system components (Figure 3.8).

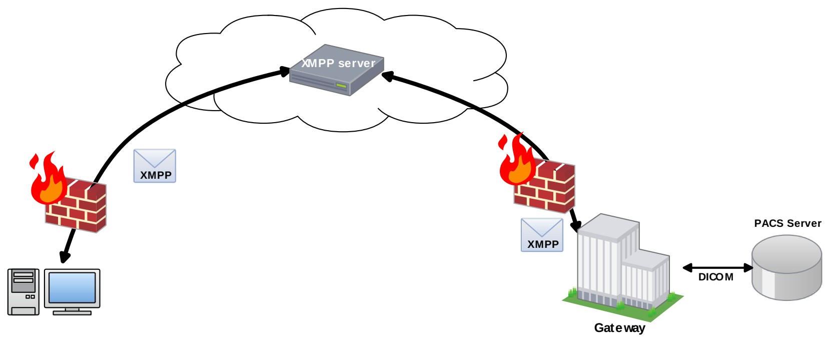


Figure 3.8: Firewall bypass using XMPP server as Cloud relay. The XMPP protocol is used as a message-oriented middleware to allow communication between every entities in the system.

Due to the utilization of a message-oriented middleware to connect every system's components, it becomes possible to send message to a given gateway that is inside a health care institution. This middleware uses XMPP protocol, and it allows to exchange message between components. By taking advantage from XMPP flexibility, it was possible to create custom functionalities, which follow the remote procedure call paradigm, where a component sends a message to a given component through the Cloud relay, and this messages contains information about which action must be executed in the target. Note that the system can handle more than one gateway from each client; therefore it is designed in such way that it allows the client discriminate them, using XMPP resource identifier [32].

In addition, in the system modelling, some access conditions and security issues were considered. Thus, it was created a kind of packet filter in the Cloud relay system. This packet filter aims to control the access to all PACS gateway components. Thus, the gateway that enables the access to a PACS server is controlled by an access list, which is persisted on a SQL database in the Cloud. So, only users allowed to contact a given PACS server can execute actions in the target gateway. The packet filter working process is defined in Figure 3.9. In short, it acts like a layer 7 firewall [37] that interprets all the XMPP messages that pass through the Cloud relay XMPP server, and forwards or blocks the message accordingly to access central rules. The system design allows the management of the firewall rules of each PACS gateway only by the resource owner. Thus, the owner of a PACS gateway can define who can access the repository through the management of its access list.

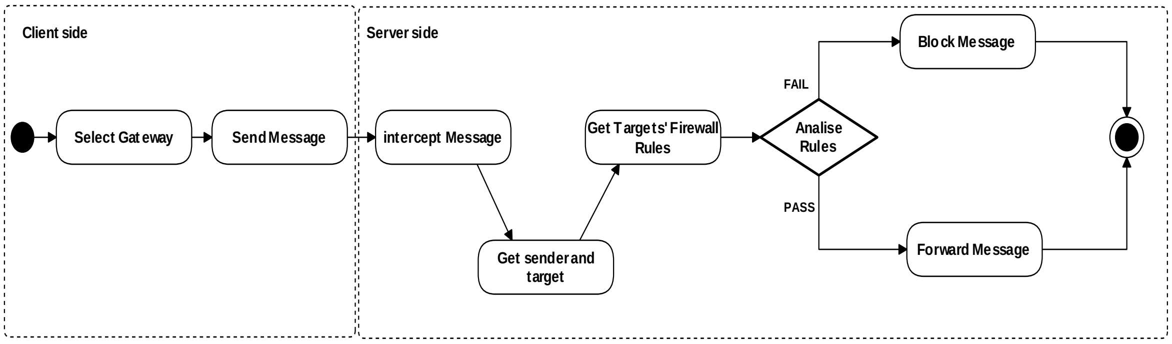


Figure 3.9: Cloud relay gateway packet filter diagram. The diagram represents a packet interceptor that acts when a message is received. The packet filter analises the target and the source of this packet, and if the rules permit, the messaged is forwarded to the target otherwise it is blocked.

The following diagram, represented in Figure 3.10, shows how the system provides a way to enable the communication with the PACS gateway components, thus creating an integrated environment, with access to information from inside the health care institutions from everywhere. The gateway plays an important role in the underling system by enable the translation of messages from XMPP protocol to DICOM protocol. As a result, it becomes possible to access to any entity in a PACS from the XMPP network, and consequently from the entire Internet, because the system will provide a SaaS that enables users to access the XMPP network through a Web application.

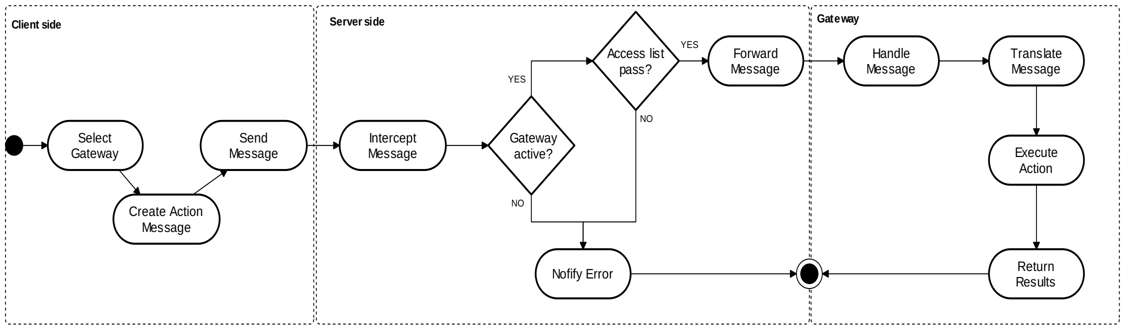


Figure 3.10: Remote action execute diagram. This diagram models the interaction between XMPP entities and PACS server through a gateway. This interaction is based on remote procedure call. A message is sent to the target gateway that has the role to act as an interface between the XMPP network and the target PACS infrastructure.

3.4.2 Data Model

The CloudMed is designed as a system with distributed resources that are plugged in the core infrastructure and provided as SaaS. To make easy the management of the system, it was decided to decouple of the core business infrastructure from data sources. The system is composed by many data sources (Figure 3.11):

- **System information SQL database**

  SQL database to store information about file store, shares and packet interceptor rules.

- **Cloud blob storage**

  It represents the medical images storage and it is implemented using an autonomous Cloud storage service.

- **Many PACS servers**

  There may be some PACS archive servers plug-in to CloudMed system to enable users to access to institution repositories and to share the resources with friends.

The CloudMed system features a completely distributed design, in which the resources are mostly spread by the Internet. In the system, the Cloud blob storage and the SaaS can cohabit in the same provider or they can be deployed on different Cloud providers. Futhermore, there may be several PACS servers available through CloudMed system and, as stated in previous sections, the access to all the data held by a given PACS server is through a gateway, which serves as an interface between the PACS infrastructure and CloudMed platform.

In addition, there was the need to use a SQL database to store information about the core business like, for instance, index information about files ownership, files sharing and the Cloud relay's firewall rules for each PACS gateway.

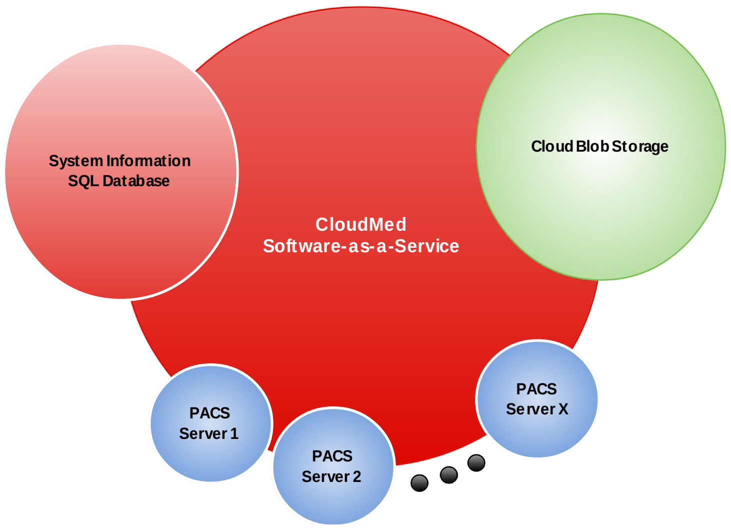


Figure 3.11: CloudMed system data model diagram. As can be seen in the diagram, CloudMed system creates a completed integrate environment, wherein users can access data from Cloud blob storage and from remote PACS servers.

The core business database structure is based on a SQL database. This database is managed directly by the core business infrastructure that commits information into this database. The physical diagram that represents how data is organized in this database is represented in Figure 3.12 and Figure 3.13. Figure 3.12 represents the data organization of the database to persist the file sharing and ownership and the Figure 3.13 shows how the system store the information about the packet filter's rules for message filtering in the Cloud relay.

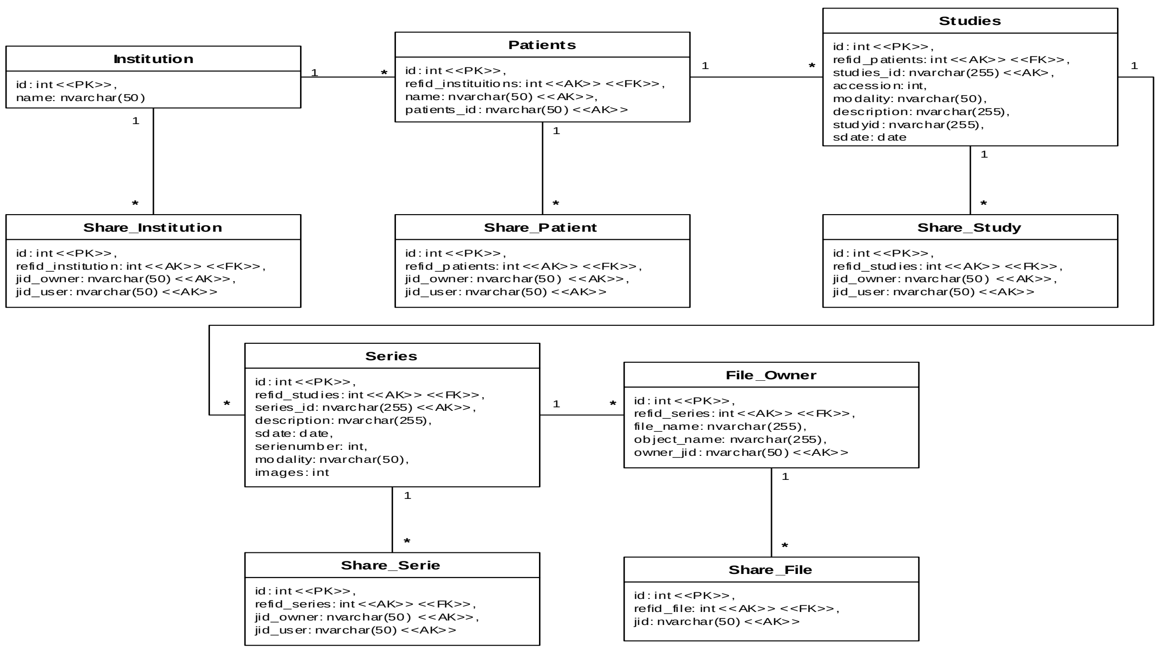

Institution
id: int <<PK>>,
name: nvarchar(50)
Patients
id: int <<PK>>,
refid_instituitions: int <<AK>> <<FK>>,
name: nvarchar(50) <<AK>>,
patients_id: nvarchar(50) <<AK>>
Studies
id: int <<PK>>,
refid_patients: int <<AK>> <<FK>>,
studies_id: nvarchar(255) <<AK>,
accession: int,
modality: nvarchar(50),
description: nvarchar(255),
studyid: nvarchar(255),
sdate: date
Share_Institution
id: int <<PK>>,
refid_institution: int <<AK>> <<FK>>,
jid_owner: nvarchar(50) <<AK>>,
jid_user: nvarchar(50) <<AK>>
Share_Patient
id: int <<PK>>,
refid_patients: int <<AK>> <<FK>>,
jid_owner: nvarchar(50) <<AK>>,
jid_user: nvarchar(50) <<AK>>
Share_Study
id: int <<PK>>,
refid_studies: int <<AK>> <<FK>>,
jid_owner: nvarchar(50) <<AK>>,
jid_user: nvarchar(50) <<AK>>
Series
id: int <<PK>>,
refid_studies: int <<AK>> <<FK>>,
series_id: nvarchar(255) <<AK>>,
description: nvarchar(255),
sdate: date,
serienumber: int,
modality: nvarchar(50),
images: int
File_Owner
id: int <<PK>>,
refid_series: int <<AK>> <<FK>>,
file_name: nvarchar(255),
object_name: nvarchar(255),
owner_jid: nvarchar(50) <<AK>>
Share_Serie
id: int <<PK>>,
refid_series: int <<AK>> <<FK>>,
jid_owner: nvarchar(50) <<AK>>,
jid_user: nvarchar(50) <<AK>>
Share_File
id: int <<PK>>,
refid_file: int <<AK>> <<FK>>,
jid: nvarchar(50) <<AK>>
1
*

Figure 3.12: Physical Data Model of file shares and file ownership.

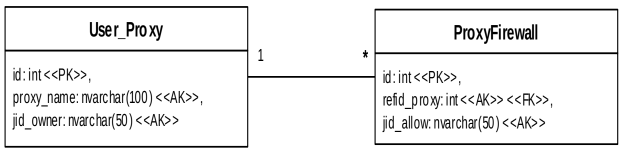

User_Proxy
id: int <<PK>>,
proxy_name: nvarchar(100) <<AK>>,
jid_owner: nvarchar(50) <<AK>>
ProxyFirewall
id: int <<PK>>,
refid_proxy: int <<AK>> <<FK>>,
jid_allow: nvarchar(50) <<AK>>
1
*

Figure 3.13: Physical Data Model for packet filter's rules.

3.4.3 Architecture

As result of the modelling stage, it was designed an architecture (Figure 3.14) to fit perfectly to the system requirements. This architecture considers four major components, which are the core business XMPP server, the web application for access the SaaS, the storage and the CloudMed PACS gateway.

The XMPP server is the CloudMed system core. Therefore, it is responsible to support all the core business layer of the infrastructure. The server runs in a Cloud provider infrastructure. Furthermore, the server supplies a SaaS, which provides a XMPP message-oriented middleware to allow the access through the Internet to the entire system.

To access the SaaS, provided by CloudMed infrastructure, it was created a Web application, developed according with Web 2.0 paradigm. This application acts as a XMPP client in the XMPP network. Thus, using a message-oriented middleware, users can interact with colleagues and manage their resources, which can be files stored on personal repository or data from a given PACS server that they own.

The access to PACS servers is done using gateways. These gateways are installed inside a health care institution, on a workstation that can connect to the target PACS server. Also these gateways are seen as a component in the XMPP network. Thus, it uses the middleware to interact with the core business server and with the Web application.

Finally, there is the storage component that is used by the system to store medical images files in the Cloud. This element is an abstract component. Indeed, to be able to implement the CloudMed system using any Cloud storage service, it was necessary to create some abstraction in the definition of the storage component. This abstraction lead to the creation of a well defined interface that can be implemented to use any Cloud storage provider that most fits into the system needs. Therefore, by using a well defined interface and plug-in based system, it is possible to add any new component that implements the CloudMed storage interface. Following this approach, the system decouples computation infrastructure from the storage infrastructure.

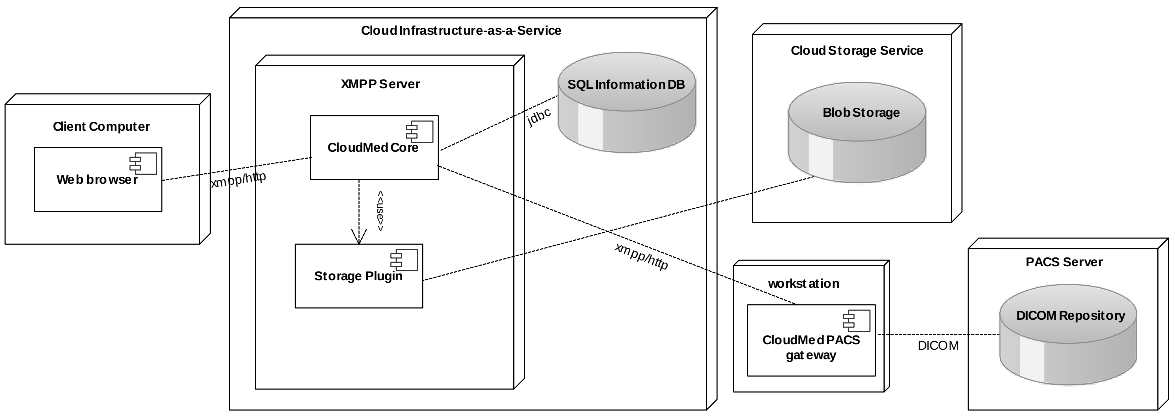


Figure 3.14: System architecture diagram.

Chapter 4

System Implementation and Results

The choices made while implementing the system were very important to the success in the final result. Some options were made to tackle specific engineering problems like, for instance, enable access to the system even inside very restrictive firewalls that allow only communication to HTTP protocol on port 80, etc. This chapter explores bases for the implementation of the underling system, the problems found related with building it and the engineered solutions that led to a successful system.

4.1 CloudMed SaaS - Computer Supported Social Network for Teleradiologia

Focus on the implementation of the desired architecture, it is possible to see that all business logic and persistence storage is held in the Cloud, thus the implementation of this component was a critical stage for the system's realization.

In CloudMed system implementation, it was very important to avoid any dependency associated to any operating system or any Cloud provider. Consequently, its core has to be able to run in any platform. Hence, there was no doubt that Java programming language is the right choice to develop the system. Furthermore, by using Java to develop the system, it becomes possible to deploy the core in any cloud provider's IaaS, independently if they provide an IaaS with Windows operative system or Unix based operative system.

In the modelling stage, was defined that it should use a message-oriented middleware to connect all components. Furthermore, was chosen the open standard XMPP to support its message-oriented middleware. Thus, it was necessary to use a XMPP server in the core of the CloudMed system. Moreover, it was not even considered to implement a XMPP server from begin, because there are a lot of open source XMPP server [38] already implemented. Thus, it was necessary to choose the one that fits well to the system requirements. The choice fell on the Openfire server [39], which is developed using Java programming language and allows programmers to extend its capabilities with new plug-ins and is well supported by the community.

4.1.1 Message-Oriented Middlware eXtensible CloudMed Communication Protocol - XCMCP

The CloudMed system uses a message-oriented middleware, so it is possible to define some custom messages to handle specific actions. Taking advantage of XMPP based message-oriented middleware extensibility, it was created a group of messages that defines a custom protocol, named eXtensible CloudMed Communication Protocol (XCMCP). This group of messages represents custom extensions to the XMPP protocol, which extends XML stanzas of <**iq**\> and <**message**\> with payloads containing specifics namespaces and elements' name.

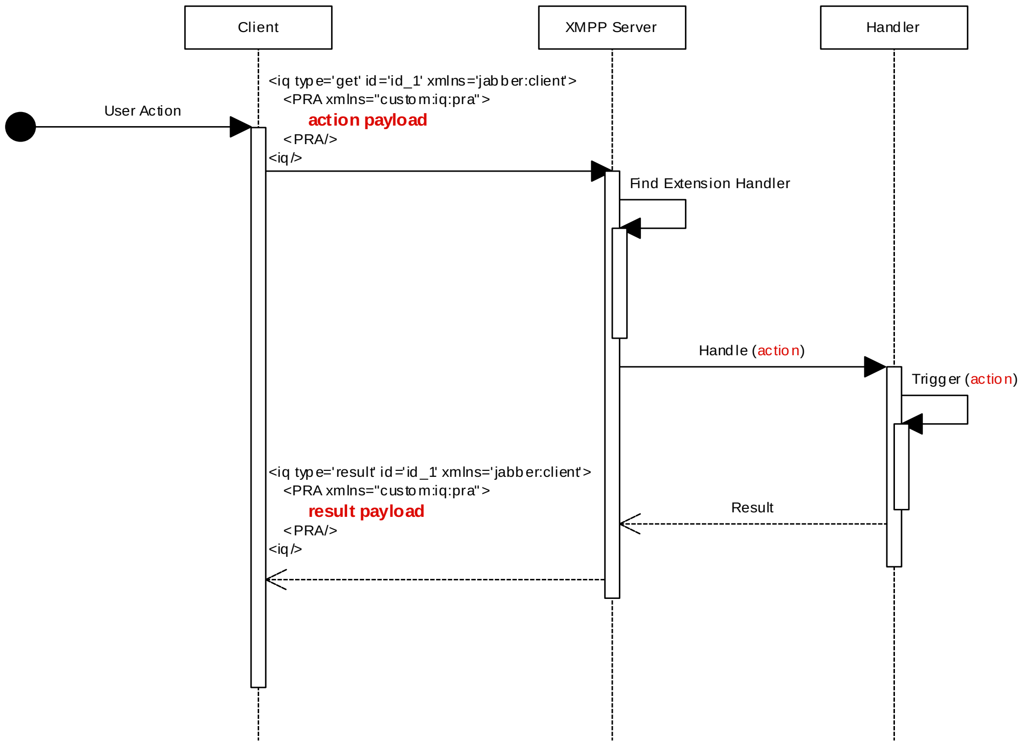
To handle these custom extensions, the entities in the XMPP network have to understand the meaning of those new extensions in order to be able to act accordingly. The entities have to implement some <**iq**\> handlers and <**message**\> handlers to support these specifics extensions and trigger action according to XML stanza's payload extensions.

The functionalities introduced with the XCMCP extensions can be divided in four groups:

- Manage personal remote repository
- Manage packet filter's rules
- Manage remote interaction with PACS gateways
- Manage proxy's anonymization

Manage personal remote repository

This functionality allows clients to control their repositories. This management is done using custom XMPP <**iq**\> stanzas, in which is provided an action to be executed and some parameters as its payload. To identify this XMPP extension, it was created a custom <**iq**\>, which starts with the element tag name ***PRA*** and uses the namespace ***custom:iq:pra***. So, when the server receives an <**iq**\> that belongs to this extension, it triggers the handler responsible to this custom XMPP extension, and the IQ handler executes the action, taking in consideration some parameters provided as element child of the payload, and returns a reply depending on the action and it execution result. The action execution flow is represented in the Figure 4.1.



```
<iq type='get' id='id_1' xmlns='jabber:client'>
   <PRA xmlns="custom:iq:pra">
        action payload
   <PRA/>
<iq/>
```

```
<iq type='result' id='id_1' xmlns='jabber:client'>
   <PRA xmlns="custom:iq:pra">
        result payload
   <PRA/>
<iq/>
```

Figure 4.1: Remote repository manage diagram.

There was a set of defined actions that can be performed:

- **Search**

  This action allows searching for studies in a given repository. In the action's parameters can be supplied the institution name, patient name, patient id, study instance id, series instance id or file name to be considered in the search.

  Also must be provided in all search actions the **level** parameter that identifies in which level must be performed the search. For instance, if an user is searching for every medical image from a given institution he must provide the **level**=*"IN"* and the institution name, but if he wants to search for all medical images from a given patient's series, he must provide de **level**=*"SE"*, the institution name, the patient name, the patient id, the study instance id, and the series instance id (Listing 4.1).

Listing 4.1: Search action's payload

```
<search in='Univ.Kl.AKH.Wien' pn='MR' pi='LSPINE'
              st='TSE' se='512' level='SE'/>
```

As result of a **seach** action, the server returns an IQ-result with a payload (e.g Listing 4.2) containing the list of files that satisfy the search based on the provided parameters.

Listing 4.2: Result's payload

```
<files>
        <file owner='eriksson@ieeta.ua.pt'
                  fname='UCLA_300_MP/PNEUMATIX/NOID/1.2.840.1137...'/>
        <file owner='eriksson@ieeta.ua.pt'
                  fname='UCLA_300_MP/PNEUMATIX/NOID/1.2.840.1137...'/>
        <file owner='admin@ieeta.ua.pt'
                  fname='UCLA_300_MP/PNEUMATIX/NOID/1.2.840.1137...'/>

         ...

        <file owner='admin@ieeta.ua.pt'
                  fname='UCLA_300_MP/PNEUMATIX/NOID/1.2.840.1137...'/>

</files>
```

- **List shared files**

  This action (Listing 4.3) returns the list of all files that are currently shared with the user that executes this action. The result is a list of <**file**\> with the attributes *fname* and *owner* that represent, respectively, the name of this file and its owner (e.g Listing 4.2).

Listing 4.3: List shared files action's payload

```
<listSharedFiles/>
```

- **List User's Files**

  This action (Listing 4.4) returns the list of all files that belong to the user that is executing this action (Listing 4.5).

Listing 4.4: Action's payload

```
<listMyFiles/>
```

Listing 4.5: Result's payload

```
<files>
        <file owner='eriksson@ieeta.ua.pt'
                  fname='UCLA_300_MP/PNEUMATIX/NOID/1.2.840.1137...'/>
         ...
        <file owner='eriksson@ieeta.ua.pt'
```

```
                         fname='UCLA_300_MP/PNEUMATIX/NOID/1.2.840.1137...'/>
</files>
```

- **Show object shares**

  An user can share an object with some colleagues. Here object can be a folder that represents an institution, a patient, a study or a series. Associated with each object there is a list that identifies who can access it. This list is the share's list. When an user executes this action, it is returned a list with all users that belong to the share's list of an object (Listing 4.7).

  To execute the action must be provided the object identification. This identification is made by supplying the object **level**, which can be *IN*, *PN*, *ST* or *SE*, depending on whether the object is, respectively, an institution, a patient, a study or a series folder. Besides the *level*, must be provided also the institution name, the patient name and id, the study instance id and the series instance id, depending on the level selected.

Listing 4.6: Action's payload

```
<IsSharedWith level='SE' in='UCLA_MP300' pn='FELIX' pid='7DfDKDK'
 st='1.2.840.113745.101000.1008000.38446.6272.7138759'
 se='1.3.12.2.1107.5.2.13.20561.30000005042216091690600002608'/>
```

Listing 4.7: Result's payload

```
<users>
        <user name='carlos@ieeta.ua.pt'/>
        <user name='luis@ieeta.ua.pt'/>
</users>
```

- **Delete file**

  This action (Listing 4.8) allows user to remove an object from his repository. The object identification is made in the same way as described above. If an user wants to remove a given object from his repository he needs to identify this object in the remove action's parameters.

  There is no specific result. If the server responds with an IQ-result the action was successfully executed. If the server responds with an IQ-error the remove did not occur.

Listing 4.8: Action's payload

```
<delete level='SE' in='UCLA_MP300' pn='FELIX' pid='7DfDKDK'
 st='1.2.840.113745.101000.1008000.38446.6272.7138759'
 se='1.3.12.2.1107.5.2.13.20561.30000005042216091690600002608'/>
```

- **Set store permission**

  The data storage is made out-of-band. In other words, the CloudMed system does not use XMPP to send medical images to be stored in the Cloud because XMPP is not optimized to handle binary data. Thus, to control the out-of-band data storage, in an user's Cloud repository, it is necessary to request store permission before store data in the repository. If there is a try to store data that are not expected, or do not have store permission, the action is denied.

  An user can only set store permission to his repository, but he can get other users' permission to store in their repository.

  The store permission action (Listing 4.9) must be invoked providing a **level** parameter that can be set as **study** or **series**, to identify that the given studies or series (with the ids provided in the **ids** parameter) can be stored in the repository. Additionally, there is the parameter **allowuser** that identifies which user can store the required studies or series.

  This action does not require any result.

Listing 4.9: Action's payload

```
<storePermission allowuser='eriksson@ieeta.ua.pt' level='study'
ids='1.2.840.113745.101000.1008000.38446.6272.7138759,
1.2.840.113745.101000.1008000.38446.6272.7138760,
...,
1.2.840.113745.101000.1008000.38446.6272.7138769'/>
```

- **Share object**

  The share action (Listing 4.10) allows the user to share an object with other users. To perform this action the user must identify the object to be shared, as described above. In addition, he must provide a list of users, using the **sharewith** parameter, to be added to the object share's list.

Listing 4.10: Action's payload

```
<share>
        <item shareWith='eriksson@ieeta.ua.pt' level='SE'
        in='UCLA_MP300' pn='FELIX' pid='7DfDKDK'
        st='1.2.840.113745.101000.1008000.38446.6272.7138759'
```

```
          se='1.3.12.2.1107.5.2.13.20561.30000005042216091690600002608'/>
          ...
          <item shareWith='admin@ieeta.ua.pt' level='SE'
          in='UCLA_MP300' pn='FELIX' pid='7DfDKDK'
          st='1.2.840.113745.101000.1008000.38446.6272.7138759'
          se='1.3.12.2.1107.5.2.13.20561.30000005042216091690600002618'/>
<share/>
```

- **Remove share**

  As like as the share action, this action also requires the parameters to identify the object that will be the target and also a list of users that will be removed from this object's share list. This action's payload is almost equal to the payload associated with the share object action (Listing 4.10), but instead of starting with element name <**share**> it starts with element name <**unshare**>.

Manage firewall rules

As like the extension defined in the previous topic, this is also based in custom <**iq\**>, identified by the element tag name ***FIREWALL*** and namespace ***custom:iq:firewall***. This custom <**iq\**> is used to trigger actions in order to allow an user to manage the access to his PACS server through a gateway.

The possible actions triggered with this custom extension are listed below:

- **Get a gateway ACL**

  This action can be invoked by sending a <**iq\**> with a payload identifying the target gateway (Listing 4.11). It will be returned a copy of its access list, where is presented the users that can access the gateway.

Listing 4.11: Action's payload

```
<getACL proxy='HP_PACS'/>
```

- **Deny access**

  This action allows an user to deny, to a group of users, the access to a given gateway. As represented in Listing 4.12, must be provided a list of rules where is identified the CloudMed PACS gateway by the attribute ***proxy*** and the user to be blocked by the attribute ***denyUser***.

Listing 4.12: Action's payload

```
<denyAccess/>
          <rule proxy='HP_PACS' denyuser='eriksson@ieet.ua.pt'/>
```

```
        <rule proxy='SIEMENS_PACS' denyuser='eriksson@ieet.ua.pt'/>
        <rule proxy='HP_PACS' denyuser='admin@ieet.ua.pt'/>
<denyAccess/>
```

- **Allow access**

  To allow a list of user to access a given gateway, must be sent a message where is provided the gateways and the users to grant access permission.

Listing 4.13: Action's payload

```
<allowAccess/>
        <rule proxy='HP_PACS' allowuser='eriksson@ieet.ua.pt'/>
        <rule proxy='SIEMENS_PACS' allowuser='eriksson@ieet.ua.pt'/>
        <rule proxy='HP_PACS' allowuser='admin@ieet.ua.pt'/>
<allowAccess/>
```

Manage Proxy's Anonymization

Some concerns related with medical data privacy usually create some management limitations. Consequently, the system had to create some features to avoid undesired situations and enable users to use the service provided by this system without concern of violating any privacy rule. One way found to tackle this problem was to introduce an anonymization tool that allows users to define that a given gateway must anonymize files that it sends to cloud as result of a Query/Retrieve. Moreover in CloudMed system, when users share studies there are some implicit constraints that impose some limitations about those medical images access. This limitations consists in not allowing users to share files that was shared with him with other colleagues. Furthermore the system does not provide tools for direct download of shared images.

To indicate the gateways that they must anonymize all files they send to a given user is used the XMPP Private XML Storage extension [35], which is a standard XMPP extension identified by a <**query**\> child scoped by the 'jabber:iq:private' namespace.

- **Set Anonymization Rules**

  This action allow users to indicate, for a given gateway, if it must or not anonymize files consumed by a third user. It uses the XML Storage extension to sent an IQ-set with an XML containing this information.

  To store information about anonymization, for a given proxy, is used an payload to identify the gateway, where the element name is the gateway name and the child element is scoped by the 'custom:proxy:anon' namespace (Listing 4.14).

Listing 4.14: Messages exchanged

```
CLIENT:
<iq type='set' id='7594:saveXML' xmlns='jabber:client'>
   <query xmlns='jabber:iq:private'>
      <HPPACS xmlns='custom:proxy:anon'>
         <users>
            <user name='carlos.costa@ieeta.ua.pt' anon='true'/>
            <user name='luis@ieeta.ua.pt' anon='false'/>
            <user name='samuel@ieeta.ua.pt' anon='false'/>
            <user name='sergiomatos@ieeta.ua.pt' anon='true'/>
         </users>
      </HPPACS>
   </query>
</iq>

SERVER:
<iq type="result"
    to='bastiao@ieeta.ua.pt/webclient'
    id="7594:saveXML"
    xmlns='jabber:client'/>
```

- **List Anonymization Rules**
  This action allows a gateway to get information about defined anonymization rules. To get its specific anonymization rules, a gateway has to send an IQ-get with a payload where the element name is the gateway name and the child element is scoped by the 'custom:proxy:anon' namespace (Listing 4.15).

Listing 4.15: Messages exchanged

```
CLIENT:
<iq type='get' id='7597:loadXML' xmlns='jabber:client'>
   <query xmlns='jabber:iq:private'>
      <HPPACS xmlns='custom:proxy:anon'/>
   </query>
</iq>

SERVER:
<iq xmlns='jabber:client' to='bastiao@ieeta.ua.pt/webclient'
   type='result' id='7597:loadXML'>
   <query xmlns='jabber:iq:private'>
      <HPPACS xmlns='custom:proxy:anon'>
         <users>
            <user anon='true' name='carlos.costa@ieeta.ua.pt'/>
            <user anon='false' name='luis@ieeta.ua.pt'/>
            <user anon='false' name='samuel@ieeta.ua.pt'/>
            <user anon='true' name='sergiomatos@ieeta.ua.pt'/>
         </users>
      </HPPACS>
   </query>
</iq>
```

Manage remote interaction with PACS gateways

Unlike the extensions referenced above that represent new features in the client-to-server interaction, this extension adds new functionalities in the client-to-client communication. It is responsible for introduce new messages in the protocol layer, which can be used by users to manage and interact with their remote CloudMed PACS gateways.

This extension is based on <**message**\> XML stanza. The extension is identified by a <**XMPPCloudMed**\> child element with **http://www.ieeta.ua.pt/dicomproxy** as it namespace.

Moreover, all extensions' payloads contain the same parameters (Table 4.1):

Parameter	Description
Id	It identifies the message. It should be unique in a session.
Type	It identifies the kind of message. It can take the fallowing values: **QUERY** to identify a query message; **QR** to indicate that this message should be transformed in a DICOM Query/Retrieve; **SYNCH** is used to order this gateway to synchronize a given folder from the Cloud with the workstation where it is installed (it can only be executed by the owner of this gateway).
Status	It is used to identify the success or failure of a given request. This parameter takes the value **OK** to identify a reply that was successfully executed. It takes the value **ERROR** to identify an failure in the execution of a requested action.
Data	This parameter can contain any kind of text. It is used to transport formatted extra parameters for a request or formatted results in a reply message.

Table 4.1: Message format for manage remote PACS gateways.

This extension adds new functionalities to the message-oriented middleware that allow users to execute the fallowing actions in their PACS gateway.

- **Do queries**

  This action is used to send a query to a given PACS gateway. In the parameter **Data** is sent some extra formatted text that contains the parameters to be used to create the DICOM C-FIND command.

- **Retrieve from PACS server**

  This action allows the user to retrieve medical image from a PACS archive server, through the PACS gateway, and to store them in the Cloud personal repository. It also uses the parameter **Data** to send the parameters used in the construction of the DICOM C-FIND and C-MOVE commands.

- **Synchronize data** This action is used to retrieve medical images from a given folder, from a Cloud repository, and synchronize it with a folder in the workstation where the gateway is installed. In the **Data** parameter is sent the information that identifies the folder to be synchronized with this workstation.

4.1.2 XMPP Server - Openfire

To handle the message-oriented middleware, the system uses the Openfire XMPP server, which is easy to setup and administer, but offers solid security and performance. Moreover, one of the most important characteristics of Openfire server, that led to it usage as CloudMed core server is because this server has a well designed implementation, which allows developers to easily extend its functionalities using plug-ins.

Plug-in system

Openfire server provides a well defined interface that allows developers to implement new plug-ins in a simple way. Thus, to implement all the CloudMed core business layer of our system, it was necessary to develop a plug-in that runs in Openfire server.

To create a plug-in for Openfire, the developers need to implement a plug-in initializer class that implements *org.jivesoftware.openfire.container.Plugin*. This class is responsible to implement the methods initializePlugin and destroyPlugin, which make, respectively, the plug-in code initialization and clears the plug-in on destroy (Listing 4.16).

Listing 4.16: Class that initializes a Plug-in

```
package pt.ieeta.cloudmed.openfireplugin;
import org.jivesoftware.openfire.container.Plugin;
public class CloudMedOpenfirePlugin implements Plugin {

    public void initializePlugin(PluginManager manager,
                                 File pluginDirectory) {
        //initialize code
    }

    public void destroyPlugin() {
        //destroy code
    }
}
```

In addition, to load the plug-in is necessary to identify the class, among others in the *.jar* file, which is responsible to make its load. To do this, developers should place in the plug-in's *.jar* root a file named plugin.xml. Openfire plug-in loader reads this file and identifies the class needed to load the plug-in and execute the method initializePlugin (Listing 4.17) of this class.

Listing 4.17: Plug-in configuration

```
<plugin>
  <!-- Main plugin class -->
  <class>pt.ieeta.cloudmed.openfireplugin.CloudMedOpenfirePlugin
  </class>

  <!-- Plugin meta-data -->
  <name>CloudMedOpenfirePlugin</name>
  <description>Interact with Cloud providers</description>
  <author>eriksson.monteiro</author>
  <version>1.0</version>
  <minServerVersion>3.3.0</minServerVersion>
  <licenseType>gpl</licenseType>
</plugin>
```

4.1.3 Personal Remote Archive - PRA

Cloud repositories are nowadays very common in users' daily activity. Currently, users are using services like Dropbox, Google Drive, Sky Drive, etc, to store their files in the Cloud, and thus, being able to access those files from any-where.

Using the same concept introduced by some Cloud file storage provider, the CloudMed system also provides a mechanism that allows storage of medical images in the Cloud and sharing functionalities. Moreover, the users have the notion of personal repository, which they can use to store files to be accessed from any-where using workstations or mobile devices. The repository is managed through the storage plug-in that is plugged in the core CloudMedOpenfire plug-in in order to provide an input/output stream to write or read files from any Cloud provider, according to the plug-in implementation.

In CloudMed system, each user owns one repository, which can contain several folders, i.e studies. The user can share his folders with other users. By providing access to others, they became able to visualize or manage, according with shared permission. Each folder can contain many medical images, stored in one Cloud blob storage provider. The relation between users an repositories is represented in the Figure 4.2.

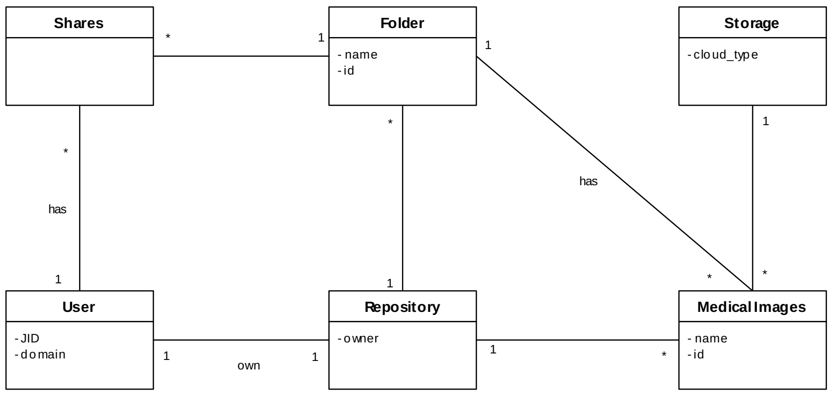


Figure 4.2: Personal Remote Archive diagram shows the relation between users and repositories in the cloud.

4.1.4 System Core Organization

The system core is an important component that is deployed as an Openfire plug-in. This component brings all the semantic to the system. It enables the server to interpret the custom functionalities build on top of XMPP protocol, named XCMCP (section 4.1.1) and has also the task to load the plug-in that handles the storage.

All the system logic lays on the XMPP message-oriented middleware, thus the core system implements some handles to deal with the middleware, in order to support the communication protocol and enable the system to work properly.

The system core has 3 main components:

- **Storage**

  A storage component that is responsible to manage the access to a given Cloud blob storage. This storage component is used to store, in a Cloud provider, all medical images held by this system.

- **Handler Register**

  The Handler Register is the component that registers some handlers to deal with each of the new XCMCP extensions, builded over XMPP protocol.

- **Upload/Download servlet**

  This component is used to enable the out-of-band upload and download of medical images.

Storage

The storage component is also an Openfire plug-in. However this plug-in has a specific parent, which is the core CloudMed Openfire plug-in.

This plug-in is the component that enables the core system to be decoupled from any Cloud Storage provider. This decoupling is possible because the core system defines an interface (Listing 4.18) that should be implemented by any storage plug-in in order to be attached to the core system, thus allowing the core system to access a specific Cloud blob storage.

Listing 4.18: Storage interface

```
/**
 *
 * @author Eriksson Monteiro <eriksson.monteiro@ua.pt>
 */
public interface ArchiveStream {
        public void writeFile(String owner, DicomTreeInfo dicomTree,
          String id,byte[] in, boolean append) throws IOException;
        public void writeFile(String owner, DicomTreeInfo dicomTree,
          String id, InputStream in) throws IOException;
        public InputStream readFile(String owner, String id)
                throws IOException;
        public boolean removeFile(String owner, String id)
                throws IOException;
        public boolean removeFile(String owner, DicomInfo info)
                throws IOException;
        public boolean exists(String owner, String id)
                throws IOException;
        public List<String> listDirectory(String owner,
                DicomFileFilter filter)throws IOException;
}
```

In order to be available, the storage plug-in must be added to an archive manager (Figure 4.3) in the core system. This manager allows the system access to a specific storage plug-in, identified by its class name. It is possible to plug, to the core system, more than one storage plug-in. Thus, there may have some flexibility to choose one storage plug-in to use. Nevertheless, there is a default storage plug-in that is the first one plugged into the core system.

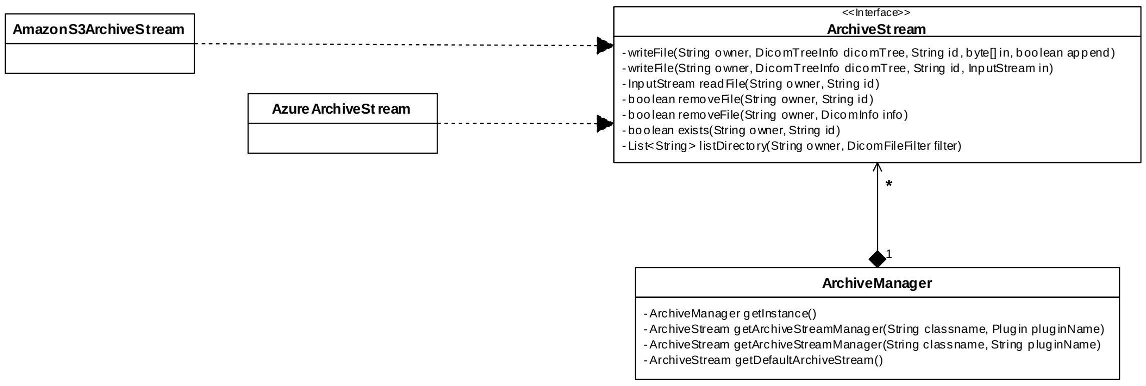

AmazonS3ArchiveStream
AzureArchiveStream
<<Interface>>
ArchiveStream
- writeFile(String owner, DicomTreeInfo dicomTree, String id, byte[] in, boolean append)
- writeFile(String owner, DicomTreeInfo dicomTree, String id, InputStream in)
- InputStream readFile(String owner, String id)
- boolean removeFile(String owner, String id)
- boolean removeFile(String owner, DicomInfo info)
- boolean exists(String owner, String id)
- List<String> listDirectory(String owner, DicomFileFilter filter)
*
1
ArchiveManager
- ArchiveManager getInstance()
- ArchiveStream getArchiveStreamManager(String classname, Plugin pluginName)
- ArchiveStream getArchiveStreamManager(String classname, String pluginName)
- ArchiveStream getDefaultArchiveStream()

Figure 4.3: Archive manager diagram.

For first implementation tests, it was created a plug-in that writes the files directly to the file-system in the HP Cloud IaaS. Later, it was created a plug-in that uses the Amazon S3 cloud storage to store the digital medical images in that Cloud provider (Figure 4.4). For security and privacy reasons, the Amazon S3 remote archive plug-in encrypts the files before send them to the Cloud provider infrastructure and decrypts them when an user wants to consume them.

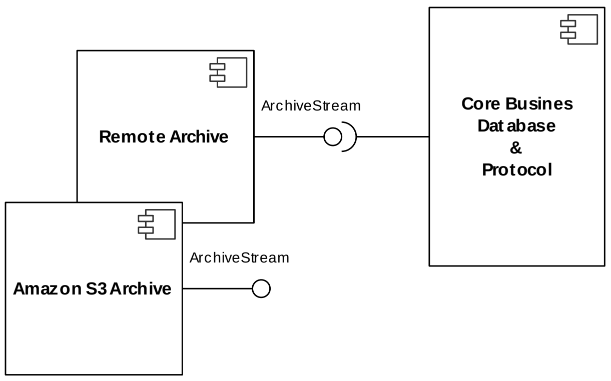


Figure 4.4: Remote Archive plugins implementation.

Handle register

The core system implements a handler register that instantiates some handlers to deal with the XCMCP extensions of XMPP. The XCMCP extensions group defines two client-to-server extensions, which are the extension to manage personal remote archive and to manage packet filter's rules. Thus, to handle each of these extensions, it was implemented some specifics handlers. Each handler is able to understand the respective extension and acts accordingly to message it received.

Furthermore, all the two client-to-server extensions, handled by the Openfire core plug-in, are based in IQ XMPP stanzas. Thus, to implement the handlers, to deal with some specifics IQ, were created some classes that extend the Openfire server SDK class IQHandler and implement the method *handleIQ (IQ packet)*. In the implementation of the method *handleIQ (IQ packet)* the handler analyses the packet received and triggers an method to execute an action, depending on the content of the packet that was received (Figure 4.5).

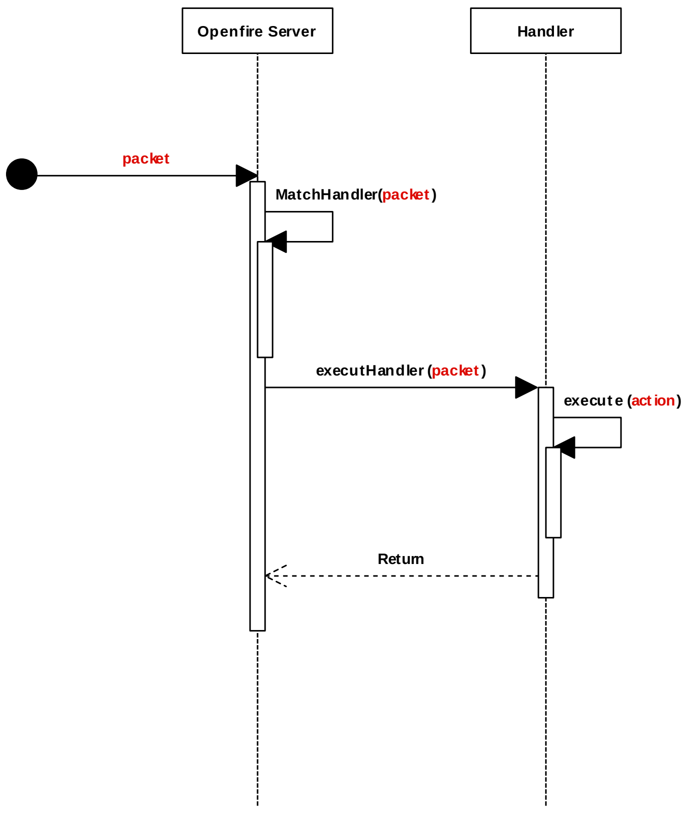


Figure 4.5: Extension packet handle diagram.

Upload/Download servlet

The middleware used by CloudMed system reveals some limitation dealing with binary data. XMPP message-oriented middleware is quite inefficient to handle in-band binary data transfer. This limitation is due to the fact that XMPP connection is encoded as a long-lived XML stream, thus to transfer binary data is necessary to encode it to Base64 before it can be transmitted in-band. The usage of Base64 [40] is not efficient to transfer large amount of data because number of output bytes per input byte is approximately 4 / 3 (33% overhead).

To bypass this XMPP limitation, transferring binary data, there are some methods that consist in sending the data directly from one entity to another over a connection that has no intermediaries and making the signalization in-band using XMPP. For instance, using SOCKS5 specified in SOCKS5 Bytestreams [XEP-0065] [41] or using Jingle RTP session [XEP-0167] [42] to establish video and voice connection. Nevertheless, in the proposed system the digital medical images' data are sent via HTTP POST to the server and also there is an in-band signalization using XMPP middleware to inform the server that one client is sending a given image to be stored in the Cloud.

The system provides a servlet to enable the out-of-band upload of digital medical images to the Cloud. Obviously, to upload a file to a given user's personal repository in the Cloud there are some security concerns that were taken in account. In CloudMed system, each user has his personal repository in the Cloud, thus only he can upload files to his repository or provide permission to a friend's CloudMed PACS gateway upload files to his repository.

There is an in-band signalling mechanism that provides a list of permitted studies or series that an user is expecting from a friend's CloudMed PACS gateway. This features uses XCMCP messages (section 4.1.1) to provide the desired information to the system core.

When there is a Query/Retrieve of medical images, from a friend's CloudMed PACS gateway, it is necessary to execute some steps before the files can be stored in the user's personal repository. These steps are represented in the following diagram (Figure 4.6):

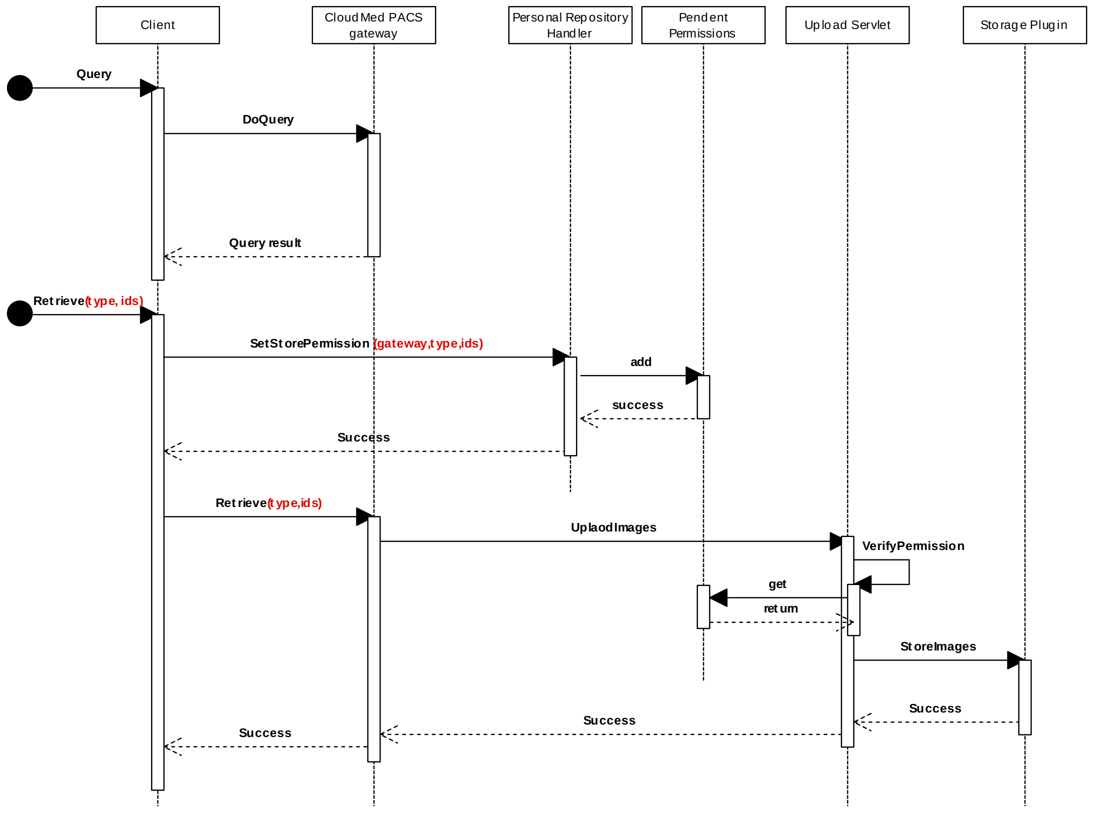


Figure 4.6: Uploading files to Cloud storage.

As referred before, the upload is made using an out-of-band HTTP POST. Thus in the POST method the client must provide information about which user's personal repository the file must be stored at. The JID of the user sending the file and an hash is used to verify the authenticity (Figure 4.7) of this POST.

To authenticate files that users send to Cloud, the system uses SHA1 HMAC function [43]. The system core uses the HMAC to verify if the file was uploaded by an user with write permission in the desired repository. Before make the upload, the client must calculate a SHA1 HMAC of the data using the user's password. Thereby, the system is able to verify the HTTP POST authenticity because the password is shared between the client and the server [44].

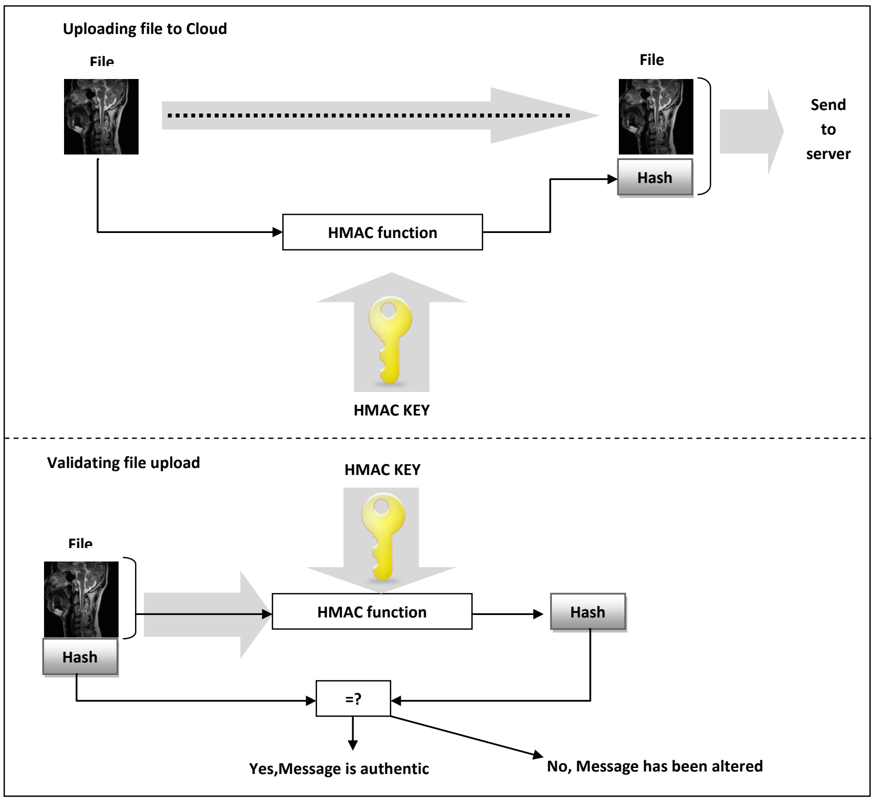


Figure 4.7: File upload authentication.

In addition, to download a file from a personal repository, the system also requires the authentication of the HTTP GET method. The authentication is made by sending, in the **hmac** parameter of the query string the SHA1 HMAC of the name of the file that is being downloaded, created using the user password.

To download a file also it is necessary to send the parameters **jid** and **owner**. The parameter **jid** identifies the user that is executing this action, thus the **hmac** parameter is calculated using his password. And to choose the **owner** of the repository, from where is intended to be downloaded the image, is used the **owner** parameter. If the **owner** parameter is equal to **jid** parameter, then it is a request to download a file sent by the repository owner (Figure 4.8). Otherwise, it is an access to a shared repository, thus it is necessary to verify if the user has permission to access the repository before permit him to download images (Figure 4.9).

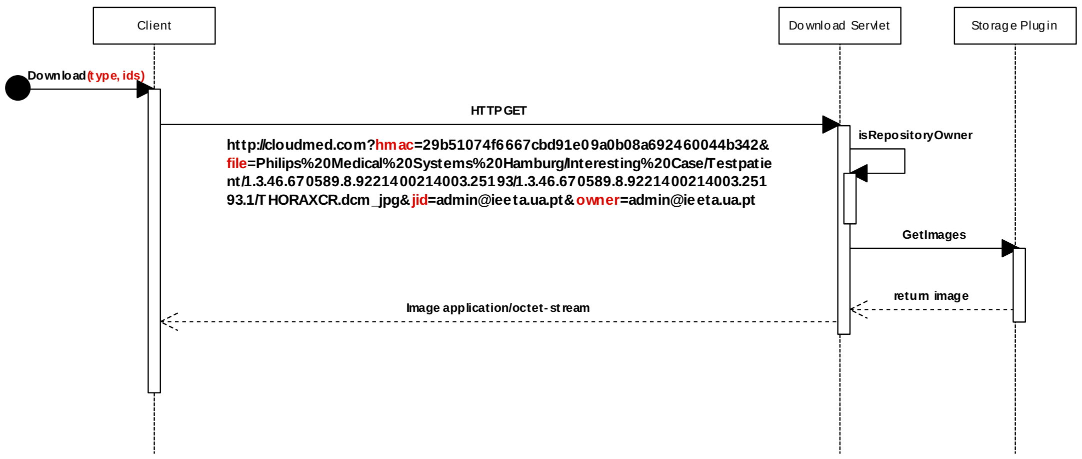

Client
Download Servlet
Storage Plugin
Download(type, ids)
HTTP GET
http://cloudmed.com?hmac=29b51074f6667cbd91e09a0b08a692460044b342&file=Philips%20Medical%20Systems%20Hamburg/Interesting%20Case/Testpatient/1.3.46.670589.8.92214002140003.25193/1.3.46.670589.8.92214002140003.25193.1/THORAXCR.dcm_jpg&jid=admin@ieeta.ua.pt&owner=admin@ieeta.ua.pt
isRepositoryOwner
GetImages
return image
Image application/octet-stream

Figure 4.8: Download file diagram represents the workflow when a client download a file from his repository.

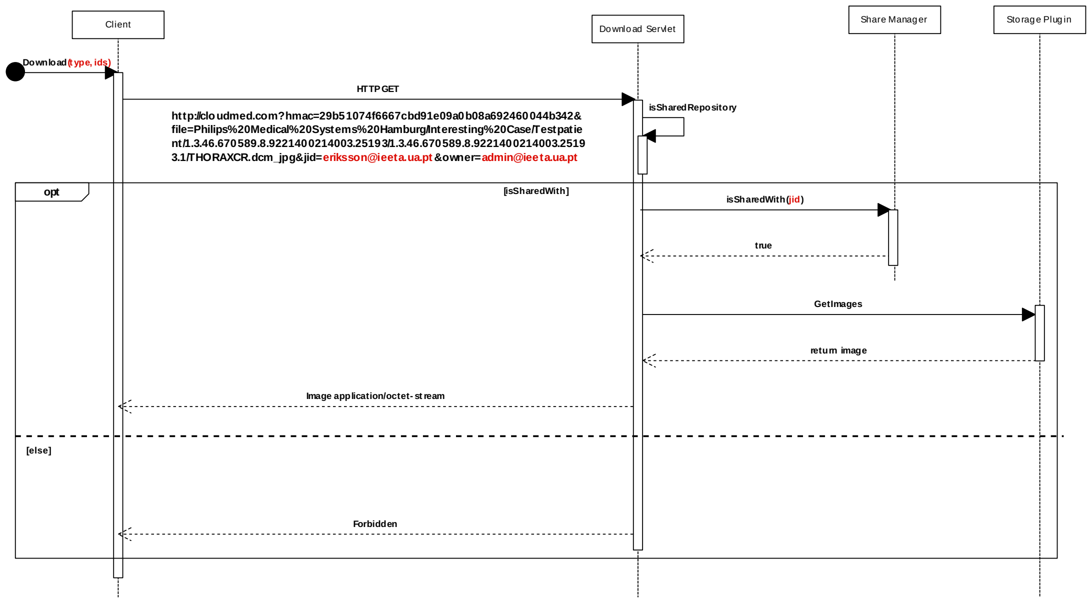

Client
Download Servlet
Share Manager
Storage Plugin
Download(type, ids)
HTTP GET
http://cloudmed.com?hmac=29b51074f6667cbd91e09a0b08a692460044b342&file=Philips%20Medical%20Systems%20Hamburg/Interesting%20Case/Testpatient/1.3.46.670589.8.92214002140003.25193/1.3.46.670589.8.92214002140003.25193.1/THORAXCR.dcm_jpg&jid=eriksson@ieeta.ua.pt&owner=admin@ieeta.ua.pt
isSharedRepository
opt
[isSharedWith]
isSharedWith(jid)
true
GetImages
return image
Image application/octet-stream
[else]
Forbidden

Figure 4.9: Download share file diagram represents the workflow when a client download a file from a repository that was shared with him.

Moreover, the communication channel between clients and the upload/download servlet is secure. To download a file from a repository or upload a file to a repository, the client must establish one HTTPS connection with the server, to be able to access the servlet. Therefore enhance the communication channel avoids eavesdrop attacks, and unprivileged users access to repositories.

4.2 Medical Management Information System Gateway - DICOM Gateway

The gateway is a very important component in the system architecture because it is the bridge that enables the communications between the XMPP network and a health care institution PACS infrastructure. The MMIS (Medical Management Information System) is composed by two parts, where one acts as a XMPP client and creates an interface that allows the gateway to connect with the XMPP network, and the other part implements some features that enables communicate with the PACS architecture, using DICOM protocol.

To communicate with a PACS archive server, the MMIS uses the DICOM dcm4che2 library, which provides all functionalities to query and retrieve information from a PACS server using DICOM protocol.

On other hand, the MMIS uses the Smack API library to create a XMPP client. By acting as a XMPP client, it was necessary to implement some handlers to handle some specifics <**iq**\> to be able to understand the XCMCP features.

The way how MMIS allows XMPP Cloud clients to interact with PACS archive server is based in a translation message mechanism. It works based on a XMPP packet listener that process every packet received by the gateway. After receiving a packet, the gateway analyses what kind of action this massage pretends to invoke and, according to the action and parameters provided, the gateway translates it into a DICOM command.

Moreover, the gateway allows the users to synchronize their remote repository with the workstation where the gateway is running. To do so, the users must send a specific <**iq**\>, like defined in the XCMCP, where explicitly informs about folder should be synchronized. After receive this <**iq**\>, the gateway proceeds with the download of all the files that belong to the folder and stores them into a specific workstation folder.

4.2.1 CloudMed PACS Gateway Architecture

This component acts as an interface between XMPP Cloud network and the DICOM universe. Thus, the system implementation is builded over two main components (Figure 4.10) that provide interface with each network:

- **DCM4CHE**

  This component provides an API with some tools used in the creation of DICOM application, which is able to interact with the PACS network.

- **SMACK**

  This library provides an API that can be used to implement a XMPP client to interact with the XMPP network.

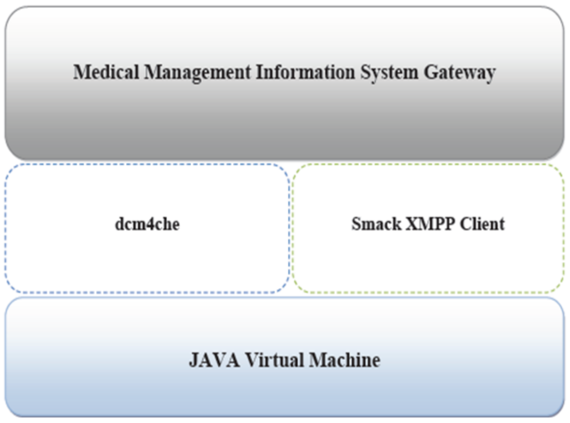


Figure 4.10: CloudMed PACS gateway components diagram.

4.2.2 XMPP Interface

The gateway has a Web compliant interface that uses the HTTP protocol to create a XMPP stream. But, HTTP protocol does not provide bidirectional streams. Thus, it was used BOSH protocol defined by “XEP-0124: Bidirectional-streams Over Synchronous HTTP” [45] and “XEP-0206: XMPP Over BOSH” [46]. BOSH protocol uses HTTP POST method and long pooling to empower the XMPP stanza streaming. This protocol and XMPP client was implemented using SMACK API, which had to be patched with some features to enable the usage of BOSH protocol in the connection between clients and the server. Moreover, to enable CloudMed PACS gateway to interpret and react when it receives a custom XMPP stanza, it was implemented some messages listeners and packet providers. The following class diagram (Figure 4.11) represents the entities that enable the gateway to communicate with the XMPP network.

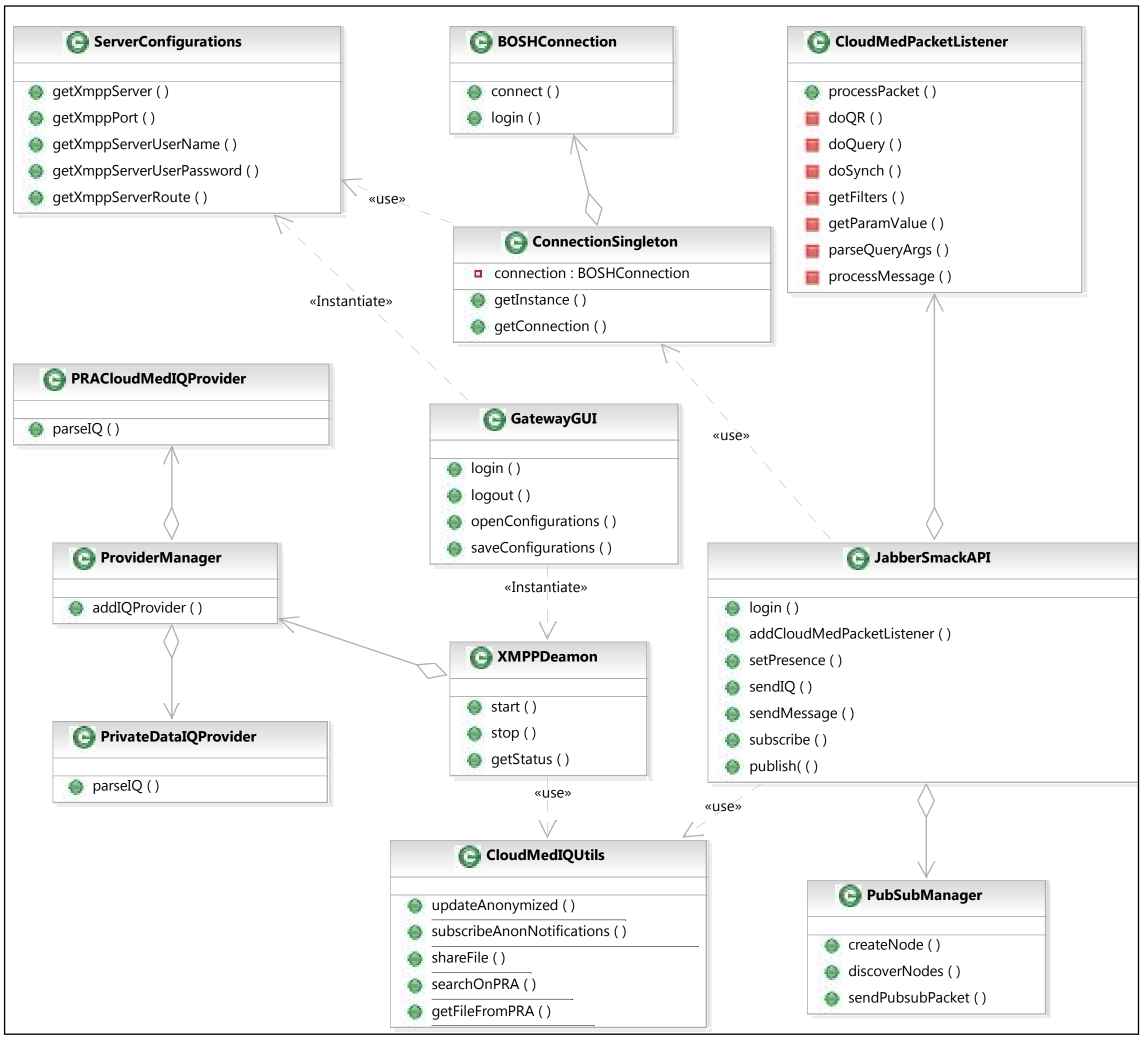


Figure 4.11: This class diagram shows the entities that interact with the XMPP layer.

JabberSmackAPI is the XMPP client entity that implements several useful features like, for example, to login, to send XMPP messages, to send XMPP IQs, to publish notification, etc. In addition, it uses the function **addCloudMedPacketListener** to register a **CloudMed-PacketListener**, which will intercept all XCMCP messages received by the gateway.

4.2.3 DICOM Interface

To interface with the PACS infrastructure, it was used DCM4CHE library. This library allowed the implementation of DICOM communication layer between the CloudMed PACS gateway and a given PACS archive server. It was implemented some DICOM services, which allow the gateway to query (C-Find) and retrieve (C-MOVE) medical images from a the PACS archive (Figure 4.12). To improve the multi-threading mechanism in task performed by the gateway, it was implemented a layer before call the DICOM services. This layer is responsible for manage two port pooling, one for Query and another for Query/Retrieve. The most complicated part in the multi-threading tasks is to handle several Query/Retrive at same time. That is because, the gateway has to identify each Query/Retrive (C-MOVE) request. This is possible, by using the DICOM tag MoveOriginatorMessageID. So, when the DICOM receiver service receives a file it is able to know which XMPP messages requested the file move to the Cloud and consequently which user made the request and where this file must be stored. Nevertheless, the store is committed only if the owner of the target repository has granted the necessary permission to the user requesting the storage (Figure 4.13).

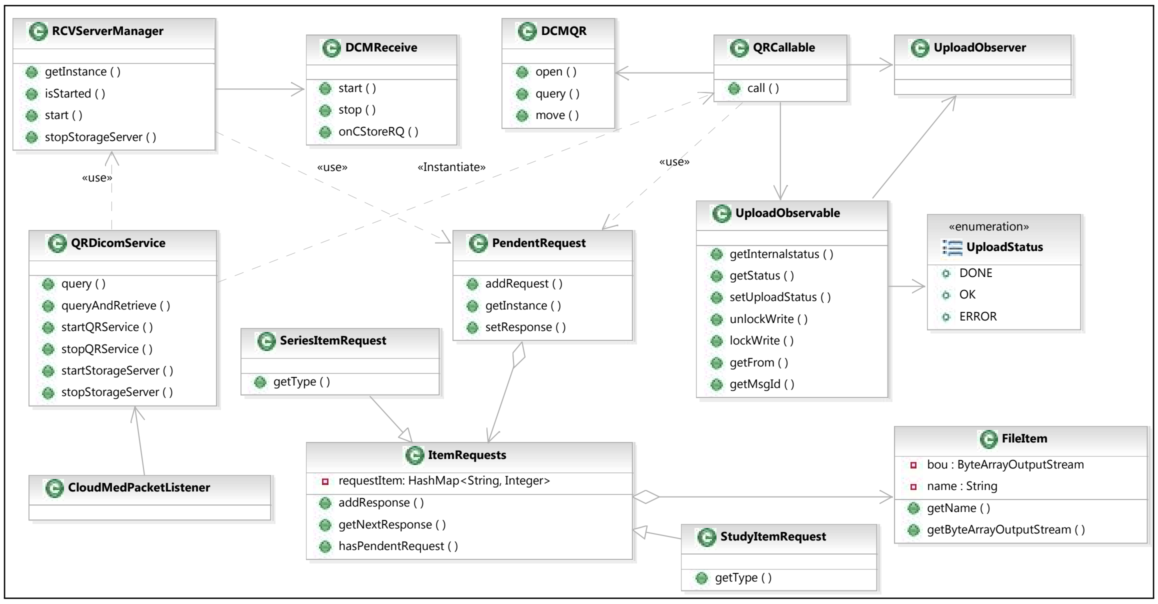


Figure 4.12: DICOM layer class diagram.

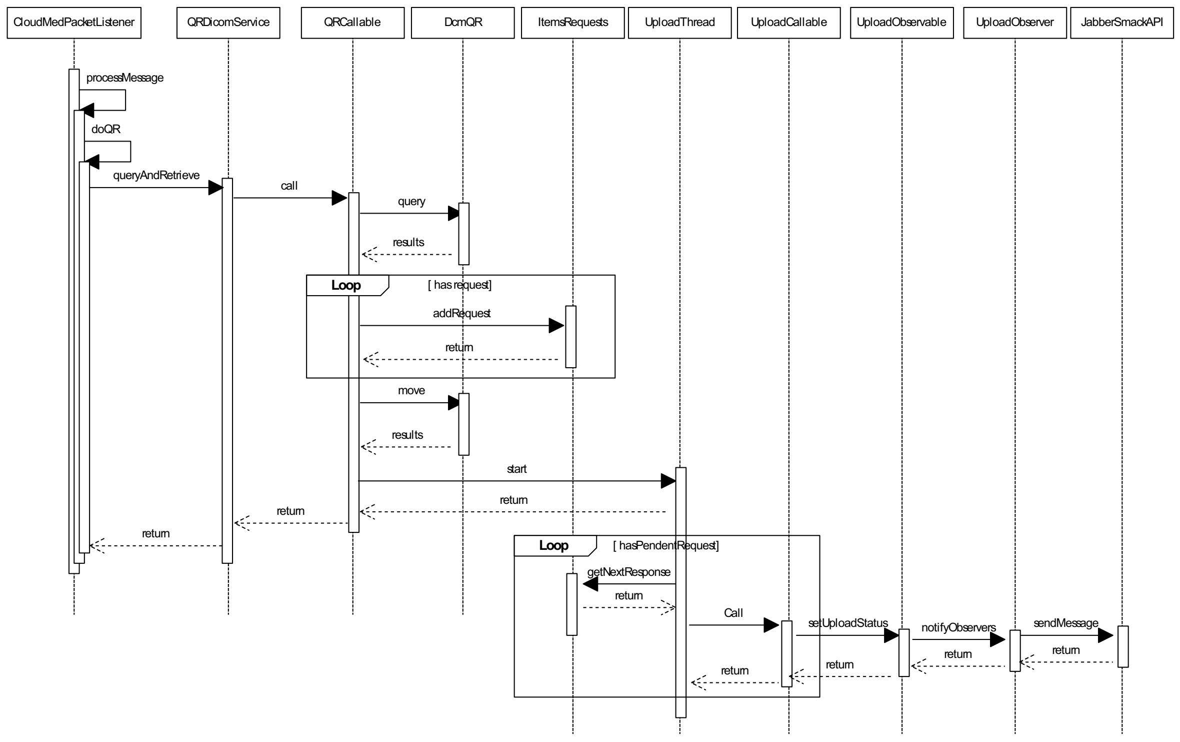


Figure 4.13: DICOM Query/Retrieve diagram.

4.2.4 Folder Synchronization

The system also provides mechanism to allow users to synchronize Cloud repository folders with a local workstations. The synchronization feature allows selective synch. Thus, users can synchronize just a given folder that contains a pretended study or a series. The synchronization action (Figure 4.14) is triggered by the Web client application, which sends custom messages (section 4.1.1) containing information about every folder being synchronized.

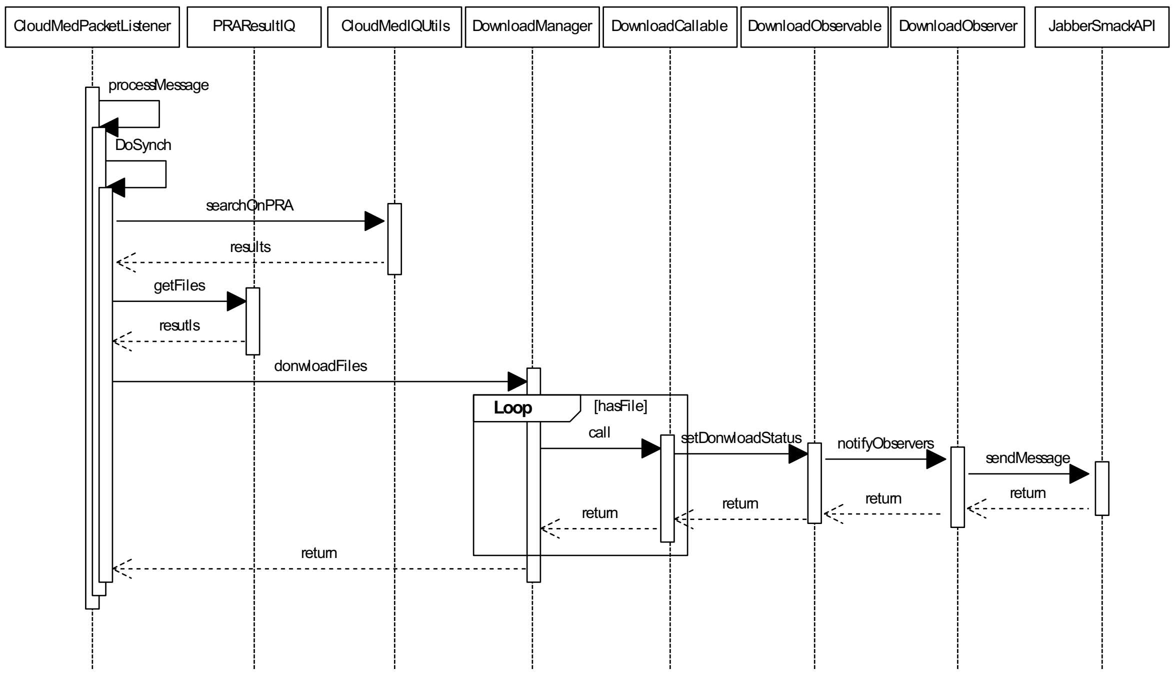


Figure 4.14: Folder synchronization diagram

4.2.5 Anonymous Images

Users are able to manage the access to their CloudMed PACS gateways through a Web application provided as SaaS. The Web application takes advantage of the message-oriented middleware to communicate with CloudMed PACS gateways. Thus, there are specifics messages that can be used to share gateways access and set anonymization rules. The anonymization rules are stored in the server using Private XML storage extension. And, to notify gateways about rules change, it is used a Publish-Subscribe pattern, where gateways subscribe a channel used by resource owner to publish the anonymization rules. After rules change, the gateway is notified and updates the new rules (Figure 4.15).

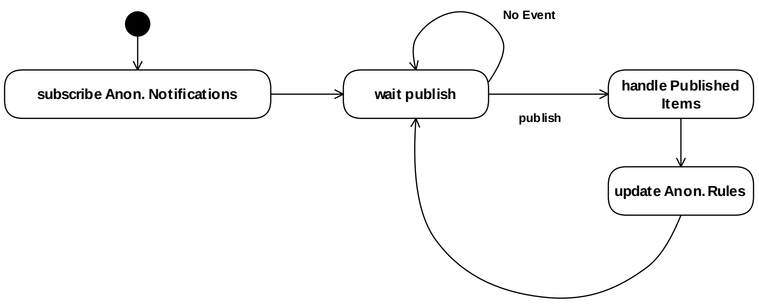


Figure 4.15: Anonymization rules update via Publish-Subscribe mechanism.

The anonymization rules are verified before upload any image to the Cloud. The rules are used to verify if a file must be sent as anonymized to a given user. By default, the files are sent anonymously but the gateway's owner can manage the anonymization rules to every share defined.

When the gateway receives a message to execute Query/Retrieve, it starts the DICOM receiver (if it is not started yet) to receive the DICOM images and, after that, it executes the DICOM C-MOVE commands. As soon as it receives an image, it verifies the anonymization rules in order to realize if it should (or not) anonymize the image before upload it to Cloud. The Figure 4.16 depicts the medical images retrieve with anonymization capabilities.

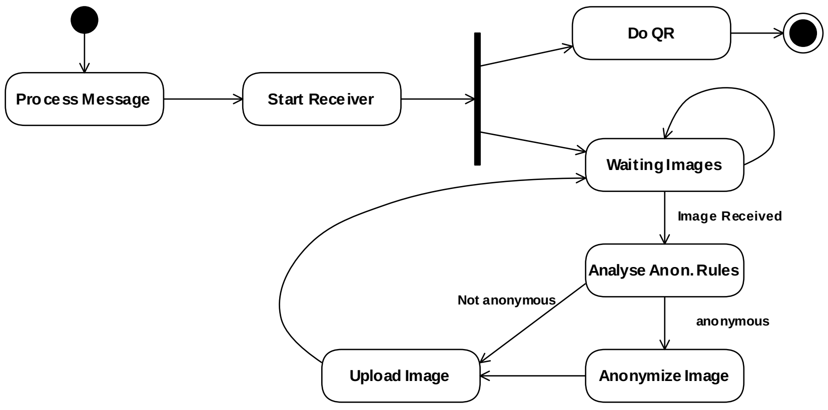


Figure 4.16: Image upload with anonymization capabilities.

4.3 CloudMed Web2.0 Client

Currently, web applications are being very improved by available programming interfaces supported by the browsers. The great improvements of the Web APIs came from the under-construction HTML5 standard, which allows programmers to create very complex Web applications using APIs like, for instance, geolocation, Web SQL Database, File API, Drag-and-drop, Offline Web application, etc.

Web applications are becoming very sophisticate, and some are inclusively replacing desktop applications. Web 2.0 applications are commonly used to consume services available in the cloud, thus allowing users to access the service from any-where just using a web browser. So, they do not have to install any specific desktop application to consume the service.

Admittedly, the creation of a Web application is an added value to the software-as-a-service provided by CloudMed system. This Web application allows users access all CloudMed functionalities, through a Web browser. Furthermore, some new HTML5 APIs improve the user experience and application quality in general.

CloudMed Web 2.0 client was completely developed using JavaScript programming language and HTML5 APIs. Moreover, it was implemented using the MVC design pattern [47] (Figure 4.17). As we can see in Figure 4.17, the file app.js is the main model of CloudMed Web2.0 application. This model uses some controllers like, for instance, html5uploader.js, messaging.js, filebrowser.js, etc, to control components in the user interface (Figure 4.17). These controllers have some functions that are triggered by the user interaction with the UI (User interface). Events like, drag and drop a file to the uploader UI component, trigger the function upload in the html5uploader.js controller. The upload action can send multiple files to the server, thus the controller has to update the UI component state with the progress, giving user feedback about current state.

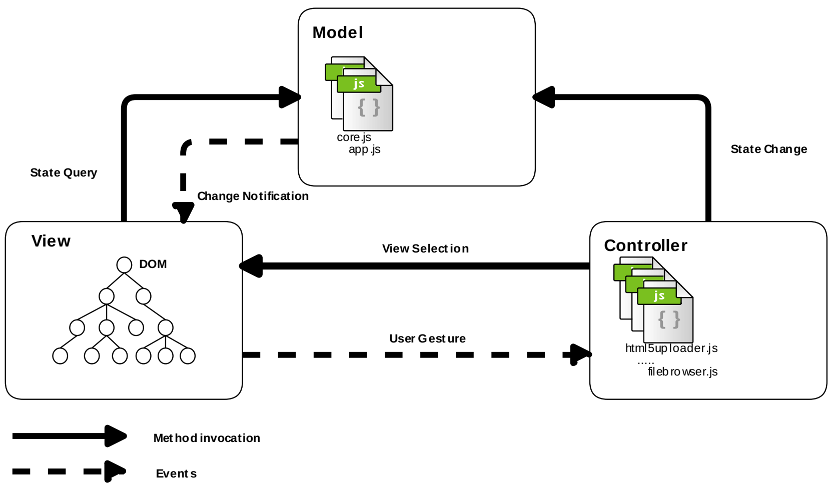


Figure 4.17: Web application MVC design pattern

The proof-of-concept UI has a simple interface layout (Figure 4.18) that enables users to have an intuitive interaction with the Web application. Most of the features are directly accessible through the master page. This page allows users to easily identify features like, for example, chat, access to remote PACS, access to shared PACS, access to cloud repository, etc. Furthermore, the master page has a dynamic section that is used to render dynamic content, according with context and the selected feature.

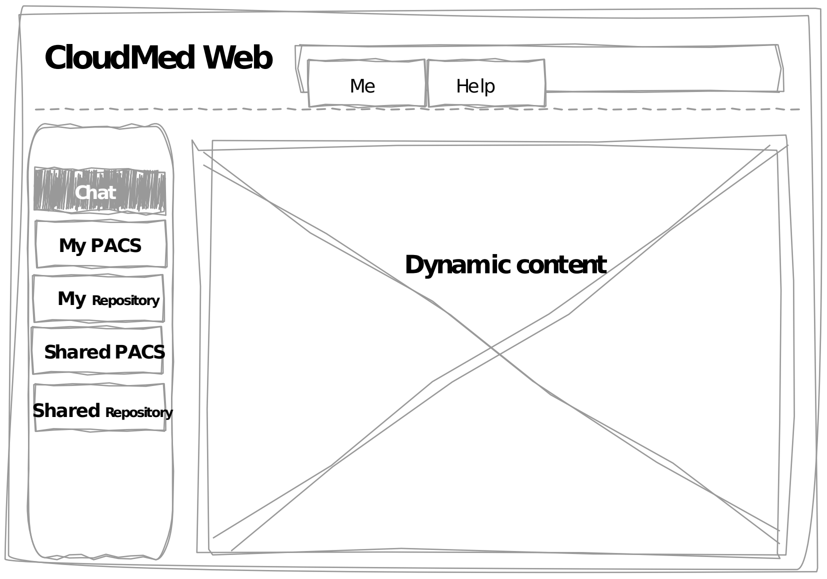


Figure 4.18: Web application layout prototype.

4.3.1 JavaScript XMPP Client

CloudMed system uses the XMPP message-oriented middleware, so the Web application has to act as a XMPP client to interact with other components on the XMPP network.

CloudMed Web 2.0 client uses Strophe.js [48] library to implement the XMPP web client. This library allows a Web application to speak XMPP but, unlike many others library available, it was designed to support chat-based applications and also to implement real-time games, notification systems, search engines, cloud services, etc.

As anything based on Web, Strophe.js connection to remote server is based on HTTP and Websockets protocol. Primarily, Strophe.js library uses Bidirectional-streams Over Synchronous HTTP (BOSH) to connect with XMPP server. BOSH is a technology for two-way communication over the Hypertext Transfer Protocol (HTTP), which uses long pooling [45, 46]. Beside the BOSH based implementations it is possible to find another implementation using Websockets. The possible use of Strophe.js based on Websocket was withdraw because currently it is not possible to proxy Websockets and an application server on same port. Thus, this limitation does not allow us to run the Web server (for serve Web application) and the Websocket on some port, i.e. the port 80. Moreover, if the Websockets is running on another port, the service can face some problems related with firewalls. The issues that affect Websocket proxy do not affect BOSH implementation because it is based on pure HTTP POST method, thus it is possible to proxy BOSH and the Web server on same port.

The following diagram (Figure 4.19) represents the organization of the JavaScript XMPP client that was implemented to be used by CloudMed Web2.0 client.

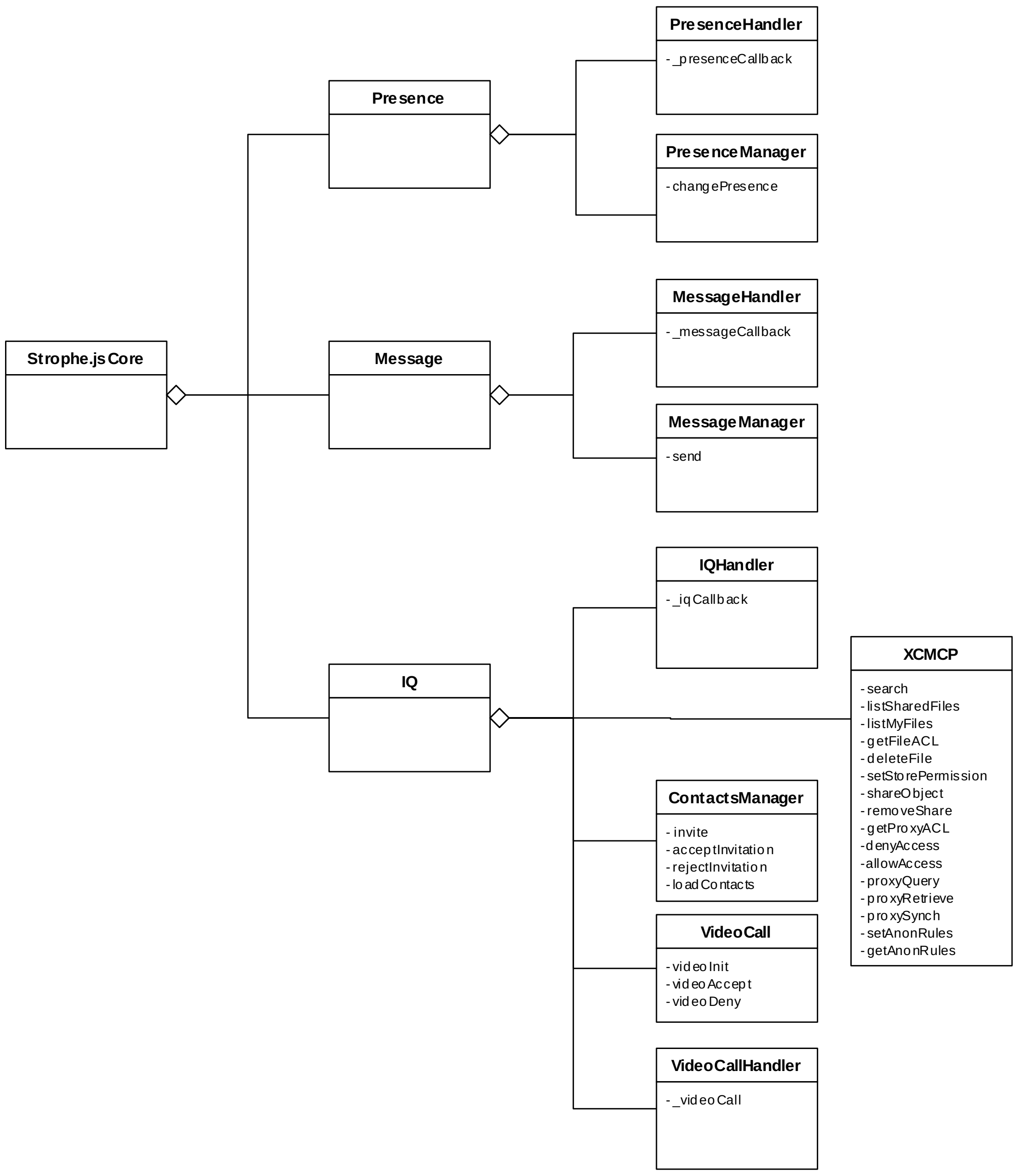


Figure 4.19: XMPP JavaScript client organization.

The Strophe.js core library only allows the Web application to understand default XMPP messages, which allow to chat, view users presence, manage contacts. Thus, was necessary to implement some features over this library in order to allow CloudMed Web2.0 client to interpret the new IQs introduced by the XCMCP (described in section 4.1.1), and features make signalization on video calls.

To show users' presence, the system implements a presence handler that is called whenever the Strophe.js core receives a <**presence**\> from a contact. This handler interacts with the user interface in order to notify the user about a contact presence change. And, in order to receive a contact <**presence**\>, the user must subscript to the contact's presence. This subscription is made by using the ContactManager.

4.3.2 Html5 and JavaScript User Interface

Drag And Drop

One natural action that users usually do on their computer is to drag and drop files from one place to another. Therefore, this action is available in CloudMed Web2.0 client, where users can drag and drop files from their computers to their personal repository in the Cloud. Currently, HTML5 defines direct APIs for handler drag and drop actions.

The HTML5 Drag&Drop API defines some native events to handler user interactions:

- dragstart

  Called when the "draggable" element first starts being dragged.

- dragend

  Called at the end of a drag event, successful or cancelled.

- dragenter

  Called on a target drop container when the drag first moves over the target.

- dragleave

  Called when the drag moves off the target.

- dragover

  Called continuously while the drag is over the target.

- drop

  Called when the drag event is completed by dropping the element onto the target drop container.

In the Web client, the JavaScript Drag&Drop API is used to tell the browser where a file can be dropped. If one file is dropped into a container that allows the drop action, an event is fired and a handler is invoked to take care of this event.

Nevertheless, to access the file's bytes is necessary to use the HTML5 File API. This API allows the access to a file's bytes using the class FileReader. The FileReader, provide several ways to read a file data, such as:

- readAsArrayBuffer

  Creates an ArrayBuffer containing the file data.

- readAsBinaryString

  Creates a string that contains the raw binary data from the file

- readAsDataURL

  Creates an URL representing the file's data.

- readAsText

  Creates a text string that contains the contents of the file.

The HTML5 Drag&Drop and File APIs were used to implement the feature that allows users to drag files from local computer hard disk into a specific container in the Web application, which grab the files contents and enables the Web application to send these to the Cloud.

File Upload

To upload a file, the Web application uses the HTML5 File API. This API is used to obtain the files' data and, after those bytes being available, the application is ready to upload the files' content to the Cloud.

Security concerns bring the need to authenticate the upload because it is made out-of-band, through a direct HTTP POST to the server in the Cloud, because XMPP has some limitation to handle binary data [49, 50]. One first approach to upload file to the Cloud was based in an in-band file send. But because XMPP messages are XML based, there was a need to convert the file's content to Base64 and send it to Cloud using a specific <**iq**\> stanza. Therefore that approach proved to be very inefficient and it was withdraw in the detriment of a HTTP POST strategy.

The file storage, which uses an out-of-band channel, implies that the HTTP POST is sent with some information that identify the user that is sending the file and the user that will be the owner of the file in the Cloud. In addition, an extra security was added, wherein the sender authenticates the files content calculating the HMAC of the file data, as defined in the RFC2104, and sends the calculated hash as a HTTP POST parameter.

File share

In telemedicine, information sharing at a distance is a key factor to provide assistance to others colleagues or to get assistance in some examination analysis. Moreover, if the information is easily available from any-where like, for instance, using just a Web browser, the users are more likely able to cooperate with each other.

Meanwhile, the system is available as an Internet SaaS, which provides features that permit users to share some folders, that are in the Cloud, with their friends. The XCMCP is used to manage the access of every folder that an user has in his remote repository.

To ensure that only the owner of a folder is able to share it, this action is limited to the scope of an user's XMPP connection stream, thus if a Web client send a share action to the server, the server tries to execute it by taking in account the authenticated user that is sending the XML stanza.

Integrated Messaging

To improve cooperation between colleagues, some features related with messaging and presence notification appear as crucial features to improve the cooperation. In telework, users needs to be in contact, thus a messaging functionality is very important to allow users to communicate with each other, in order to get information from some patient data examination, or any other kind of cooperation. Moreover, the presence exchange allows users to be notified about colleagues availability, and thus be able to know when a colleague is ready to chat with him.

To create a cooperative environment, the Web client uses XMPP server to manage the user contact list. The contact list is manages using XMPP specifics XML stanzas that are sent to the server to add or remove friends from the contacts. Besides, add or remove friends from user's roster, to be able to receive a friend presence notification the user must to subscribe to his friends' presence and wait them to grant permission to access their presence. The roster and presence subscription are made using some specifics <**iq**\> stanzas.

The chat feature is based on exchange of XMPP <**message**\> stanza between users. The Web application is able to process those <**message**\> using a message handler that intercepts all <**message**\> stanza received by the Web client. The message is introduced in the respective thread and the Web client core triggers a notification in order to update the user interface.

After being subscribed, the user receives notifications about the colleague presence change. To be able to notify, in the user interface, friends' presence change, there is a handler to process all <**presence**\> stanza received by the Web client.

WebRTC-based Video conference

The system uses HTML5 WebRTC API [51] to establish a connection between two browsers. This connection is used to transport the video stream. WebRTC is a protocol that enables Web browsers to support Real-Time Communications (RTC) capabilities via simple Javascript APIs. WebRTC empowers rich, high quality, RTC applications to be developed in the browser via simple Javascript APIs and HTML5.

As result of using WebRTC, to implement the video conference feature, it was possible to enrich the CloudMed Web2.0 application with video conference capabilities without the necessity to install any plug-in in the Web browser. Thus, browsers that are implementing HTML5 API, and support WebRTC allow users to take advantage of this feature.

To establish a WebRTC connection, it is necessary to exchange some messages in order to make the media negotiation. To make this negotiation, WebRTC uses SDP [52] (Session Description Protocol). But, the SDP messages need to be exchanged between browsers. So, CloudMed Web2.0 application uses XMPP to make the signalling. It was implemented an extension to Strophe.js library to handle the signalization layer in WebRTC connection establishment (Figure 4.20). This extension is the VideoCall entity, represented in Figure 4.19. So, this entity allows to initialize, accept and deny video calls.

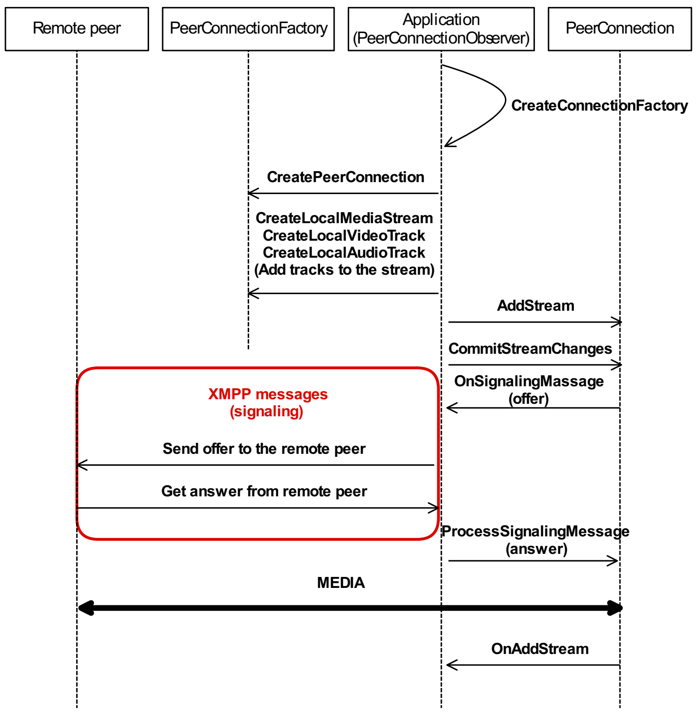

Remote peer
PeerConnectionFactory
Application (PeerConnectionObserver)
PeerConnection
CreateConnectionFactory
CreatePeerConnection
CreateLocalMediaStream
CreateLocalVideoTrack
CreateLocalAudioTrack
(Add tracks to the stream)
AddStream
CommitStreamChanges
OnSignalingMassage (offer)
XMPP messages (signaling)
Send offer to the remote peer
Get answer from remote peer
ProcessSignalingMessage (answer)
MEDIA
OnAddStream

Figure 4.20: WebRTC session establishment.

Embedded Image Viewer

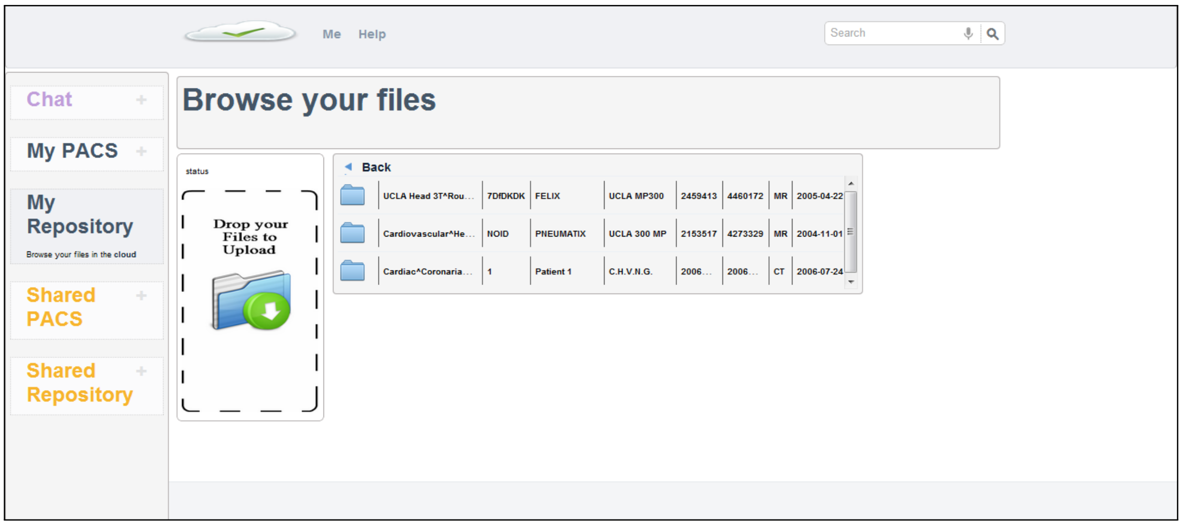
CloudMed provides an embedded image viewer that can be used in the Software-as-a-Service by radiologists to analyse medical images, and thus, providing them some tools to support a decision making on an examination.

In telemedicine, image viewer is usually very important to leverage the system functionalities. Medical image viewer is an asset for almost all telemedicine application [53, 54] because it provides some features very useful for physicians like, for instance, tools for annotate medical images, image processing based on local contrast enhancement, adaptive interpolation technique, colour transformation, cine loop, etc.

The viewer created uses some JavaScript libraries like, for example, jquery.svg.js [55], pixastic.core.js [56], and other drawing libraries to implement the visualization layer. These tools enable user to draw overlays, manipulate contrast, apply filters, measure pixel density, and other varieties of operations like fancy effects on images using just a bit of JavaScript.

4.4 Results

The objectives defined when modelling the system were successfully fulfilled and, as result, created a system with a large potential to empower telemedicine in a cooperative network. The results obtained show the potential use of Cloud computing for delivery telemedicine services. The deployment of Software-as-a-Service for telemedicine is feasible, therefore it has a great potential to be applied in this area. To implement a system with high capabilities for telework, several aspects were focused like, for instance, data access availability, cooperative work, medical images visualization, remote PACS access, ubiquitous computing, etc. In telemedicine, data availability is very important. To take advantage of Cloud services features was implemented in the Cloud a personal repository (Figure 4.21) where users can store digital medical images. Thus, it becomes possible to inherent from Cloud services features the data availability. Therefore users can access their data stored in their personal repository any-where and any-time.



Browse your files

🗀	UCLA Head 3T^Rou...	7DfDKDK	FELIX	UCLA MP300	2459413	4460172	MR	2005-04-22
🗀	Cardiovascular^He...	NOID	PNEUMATIX	UCLA 300 MP	2153517	4273329	MR	2004-11-01
🗀	Cardiac^Coronaria...	1	Patient 1	C.H.V.N.G.	2006...	2006...	CT	2006-07-24

Figure 4.21: Remote repository in the Cloud.

Also, using this personal remote repository feature, it was possible to leverage telemedicine's services with the enhancement of cooperation between colleagues by enabling users to share their data with colleagues (Figure 4.22). This feature improves the system capabilities, due to the fact that it creates an agile way to make dataflow in telemedicine. Thus, if a radiologist needs to send an exam to be analysed by a colleague he just has to share this exam with his colleague that becomes able to access it from any-where and any-time (Figure 4.23).

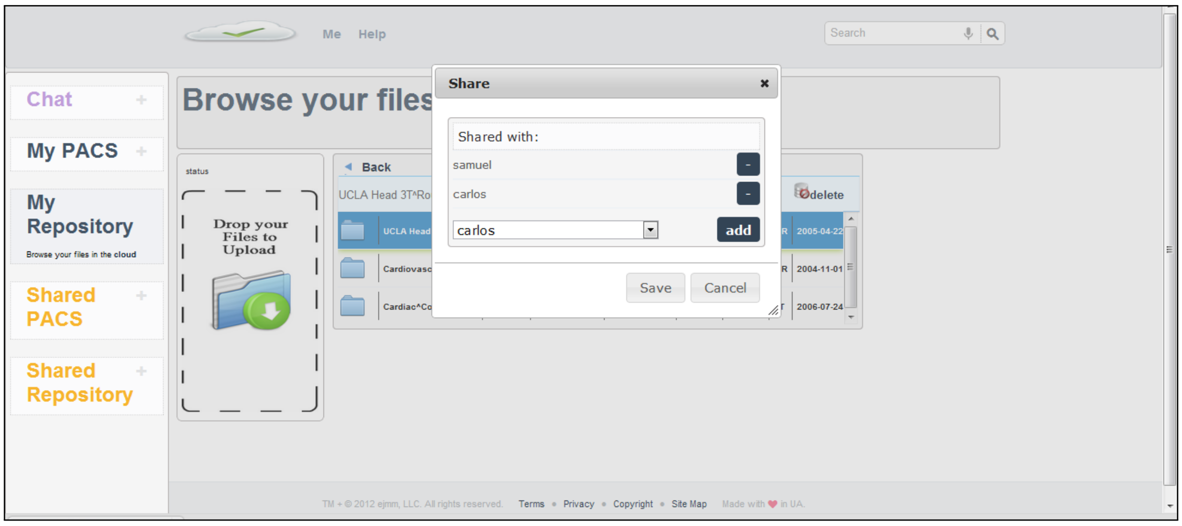


Figure 4.22: Sharing files with colleagues.

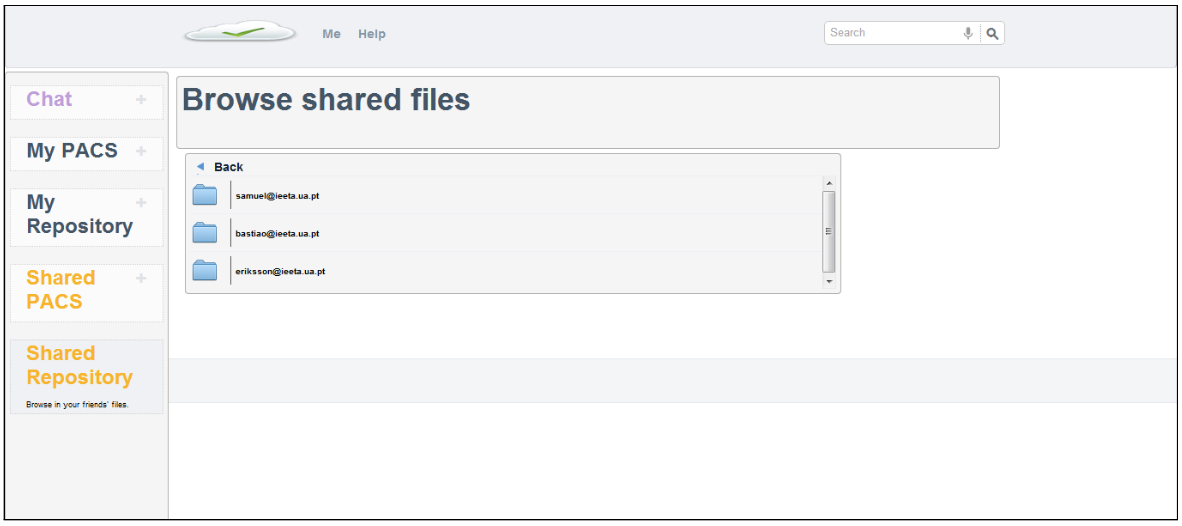
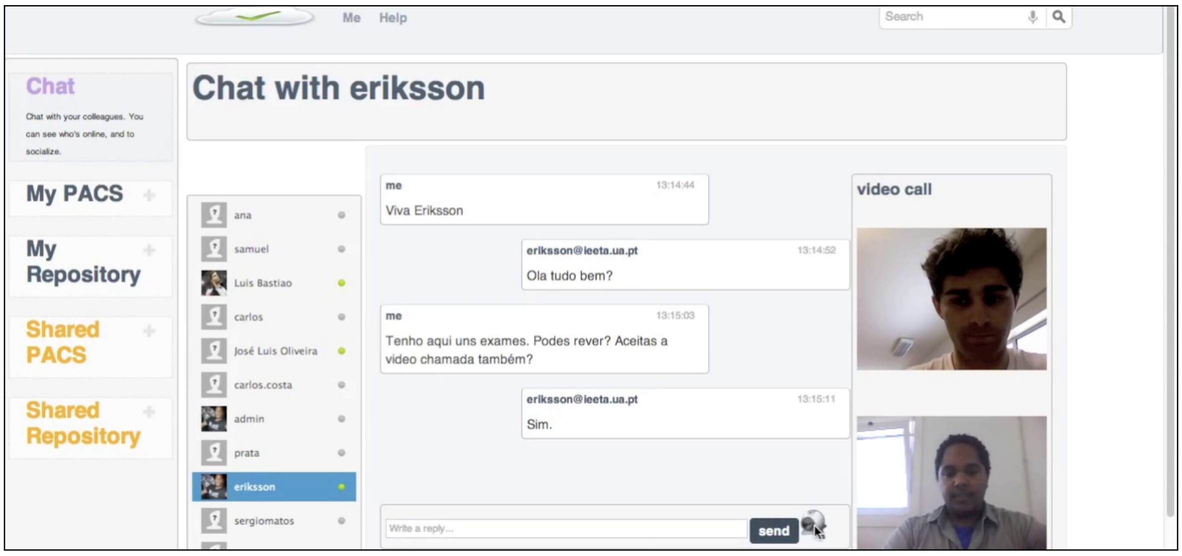


Figure 4.23: Accessing shared files.

Moreover, this system focused itself in improve the cooperation and collaboration in telework, and some of the implemented features follow the socialnetwork paradigm. This paradigm empowers the cooperation and collaboration features. Users can add others as friends and thus become able to share data with them, interact with them in exams' analyses and do video conferences (Figure 4.24). Another important feature for collaboration that this system provides is the awareness of presence status of each user. This feature is very important because it allows radiologists to know when they can interact with each other and when their colleagues are able to reply, which is not possible in systems such as email.



Chat with your colleagues. You can see who's online, and to socialize.

Chat with eriksson

me 13:14:44
Viva Eriksson

eriksson@ieeta.ua.pt 13:14:52
Ola tudo bem?

me 13:15:03
Tenho aqui uns exames. Podes rever? Aceitas a video chamada também?

eriksson@ieeta.ua.pt 13:15:11
Sim.

Figure 4.24: collaborative tools.

Some users usually work in several workstations (e.g. at home, office, laptop, and so forth) and some times they need to synchronize data between different workstations. So, to leverage the system capabilities, it was implemented a feature that allows users to do selective data synch by selecting a given folder, from the Cloud repository, to be synchronized with a workstation (Figure 4.25).

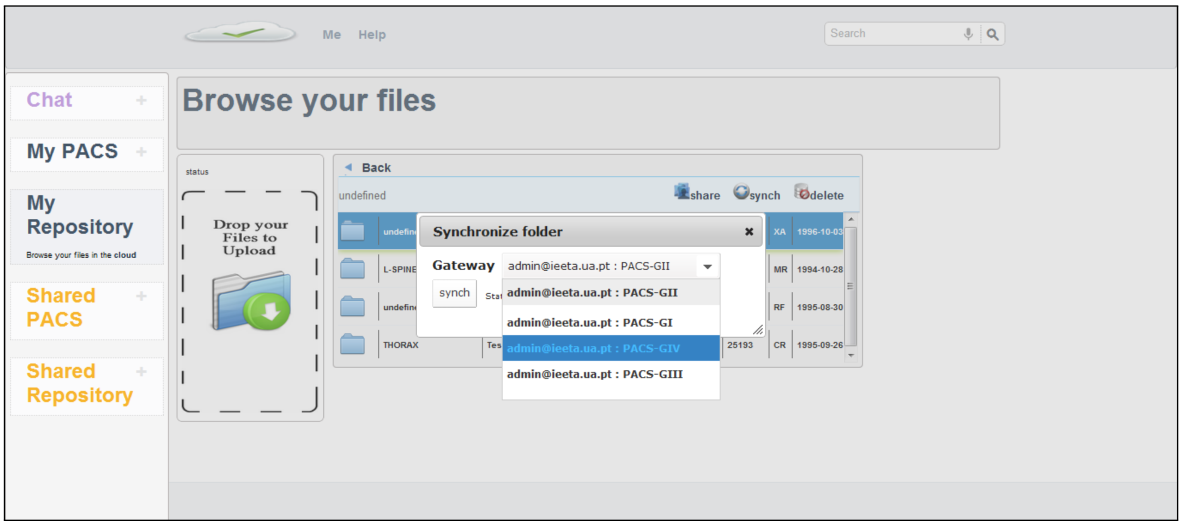


Figure 4.25: Synchronizing data from the Cloud with a given CloudMed PACS gateway

The synchronization feature allows users to synchronize Cloud files with a workstation. This will reduce the latency and allow users to open medical images studies with specialized applications. However, CloudMed system provides an embedded DICOM viewer, implemented with HTML5 and JavaScript, which enable the visualization of DICOM medical images directly from the Cloud, without have to download them (Figure 4.26).

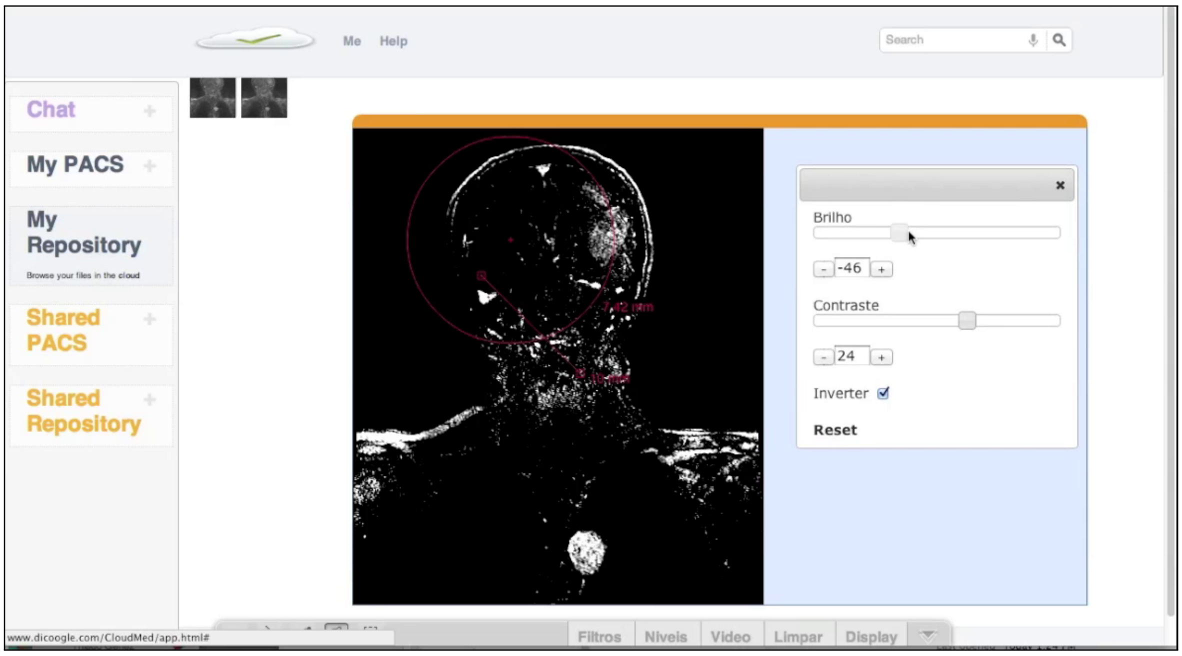
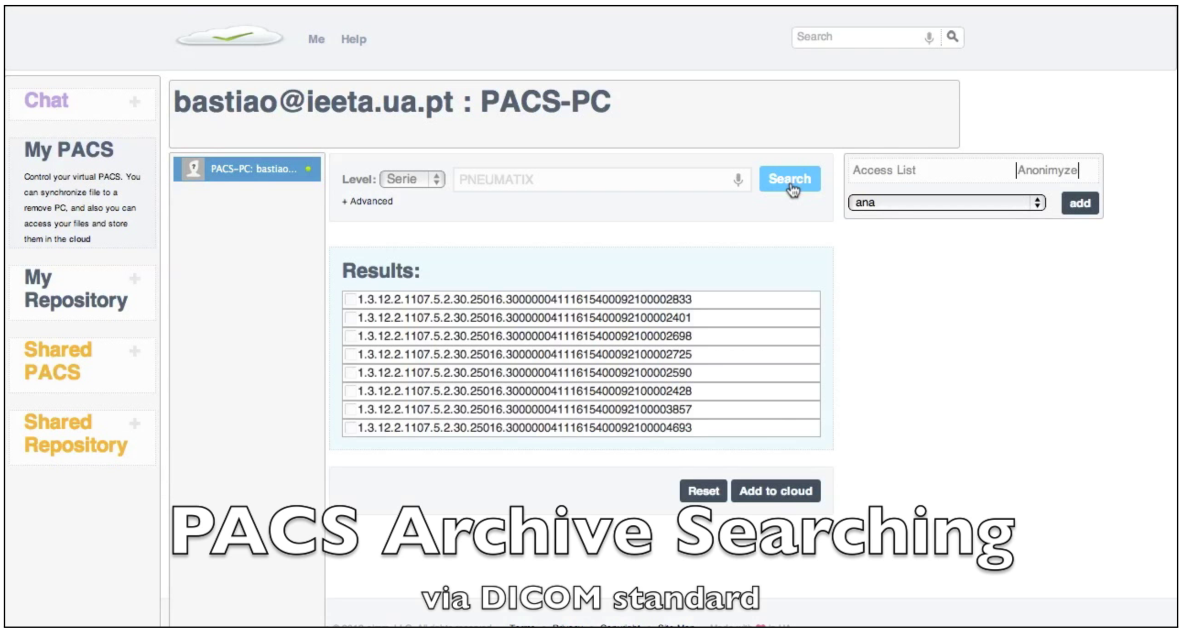


Figure 4.26: Embedded HTML5/JS DICOM viewer

As result, this system also enhance its ubiquity by providing features that allow radiologists to do regular examinations, which they are used to do in their workplace, from any-where by accessing the CloudMed system. This is possible because CloudMed system enables radiologists to connect to their PACS via Cloud infrastructure. Thus, through a gateway, the radiologist becomes able to access remote PACS for search (Figure 4.27) and retrieve (Figure 4.28) exams.



bastiao@ieeta.ua.pt : PACS-PC

Control your virtual PACS. You can synchronize file to a remove PC, and also you can access your files and store them in the cloud

Level: Serie PNEUMATIX

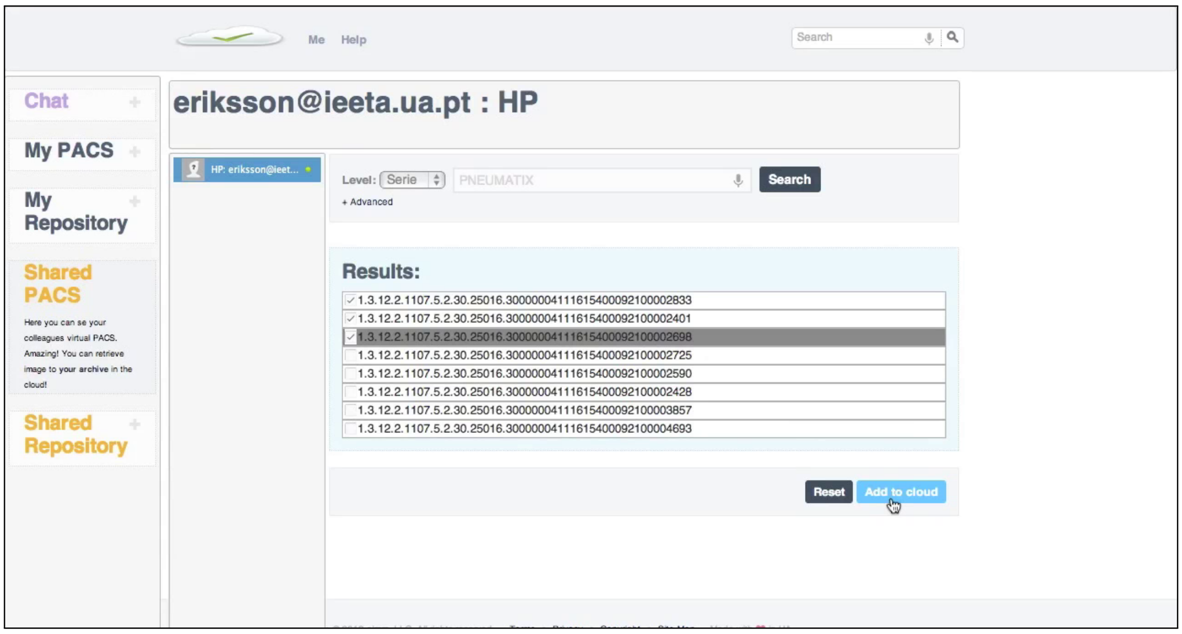
+ Advanced

ana

Results:

```
1.3.12.2.1107.5.2.30.25016.30000004111615400092100002833
1.3.12.2.1107.5.2.30.25016.30000004111615400092100002401
1.3.12.2.1107.5.2.30.25016.30000004111615400092100002698
1.3.12.2.1107.5.2.30.25016.30000004111615400092100002725
1.3.12.2.1107.5.2.30.25016.30000004111615400092100002590
1.3.12.2.1107.5.2.30.25016.30000004111615400092100002428
1.3.12.2.1107.5.2.30.25016.30000004111615400092100003857
1.3.12.2.1107.5.2.30.25016.30000004111615400092100004693
```

Figure 4.27: Searching on remote PACS archive from web application



eriksson@ieeta.ua.pt : HP

Here you can se your colleagues virtual PACS. Amazing! You can retrieve image to your archive in the cloud!

Level: Serie PNEUMATIX

+ Advanced

Results:

```
✓ 1.3.12.2.1107.5.2.30.25016.30000004111615400092100002833
✓ 1.3.12.2.1107.5.2.30.25016.30000004111615400092100002401
✓ 1.3.12.2.1107.5.2.30.25016.30000004111615400092100002698
  1.3.12.2.1107.5.2.30.25016.30000004111615400092100002725
  1.3.12.2.1107.5.2.30.25016.30000004111615400092100002590
  1.3.12.2.1107.5.2.30.25016.30000004111615400092100002428
  1.3.12.2.1107.5.2.30.25016.30000004111615400092100003857
  1.3.12.2.1107.5.2.30.25016.30000004111615400092100004693
```

Figure 4.28: Select and retrieve images from remote PACS arhive

In addition, to promote collaboration and cooperative environment in imagiology, users can also share their PACS with colleagues to supply outsourcing, for instance, of reporting services. Due to privacy concerns, the PACS owner can define that data will be provided in an anonymous way. Thus, when his colleagues retrieve exams from that PACS, the sensible information is returned anonymous (Figure 4.29).

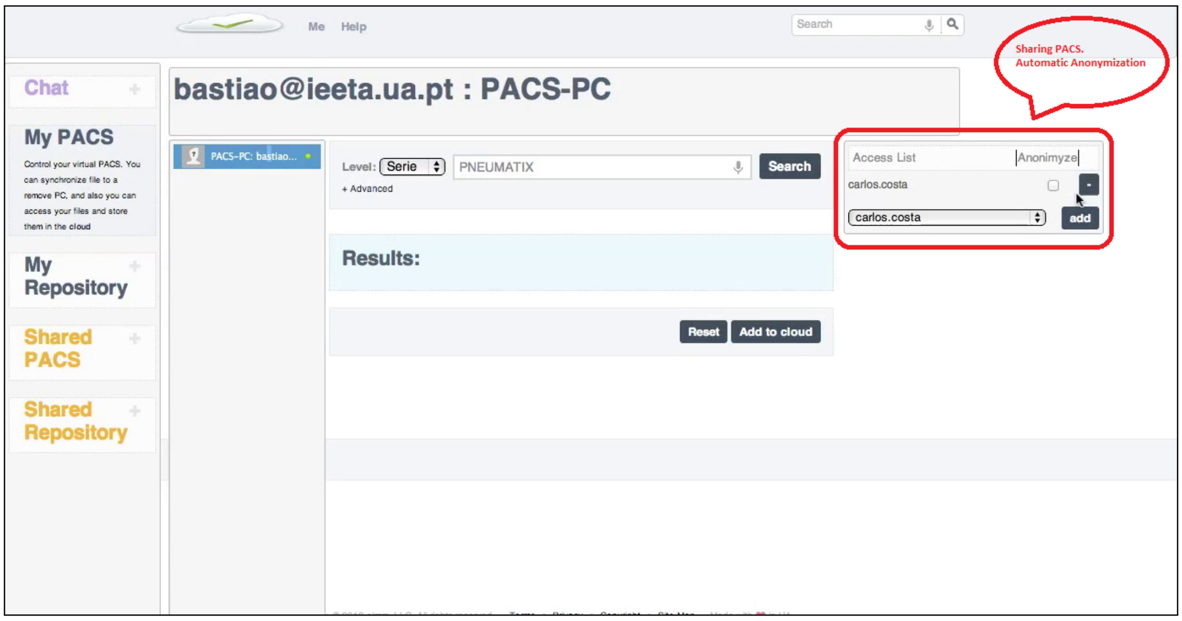
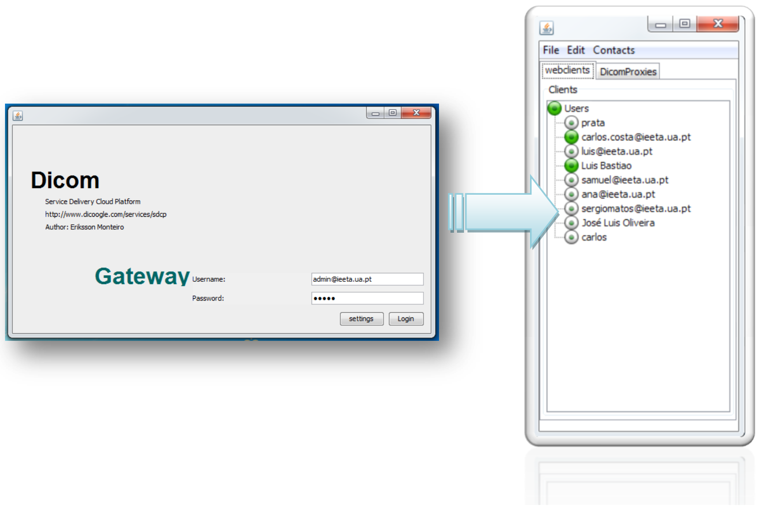

Me
Help
Search
Sharing PACS, Automatic Anonymization
Chat
My PACS
Control your virtual PACS. You can synchronize file to a remove PC, and also you can access your files and store them in the cloud
My Repository
Shared PACS
Shared Repository
bastiao@ieeta.ua.pt : PACS-PC
PACS-PC: bastiao...
Level: Serie
PNEUMATIX
Search
+ Advanced
Results:
Reset
Add to cloud
Access List
Anonimyze
carlos.costa
carlos.costa
add

Figure 4.29: Sharing PACS with colleagues

The gateway client contain a GUI which allows the administrator to visualize which are the active users in this gateway, and also it provides in the GUI the possibility to set some configurations for the gateway (Figure 4.31). For instance, it is possible to set the address and port of the PACS archive server, the address route and port of the XMPP server, port pooling range, the local folder where is stored the files synchronized from the Cloud, etc. These configurations are stored in a property file which is loaded when the gateway starts. After configure the gateway and insert the credentials to authenticate (Figure 4.30) the gateway into the XMPP network, it becomes fully operational and ready to receive and reply to any messages.


Dicom

Service Delivery Cloud Platform

http://www.dicoogle.com/services/sdcp

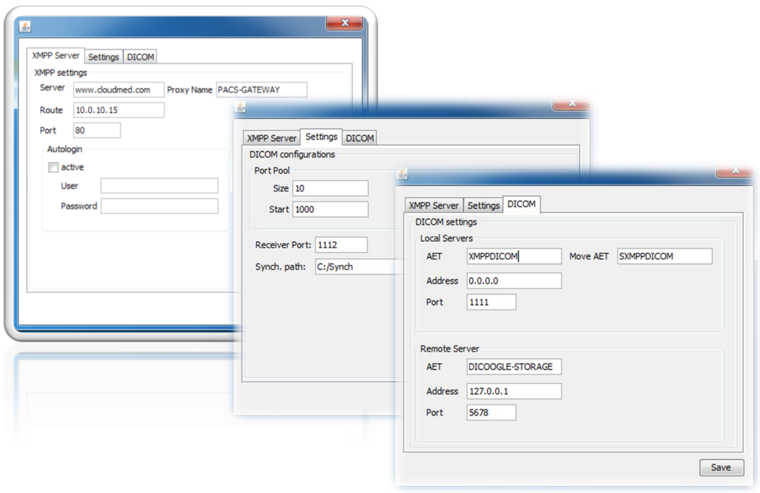
Author: Eriksson Monteiro

Gateway

Username: admin@ieeta.ua.pt

Password: •••••

Figure 4.30: Gateway graphical user interface.


XMPP settings

Server www.cloudmed.com Proxy Name PACS-GATEWAY

Route 10.0.10.15

Port 80

Autologin

☐ active

User

Password

DICOM configurations

Port Pool

Size 10

Start 1000

Receiver Port: 1112

Synch. path: C:/Synch

DICOM settings

Local Servers

AET XMPPDICOM Move AET SXMPPDICOM

Address 0.0.0.0

Port 1111

Remote Server

AET DICOOGLE-STORAGE

Address 127.0.0.1

Port 5678

Figure 4.31: Gateway settings GUI.

Chapter 5

Conclusions

The major objective, which defined the path followed by this thesis, was to study and realise how new technologies, in particular Cloud computing, can improve telemedicine workflow and dataflow. Moreover, to leverage telemedicine support, it was studied how could computer supported system improve the way that radiologists interact with each other in order to share knowledges in medical images analysis. Generally, this thesis aim was to create an integrated environment where radiologists could share medical data and collaborate with each other in order to improve telework. Furthermore, this thoroughly integrated environment takes advantage of Cloud computing features to provide a SaaS, which users can access from anywhere using an Internet connection.

To implement the system that achieved these goals, it was created an infrastructure that integrates several distributed data sources into an unique system using a message-oriented middleware to connect those data sources with the system core. Furthermore, in the system deployment, it was used an IaaS wherein was installed an XMPP server to deal with the message-oriented middleware, a SQL database (for index information about users and the data sources) and a Cloud storage service to store digital medical images' data.

Undoubtedly, the implementation of this system, using Cloud computing, was very important to leverage cooperation and collaboration in telemedicine. Thereby, it was used one XMPP server to create a cooperative environment where users can interact with each other by chat and video conference. In addition, the collaborative system provides presence notification mechanism to allow users to be aware of their colleagues availability.

The final system allows users to share medical images with colleagues with a simplicity similar to actual social networking application. To enable this feature, it was important the usage of Cloud blob storage to archive those medical images, in order to be available any-time and any-where. Furthermore, the system uses a plug-in based approach to enable decoupling compute infrastructure from storage infrastructure. This decoupling is archived by creating a plug-in with a specific interface, which has the role to enable the communication between the Cloud computation's infrastructure and the Cloud storage's infrastructure.

Moreover, to enhance telemedicine service quality and the collaboration between radiologists, the system also allows users to access and share their PACS archive server with colleagues, by using a gateway to enable the communication between this system and the PACS archive server inside a given health care PACS.

Nevertheless, this system presents a new concept using new techonlogies, which can be seen as a threat to it adoption. The adoption of Cloud-based systems to support teleimagiology services is not very wide in Portuguese's medical environments. That is because, there are some concerns about medical images privacy and protection. But, with the evolution of Cloud services and the improvement of their security and privacy models, there will be an increasingly adoptions of this concept to implement medical informatics systems.

The system requirements, listed in the chapter 3, were fulfilled by creating a fully integrated system for teleimagiology, where there are available several medical images data sources like repositories in the Cloud and remote PACS archive available through a gateway.

5.1 Future Work

Some features could be more explored in order to take advantage of the developed system core. For instance, tools like cooperative image viewing could be developed, extending actual proposal with some extra functionalities to improve the decision making in exam analysis. It could include features like interactive real-time image viewer with multiple users manipulating the same image, including multimedia annotations.

Distributed collaboration environment wherein multi-user interaction is the main problem is a theme that has been focused in literature. It can be found some systems focused in the implementation of multi-user interaction for collaboration [57–60]. From [60], it can be concluded that a middleware can facilitate the development of distributed and collaborative application. The middleware is useful to implement collaboration bus, to enable resources to be plugged in and allow users to interact, using the resources, collaboratively and transparently. So, implementing a multi-user interaction in CloudMed system will be more easy because the system core is thus grounded on XMPP message-oriented middleware.

Moreover, additional security constraint could be taken in account in further work. In order to enhance the control over the PACS share, the system could implement features enabling the PACS owner to limit some user queries to a set of patient or within a temporal constraint.

The system developed represents a well defined core to other applications, built over a message-oriented middleware. It could be extended with some components able to communicate with the middleware. For instance, could be implemented gateways to communicate with others kind of medical information data sources like, for example, HL7.

Bibliography

[1] M.H.A. Al-Taei. “Telemedicine needs for multimedia and integrated services digital network (isdn)”. In *Computational Intelligence Methods and Applications, 2005 ICSC Congress on*, page 4 pp., 0-0 2005.

[2] I. Chorbev, M. Mihajlov, and I. Jolevski. “Wimax supported telemedicine as part of an integrated system for e-medicine. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 589 –594, june 2008.

[3] A. Rodrigues, C. Resende, L. Carvalho, P. Saleiro, and F. Abrantes. Performance analysis of an adaptable home healthcare solution. In *e-Health Networking Applications and Services (Healthcom), 2011 13th IEEE International Conference on*, pages 134 –141, june 2011.

[4] M.M. Maheu, P. Whitten, and A. Allen. *“E-Health, Telehealth, and Telemedicine: A Guide to Start-Up and Success”*. Jossey-Bass Health Series. Jossey-Bass, 2001.

[5] Anonymous. Sending dental x-rays by telegraph. *Dent Radiogr Photogr.*, pages 6–12, 1929.

[6] H.K. Huang. *PACS and Imaging Informatics: Basic Principles and Applications.* Wiley, 2004.

[7] Sajeesh Kumar. “Introduction to teleradiology”. In Sajeesh Kumar and Elizabeth A. Krupinski, editors, *Teleradiology*, pages 1–9. Springer Berlin Heidelberg, 2008.

[8] Chao-Tung Yang, Lung-Teng Chen, Wei-Li Chou, and Kuan-Chieh Wang. “Implementation of a medical image file accessing system on cloud computing”. In *Proceedings of the 2010 13th IEEE International Conference on Computational Science and Engineering*, CSE ’10, pages 321–326, Washington, DC, USA, 2010. IEEE Computer Society.

[9] A. Silva L. Bastião, C. Costa and J. L. Oliveira. “A pacs gateway to the cloud”. *6th Iberian Conference on Information Systems and Technologies (CISTI 2011)*, 2011.

[10] Charalampos Doukas, Thomas Pliakas, and Ilias Maglogiannis. “Mobile healthcare information management utilizing cloud computing and android os.”. *Conference Proceedings of the International Conference of IEEE Engineering in Medicine and Biology Society*, 2010(8):1037–1040.

[11] Carlos Ferreira Viana, Daniel Ferreira, Frederico Valente, and Eriksson Monteiro. “Dicoogle mobile: a medical imaging platform for android”. *XXIV Conference of the European Federation for Medical Informatics*, 2012.

[12] Cosmin Porumb, Sanda Porumb, Bogdan Orza, and Dinu Budura. "Computer-supported collaborative work and its application to e-health". *Advances in Mesh Networks, International Conference on*, 0:75–80, 2010.

[13] Kevin L. Mills. "Computer-supported cooperative work (cscw)", 2003.

[14] J.K.H. Tan. *"E-health Care Information Systems: An Introduction For Students And Professionals"*. Jossey-Bass, 2005.

[15] Carlos Costa, José L. Oliveira, Augusto Silva, Vasco Gama Ribeiro, and José Ribeiro. Design, development, exploitation and assessment of a cardiology web pacs. *Comput. Methods Prog. Biomed.*, 93:273–282, March 2009.

[16] Oleg S. Pianykh. *Digital Imaging and Communications in Medicine: A Practical Introduction and Survival Guide.* Springer Publishing Company, Incorporated, 1 edition, 2008.

[17] NEMA. Digital imaging and communications in medicine (dicom) - part 5: Data structures and encoding.

[18] NEMA. Digital imaging and communications in medicine (dicom) - part 14: Grayscale standard display function.

[19] William Voorsluys, James Broberg, and Rajkumar Buyya. *Introduction to Cloud Computing*, pages 1–41. John Wiley & Sons, Inc., 2011.

[20] R. Buyya, J. Broberg, and A. Gościński. *"Cloud Computing: Principles and Paradigms"*. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2011.

[21] Amazon, amazon virtual private cloud. `http://aws.amazon.com/vpc/`. Accessed: 24/03/2012.

[22] Skytap, virtual lab. `http://www.skytap.com/`. Accessed: 24/03/2012.

[23] Cohesiveft, vpn-cubed. `http://www.cohesiveft.com/vpncubed/`. Accessed: 24/03/2012.

[24] E. Cerami. *"Web services essentials"*. Cookbooks Series. O'Reilly, 2002.

[25] S. Allamaraju. *"RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity"*. O'Reilly Series. O'Reilly Media, 2010.

[26] J. Governor, D. Nickull, and D. Hinchcliffe. *"Web 2.0 Architectures"*. O'Reilly Series. O'Reilly Media, Inc, 2009.

[27] M.D. Lytras, E. Damiani, and P.O. de Pablos. *"Web 2.0: The Business Model"*. Springer, 2008.

[28] Google app engine. `https://developers.google.com/appengine/`. Accessed: 4/07/2012.

[29] Windows azure. `http://www.windowsazure.com`. Accessed: 4/07/2012.

[30] Maha Abousharkh and Hussein Mouftah. Xmpp-enabled soa-driven middleware for remote patient monitoring system. In *Information Technology and e-Services (ICITeS), 2012 International Conference on*, pages 1 –5, march 2012.

[31] A. Hornsby. Xmpp message-based mvc architecture for event-driven real-time interactive applications. In *Consumer Electronics (ICCE), 2011 IEEE International Conference on*, pages 617 –618, jan. 2011.

[32] Extensible messaging and presence protocol (xmpp):core, and related others rfc. `http://xmpp.org/rfcs/rfc3920.html`. Accessed: 23/02/2012.

[33] Xep-0114: Jabber component protocol. `http://xmpp.org/extensions/xep-0114.html`. Accessed: 03/07/2012.

[34] Xep-0184: Message delivery receipts. `http://xmpp.org/extensions/xep-0184.html`. Accessed: 27/06/2012.

[35] Xep-0049: Private xml storage. `http://xmpp.org/extensions/xep-0049.html`. Accessed: 25/02/2012.

[36] Chesspara, online chess game. `https://secure.chess.com/chesspark.html`. Accessed: 23/05/2012.

[37] R. Bejtlich. *Extrusion detection: security monitoring for internal intrusions*. Addison-Wesley, 2006.

[38] Xmpp servers. `http://xmpp.org/xmpp-software/servers/`. Accessed: 14/06/2012.

[39] Openfire - jive software. `http://www.igniterealtime.org/projects/openfire/`. Accessed: 21/02/2012.

[40] The base16, base32, and base64 data encodings. `http://tools.ietf.org/html/rfc4648`. Accessed: 27/06/2012.

[41] Xep-0065: Socks5 bytestreams. `http://xmpp.org/extensions/xep-0065.html`. Accessed: 27/06/2012.

[42] Xep-0167: Jingle rtp sessions. `http://xmpp.org/extensions/xep-0167.xml`. Accessed: 27/06/2012.

[43] Hmac: Keyed-hashing for message authentication. `http://www.ietf.org/rfc/rfc2104.txt`. Accessed: 27/06/2012.

[44] S. Garfinkel, G. Spafford, and A. Schwartz. *Practical UNIX and Internet Security*. O'Reilly Media, 2011.

[45] Xep-0124: Bidirectional-streams over synchronous http. `http://xmpp.org/extensions/xep-0124.html`. Accessed: 04/06/2012.

[46] Xep-0206: Xmpp over bosh. `http://xmpp.org/extensions/xep-0206.html`. Accessed: 04/06/2012.

[47] S.J. Metsker and W.C. Wake. *Design Patterns in Java*. The Software Patterns Series. Addison-Wesley, 2006.

[48] Strophe.js. `http://strophe.im/strophejs/`. Accessed: 25/02/2012.

[49] J. Moffitt. *Professional Xmpp Programming with JavaScript and Jquery*. John Wiley & Sons, 2010.

[50] P. Saint-André, K. Smith, and R. Tronçon. *XMPP: The Definitive Guide : Building Real-time Applications with Jabber Technologies*. Oreilly Series. O'Reilly, 2009.

[51] Webrtc. `http://www.webrtc.org/`. Accessed: 4/07/2012.

[52] Session description protocol. `http://www.ietf.org/rfc/rfc2327.txt`. Accessed: 4/07/2012.

[53] P. Suapang, K. Dejhan, and S. Yimmun. A web-based dicom-format image archive, medical image compression and dicom viewer system for teleradiology application. In *SICE Annual Conference 2010, Proceedings of*, pages 3005 –3011, aug. 2010.

[54] P. Suapang, S. Yimmun, and A. Puditkanawat. Web-based medical image archiving and communication system for teleimaging. In *Control, Automation and Systems (ICCAS), 2011 11th International Conference on*, pages 172 –177, oct. 2011.

[55] jquery svg. `http://keith-wood.name/svg.html`. Accessed: 4/07/2012.

[56] Pixastic. `http://www.pixastic.com/`. Accessed: 4/07/2012.

[57] V.R. Watson. Supporting scientific analysis within collaborative problem solving environments. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, page 9 pp., jan. 2001.

[58] Atul Prakash and Hyong Sop Shim. Distview: support for building efficient collaborative applications using replicated objects. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 153–164, New York, NY, USA, 1994. ACM.

[59] B. Steinert, M. Grunewald, S. Richter, J. Lincke, and R. Hirschfeld. Multi-user multi-account interaction in groupware supporting single-display collaboration. In *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, pages 1 –9, nov. 2009.

[60] Yongwang Zhao, Dianfu Ma, Chunyang Hu, Min Liu, and Yonggang Huang. Socom: A service-oriented collaboration middleware for multi-user interaction with web services based scientific resources. In *Parallel and Distributed Computing, 2007. ISPDC '07. Sixth International Symposium on*, page 28, july 2007.