An Open-Source PACS Model for University Hospitals

by

Hamidreza GHADERI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR A MASTER'S DEGREE
WITH THESIS IN SOFTWARE ENGINEERING
M.A.SC.

MONTREAL, JANUARY 3, 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Alain April, Thesis Supervisor
Department of Software Engineering and Information Technology, École de technologie supérieure

Mr. Jacques de Guise, Thesis Jury President
Department of Systems Engineering, École de technologie supérieure

Mr. François Coallier, Thesis Jury
Department of Software Engineering and Information Technology, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

MONTREAL, DECEMBER 7, 2020

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

Un modèle d'interopérabilité de PACS, du domaine du logiciel libre, pour les hôpitaux universitaires

Hamidreza GHADERI

RÉSUMÉ

La gestion et l'accès à la quantité croissante de données produites dans les établissements de santé sont un enjeu important. Les systèmes d'archivage et de transmission d'images (PACS) peuvent aider à transmettre, stocker, archiver et accéder aux données d'imagerie médicale. Cependant, les PACS ne sont pas abordables pour tous les hôpitaux, en particulier ceux qui se situent dans les pays en développement. Utiliser un PACS gratuit disponible dans le domaine du logiciel libre pourrait être une solution intéressante, mais en choisir un parmi les nombreux PACS disponibles et s'assurer qu'il s'intègre bien aux autres systèmes d'information de l'hôpital est une problématique de taille. Les PACS gratuits disponibles dans le domaine du logiciel libre sont tous différents les uns des autres en termes de conception logicielle, d'architecture interne, d'interopérabilité, de support et de fonctionnalité utilisateur. Ainsi, afin de mieux comparer ces logiciels disponibles gratuitement et d'en sélectionner un qui pourrait être implémenté dans un hôpital universitaire africain, il est intéressant d'avoir une liste de critères de sélection. Dans cette recherche, tout d'abord, quatre critères de sélection sont définis afin de faire ressortir les caractéristiques requises d'un PACS qui serait utile aux chercheurs localisés dans les hôpitaux universitaires:

- critère 1: le niveau d'activité communautaire;
- critère 2: le type de licence utilisée par le logiciel libre;
- critère 3: le niveau de participation, de support de la communauté et de la documentation;
- critère 4: les fonctionnalités disponibles et les caractéristiques techniques du logiciel.

Les référentiels de code source, les sites Web des PACS et les articles publiés sont utilisés pour collecter des données pour cette évaluation. Seize PACS populaires sont évalués à l'aide de ces critères. Le résultat de l'évaluation démontre qu'Orthanc, DCM4CHE, DCMTK, Dicoogle et MRIdb sont les PACS du domaine du logiciel libre qui se sont les mieux classés. Par la suite, l'architecture logicielle du PACS Orthanc est décrite afin de l'utiliser dans une étude de cas pour l'hôpital universitaire Donka, de Guinée Conakry, en Afrique.

Des composants incontournables d'un système d'information hospitalier moderne pour la radiologie sont généralement : le système d'information hospitalier (SIH), le système d'information de la radiologie (SIR), le PACS lui-même et sa visionneuse d'images médicales. L'hôpital Donka a acquis, en 2020, un SIH, nommé eHospital, qui inclue toutes les fonctionnalités requises pour un hôpital universitaire moderne, y compris un SIR. Par contre, l'administrateur de l'hôpital n'avait pas prévu l'acquisition d'un PACS commercial nécessaire pour stocker, archiver et accéder aux données d'imagerie médicale de l'hôpital. Dans cette recherche, une étude de cas expérimente l'utilisation d'un intergiciel entre eHospital et le PACS Orthanc. L'intergiciel choisi, nommé Mirth Connect, est un projet mature de la communauté de logiciel libre qui facilite l'interopérabilité et l'échange de données de systèmes informatiques hétérogènes afin qu'ils puissent se transmettre des messages sous différents

formats, dont le FIHR/HL7 très populaire dans le domaine de la santé. Un modèle d'interopérabilité expansible est proposé afin d'effectuer l'intégration du PACS Orthanc avec le système d'information hospitalier eHospital. Les principaux composants nécessitant d'échanger des transactions sont : le SIH eHospital, le bus de communication Mirth Connect, le PACS Orthanc, une visionneuse d'images médicales et d'autres interfaces futures.

Dans le modèle d'interopérabilité proposé, différents scénarios de communication sont décrits et expérimentés afin de décrire le fonctionnement des transactions entre ces composants. Six scénarios de transactions sont décrits et expérimentés:

1. HIS et PACS (deux scénarios);
2. Modalité et PACS (deux scénarios);
3. Interface PACS et Mirth Connect;
4. Tableau de bord et Mirth Connect;
5. Visionneuse d'images et PACS;
6. Modèle TensorFlow et PACS.

Chacune des transactions traversant le bus de communication SOA Mirth Connect est expliquée et mise en œuvre pour démontrer l'implantation du PACS Orthanc à l'hôpital universitaire Donka ainsi que la mise en œuvre d'un BUS de communication SOA FIHR/HL7 permettant l'interopérabilité future de n'importe quel composant future.

Mots-clés: PACS, logiciel libre, critères d'évaluation PACS, bus d'interopérabilité SOA, SIH, SIR, FIHR, HL7, hôpital universitaire.

An Open-Source PACS Model for University Hospitals

Hamidreza GHADERI

ABSTRACT

Managing the increasing amount of data that is produced in healthcare centers is a challenging problem. Picture Archiving Communication System (PACS) helps healthcare managers to transmit, store, archive, and access medical imaging data. However, PACS are not readily affordable for all hospitals, especially those in developing countries. Using an open-source PACS could be a viable solution but selecting one and integrating it with other hospital information systems is a challenging problem for hospitals. Open-source PACS are different from each other in terms of software design, internal architecture, interoperability, support, and user functionality. Thus, in order to have a better understanding of available open-source and be able to select one, it is critical to have good selection criteria. In this research, firstly, four criteria are defined as the required characteristics of an open source PACS to be used in a university hospital, which are following:

- criteria 1: community activities;
- criteria 2: licensing models;
- criteria 3: activity, support, and documentation;
- criteria 4: enterprise functions and software characteristics.

These criteria are used to assess sixteen open-source PACS, such as available support from the software creator, future development and distribution possibility, and implemented and developed functions. To achieve this assessment, PACS project source code repositories, PACS websites and research papers are used to collect data for this evaluation. The result of the assessment shows that: Orthanc, DCM4CHE, DCMTK, Dicoogle, and MRIdb are the top-ranked open-source PACS using these criteria. Orthanc is then selected in this research to conduct an interoperability case study for the Donka university hospital in Guinea, Africa.

The main components of a modern hospital information systems for radiology are the hospital information system (HIS), the radiology information system (RIS), a PACS, and a radiology image viewer. The Donka hospital management has already acquired an HIS named eHospital which includes all the modern hospital functionality, including an RIS. But the hospital administration did not plan the acquisition of a commercial PACS required to store, archive and access the many imaging files of the hospital. In this research, the case study looks at the use of a middleware between the HIS/RIS and the open source PACS Orthanc. The middleware chosen for the case study, named Mirth Connect, is a mature open-source project which facilitates interoperability and data exchange of heterogeneous IT systems to allow them to exchange messages between each other using different exchange format, like FIHR/HL7 which is very popular in the healthcare industry.

A seamlessly extensible interoperability model is proposed to interconnect the SIH/RIS eHospital and the open source PACS Orthanc. The main components involved in the exchange of data are: the HIS/RIS, Mirth Connect, the PACS Orthanc, and image viewer, and other

future interfaces. In this model, different scenarios are defined as routine and required data transactions between mentioned components. The six data transactions scenarios experimented are:

1. HIS and PACS (two scenarios);
2. Modality and PACS (two scenarios);
3. PACS interface and Mirth Connect;
4. Dashboard and Mirth Connect;
5. Image viewer and PACS;
6. TensorFlow model and PACS.

Each of these transactions is explained and implemented with the assistance of Mirth Connect for the Donka university hospital. Each of the transactions going through the Mirth Connect SOA communication bus is explained and implemented to demonstrate how to integrate Orthanc PACS at the Donka University Hospital as well as the use of the FIHR / HL7 communication protocol allowing the future interoperability of any future component.

Keywords: PACS, open-source software, PACS evaluation criteria, interoperability SOA bus, HIS, RIS, FIHR, HL7, university hospital.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

AET	Application Entity Title
AI	Artificial Intelligence
AGPL	Affero General Public License
ASP	Application Service Provider
BSD	Berkeley Software Distribution
CAD	Computer-aided diagnosis
CBCT	Cone beam computed tomography
CMS	Clinical Management System
CNN	Convolutional Neural Network
CT	Computed Tomography
CTN	Central Test Node
DCE	Dynamic Contrast-Enhanced
DCMTK	DICOM Toolkit
DEXA	Dual Energy X-Ray Absorptiometry
DICOM	Digital Imaging and Communications in Medicine
DICOM-SR	DICOM Structured Reporting
DIN	Digital Imaging Network
DVD	Digital Video Disc
DVTK	The Healthcare Validation Toolkit
EMR	Electronic Medical Report
EPR	Electronic Patient Record

FNIR	Functional Near-Infrared Spectroscopy
FTP	File Transfer Protocol
GPL	General Public License
HIPAA	Health Insurance Probability and Accountability Act
Hi-PACS	Hospital Integrated PACS
HIS	Hospital Information System
HL7	Health Level 7
HTML	Hypertext Markup Language
ICD	International Classification of Diseases
ICR	Institute of Cancer Research
IHE	Integration the Healthcare Enterprise
IT	Information Technology
JEE	Java Enterprise Edition
JMX	Java Management Extensions
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
LOINC	Logical Observation Identifiers Names and Codes
MPI	Magnetic Particle Imaging
MRI	Magnetic Resonance Imaging
NCI	National Cancer Institute
NCPDP	National Council for Prescription Drug Programs
NIH	National Institute of Health
NM	Nuclear Medicine

NMR	Nuclear Magnetic Resonance
OHIF	Open Health Imaging Foundation
OS	Operating System
OSS	Open Source software
PACS	Picture Archiving and Communication System
PET	Positron Emission Tomography
PNG	Portable Network Graphics
REST	Representational State Transfer
RIS	Radiology Information System
SCP	Service Class Provider
SCU	Service Class User
SNOMED-CT	Systematized Nomenclature of Medicine - Clinical Terms
SPECT	Single Photon Emission Computed Tomography
SPIE	International Society for Optical Engineering
UI	User Interface
USB	Universal Serial Bus
VNA	Vendor Neutral Archive
XDS-I	Cross-Enterprise Document Sharing for Imaging
XNAT	Extensible Neuroimaging Archive Toolkit

INTRODUCTION

Imaging plays a vital role in modern medical care services and medical research (Salvador, Nogueira, & Goncalves, 2014). The management of the growing amounts of medical data (i.e., from terabytes to petabytes) that is produced by healthcare centers every year is a current concern for hospital managers (Bui et al., 2007). Picture Archiving and Communication System (PACS) is a technology for managing medical images in healthcare (Law & Zhou, 2003). It facilitates electronic access to the medical images and allows their storage, transmission, and archiving (Arora & Mehta, 2014). Having acquired digital imaging management systems, hospitals and clinics report a decrease in their imaging costs (i.e., such as material cost, physical storage space, and manual labor**)** as opposed to using traditional radiology technology (Xue & Liang, 2007). As well, hospitals report that imaging service delivery has also improved because of PACS technology. This technology has eased the imaging workflow, increased the efficiency and productivity of the imaging service, and allowed time saving overall (Liu & Huang, 2008). Furthermore, a PACS has become the basis for supporting imaging specialists' decision-making process and providing a better quality diagnosis overall (Valente, Silva, Godinho, & Costa, 2016).

The development of PACS systems dates back to the 1970s (Top, 2012) and over the years has seen several advancements. The history of PACS development could be described in key evolutionary stages (van de Wetering & Batenburg, 2014). The first stage of earlier PACS, in the 1970s, has seen the development of the initial electronic imaging system repository. In the late 1980s, this first PACS system integrated with Health Information Systems (HIS) and Radiology Information System (RIS) was created. Then in the early 1990s, the development of the international standard on Digital Imaging and Communication in Medicine (DICOM) emerged allowing standard protocols between medical devices. In the most recent evolutionary stage, the PACS workflow and server application such as the enterprise PACS and the Web-based PACS have emerged (Huang, 2010). In the USA, the practical implementation of PACS in hospitals started during the 1980s (Top, 2012) and only in a very few selected hospitals decided to use them (Duerinckx, 2003). The success demonstrated and published by these

precursors was followed by the wide adoption of the technology and PACS were implemented progressively in many hospitals all around the world. For example, they were adopted widely in Asia (Huang, 2011), Europe (Inamura et al., 2003), and North America (Huang, 2011). Today, a large number of hospitals, in developed countries, have PACS. In fact, a hospital that does not have one, in the G8 countries, is considered a hospital that has not understood the value of the technology or cannot afford it. Some developing counties are also beginning to use PACS (Mendel & Schweitzer, 2015) but the affordability level of a commercial versions of a PACS prevents a large number of them from acquiring it.

Consequently the arrival of mature open-source PACS offering provides a potential solution to this problem for these poorer countries. An open-source version of a PACS provides a foundation for implementing an imaging repository and gradually offering more advanced application when needed. In the choice of an open-source PACS solution, some factors should be considered, such as cross-platform development and deployment, compliance with the present and upcoming DICOM standards and extensibility of the solution. The goal of developing open-source PACS is to provide suitable tools that can then be used by software engineers to implement PACS functions in their hospital without the high cost demanded by commercial suppliers (Bui et al., 2007)**.**

The objective of this research is to study the state of the open-source PACS offering in order that a candidate solution can be used by University Hospitals in Africa. University hospitals have additional research and teaching responsibilities that potentially affect the functionality required from an open-source PACS solution. These additional requirements aim at teaching interns using the diagnostics and imaging processes.

CHAPTER 1

PACS OVERVIEW AND STUDY STRUCTURE

1.1 Introduction

This research thesis aims to firstly provide the background of PACS development through the last decades and its key support for medical imaging. Secondly, it investigates the potential extended role and needed functionalities of an open-source PACS solution when used by research and teaching hospitals in Africa. An experimental objective of this research project is also to experiment an open-source PACS model for an African university hospital, the DONKA hospital of Guinea. This topic will be introduced at the end of this chapter. This introduction presents an overview of the research PACS features and their availability in open-source software. It follows by identifying how artificial intelligence, particularly computer vision, could shape the future of medical imaging and the future functionalities of PACS.

Medical imaging technology processes a growing number of medical images and countless amount of related information. The necessity of a medical imaging system in healthcare is undeniable. Traditional systems have difficulties dealing with the growing demand by the clinical departments and the increasingly large number of medical images they consume. Modern medical imaging technology eliminates the need to manually file, retrieve, or transport film jackets, the folders used to store and protect X-ray film. As a result, digital medical image management is a field of research now being recognized (Xiong, Du, Nie, Huang, & Zhou, 2017).

During recent years, the PACS industry has grown, and now, it is considered a profitable industry (P. G. Nagy, 2007). Combined with available and emerging Web technologies, PACS have the ability to deliver timely and efficient access to images, interpretations, and other related data. Many medical professions use PACS for their decision making and treatment procedures and consider it a valuable tool (Valente et al., 2016).

Because of the growing importance of PACS in medical practice during the last two decades, many advantages of its use have been reported, such as facilitated image manipulation and interpretation for value-added diagnosis (Silva, Pinho, Monteiro, Silva, & Costa, 2018), as well as quick access to historical data and convenient transmission (Xiong et al., 2017). Furthermore, PACS provide support for advanced and improved patient assessment workflow, which leads to quicker healthcare service delivery and lower operational costs (Huang, 2011). In addition to these advantages, due to a higher demand in imaging services, researchers are testing and using new technologies and developing new cutting-edge PACS services that can operate on cloud computing (Teng et al., 2010), on distributed and heterogeneous computing grids (Vossberg, Tolxdorff, & Krefting, 2008), (Yang, Chen, & Yang, 2010), offer knowledge extraction using indexing engines (Costa, Freitas, Pereira, Silva, & Oliveira, 2009), and also can operate on peer-to-peer networks (Costa et al., 2011).

It is not a surprise to see that healthcare organizations have heavily invested in developing and enhancing their PACS. Manufacturers have also conducted a lot of research to develop modern, reliable, safe, and fault-tolerant PACS systems. According to the Zion Market Research study, published on October 2018, the global market for RIS (Radiology Information System) and PACS in 2017 was valued USD 2.6 billion, and it's predicted that by 2024 it will reach USD 4.3 billion. This growing interest and sophistication of functionalities lead to more expensive solutions (Kagadis, Alexakos, Langer, & French, 2012). To counterbalance the accessibility problem caused by the high price of modern PACS systems, open-source PACS have started to emerge after the year 2000. Initial open-source PACS were initially targeted to small healthcare organizations to allow them to obtain a PACS at a lower cost, with basic functionality and without too much quality compromise (Erickson, Langer, & Nagy, 2005). Open source also meant the possibility of customizing and adding PACS functionalities for healthcare organizations that could not afford a commercial product. According to Nagy, this option can quickly achieve the same goals with similar performance and features to a commercial PACS (P. G. Nagy, 2007).

In summary, we have seen that the popularity of PACS revolutionized the practice of radiology (Top, 2012). According to Wetering & Batenburg, and presented in the next section, the development of PACS can be summarized in key stages.

1.2 Background of this field of study

Picture archiving and communication system (PACS) have revolutionized the practice of radiology by changing the medical imaging process, the information communication technologies, the storage and display of medical images and related information, and the clinical workflow itself. Additional to these many impacts, PACS have the ability to integrate with different healthcare information systems such as Hospital Information System (HIS), Radiology Information System (RIS), Clinical Management System (CMS) and other medical information systems to be more integrated and effective. Progressively, all these systems need to be interrelated. Interrelation, in health systems, is facilitated by using industrial and normalized communication standards, including HL-7 and DICOM communication protocols that facilitate PACS clinical interoperability (Huang, 2011). In this section, the development of PACS during the last decades is summarized.

Digital radiology and digital image communication were firstly introduced in the late 1970s and early 1980s. In 1979, the concept of digital image communication and display was introduced by Professor Heinz U. Lemke (Huang, 2011). The idea of a "Photoelectronic Radiology Department" was introduced by Dr. M. Paul Capp (Capp et al., 1981) and his colleagues at the conference on Digital Radiography sponsored by International Society for Optical Engineering (SPIE). This team of researchers also presented a "system block diagram" describing a prototype facility located at the University of Arizona Health Sciences Center (Capp et al., 1981). The cost of managing digital diagnostic images, in a typical radiology department, was also depicted by Professor S.J. Dwyer (Dwyer et al., 1982). During the first International Conference and Workshop on PACS conference in California, in January 1982, the terminology PACS was coined. Afterward, Medical Imaging and PACS conferences

combined into a joint SPIE meeting, which was held each February in California or Florida over the next years (Huang, 2011).

Another effort emerged, in 1983, from the U.S. army "Teleradiology project" that was one of the earliest research projects concerning PACS technology in the United States. The next important PACS pilot project, managed by the MITRE Corporation and funded by the U.S. Army, was the "Installation Site for Digital Imaging Network and Picture Archiving and Communication System" (DIN/PACS) conducted in 1986. In this pilot project, the George Washington University Consortium (located in Washington D.C.) and the University of Washington (located in Seattle) with the participation of AT&T and Philips Medical Systems were selected for the implementation. Two other related projects (e.g. PACS research project started at the mid-1980s and large-scale program project started at early 1990s) were funded by the National Institutes of Health (NIH) and U.S. National Cancer Institute (NCI) during these years. These two projects were given the names "Multiple Viewing Stations for Diagnostic Radiology, Image Compression, PACS in Radiology" (Huang, 2011).

With the results of all these initiatives being published, quickly it was realized that PACS had the potential to be used at a large scale. At that time, the notion of "a large scale use" was defined as a PACS system which satisfied one of the following conditions:

1. A daily clinical operation;
2. The ability to connect to at least three modalities (Modality is a type of equipment used to acquire functional or structural images of the body such as magnetic resonance imaging (MRI), visible light, computed tomography (CT), nuclear medicine, ultrasound and radiography);
3. Having workstations outside and inside of the Radiology Department that could handle at least 20,000 radiological procedures per year. This definition separated the concept of small and large-scale use of a PACS. Even in 1996, most PACS were already meeting or exceeding this requirement (Bauman, Gell, & Dwyer, 1996).

Until the early 1990s, it was reported that PACS technologies had remained in the radiology department. The University of California, San Francisco developed the first hospital-integrated PACS (Hi-PACS) (Huang et al., 1996) in the mid-1990s. To be integrated for daily hospital use, a workflow, named Hi-PACS needed a Radiology Information System (RIS) as the engine for clinical use, and this concept opened the future PACS hospital clinical applications and development in imaging informatics. The next years showed that with its growing popularity, manufacturer and hospitals all over the world had a growing interest in researching and developing additional PACS functionality for clinical use at different levels of complexity. Huang, proposed a model having six levels of complexity associated with its method of implementation. These levels are the home-grown model, the two-team effort model, the turnkey model, the partnership model, the application service provider (ASP) model, and the open-source model (Huang, 2011).

Gradually, manufacturers and many universities researchers started to contribute their results in the public domain towards open-source PACS projects. This new phenomenon allowed healthcare centers to adapt these open-source PACS application servers and Web servers to their specific requirements without the involvement of suppliers. Many research hospital home-grown PACS development teams developed open-source PACS functionality and Web services (Huang, 2011). As a result, many open-source PACS solutions have appeared, in recent years, aimed at providing additional functionality for education, research, and clinical trials based on open-source PACS. Chapter 2 presents some of the most popular open-source PACS projects.

All of this would not have happened if the funding had not been available at the onset. The initial growth of PACS technologies was heavily funded by:

1. The US Federal Government academic research;
2. The imaging research community;
3. The manufacturers.

Funding then went to many universities where the “medical imaging informatics” domain emerged as a research specialty in universities (Huang, 2011). This is an important part of this history of PACS allowing it to become more popular and causing a significant increase in the number of hospitals equipped with PACS compared to the use of RIS, electronic patient record (EPR), and HIS (Inamura & Kim, 2011) (see Figure 1.1). Table 1.1 shows the penetration of PACS technology over time.

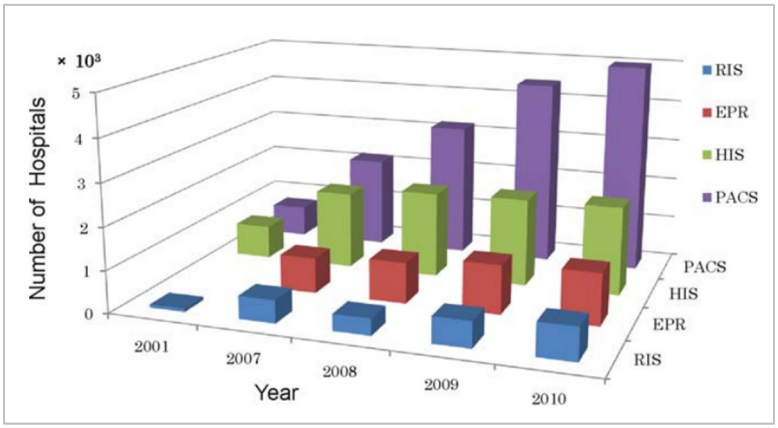


Figure 1.1 PACS installation in comparison with the RIS, EPR, HIS from 2001 to 2010)
Taken from Inamura and Kim (2011, p. 186)

From a clinical use perspectives, researchers started to study the domain of Computer-Aided Diagnosis (CAD) in the early 1980s, and it gradually became one of their most important clinical support tool. CAD functionalities enhanced the radiologist diagnostic accuracy as it could be used as a second reader. Quickly, CAD functions became integral functions of PACS (Doi & Huang, 2007). This integration provided CAD interoperability to the PACS image resources and increase its clinical value. Next, the integration of CAD with HIS/RIS/PACS (Hospital and Radiology Information Systems) became the most popular research topic for DICOM (Digital Imaging and Communication in Medicine), HL7 (Health Level 7) and Integration the Healthcare Enterprise (IHE) projects. They all aimed at developing integrated workflows to comply with the Health Insurance Probability and Accountability Act (HIPAA) requirements to allow an interoperable and integrated healthcare service. In recent years,

developed CAD-PACS integration kit based on IHE workflow profiles and DICOM-SR (structured reporting) are used frequently and allow unified integration of CAD and PACS (Le, Liu, & Huang, 2009).

Another interesting use of PACS is for surgical operations. EPR system with PACS images can be used during surgical applications. In order to develop a patient-centric information system, the professionals use the concept of Web-based EPR. During the pre-operation consultation, all medical and surgical information of a patient can be acquired, and throughout the operation, live information and surgical information can be collected in real-time. Also, during post-operation, the patient recovery data can also be acquired. After the patient leaves the hospital, all this information will be available for further diagnostic, research and patient follow-up (Huang, 2003).

Table 1.1 Medical imaging, PACS and imaging informatics R&D progress over time
Taken from Huang (2011, p. 172)

Decade	R&D Progress	R&D Topics
The 1980s	Medical imaging technology development	CR, MRI, CT, US, DR, WS, storage, networks
The late 1980s	Imaging systems integration	PACS, ACR/NEMA, DICOM, high-speed networks
The early 1990s	Integration of HIS/RIS/PACS	DICOM, HL7, Intranet, and Internet
Late 1990s - present	Workflow & application servers	IHE, EPR, enterprise PACS, Web-based PACS
The 2000s - present	Imaging Informatics	CAD, image contents indexing, knowledgebase, decision support tools, image-assisted diagnosis, and treatment

As we have seen, standardization has played a central role in the popularity of medical imaging for clinical use. Before the presence of the DICOM international standard there was no easy and standard way of communicating between different vendors systems. Each hospital purchased a proprietary system form a vendor, including work stations, modalities, archive, and film printers all on an isolated network. In the early 1990s, the Radiological Society of North America commissioned two groups to develop DICOM communications tools. The

DCMTK which is a collection of open-source applications and libraries implementing large parts of the DICOM international standard, was developed by the OFFIS group (Oldenburg, Germany). DCMTK includes software for constructing, analyzing, converting DICOM image files, sending and receiving images over the network connection, handling offline media, and other features. Also, the central test node (CTN), which is an application that implements a simple image archive was developed by the Electronic Radiology Lab at the Mallinckrodt Institute of Radiology (locate in St. Louis, Missouri) was used to support cooperative demonstrations by medical imaging vendors. Everyone are permitted to use DCMTK and CTN for handling and simulating the DICOM international standard transactions, and the source code could help the industry understand and use DICOM faster. These open-source libraries remain open to the public today and are used by everyone to implement and test DICOM components with their PACS software. These initiatives have transformed the medical imaging industry from an ad-hoc approach to today's best-of-breed industry (P. Nagy, 2007).

The previous paragraphs presented a literature review of PACS related topics summarized from many conferences and research papers dated from 1982 to 2010. However, in more recent years, researchers apply new technology to develop PACS enhanced functionality in order to extend the reach of PACS data for education, research, and clinical trials. One of the important use of PACS data, in a teaching hospital, is the "research PACS". Doran et al. (Doran et al., 2012) study the required functionality of a research PACS. They mention some of desirable features of a research PACS in their paper, such as: data visualization, flexibility and security when accessing data, the need to sort PACS data according to arbitrary criteria, the need for image co-registration between time points, having access to metadata describing the relationship of digitized histologic data and noninvasive imaging data, the possibility of creating data processing pipelines and finally the need to audit of the results of any data processing done on PACS data. Some of these features are already available and provided by the open-source project named Extensible Neuroimaging Archive Toolkit (XNAT) software developed at the Washington University. Doran et al. (2012) propose that a research PACS has additional data available, allowing specialized research and teaching applications to retrieve and use this data. They describe a prototype research PACS framework. This framework has

been used by a clinical MR imaging group, at the Institute of Cancer Research (ICR), to develop dynamic contrast-enhanced (DCE) MR imaging, diffusion analysis, data visualization, distortion correction, and breast screening functionalities. All of these functionalities are being included in the XNAT system research PACS framework and demonstrates how functionalities are originally developed to work as standalone functions can interact between themselves by means of enhancing PACS data sharing (Doran et al., 2012).

Another example of applying new technologies to develop PACS is published by Zhang et al. (2018). This team of researchers developed a cloud-based research PACS functionality for diabetic retinopathy prescreening with the help of deep learning algorithms. They used a convolutional neural network (CNN) technique on 30,000 annotated images from their research PACS database to design a prescreening functionality. They claim that their initial research results are very encouraging and that such PACS based research application, joined with a CAD functionality could be valuable for the next generation of PACS for research hospitals (S. Zhang et al., 2018).

This PACS functionality review showed how Integrating PACS data with other health data and systems can provide new opportunities for researchers and practitioners. The next section explores the opportunities of future research in this domain in more details.

1.3 The opportunities of future research PACS

We have seen that clinically, PACS have demonstrated their many benefits when used with patients, as they provide secure and easy access to the clinical image repository. We have also seen that PACS basic functionality has been reported to be insufficiently flexible to be used for research and academic purposes. Doran proposes that PACS are not solely to blame but instead it's the academic and research processes that fail in using this type of technology because of: "an excessive dependence on individual researchers to keep track of a large amount of data, a significant overhead in organizing and indexing imaging files, problems with data duplication and possible data corruption, data loss, and an excessive need to respect patient

data protection" (Doran et al., 2012). It is also reported that clinical and research workflows have many differences. Research data and clinical data are often stored and processed by different workflows in a hospital. In a typical clinical workflow, images are transferred from local imaging devices to the institutional PACS and typically processed by the hospital proprietary software. To ensure patient information security, the patient imaging data sharing between hospitals is still difficult because institutional PACS data and images are kept behind security firewalls.

In comparison, in a typical research workflow, imaging data is obtained from a variety of sources, and it is stored with additional information annotating and enriching it (i.e., patient history data, environmental data, medication, and genetic data). This combination of data is generally not used together in clinical management. For research purposes, anonymized patient images are used to collaborate between researchers and often shared on workstations, either via secure File Transfer Protocol (FTP), DICOM protocols, or via a portable medium like the Universal Serial Bus (USB) drive or a Digital Video Disc (DVD). Alternatively, the researcher may also download images from an open-source repository available online. After processing this data, he can store the original image and the annotated or processed image on his local storing space without any centrally available electronic history of processing or typical organizational structure. Doran also reports that researchers also analyze medical images using many different applications, which are executing on the different operating system and host computers (Doran et al., 2012).

We have discussed that in a typical hospital, PACS are generally designed to fulfill routine radiology tasks for the patients, but in terms of research, it has limited use because of its lack of flexibility. Because of regulatory compliance, integration of PACS with custom third-party software is currently still difficult. Doran reports that the development of specific PACS functionality for scientific research purposes has too small a market potential and does not seem commercially interesting for manufacturers yet. Researchers also report that since they manage their own research process, they experience a constant need to upgrade their technologies when PACS manufacturers release new versions of their software. This leads to

an excessive impact and redesigns of their research process protocols and can also invalidate some past analysis results causing much rework (Doran et al., 2012).

1.4 Research gap

It is planned that PACS that will include artificial intelligence (AI) will likely replace the PACS that do not have this imbedded functionality in the future (Dugar, 2018). Computer vision and artificial intelligence, using PACS digital images, have the ability to automate/assist the many human intensive visual tasks such as: processing, analyzing, and understanding the patient images in order to emit a diagnostic. It is also reported that using machine learning and artificial intelligence algorithms improve dramatically the detection accuracy where humans cannot compete. The increasing amount of research results, in this area of innovation, has proven that human analysis of PACS images is less precise than its computer vision counterpart. However, the skills of the professionals are still required to: train, validate, and approve the computer vision system results at this time. While computer-vision paired with AI has had impacts on many industries, the availabilities of proven and tested functionalities, for radiologists, is still limited.

Many PACS commercial suppliers have failed to integrate computer vision and AI in their current PACS commercial offerings. It is also claimed that there is a need for PACS commercial offering for the following diagnostics: CT lung nodule CAD, mammography CAD, brain CT anomaly CAD, fracture detection CAD, and chest x-ray shadow detection CAD. Secondly, PACS offerings are needed also for the evolution detection of sclerotic follow-up and tumor follow-up.

In summary, CAD and AI have the potential to provide support for radiology departments and radiologists in the future. These future functionalities need to be integrated into future PACS software and use DICOM CAD format standards for easier interoperability. A trend in this research domain is to try to have the radiologist reporting workflow integrate computer-vision and AI algorithms to improve the accuracy and efficiency of radiology diagnostics. Further

away, is the research concerning the context-aggregation and AI functionalities that could help the radiologist to extract information, from different healthcare systems, and use it as well. When radiologists have more information, such as blood test result, renal function tests, tumor markers, and inflammatory markers tests results, they will likely make more accurate clinical diagnostics.

Dugar (Dugar, 2018) also reports that computer vision and content aggregation will be the most interesting new research field to produce novel technologies for the radiology professionals and researchers in order to help them make smarter and more efficient diagnostics by using endoscopy and histopathology images, as well as other clinical images and documents (Dugar, 2018). Research and development of new PACS functionalities for research and education purposes are currently in its infancy. In the context of African university hospitals, their requirements for research PACS functionalities and training PACS functionalities are still unclear. In fact, there is little information available about their current and most urgent training needs for their interns.

1.5 Objectives

The objective of this research is to develop an open-source PACS model to be experimented in a university hospital in Africa. The following sub-objectives of the research are:

1. To identify the functionality required by “research PACS” to be useful for research and teaching hospitals in Africa;
2. To assess the available open-source PACS with regards to the fit of their functionality with the specific requirements of research and university hospitals in Africa and choose an open-source PACS candidate for experimentation;
3. To adapt the selected open-source PACS for an experimentation (i.e. the case study) applied to the Donka university hospitals in Guinea, Africa.

1.6 Organization of this thesis

The first chapter of this thesis, the introduction, provided a general overview of this research project. It is followed by a literature review that will be presented in chapter two. Chapter three investigates the current open-source PACS and identifies a candidate to be used in a case study with the Donka hospital of Guinea, Africa. And then, the PACS interoperability model is proposed. Chapter four will present the Donka hospital case study objectives as well as the "research PACS" model to guide university and teaching hospitals in Africa in the future and will present the results of the case study. The final chapter will summarize the results of this research, its limitations as well as future research directions.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

We have seen that this research aims to propose a "research PACS" interoperability model for university hospitals that conduct teaching and research on top of normal clinical activities. As well, since this study is undertaken to try to help African University hospitals, one key literature review focus is to investigate and understand some leading open-source PACS systems functionality to have a better understanding of the best available open-source PACS software design, internal architecture, interoperability, and user functionality for this research. This will also help in assessing their potential to be used in a teaching and research university hospital located in Guinea, Africa.

Imaging is challenged to continually identify, develop, embrace, and promote new services that have profound impacts on how healthcare services are delivered to patients. Specific interactions, computational intensity, and a large range of applications are done using imaging. Sharing source code and programs have played an important role, and accelerated the adoption of the DICOM international standard (Erickson et al., 2005). The radiology open-source community is a vibrant collection of users and developers working on collaborative software projects. The open-source community, which includes several commercial vendors, has a rich history in supporting the success of the DICOM international standard and nowadays is promoting interoperability by embracing the in Integrating the Healthcare Enterprise (IHE) (Nagy, 2007). IHE promotes the coordinated use of established standards such as DICOM and HL7 to address specific clinical needs in support of optimal patient care. Systems developed in accordance with IHE communicate with one another better, are easier to implement, and enable care providers to use information more effectively. Some benefits of open-source software for medical imaging are:

1. Reducing support costs (Christensen & Raynor, 2003);
2. Reducing development costs;

3. Adding business and patient value (Nagy, 2007).

In this chapter, in order to understand the architecture of open-source PACS and DICOM in more detail, in the next sections, the Modality, popular open-source PACS architectures, the DICOM file format, and the DICOM network protocol are explained.

2.2 What is the modality in medical imaging?

A modality, in medical imaging, refers to a process and technique of creating visual representations of the inner body organs for medical intervention and/or clinical analysis. Also, a modality could be used for the function representation of some tissues or organs. Medical imaging reveals the internal structure concealed by the skin and bones; this helps doctors diagnose and treat diseases. There are different types of medical imaging, explained in the following list.

1. **Radiography**: is an imaging technique that uses gamma rays, X-Rays, or similar ionizing radiation and non-ionizing radiation to view the internal form of objects. There are many types:
   a. Projection radiography: creating images by exposing an object to X-rays;
   b. Computed tomography (CT scan): using ionizing radiation in conjunction with computers to create images of both hard and soft tissues;
   c. Dual energy X-ray absorptiometry (DEXA or bone densitometry): which is used for osteoporosis (a type of disease that bone weakening increase the risk of a broken bone) tests;
   d. Fluoroscopy: used to view the movement of tissue, in order to guide a medical intervention or a joint repair/replacement.
2. **Magnetic resonance imaging (MRI scanner) or nuclear magnetic resonance (NMR):** is a radiology technique used to form images of the anatomy and the physiological processes of the body. MRI is a medical application of nuclear magnetic resonance (NMR);
3. **Nuclear medicine:** refers to the use of radioactive substances in the diagnosis and treatment of disease. Nuclear medicine has two common imaging modalities: single photon

emission computed tomography (SPECT) uses gamma rays and scans the level of biological activity and positron emission tomography (PET) to visualize and measure metabolic in the body;

4. **Ultrasound:** uses high-frequency broadband sound waves that are reflected by tissue to produce images. Is largely used for imaging the fetus in pregnant women and detecting tumors;
5. **Elastography:** is a relatively new imaging modality and emerged in the last two decades to draw the elastic properties of soft tissues;
6. **Photoacoustic imaging:** is a hybrid biomedical imaging modality that has recently been developed based on the photoacoustic effects. Early tests show that it could be used in skin melanoma detection, functional brain imaging, blood oxygenation mapping, and vivo for tumor angiogenesis monitoring;
7. **Tomography:** is a technique of imaging by sections. The main methods are CT, PET, and MRI;
8. **Echocardiography:** is a technique which is using ultrasound to image the heart;
9. **Functional near-infrared spectroscopy (FNIR):** is a widely accepted technique for brain imaging technique is used for functional neuroimaging; and
10. **Magnetic Particle Imaging (MPI):** is used for tracking superparamagnetic iron oxide nanoparticles with high sensitivity and specificity. In medical research, MPI is used to image cell tracking, neuroperfusion, and cardiovascular performance.

2.3 Open-source PACS architectures

Open-source software (OSS) development is a popular approach for creating and distributing software at low costs. OSS is transforming the software industry (Morgan & Finnegan, 2014) and is used in different domains (von Krogh & von Hippel, 2006). OSS has also demonstrated significant results in the software industry (Morgan & Finnegan, 2014). Many firms' success now depends on OSS (Gulati, Puranam, & Tushman, 2012). Developers' attention and Knowledge (Grant, 1996) are key factors assessed by the best OSS projects (Singh, Tan, & Mookerjee, 2011). The notion of attention is "noticing, encoding, interpreting, and focusing

of time and effort". Attention refers to the effort and time expended on a project by a developer. Attention has been identified as a key strategic resource (Ocasio, 1997). Researchers highlighted the relation of developers' effort and attention to the level of OSS projects' success. Online repositories such as Github and Sourceforge provide functionalities to facilitate the development and allows developers to join and leave the project at will (Seidel & Stewart, 2011). This is how OSS projects draw knowledge from a wide range of professional developers (Ye & Kishida, 2003) and also from a broad array of other open-source projects (Singh et al., 2011). Contributors introduce knowledge in different ways, such as posting comments in project discussions or better by contributing to a project's source code (Hann, Roberts, & Slaughter, 2013).

According to Kenwood (2001), the decision between commercial products and OSS is based on three main factors, which are including:

1. Direct costs (e.g., software price) and indirect cost (e.g., end-user downtime);
2. Benefits of using each product such as performance and enterprise functions;
3. More intangible criteria like the quality of support.

Costs vary when acquiring/using a commercial PACS based on various factors such as the number of diagnostic and size of the practice. This cost varies between $5,000 to $100,000 in the literature. Also, costs should consider the entire life-cycle costs of using a PACS, such as customizations and support costs. Open-source PACS are available freely and can provide some level of support through forums, from product distributers and/or from volunteers. Although PACS vendors could provide more completle PACS functionalities for their customers that require higher PACS performance, recent progress in open-source developments makes it possible for healthcare organizations to adopt an open-source PACS and customize it according to their requirements. The quality of support depends on each PACS open-source project where a forum and mailing lists are available to their users.

Open source PACS software provides some basic, necessary, and cost-effective functionalities for clinical use and even for some for research and training activities. Additionally, they

provide a starting point for software developers to enhance their features in order to better fit what is required by a specific hospital (Lebre, Bastião, & Costa, 2019). Apart from offering the developers a startup software code base, good software architecture and modularity is essential if you are to quickly extend the functionalities of an open-source PACS solution. Therefore, evaluating open-source PACS applications is helpful for selecting the right software for your need.

In the next subsections, the architectures of DCM4CHE, DCMTK, Dicoogle, MRIdb, and Orthanc are described.

2.3.1 DCM4CHE

DCM4CHE is a popular open-source PACS which is mostly developed before 2010 (Figure 2.1) that has been used in many hospitals. In addition, it is used in the architecture of other PACS (e.g., Dicoogle) for development. DCM4CHEE, as a cross-platform application, is a collection of open-source utilities and applications about archiving and managing images and based on JEE, JMX, and the JBoss Application Server (Maniadi, Spanakis, Karantanas, & Marias, 2015).

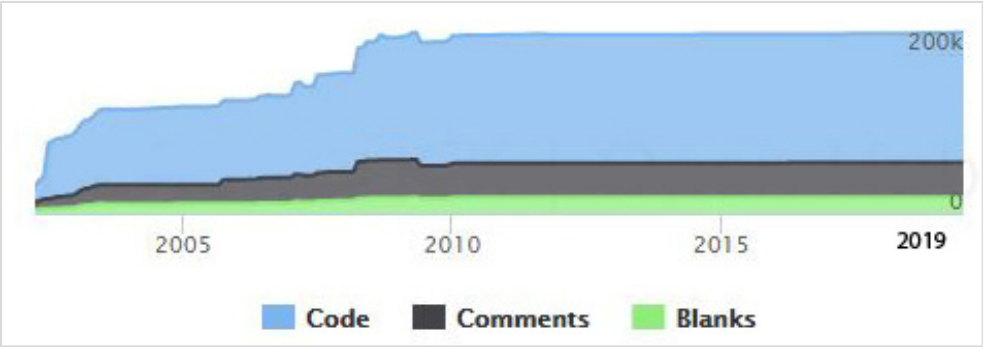


Figure 2.1 DCM4CHEE line of code and programming language pie chart
Taken from Openhub (2019)

This framework ensures broad compatibility and versatility and excellent performance (Valeri et al., 2015). DCM4CHEE provides HL7 and DICOM services and interfaces that are required for retrieving and storing data and managing the workflow in a complex environment like diagnostic imaging (Warnock, Toland, Evans, Wallace, & Nagy, 2007).

DCM4CHE contains some software components such as PACS server (Archive 2 and Archive 5), toolkit and utilities (DCM4CHE 2 and DCM4CHE 5), and web viewer (Weasis, Oviyam, and Mayam). DCM4CHEE has a web-based application for administration tasks, which is compatible with popular database management systems such as Oracle, SQL Server, MySQL, and PostgreSQL. Each hospital is responsible for managing its own imaging data. Imaging data can be uploaded through a user interface.

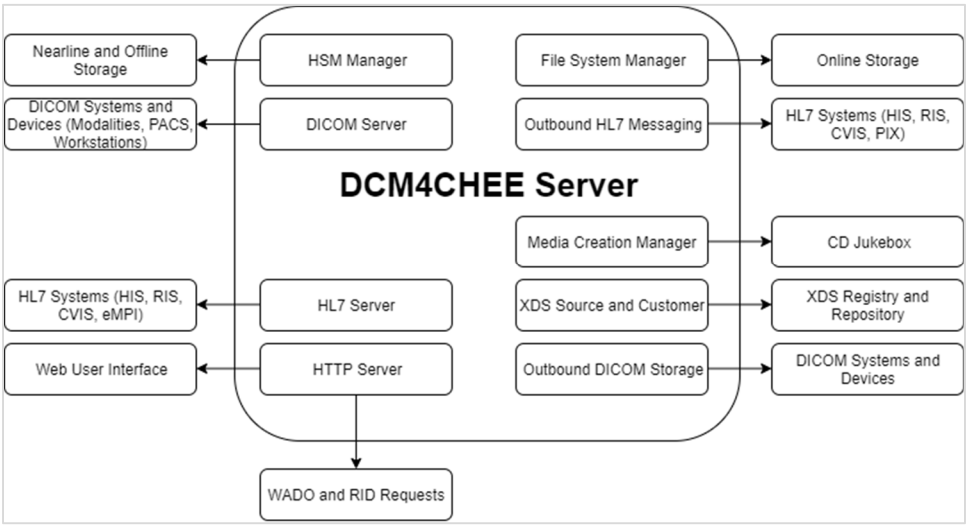


Figure 2.2 DCM4CHEE Server system architecture
Adapted from Warnock et al. (2007, p. 126)

When files upload to the PACS, based on their annotations, DCM4CHEE automatically indexes and stores them using their DICOM data elements. Imaging specialists can filter patient data, for example, based on modality. Also, files can be download. Users can also delete DICOM files (Maniadi et al., 2015). The architecture of DCM4CHEE is depicted in Figure 2.2.

2.3.2 DCMTK

DCMTK is open-source software that contains a collection of applications and libraries, including functionality for constructing, examining and converting DICOM image files, sending and receiving images over a network connection, handling offline media, and demonstrative image storage; and worklist servers. It has been used in many different situations, for example, as a tool, as a building block for research projects, and as a commercial product. DCMTK is mostly written in C/C++, and in recent years, has grown significantly (see Figure 2.3).

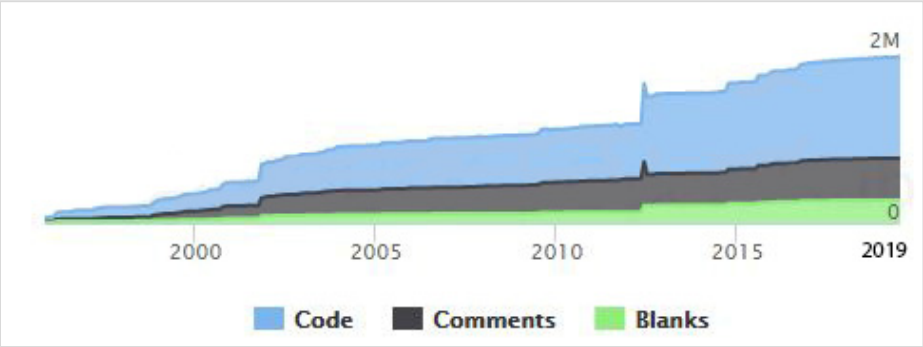


Figure 2.3 DCMTK line of code and programming language pie chart
Taken from Openhub (2019)

2.3.3 Dicoogle

Dicoogle is an open-source PACS (Valente et al., 2016) with a modular architecture (Figure 2.4) (Pinho & Costa, 2016). Its software development kit (SDK) and plugin concept encourages researchers and developers to develop new features easily. Dicoogle uses DCM4CHEE for implementing the DICOM standard functionalities (Costa et al., 2011). Also, its modular architecture is often used for teaching, education, and clinically in hospitals (Lebre et al., 2019).

The main features of Dicoogle, according to the project homepage, are included:

1. Expansible (based on plugin-based architecture and SDK);
2. Scalable (tested with over 25 million of indexed DICOM objects and optimized for big data paradigms);
3. Indexing/Query Engine (Enable DICOM study retrieval and knowledge extraction and support for complex query/retrieval solutions);
4. Web user interface;

Dicoogle has been used for various purposes, both in research and clinically. For example, it is being used as DICOM data mining tools (Santos, Bastião, Costa, Silva, & Rocha, 2011), (Valente et al., 2013), by third entities to regional PACS, and as an educational tool for students and interns. Besides, it addressed many challenges of healthcare institutions such as performing content-based image retrieval (Valente et al., 2013) and interfacing with Cross-Enterprise Document Sharing for Imaging (XDS-I) (Santos et al., 2011). Due to its software architecture, it could support this range of different applications with very varied requirements. Its extension mechanisms have permitted users to use present DICOM functionality and discover new directions in a non-intrusive manner.

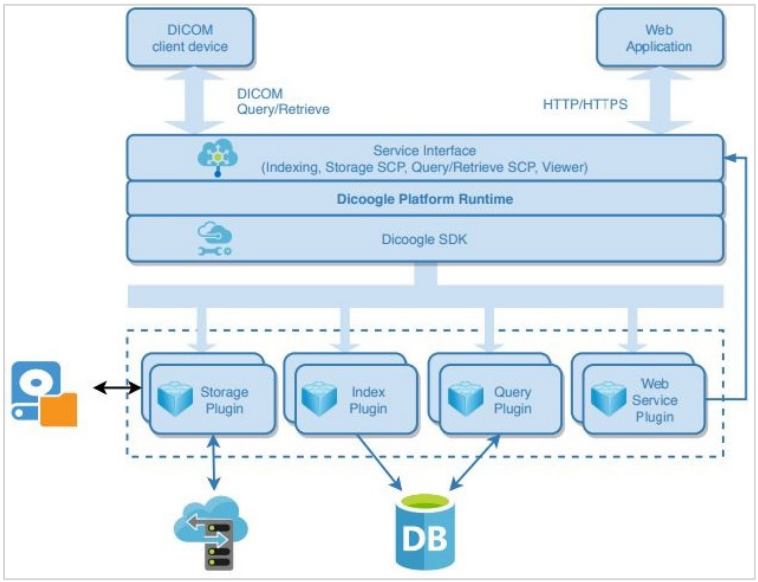


Figure 2.4 Dicoogle General Architecture.
Taken from Lebre et al. (2019, p. 3)

Due to its low difficulty in using it, the development time is reported to be reduced, whether it is a data analysis task or the development of a new experimental feature. By using the existing DICOM functionality, developers can quickly prototype, adapt, and develop features for their use case. It is reported that it provides many benefits for both small to medium medical healthcare centers and research institutions. Also, because of its low-end hardware requirements and easy deployment, Dicoogle is popular. To deal with the fast-changing PACS environment, using an extensible plugin-based software is important, as it facilitates rapid prototyping, experimentation, and validation while encouraging code and functionality reuse (Valente et al., 2016).

2.3.4 MRIdb

MRIdb is an open-source PACS that is suitable for storing and managing MRI (magnetic resonance imaging) datasets. It was designed for researchers and clinicians (Woodbridge, Fagiolo, & O'Regan, 2013). MRIdb is software composed of a suite of tools and utility scripts and a bespoke Web application. It depends on a number of other components which include a PACS, a relational database system that is scalable and an authentication service (see Figure 2.5). MRIdb is based on DCM4CHE (Woodbridge et al., 2013), a highly configurable and mature open-source PACS. It handles raw image and thumbnail retrieval facilities, metadata extraction from images into a relational database schema and raw image, and low-level functions of image archival from scanners using the DICOM protocol.

MRIdb natively provides Web and DICOM interfaces, but the former is complex and provides extensive data manipulation and administrative facilities, whilst the latter enables access to un-anonymized data. MRIdb also provides visualization; export functionality with enforced preservation of anonymity and data integrity; utilities for auditing; system monitoring; data migration and study management. It is designed to support image management and clinical research in the area of epidemiological and imaging genetics research. The User Interface (UI) is implemented using Hypertext Markup Language (HTML). The software is cross-platform, and it can execute on any modern Web browser. The back-end of the software is written in

Python and Java and is recommended to operate on Linux. MRIdb is freely available from the project Website and distributed under the GNU General Public License v3.0 (Woodbridge et al., 2013).

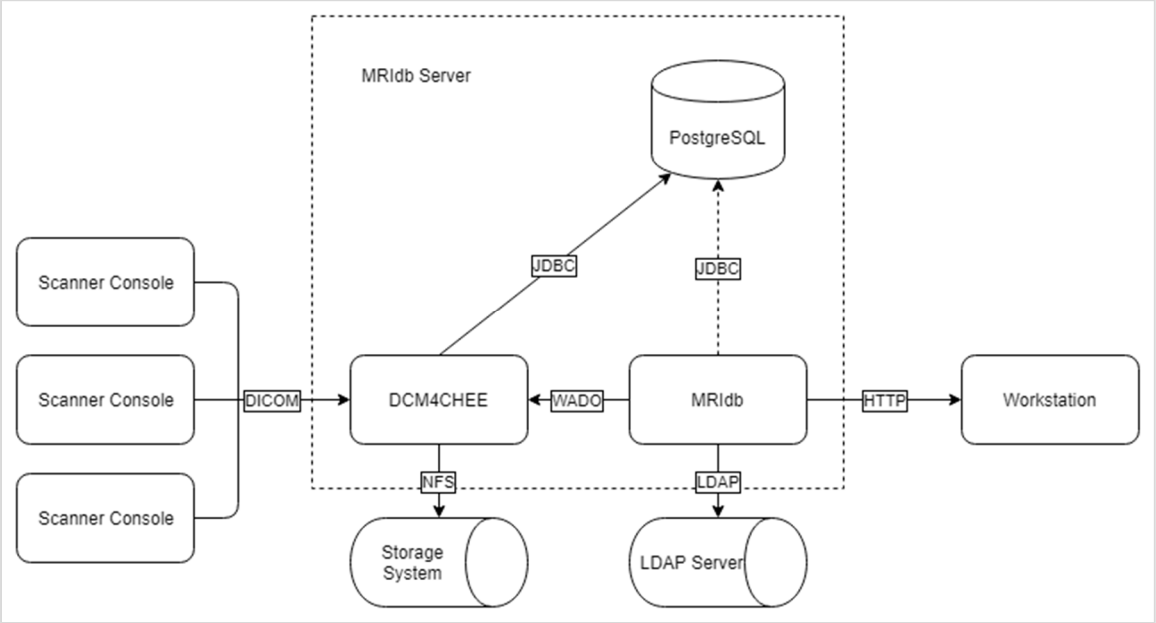


Figure 2.5 MRIdb Server system architecture
Adapted from Woodbridge, Fagiolo, & O'Regan (2013, p. 887)

A turnkey distribution of MRIdb is available in the form of a virtual appliance where it can be deployed without a lot of technical knowledge. By using this facility, the lengthy installation process is facilitated. This includes specification of the location of the storage space allocated for image archival, the address of the lightweight directory access protocol (LDAP) server used for user authentication, and the e-mail address of the system manager (to whom errors are automatically reported).

Without requiring a complex installation, it provides patient data management in a secure and scalable manner (Woodbridge et al., 2013). MRIdb was written mostly in Java, and its modular architecture is depicted in Figure 2.5. The last version of the MRIdb dates back to 2014 and this PACS has not intensively been enhanced in recent years.

2.3.5 Orthanc ecosystem

Orthanc, is an open-source PACS that provides a powerful environment to optimize and automate the imaging flows, which are always specific to each hospital. The Orthanc server has a lightweight vendor archive that can be extended using plugins. Orthanc also uses DCMTK in its Orthanc server for DICOM C-Store, C-Find, and C-Move. The advanced programming interface of Orthanc server allows research engineers and software developers to readily develop external software dealing with medical images with very little knowledge needed of the DICOM standard (Jodogne, 2018). According to the project's homepage, it is designed to meet the following benchmarks:

1. To ease DICOM scripting for clinical routine (e.g., C-Find, C-Store, and C-Move SCU);
2. To ease data management for medical research and clinical routine (mini-PACS);
3. To bring DICOM images to the Computer Vision community (to ease the automated analysis of medical images).

To meet above benchmarks it offers:

1. Fast, Lightweight (written in C++) and mostly developed in recent years (Figure 2.6);
2. Standalone (all the dependencies can be statically linked);
3. Cross-platform (at least, Windows, Linux, and OS X);
4. Compliant with the DICOM standard (as it is built on the top of DCMTK);
5. Programmer-friendly (PNG, JSON, REST API).

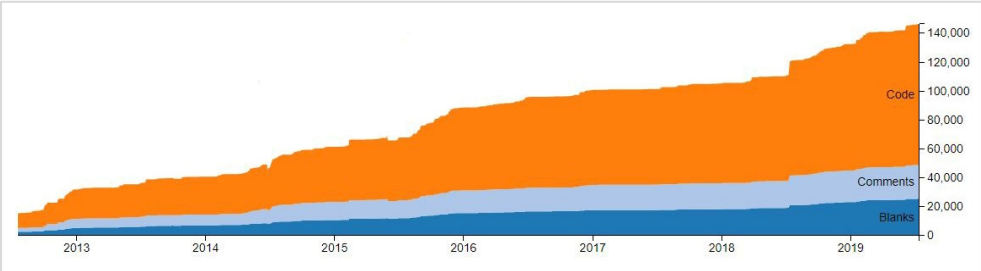


Figure 2.6 Orthanc core line of code and programming language pie chart
Taken from Orthanc-server (2019)

The software ecosystem of Orthanc contains different modules, which results in a growing number of source code, as depicted in Figure 2.7.

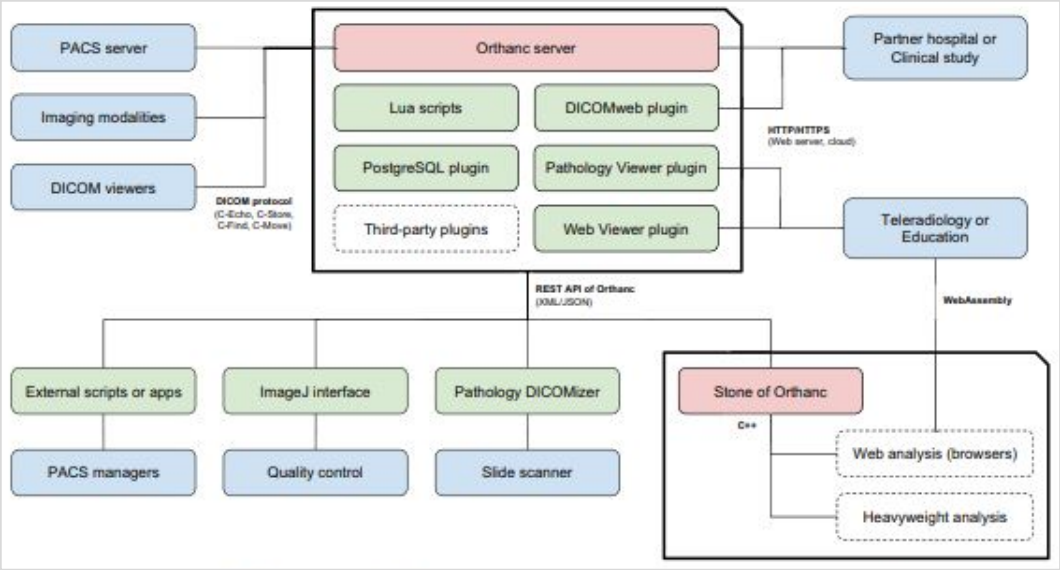


Figure 2.7 Orthanc ecosystem. The main components are shown in red color. The green components are Orthanc plugins, and the blue color shows application related to clinical research, academic activities, and medical practice.
Taken from Jodogne (2018, p. 343)

Orthanc aims to deliver a simple and powerful standalone DICOM server. It is designed to facilitate the DICOM flows in hospitals and automated analysis of medical images. Orthanc hides the complexity of the DICOM format and protocol, and it provides this opportunity for its users to have more focus on the content DICOM files. It can run on many popular operating systems such as Windows, Linux, and OS X and turn them into a DICOM store (e.g., a mini PACS system).

The Orthanc server is placed at the core of the Orthanc ecosystem and has a lightweight and standalone architecture. Thus, it does not require any complex database administration and installation of third-party dependencies. Two main features of the Orthanc, in comparison with the other open-source PACS, are its REST API that has a plugin mechanism that will be explained in the following sections (Jodogne, 2018). The Orthanc software ecosystem has

different components, depicted in Figure 2.7 in this section, we will review each part of the Orthanc architecture.

2.3.5.1 Orthanc Server

As we have seen, the Orthanc server is the main component of the Orthanc software ecosystem. The Orthanc server is a Vendor Neutral Archive (VNA) that can receive, index, store, and transmit medical images using the DICOM standard. Its internal architecture (see Figure 2.8) provides the simplicity of the packaging and deploying almost immediately. This architecture has the following properties:

1. Small footprint: it can run in many different hardware platforms such as virtual machines on the cloud, desktop computer, Raspberry Pi, or even from a USB stick;
2. Cross-Platform: It is written in C++, and it can package for the various operating system;
3. Standalone: It comes with SQLite, and it is not required for the installation of any framework such as Java, .Net, or any external software to run;
4. Compliant with the DICOM standard by using the DCMTK toolkit.

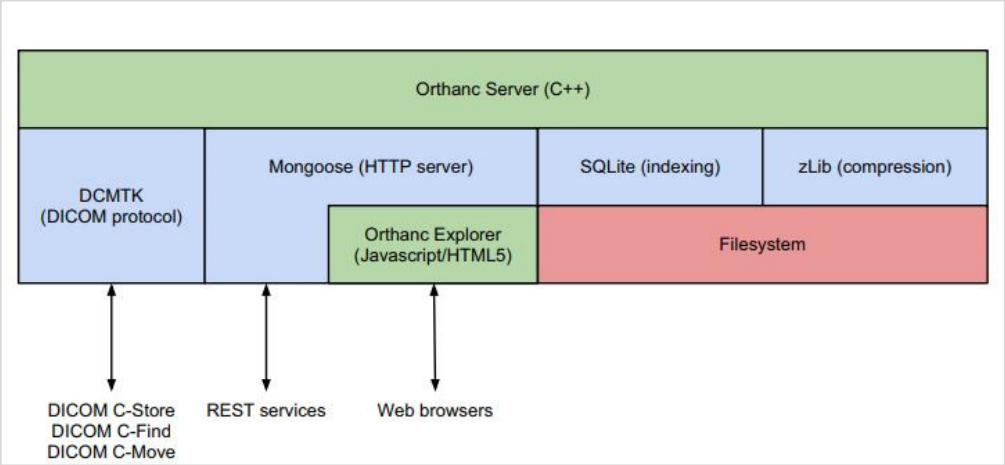


Figure 2.8 The layer of Orthanc server software architecture
Taken from Jodogne (2018, p. 343)

2.3.5.2 Orthanc Explorer

The Orthanc explorer provides an embedded Web user interface. Users can open and interact with the Orthanc explorer, which is an embedded Web user interface that allows the users to interact with the DICOM server through a browser (see Figure 2.9). It is based on an HTTP server, but it is configured to use HTTPS to be more secure. The primary functions of these components are: browsing the content of the Orthanc server; anonymization; manual upload of DICOM instances; download of a ZIP archive, query; and retrieve from remote modalities (imaging devices). The Orthanc explorer facilitates data management for medical research and clinical routine, and it is easy to install, task-specific, and fine-grained DICOM stores. These servers can connect different DICOM modalities, medical departments, and hospitals. For example, it is used at the University Hospital of Liege radiology department, which is interconnected to a nuclear medicine department to enable backups to the contours and to the research database that collects in-room images produced by treatment machines (Jodogne, 2018).

Figure 2.9 Orthanc explorer screenshot
Taken from Orthanc-server (2019)

2.3.5.3 Lua Scripting

Lua scripting is an embedded scripting engine that can be used to drive DICOM flows in an automated way and is used in Orthanc. Thanks to this major feature, Orthanc can be adjusted to any medical workflow without being driven by an external script. It means that by using this scripting engine, users can define routing rules as needed. It can also monitor the arrival of the DICOM instances and react to them if some condition is met (Jodogne, 2018). A sample of a Lua script is depicted in Figure 2.10.

```
function ReceivedInstanceFilter(dicom, origin)
   -- Only allow incoming MR images
   if dicom.Modality == 'MR' then
      return true
   else
      return false
   end
end
```

Figure 2.10 A sample of Lua scripting
Taken from Orthanc-server (2019)

2.3.5.4 REST API

For basic auto-routing tasks, Lua Scripting is very useful, but for more complex DICOM automation, the Orthanc server offers REST APIs. It is based on the internal HTTP and provides this opportunity for software developers to have full access to all the core features (Richardson & Ruby, 2008). Developers can use different programming languages such as Python, C#, and Java. It is quicker to develop a new application by giving the responsibilities of handling the DICOM files into the Orthanc server. Orthanc server works as a high-level bridge between the DICOM standards and software standards such as XML, JSON, HTTP, and PNG (Jodogne, 2018).

2.3.5.5 Orthanc Plugins

The core of the Orthanc server can be extended by using the Orthanc plugins. C and C++ programming languages are used for developing plugins and can be used to add new endpoints in the REST API or to serve new Web pages.

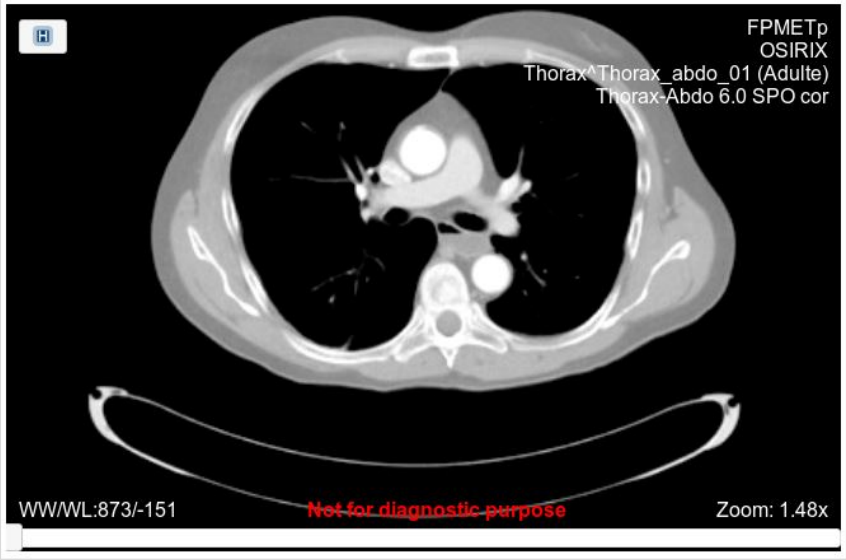


Figure 2.11 A screenshot of the Orthanc web viewer plugin
Taken from Orthanc-server (2019)

The Orthanc Web viewer is a good example of an Orthanc plugin which can be used to view a range of DICOM images (see Figure 2.11), and it can be used to meet basic teleradiology needs. The plugins engine of the Orthanc provides the opportunity to replace it with another open-source plugin. For example, the default Database plugins in Orthanc architecture is SQLite, which is capable of storing approximately 5000 DICOM instances, but it can be inadequate for enterprises as they require more capacity. Consequently, the PostgreSQL open-source plugin can be used in Orthanc as a reliable database to manage more than 10TB imaging data (Jodogne, 2018).

2.3.5.6 Digital Pathology

Recently open-source support of DICOM for digital pathology has been added to the Orthanc ecosystem (Jodogne et al., 2017). Introducing digital pathology is a new approach to Telepathology. An application of telemedicine provides a long-distance practice of anatomopathology (Jodogne, 2018) which is valuable for intraoperative consultation (Ribback, Flessa, Gromoll-Bergmann, Evert, & Dombrowski, 2014), consultation from experts (Farahani & Pantanowitz, 2016), research and education (Marée et al., 2016), and pathology archiving (Webster & Dunstan, 2014), the with electronic format images. Digital pathology has two main parts, which are included: the DICOMizer and a plugin for Orthanc server (Jodogne, 2018).

2.3.5.7 Stone of Orthanc

Finally, the Stone of Orthanc is a function for rendering 2D and 3D medical images. It supports the multiplanar reconstruction of volume images (MPR), reslicing; radiotherapy (rendering of RTDOSE and RT-STRUCT); layering (fusion of images), and accurate physical 3D world coordinates. Stone of the Orthanc can retrieve DICOM images through REST API from an Orthanc server. It is a lightweight, cross-platform C++ toolkit. Because it is entirely standalone and entirely written in C++, it can readily be embedded into heavyweight software (bindings to Java and C# are in active development) or into native mobile applications (iOS and Android). In addition, it is compatible with the emerging Web Assembly technology and without any installing browser's extension can run C++ applications in the Web (Haas et al., 2017).

As a result, it is possible to quickly develop applications from a single codebase for displaying and analyzing medical images and use in any platform (Web, native, or mobile) (Jodogne, 2018). The examples of rendering by the stone of Orthanc toolkit is depicted in Figure 2.12.

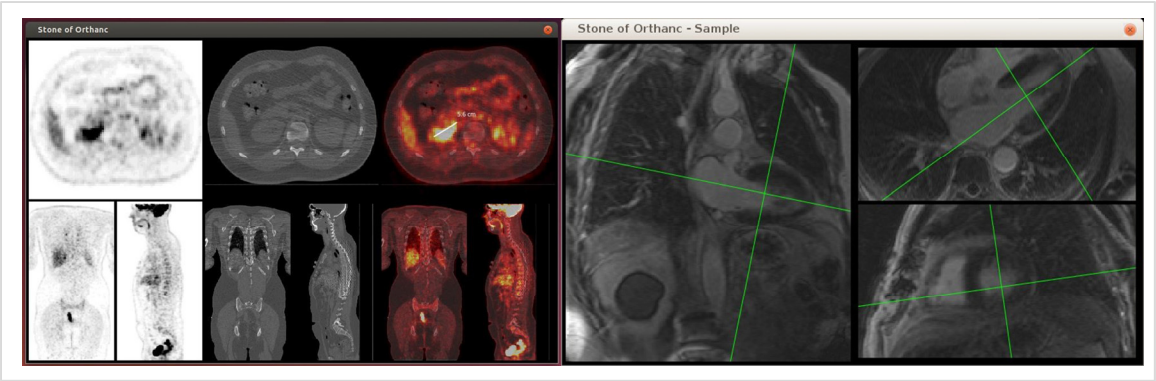


Figure 2.12 The stone of Orthanc toolkit rendering sample
Taken from Farina et al. (2004, p. 59)

2.3.5.8 Summary of Orthanc

As presented here, Orthanc has been used for different application around the world such as the automated routing or exchange of DICOM instances (both outside and inside of research centers and hospitals), the Web diffusion of (possibly anonymized) medical images, the education of stakeholders (medical physicists or physicians) and the industrial R&D or scientific research about new imaging modalities or software. The Orthanc server as a lightweight, novel, robust DICOM store provides rich scripting capabilities based on the use of the Lua engine, and REST API. The built-in capability with Web technology makes it very multipurpose. It is fully standalone and cross-platform, which facilitates the process of deployment. Several plugins revolve around the Orthanc server, and they can be used with the core features of Orthanc, by adding support for enterprise-ready databases, by improving the user interface, by interfacing with administrative servers of the hospital, by implementing recent additions to the DICOM standard such as digital pathology support or DICOMweb, and by providing teleradiology solutions. The stone of Orthanc is a C++ lightweight toolkit and CPU-based rendering engine for medical images. It is a building block to create heavyweight software such as Web interfaces (through WebAssembly) or mobile applications (Android or iOS) that require to display or process medical images. The Orthanc ecosystem is completely open-source and well documented. The Orthanc server is under the license of GPL3, and the official plugins are mostly released under the AGPLv3 license. The Orthanc ecosystem is

designed to be as simple and open as possible, to eliminate the learning time of DICOM standard, to the advantage of research centers, hospitals, companies, public organizations, or general audience. More information can be found in the online Orthanc Book (https://book.orthanc-server.com/).

The DICOM standard, further explained in the next section, is used for transmitting, storing, and exchanging medical images that are created by using imaging techniques (see Section 2.2) so that these images and their patient information can be shared with other hospital information systems.

2.4 Understanding the use of DICOM format with Orthanc

The DICOM format enables the ability to interconnect medical imaging devices such as printers, servers, scanners, workstations, and PACS from various manufacturers with other hospital information systems. It manages exchanges between two medical devices or software that are able to receive the image and patient data. Each device must support:

1. A DICOM Conformance Statement, which states which DICOM classes they support;
2. A file format definition;
3. A network communication protocol that uses TCP/IP to communicate among systems.

The DICOM standard is divided into two parts: the DICOM file format and DICOM network protocol, which are explained in the next sections.

2.4.1 DICOM file format

The medical information, which is encoded by a DICOM file, is in fact a data set that has the form of the key-value associative array. Each array could be a list of data sets, which is called a sequence. This architecture, which is similar to a JSON or an XML formatted file, leads to a hierarchical data structure internally. In the DICOM terminology, each key is called a DICOM tag. An official dictionary is available and normalizes the list of the standard DICOM tags,

which are identified uniquely by two 16-bit hexadecimal numbers. The DICOM file format also specifies which DICOM tags are mandatory or optional for each type of imaging modalities such as PET, CBCT, NM, MR, CT. This specification is known as a storage service-object pair. The DICOM standard also allows companies to develop non-standard, proprietary tags for their own use.

The DICOM tag PixelData (0x7fe0, 0x0010) is associated with the image, and the related image could be compressed by using popular image formats like JPEG. In addition, the DICOM file can act as a wrapper around encoded using H.264 or MPEG-2 protocols. A DICOM image can be multi-frame, meaning that it encodes an array of various image frames. This feature can be used to encode uncompressed video sequences that are referred to as 2D+t or cine images (e.g., for Ultrasound imaging).

The Orthanc software can send, receive, and store all kinds of DICOM images, and it supports all standard transfer syntaxes. In addition, it can convert the most uncompressed images to the PNG format. This file format was chosen by Orthanc as it is lossless, is natively supported by many popular software, programming frameworks, browsers, and capable of encoding up to 16bpp integer pixels. When previewing a DICOM image within Orthanc explorer, an on-the-fly conversion to PNG image occurs.

The patient study workflow of Figure 2.13 shows that a patient is related to a set of medical imaging studies, and each study contains a set of series. Each series is related to a set of instances. Multiple series of images can be related to an imaging study. The PET and CT series, like nuclear medicine, contain at least two separate series. Moreover, any kind of imaging study usually produces a set of separate series. In general, series could be considered as either a single 2D image (e.g., standard digital radiology), a 2D+t cine sequence, or a single 3D volume (as in a CT-scan). However, a series might also encode different files such as a single PDF report, a 3D+t image (i.e., a temporal sequence of 3D images), and a structured report.

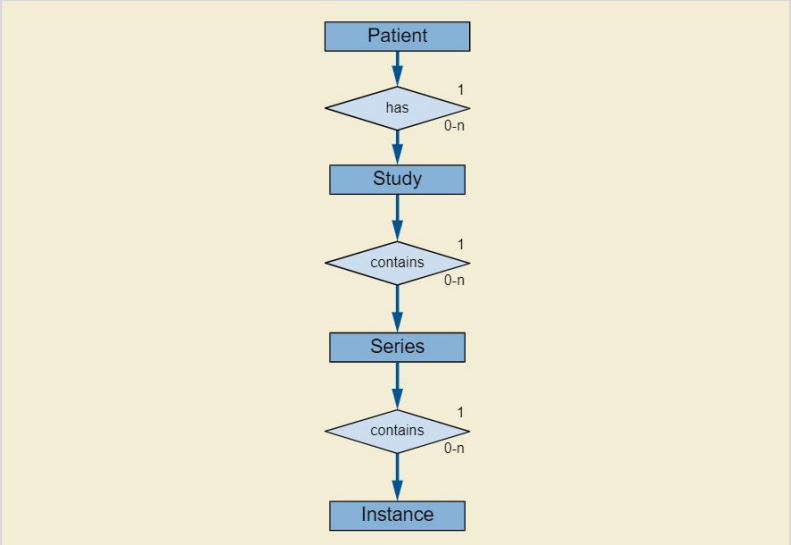


Figure 2.13 UML diagram shows a patient's study workflow Taken from Orthanc-server (2019)

For each of these four types of DICOM resources (e.g., patient, study, series, and instance), which is depicted in Figure 2.13, the DICOM standard defines a module as a set of DICOM tags that explain these resources. For example, a patient module contains the DICOM tag 'PatientName', and 'SeriesDescription' is part of the series module. Any storage service-object pair can be decomposed into a set of modules that make sense for its associated type of modality, and whose conjunction forms encode all the medical information.

2.4.2 DICOM network protocol

The DICOM protocol is known as one of the first instances of Web services, before the availability of REST and SOAP. It provides basic functionalities, which are:

1. A 'C-Echo' command: Test the connection between two devices;
2. A 'C-Store' command: Send images from the local imaging device to a remote device;
3. A 'C-Find' command: Search the content of a remote device;
4. A 'C-Move' command: Retrieve images from a remote device.

Figure 2.14 shows an overview of high-level transactions between the client and the server. The DICOM protocol uses TCP/IP in order to make a connection (i.e., the connection between a DICOM client and a DICOM server is also called an association) between a client of a DICOM service (is called a Service Class User) and a server that handles the request (is called Service Class Provider). Furthermore, each imaging device has a symbolic name, which is called an application entity title (AET) that is assumed to be unique inside the hospital Intranet. Therefore, IP address, TCP port, and AET are required for identifying a DICOM server.

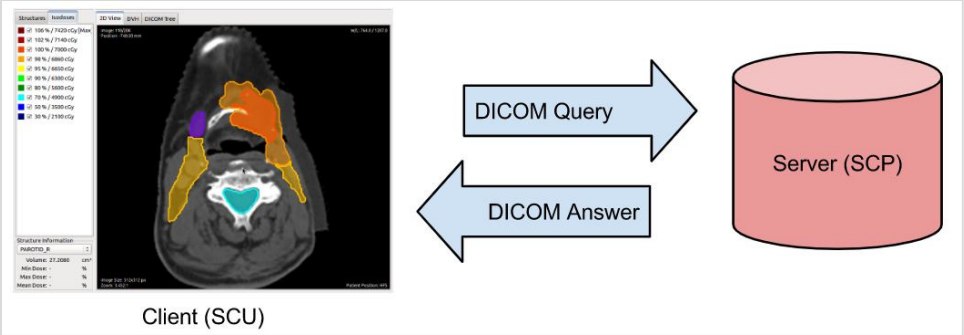


Figure 2.14 Transaction between Service Class User (SCU)
and Service Class Provider (SCP)
Taken from Orthanc-server (2019)

Orthanc can act as both as a DICOM server and as a DICOM client, depending on the value of the parameters set in its configuration file. To configure the Orthanc DICOM server, 'DicomServerEnabled' must be set as true, 'DicomAet' set to a reserved AET, and 'DicomPort' set the TCP port of the DICOM server. On the other hand, in order to configure an Orthanc DICOM client, the list of the remote DICOM servers (for each remote server should provide a symbolic name for the server that will be displayed by Orthanc Explorer, the AET of the remote server, its IP address, and its DICOM port) into 'DicomModalities' option.

The 'C-Echo' command helps users to check the connectivity of the DICOM protocol in the hospital Intranet. In practice to test a connection, first, the PACS administrator should check the TCP-level connectivity and then issue the 'C-Echo' from the client to the server to test the DICOM-level connectivity. Another command is the 'C-Store' command (Figure 2.15), which

is sending images (e.g., DICOM instances) to the server. Orthanc can act both as a C-Store server (SCP) and as a C-Store client (SCU), which can send and receive DICOM files.

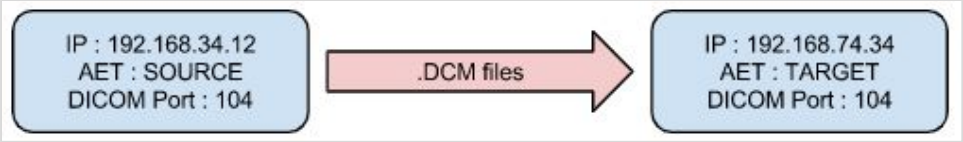


Figure 2.15 C-Store command transaction diagram
Taken from Orthanc-server (2019)

The ‘C-Find’ command (see Figure 2.16) is used to search a list of DICOM resources (e.g., patients, studies, and series) that are available at the remote DICOM server. These resources include patients, studies or series, which should be specified. Some filters tags are available. They describe the resources that users are looking for. The ‘C-find’ command is depicted as follows:

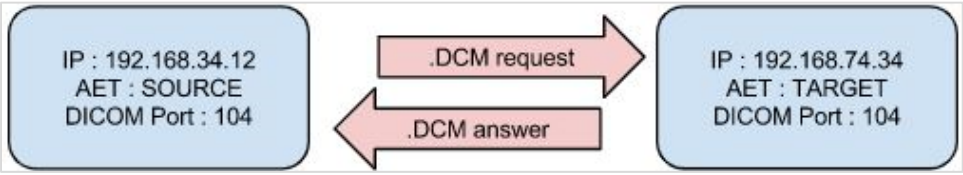


Figure 2.16 C-Find Transaction diagram
Taken from Orthanc-server (2019)

The ‘C-Move’ command (see Figure 2.17) is used to locally retrieve the result of the ‘C-Find’ command. These two sets of commands are known as the query/retrieve mechanisms, and they are the core of exchanging a DICOM file within the hospital systems. Whenever an imaging device calls a ‘C-Move’ command, it asks a DICOM server to transfer some of the resources to another DICOM server. ‘C-Move’ command drives a ‘C-store’ between two remote DICOM servers. The following diagram shows the ‘C-Move’ command.

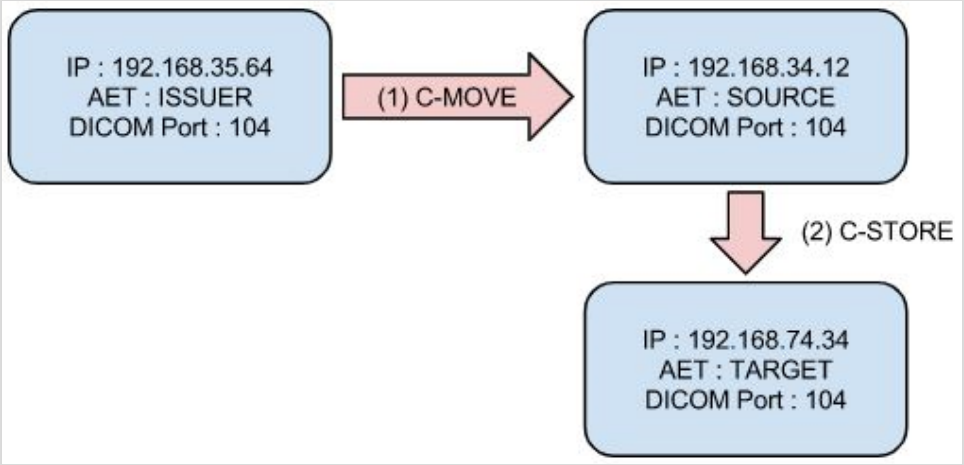


Figure 2.17 C-Move and C-Store transaction diagram
Taken from Orthanc-server (2019)

DICOM format and its functions are explained in the current section; in the next section, a laboratory test model is designed followed by a case study model to be trialed for the Donka university hospital is presented.

2.5 Conclusion

In this chapter, different imaging techniques (see Section 2.2), popular open-source PACS architectures (Section 2.3), and the DICOM format (Section 2.4) are explained. In the next chapter, open-source PACS are evaluated to recognize the most prominent open-source PACS in recent years, and for remain chapters, the selected PACS is used for the development of required functions for research-PACS to meet the requirement of Donka university hospital.

CHAPTER 3

ORTHANC MODEL SIMULATION

3.1 Introduction

In the previous chapter, popular open-source PACS architectures were explained. In this chapter, first, the available open-source PACS are evaluated (section 3.2). After this, the simulation of a model using a Modality Emulator and the Orthanc PACS in section 3.3, a model is proposed to experiment its implementation for a university hospital (i.e., the Donka Hospital of Guinea) (section 3.4).

In this literature review, various open-source PACS solutions are introduced. In this chapter, some of the most popular open-source PACS are evaluated using the proposed criteria in order to find a suitable software candidate to be experimented in this research. Therefore, before choosing an open-source PACS, it is necessary to compare them. Four criteria are extracted from the article "Open Source in Imaging Informatics" by Nagy (2007) to evaluate open-source PACS are including:

- **criteria 1**: community activities;
- **criteria 2**: licensing models;
- **criteria 3**: activity, support, and documentation;
- **criteria 4**: enterprise functions and software characteristics.

In the next section, some available open-source PACS will be evaluated using these criteria.

3.2 Evaluation of open-source PACS

Many websites, such as Medevel, Medfloss, and Idoimaging, which publish articles in the field of medical and science projects, are trying to evaluate and rank the available open-source PACS. These websites provide a list of top open-source PACS (see Table 3.1) by considering different criteria like end-user feedbacks to rank available open-source PACS. According to

these studies, see Table 3.1, Orthanc, Dicoogle, and DCM4CHE are considered as top open-source PACS solutions available currently.

Table 3.1 Top open-source PACS ranked by Medevel, Medfloss, and Idoimaging websites

Rank	Medevel	Medfloss	Idoimaging
1	Orthanc	MRIdb	DCM4CHE
2	Dicoogle	Orthanc	Orthanc
3	OHIF	Dicoogle	DCMTK
4	JVSdicom server	Xebra	ConQuest
5	EasyPACS	OSPACS	Dicoogle
6	NeurDICOM	OpensourcePACS	MRIdb
7	PacsOne Server	ClearCanvas	PACSsoft PACS
8	PACSsoft PACS	ConQuest	MyFreePACS
9	DCM4CHE	CDMEDIC PACS WEB	PacsOne Server
10	J-PACS	DCMTK	-
11	-	DCM4CHE	-

To compare leading open-source PACS, the following criteria have been investigated: community efforts, updating activities, project forum activity, Wikis documentation, open-source licenses, website appearance/usefulness, utilization, ease of installation, technical support, mailing list activity, the functionality offered, data standard, operating system (OS) and programming languages as well as developers' facilities offered. These software characteristics are categorized into four groups:

1. Community activities;
2. Licensing models;
3. Activity, support, and documentation;
4. Enterprise functions and software characteristics.

Each is presented in the next four subsections.

3.2.1 Community activity

The success of open-source software is often related to how mature it is in its community. Maturity in this context relates to its constant development/improvement as well as the level of collaboration and quality control. Few open-source projects evolve to a high level of maturity.

Table 3.2 Evaluating open-source PACS by developers' activity, updating project activity, and community activity

PACS	Fork	Watch	Star	Commits	Last Commit Date	Release	Final Version	Final Release Date	Forum Topics	Forum Member
DCMTK	96	40	195	9590	08/19	40	3.6.3	02/18	4651	3032
DCM4CHE	389	114	527	2271	08/19	56	5.15.0	11/15	4601	2417
Orthanc	40	21	-	3352	08/19	24	1.5.7	05/19	1598	713
Dicoogle	63	30	134	1122	04/19	7	2.5.0	12/17	-	-
ConQuest	10	7	19	1055	06/19	-	1.4.17	04/13	2076	106K
MRIdb	11	6	16	418	08/18	2	1.1.0	09/14	-	-
ClearCanvas	381	100	280	1621	04/15	3	13.2	04/15	-	-
EasyPACS	30	11	60	5	04/15	-		04/15	-	-
PacsOne Server							6.5.1	01/15	-	-
PACSsoft PACS	4	1	2	17	11/17	-	-	-	-	-
NeurDICOM	4	2	17	110	06/18	-	-	-	-	-
CDMEDIC	-	-	-	-	-	-	2.17.2	02/13	-	-
OSPACS	2	6	33	127	11/17	1	-	-	-	-
Xebra	-	-	-	-	04/08	2	1.0	04/08	-	-
opensourcePACS	-	-	-	-	-	-	-	-	-	-
JVSdicom	-	-	-	-	-	-	-	-	-	-

It has been observed that for every 100 developers who are using and open-source software, merely ten developers are likely to submit feature requests or bug reports and only a few developers are likely to contribute and submit a fix or enhancement. According to Eric S. Raymond, an open-source pioneers, "many eyes make bugs shallow". Due to the participants

of highly focused developers in collaborative development, open-source projects can compete and even be superior, in quality and functionality, in many cases to commercial software (Nagy, 2007).

Forums and shared documentation, in Wikis, provide a substantial level of customer support for many community-based open-source projects. Users and contributors can find answers to their questions by reviewing topics, reported by others who have faced the same problem and resolved it or even can create a new topic related to their question. A high level of pertinent forum activity is one of the useful indication of a mature open-source project. Active and vibrant forums provide responsive developers support from volunteers and code contributors. The last update, or release time, of a software project is another way to judge the maturity of an open-source project. Mature open-source projects update their software monthly, weekly, or in some cases daily. Working developers and non-programmers in a vibrant community together is another key to assess an open-source project. The mix of these types of contributors in an online community could guarantee that the software is not a developer-only tool, but one with wider appeal and utility (Nagy, 2007).

To sum up this subsection, the contribution of developers and users in a project, updating and last release time, project forum activity, and shared documentation, in Wikis, are good indicators and useful measures to be used to rank open-source PACS. Table 3.2 summarizes information extracted from the project source code repositories (e.g., Github, Sourceforge, and Bitbucket), which is sorted by the number of project fork, final release date and forum topics count. DCMTK, DCM4CHE, Orthanc, and Dicoogle have the highest rank in Table 3.2 in comparison to the other open-source PACS projects. Some developers used the DCM4CHE and DCMTK libraries in their hospital projects to handle the DICOM implementation, and these two projects are often chosen to develop a PACS from the ground up instead of starting from many other open-source PACS offerings such as Dicoogle, MRIdb, and Orthanc.

Because the purpose of open-source development is removing common contribution, development and distribution restrictions, the copyright on the open-source project is sometimes called “copyleft”. Open source projects typically use two types of licensing models:

Berkeley Software Distribution (BSD) or GNU General Public License. These two types of licensing styles are explained in the next subsection.

3.2.2 Licensing Models

In the process of selecting an open-source PACS for further development, product license needs be considered as a very important selecting criteria. There are several license model for OSS, but the Berkeley Software Distribution (BSD) and GNU Project General Public License (GPL) are the most popular licensing models. The BSD license model says you may download a program and use it for non-commercial or commercial products (Hackländer, Martin, & Kleber, 2005). The main conditions are including:

1. Credit the authors in the source code or reproduce the copyright in binary distributions;
2. May not sue the creator, if the software doesn't work as you think it should;
3. May not use the creator's name to endorse the product.

Table 3.3 Open source PACS license

PACS	License	PACS	License
Orthanc	GPLv3.0	CDMEDIC PACS	GPLv2
DCMTK	GNU 2.1 - BSD	PacsOne Server[1]	GPL
Dicoogle	GPLv3.0	OSPACS	MIT
DCM4CHE	GPLv2.0	ConQuest	License[2]
ClearCanvas	GPLv3.0	opensourcePACS	Not Listed
MRIdb	GPLv3.0	EasyPACS	Not Listed
PACSsoft PACS	GPLv3.0	NeurDICOM	Not Listed
Xebra	GPLv2	JVSdicom	Not Listed

The BSD license model is considered a business-friendly license model because companies can use it, modify it, and even sell the code without paying its authors. The CTN, which

[1] Basic edition version
[2] Refer to the project website section: Administrative / Licensing Contact, original MicroPACS (https://ingenium.home.xs4all.nl/dicom.html)

implemented the DICOM international standard, was release with a BSD open-source license. Many companies do contribute back to OSS projects that use a BSD-style license, because it is easier to implement a parallel copy of the source code and continually sync and update these different versions of the source code. Collaborating in an OSS project could also have direct benefits in terms of visibility for the companies.

The GPL open-source license model requires that if you release a modified product to the public, the modified source code must be shared with the original creators (Hackländer, Kleber, Martin, & Mertens, 2005). This approaches helps in the fact that the improvements to the OSS project will come back and coherently be delivered again to the public. Many developers that create software use a GPL-style license for non-commercial use (e.g., home and academic use). But GPL is more restrictive for commercial use. The GPL-style license allows free use for end-users but when a company sell the OSS as part of one of their solution, there is a need for royalties. It means that software is free to use, but if it is sold, some portion of the profits should go to the founders. Table 3.3 presents an overview of the popular open-source license style used in PACS open-source projects.

3.2.3 Activity, Support, and Documentation

This subsection assesses the projects activity, support and documentation using the following criterion:

1. **Website appearance and documentation**: good and current documentation for a project could be a good indicator of a successful project. Writing a document for a software project is often the last task that developers want to spend effort on. In the best open-source projects, users also collaborate with developers in preparing and keeping up to date a comprehensive documentation. Documents of a mature open-source project will include: installation guides, screenshots, user guides, and developer's guides;
2. **Activity and utilization**: statistics provided by source repositories such as Sourceforge and Github could indicate also a successful project. These websites contain some activity measures such as: how often the project has been forked or downloaded; when was the last

update the code; the bug reports open and closed, the number of contributors; the number of subscribers to the project; and information about activity of the bulletin board;

Table 3.4 Evaluating PACS by website appearance and documentation, activity and utilization, ease of installation, technical support forum, and mailing list activity.

PACS	Website appearance & documentation	Activity & utilization	Ease of installation	Technical support forums	Mailing List Activity	Platform	Website
Orthanc	•••	•••	•••	•••	+	CP	www.orthanc-server.com
DCMTK	••	•••	•••	•••	+	W/L	https://dicom.offis.de/dcmtk.php.en
DCM4CHE	••	•••	•••	•••	+	CP	www.dcm4che.org
ConQuest	•	•••	••	•••	-	W/U	https://ingenium.home.xs4all.nl/dicom.html
Dicoogle	•••	•••	••	-	+	CP	www.dicoogle.com
MRIdb	•	-	••	•	-	CP	www.imperial.ac.uk
PacsOne Server	•	-	••	•	+	CP	www.pacsone.net
NeurDICOM	•	•	•	-	-	-	-
EasyPACS	•	-	•	-	-	CP	http://mehmetsen80.github.io/EasyPACS/
PACSsoft PACS	•	-	•	-	-	CP	www.pacssoft.com
OSPACS	•	-	•	-	-	W	https://archive.codeplex.com/?p=ospacs
CDMEDIC	•	-	•	-	-	M/U	http://cdmedicpacsweb.sourceforge.net/
ClearCanvas	•	-	•	-	-	W	www.clearcanvas.ca
opensourcePACS	•	-	•	-	-	CP	-
Xebra	-	-	•	-	-	-	-
JVSdicom	-	-	-	-	-	W	http://jvsmicroscope.uta.fi/

3. **Ease of installation**: an easy installation process is another element that could be a decisive criteria in choosing an open-source software over another. Software can operate on different platforms (i.e. Operating Systems), such as Windows, Linux, and Mac OS. This does not mean that the software will install easily as a plug-and-play every time? Poor installation documentation and insufficient validation tests on different hardware platforms

is the most important cause of installation failures. If other users report that a an open-source project is hard to install, it could be a sign that the project is not mature enough;

4. **Technical support forums**: the existence of an active support forum for an open-source project, is a good sign that a large group of active contributors are helping each other to resolve the issues, evolved the software and get the most value out of the application. In a mature open-source community, response time, when a support request is issued, is typically short even for the most challenging questions from volunteers and code contributors.

Table 3.4 shows a quality rating for each criteria assessed. Three bullets (•••) indicate good development, resources, high activity and utilization in the recent year, and a software that is easy to install. Alternatively, one bullet (•) demonstrates less development, low activity and utilization and a software that is hard to install. Finally, a minus (-) indicates that not enough information was available on that characteristic to assess it. Ease of installation criteria is assessed by averaging documentation quality, platform and technical support quality. It shows that Orthanc, DCMTK, DCM4CHE, ConQuest, and Dicoogle obtain the best results. These open-source PACS software provide good documentation (e.g., Orthanc book and Dicoogle learning pack) for the developers, researchers, and users. Orthanc google group, DCMTK developer community, DCM4CHE google group, and ConQuest forum provide a platform for users and developers to discuss with experts related to their issues. These open-source PACS platforms help developers and users with the installation process, the adaptation, and maintenance.

3.2.4 Enterprise functions and software characteristics

Next, we investigate each open-source PACS different built-in functionality, which are typically: Image Archiving, Image Management, Image Communication, Image Processing, Image Viewing, and Image Distribution. Some PACS operate on limited or specific operating systems but other work on many (i.e. they are called cross-platform software).

Table 3.5 Open source enterprise functions and software characteristics

PACS	Image Archiving	Image Management	Image Communication	Image Processing	Image Viewing	Image Distribution	Worklist Provider	Application Integration	Other	**Platform**	**Programing Language**	**Extensibility**
DCM4CHE	·	·	·	·			·	·		Cross-Platform	Java	API/Library
Orthanc	·	·	·		·	·	·			Cross-Platform	C++	API/Library Scripting
Dicoogle	·	·	·	·	·	·				Cross-Platform	Java	API/Library
ConQuest	·	·	·	·			·		.	Windows/Unix	C/C++ Pascal	API/Library
opensourcePACS	·	·	·	·	·				.	Cross-Platform	Java	
DCMTK	·	·	·	·				·		Cross-Platform	C/C++	
CDMEDIC	·	·	·	·		·				Mac OS/Unix	Java	
ClearCanvas	·	·	·	·	·					Windows	C/C#	
MRIdb	·	·							.	Cross-Platform	Java	
OSPACS	·	·	·	·	·					Windows	C#	
Xebra		·	·		·	·			.		Java	
PACSsoft	·				·					Cross-Platform	C#	
EasyPACS					·					Cross-Platform	JS/Java	
PacsOne	·									Cross-Platform	PHP	
NeurDICOM											Python	API
JVSdicom										Windows	C++	

Next, the easiness of programmers to adapt the software to his/her hospital-specific needs is another concern to be considered when choosing an open-source PACS. Modular software architecture and the availability of development plugins help developers when adapting/evolving a software. A software that provides RESTful API is preferred in our assessment as it is often better architecture than software who do not use it.

Table 3.5 considers the number of built-in functions, platform, and extensibility options. DCM4CHE, Orthanc, Dicoogle, and ConQuest PACS have the best results according to these criteria.

3.2.5 Assessment Result

Overall, we have evaluated and compared 16 open-source PACS projects using a number of characteristics regrouped in four criteria:

- **criteria 1**: community activities;
- **criteria 2**: licensing models;
- **criteria 3**: activity, support, and documentation;
- **criteria 4**: enterprise functions and software characteristics.

Table 3.6 Open Source PACS assessment results using four criteria

Rank	Criteria 1	Criteria 2	Criteria 3	Criteria 4	Result 1 (BSD/GNU License)	Result 2 (All License Model)
1	DCMTK	Orthanc	Orthanc	DCM4CHE	Orthanc	Orthanc
2	DCM4CHE	DCMTK	DCMTK	Orthanc	DCM4CHE	DCM4CHE
3	Orthanc	Dicoogle	DCM4CHE	Dicoogle	DCMTK	DCMTK
4	Dicoogle	DCM4CHE	ConQuest	ConQuest	Dicoogle	Dicoogle
5	ConQuest	ClearCanvas	Dicoogle	opensourcePACS	MRIdb	MRIdb
6	MRIdb	MRIdb	MRIdb	DCMTK	CDMEDIC	ConQuest
7	ClearCanvas	PACSsoft	PacsOne	CDMEDIC	ClearCanvas	EasyPACS
8	EasyPACS	Xebra	NeurDICOM	ClearCanvas	PACSsoft	PacsOne
9	PacsOne	CDMEDIC	EasyPACS	MRIdb	Xebra	CDMEDIC
10	PACSsoft	-	PACSsoft	OSPACS	-	ClearCanvas

These Criteria are defined in order to find the best open-source PACS to be used in our case study. Table 3.6 present a synthesis of this assessment. In Table 3.6, the two columns (Result 1 and Result 2) summarize the assessment result. Result 1 is the average for criteria 1,3 and 4 with BSD or GNU license models, and Result 2 includes all averages without considering the license model. According to these results, Orthanc, DCM4CHE, DCMTK, and Dicoogle are identified as the top-ranking open-source PACS projects. Thus, Orthanc is selected for the remaining parts of this study.

3.3 Laboratory test model

Before experimenting Orthanc as a PACS server for the Donka university hospital case study, it is necessary to test it in a laboratory controlled environment to gain a better understanding of its internal process, and how to connect this PACS server to an actual hospital information systems (HIS) (in our case study the eHospital HIS from the Adroit Company). In the next subsection, this lab test is described.

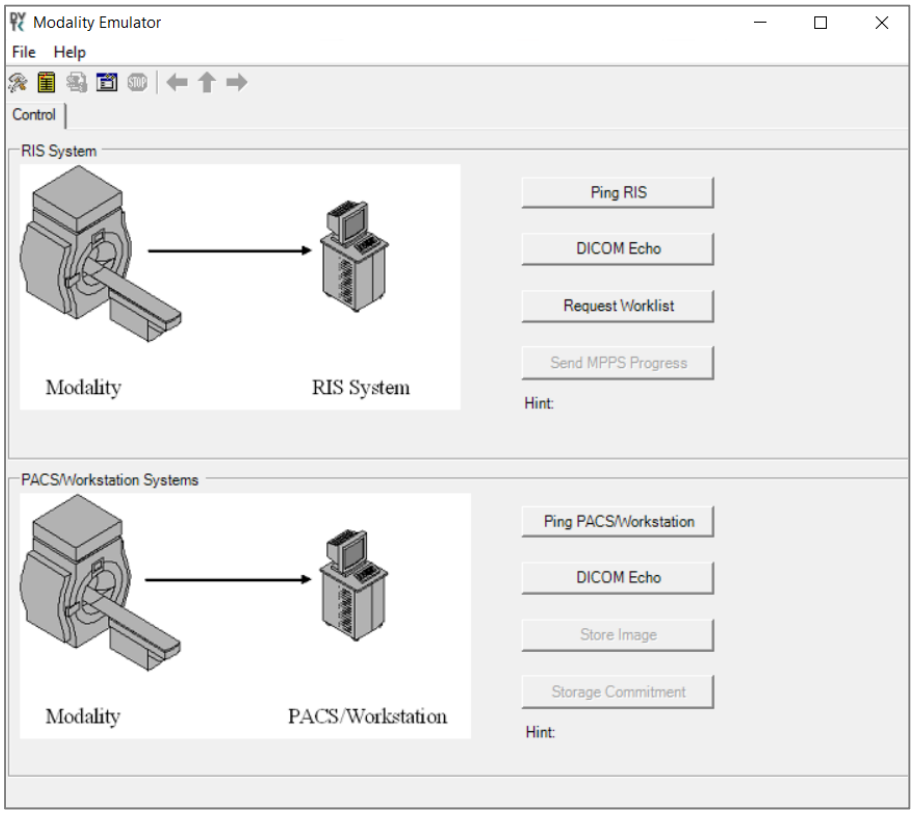


Figure 3.1 Modality Emulator software user interface

In order to simulate the process of sending DICOM files to the PACS server, reading the worklist file using a modality requires to execute the following 3 steps:

1. Download and run Orthanc in a virtual machine: in our tests, Orthanc 20.5.3 for windows is downloaded and installed on a virtual machine;

2. Modality Emulator: For the simulation of a modality, the open-source software Modality Emulator software (version 5.0.0) published in The Healthcare Validation Toolkit (DVTk) is used. The user interface of the Modality Emulator software is shown in Figure 3.1;

3. Understand the workflow: In order to connect Orthanc to the Modality Emulator in the Orthanc 'configuration.json' file in the section network topology, the AE Title, IP address, and port have been added. Also, configurations are required for the Modality Emulator in the Configure Remote System and PACS/Workstation Systems: the IP address, remote port, and AE Title are added. To check the connectivity of the Orthanc to the Modality Emulator, users can ping PACS by pressing the Ping PACS/Workstation. After this command, the Modality Emulator user can read the worklist file provided by Radiology Information System (RIS). Then the user should choose one the patient from the list. After this choice, users can send the DICOM file of the selected patient from the worklist to the PACS server by using the function 'Store Image'. The workflow of the lab test is depicted in Figure 3.2.

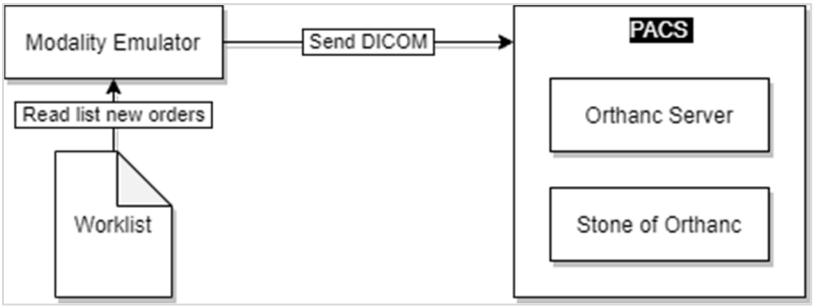


Figure 3.2 Simulation Workflow

3.4 Developing a model with PACS, HIS, RIS, and Modality

Once this connectivity test is done, integrating PACS with the hospital RIS, HIS is the next logical step. Different models for integrating PACS, RIS, and HIS exist. Figure 3.3 shows the typical hospital IT systems in most hospitals involved in connecting a PACS. This may differ based on the hospital's limitations, choice of HIS and RIS and PACS server functions. For

instance, some RIS have a feature to create and update a worklist file for modalities from the information which is provided by the HIS. But in some other designs, the PACS is responsible for this task. Orthanc has this ability to act as a DICOM worklist server by adding the modality worklist plugin. Using the Mirth Connect application is another method of integrate hospital IT systems for exchanging the information. In the next section, Mirth Connect as a powerful integration tool is explained.

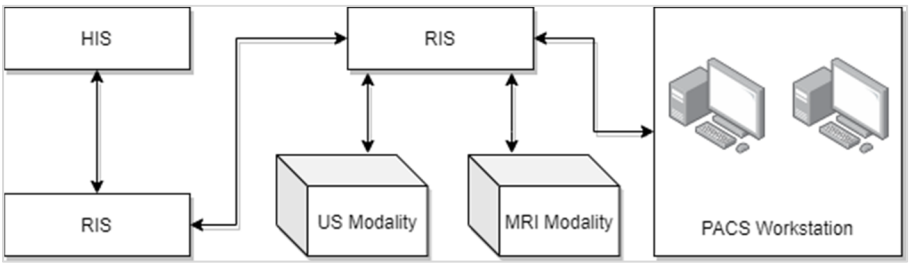


Figure 3.3 Typical connections of PACS in hospital systems

3.4.1 Using Mirth Connect

One of the goals of this research is to decide how to best integrate HIS (e.g., eHospital in this case) with Orthanc. Integration in healthcare information technology is recognized as one of the most challenging issues faced by hospitals. Hospitals acquire many technologies over time and they all have to communicate internally with many other systems and if a product/system fails to integrate, its investment value would not be realized (Henderson & Venkatraman, 1999). In Healthcare, the security and privacy concerns, the diversity of requirements and versatility of data items, the reluctance of sharing confidential data, and the scarcity of successful and published interoperability best practices makes it difficult for hospital to integrate all their healthcare technologies and obtain the maximum value from the data they produce (Katirai & Sax, 2005). Several studies have been published where a framework is proposed to integrate diverse health applications/systems and devices. In this research area, standard terminologies (e.g., SNOMED-CT, LOINC, RxNorm, and ICD-10) and messaging standards (e.g., NCPDP, HL7, and DICOM) have been proposed by the various standard

organizations. These have been progressively used and adapted to different popular technologies used in the clinical domain such as clinical information and medical imaging (Cyr, Agarwal, & Furht, 2013). While these efforts are valuable, they have introduced new challenges and complexity for customers and vendors. Customers and vendors have published many papers about their difficulty to select messaging and terminology standards, and consequently applications available from different vendors are not yet available for some operating systems and platforms. Also, upgrading existing healthcare applications, in a hospital, to support a peculiar interface is still a challenging task (Cheng, Chen, Lai, & Lai, 2010).

To address all these issues, Mirth Connect provides many interoperability benefits to a hospital, for example:

1. It is independent of the operating system;
2. It is open-source so has no licensing costs;
3. It is extremely extensible and flexible;
4. It allows its upgrade and maintenance separately from the business logic (Haque, W., Reed, A., & McCann, A. 2013).

Also, Mirth Connect can be applied for many other interoperability purposes such as: platform shift, message traffic monitoring, data mart interface creation, and adapter pattern. Between internal and external systems used by a large healthcare organization, like a university hospital, many messages have to be exchanged every day between many systems/devices. Mirth Connect provides a notification and monitoring functionality that can be employed for monitoring and evaluating the messages, to ensure immediate response to any interruption in the flow of messages, and to get notified of transactions that may be a sign of privacy or security breach. Another common scenario for applying Mirth Connect is its resilience when the technological platform evolves. When a hospital acquires new technological platforms for novel healthcare service delivery, such as a service on mobile devices or using cloud computing, the remaining business logic remains unchanged, and only the method used for

data gathering and delivering to the logical core needs to be modified (Roberge, MacLeod, Hartsock, & Asangansi, 2011).

In this research, Mirth Connect will be experimented. In the next section, the proposed PACS interoperability model for the Donka university hospital case study is proposed.

3.4.2 Overview of Donka University Hospital PACS interoperability model

Each hospital uses different information systems to automate their daily operations. These systems are often a patch work of many heterogeneous systems and medical devices trying to exchange information as best as they can. Alternatively, if all the information systems can be acquired at once it is better to acquire an integrated hospital management system (HIS). In this research case study, the Donka university hospital has already acquired an HIS (e.g. eHospital), which is an integrated hospital management system that includes all the hospital functionality, including a radiology module highlighted by a red box (see Figure 3.4).

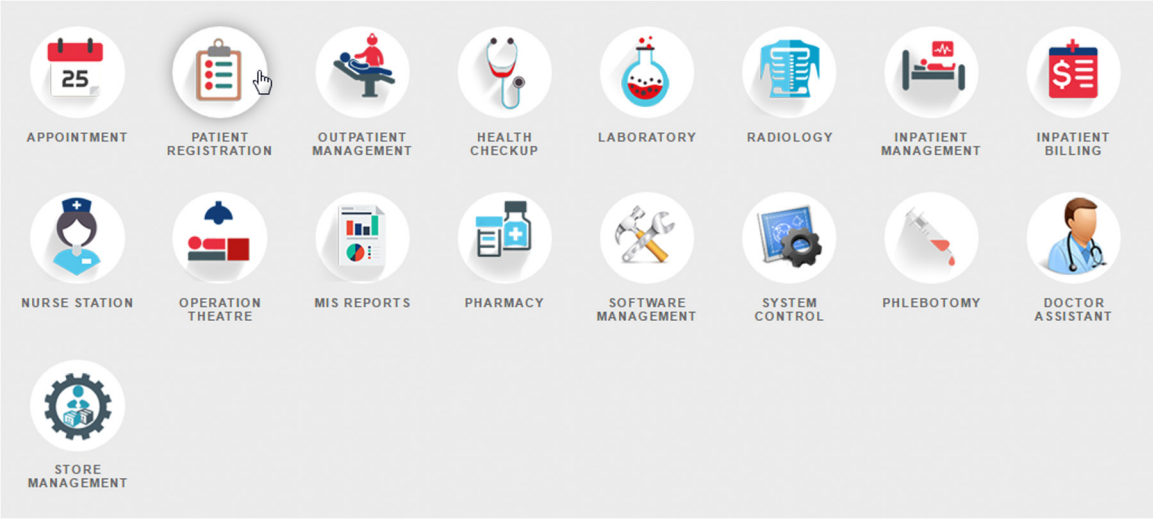


Figure 3.4 eHospital HIS modules, including Radiology

In order to integrate this HIS system with Orthanc, first, it is necessary to understand where the HIS requests and expected responses to and from the PACS will need to be added, to avoid

one to one exchanges of information. It is proposed that communication bus, like Mirth Connect, be added as a middleware between this HIS and Orthanc. Mirth connect would handle the requests from the HIS and send back expected responses.

A database would be designed in order to log transactions. This proposed design will ensure that the communication bus be used in the future for any exchange of data between the HIS and another system or medical device. Understanding the patient workflow in the hospital is important if we are to identify where the transactions between the HIS and the PACS are to take place.

Figure 3.5 shows the 10 steps of the radiology workflow that we intend to propose to this hospital. Patient goes through steps 1 to 4 for administrative purpose. Then the radiology technician selects his appointment in the radiology dashboard and sends him to the changing room. After this step he is taken to the X-Ray room where the technician will send a command to capture patient information from the DICOM worklist and send the DICOM images to the PACS server to be stored.

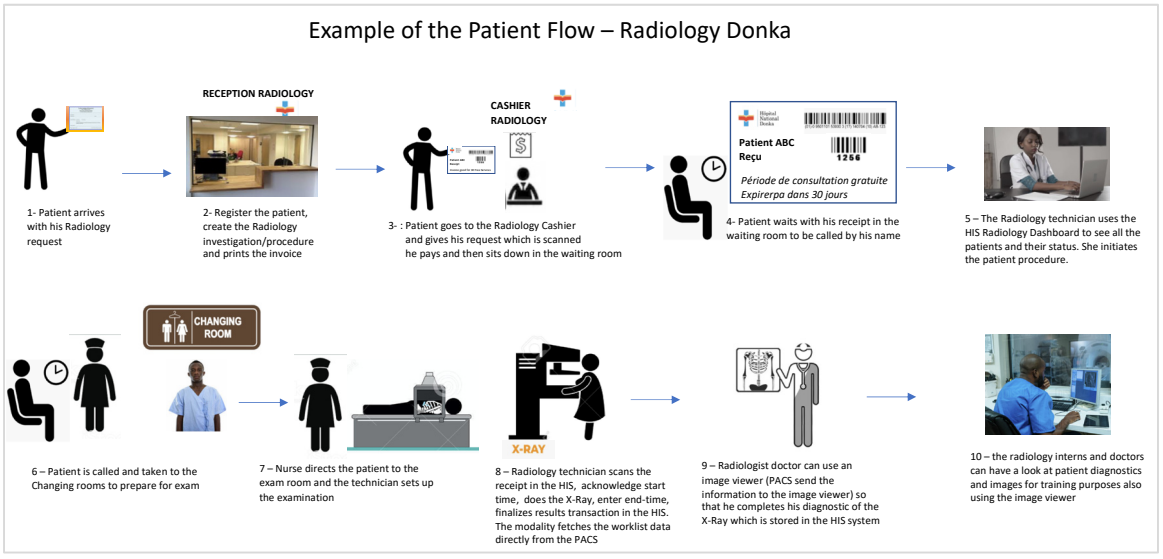


Figure 3.5 Proposed radiology workflow for the Donka hospital

In step 9, the radiologist can use the HIS system to retrieve a patient's files and document his diagnostic in the HIS system. Finally, at step 10, we show that interns (with authorized access) can look at examination results and images for later consultation or training.

Figure 3.6 shows the proposed PACS interoperability model for the Donka University Hospital case study. According to the HIS developers, the system is sending two requests to the PACS server:

1. When a new order is submitted to PACS (at steps 2 and 3);
2. When they send a query to retrieve a patient image report (steps 9 and 10).

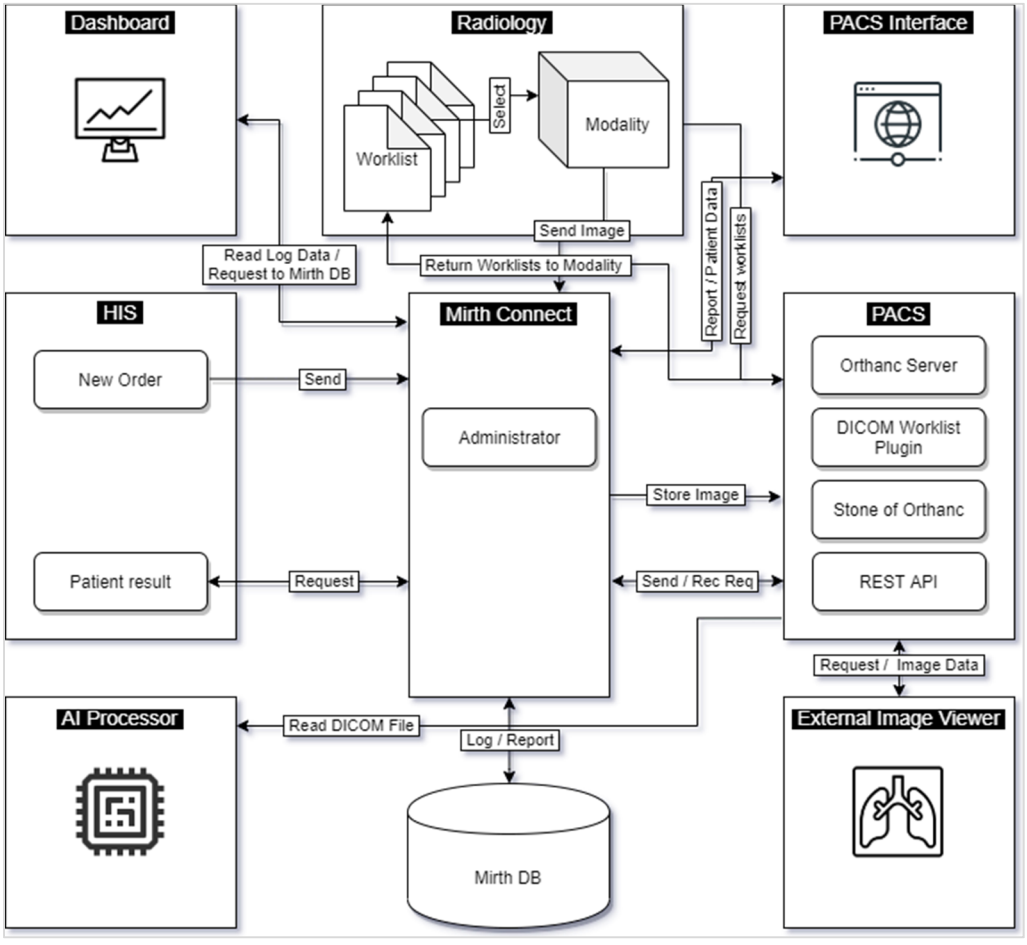


Figure 3.6 Proposed PACS interoperability model for Donka Hospital

The point of sending a new patient request to the PACS server is that it should provide to the modality a worklist table and this can be handled by Orthanc server. Orthanc has the ability to create a worklist for the radiology department from new orders that could help to automate and facilitate the data entry. Once receiving the new order from the HIS, Mirth Connect creates a ".wl" file in a folder which is accessible for the Orthanc DICOM server (adding a DICOM Worklist plugin makes Orthanc act as a DICOM Worklist server). When a patient goes to the radiology room, the Modality technician uses the C-Find command to fetch the list of worklist files from the DICOM server and then select the patient information. After the experiment Modality sends the DICOM files to the Mirth Connect and Mirth Connect redirect it to the PACS server in order to store.

PACS interface provides this opportunity for the technicians to query patient's information from the PACS and Mirth Connect database. In this web application technician can search a patient with the patient name, accession number, study description, and study date. The Dashboard shows the hospital patients' states online. It means that it shows which a patient is still in line before entering the experiment room, or which patient is waiting for the doctor to write a report on his medial image. TensorFlow, which is trained with the X-Ray images, provide the probability of a specific patient disease according to his DICOM files. This model could help to increase the accuracy and speed of disease diagnosis and provide support for the doctors.

3.5 Conclusion

In this chapter, popular open-source PACS are evaluated and Orthanc is selected for the remaining sections of the study (refer section 3.2). It is followed by the description of a lab test of Orthanc, as well as the test of an open-source Modality Emulator software that has helped in understanding the inner workings of the interconnections needed (refer to section 3.3). It is followed by the description of a proposed radiology patient workflow, as well as a proposed PACS interoperability model to be experimented for the Donka university hospital case study (see section 3.4). This proposed interoperability model will be experimented in chapter 4. In

the next chapter, the proposed PACS interoperability model design is investigated in more detail, then implemented in a prototype and is tested, in a case study, for the Donka Hospital.

CHAPTER 4

PACS INTEGRATION IMPLEMENTATION

4.1 Introduction

In the previous chapter, Orthanc Server was tested in the lab, and then an interoperability model between Orthanc and the HIS of a university hospital was proposed. In this chapter, the PACS interoperability model and different dataflow scenarios in the hospital are explained (see section 4.2) and tested for the Donka University Hospital.

4.2 Implementation of the proposed model

The implementation of the proposed PACS interoperability model requires understanding the type of requests and responses from HIS (e.g., eHospital), PACS, and modalities and implements each scenario separately. Different scenarios of data transactions in the daily routine of a hospital are categorized into:

1. HIS and PACS (two scenarios);
2. Modality and PACS (two scenarios);
3. PACS interface and Mirth Connect;
4. Dashboard and Mirth Connect;
5. Image viewer and PACS;
6. TensorFlow model and PACS.

Mirth Connect has been designed to manage transactions between different IT applications, systems, and medical devices. This means that each transaction is first sent to the Mirth Connect Channel, and then it will be redirected to its destinations, or Mirth will generate a proper response to the request. In the next subsections, the eHospital, the modalities, the PACS, and the image viewer transactions going through Mirth Connect are explained.

4.2.1 HIS and PACS dataflow

When a new patient comes to the hospital or to the radiology department reception, a technician will create a new POST request that will automatically be sent to Mirth Connect. A new channel is created for receiving this request from the HIS. Figure 4.1 shows the POST request parameters to the PACS server and the expected responses. These parameters are related to the patient information, test center, and a unique number named the “accessionNo”. This number is used for querying patient data from the PACS in order to read the patient's DICOM files and also the Mirth database. When this request is sent to the PACS, three different scenarios may occur:

1. New order and save (Figure 4.1 (2));
2. Old order, then update (Figure 4.1 (3));
3. Wrong date format (Figure 4.1 (4) (5)).

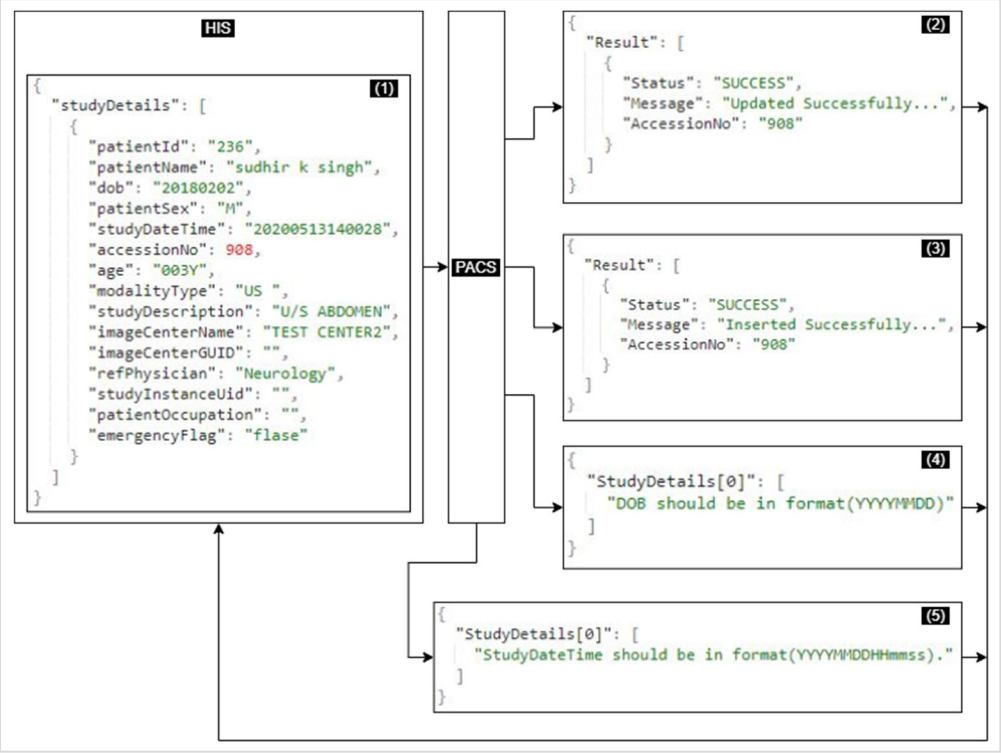


Figure 4.1 Example of new order request parameters

A new channel should be created to receive requests from the HIS. The connector type of this channel in the Mirth Connect Administrator is set as an HTTP Listener, so each request which is sent to the HTTP URL and port of this channel from the HIS is received and processed. The destination tab should define the potential target of the request. The new order post request first is inserted into the database (destination_1: Mirth_db) and then generates a worklist file (destination_2: Worklist folder). Thus, two separate destinations are defined for these two actions.

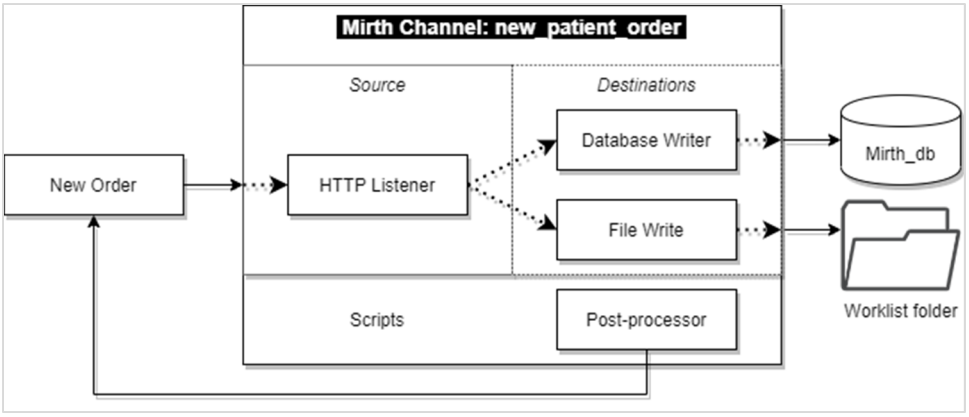


Figure 4.2 New order dataflow

In order to parse the JSON values (based on the HL7 standards) received from the POST request, a transformer should be designed to map the JSON attributes and the defined Mirth variables. After inserting a sample JSON into the Message Templates in the Edit Channel > Source > Edit Transformer page, each attribute should map to a variable. Mapping attributes and variables help in accessing the JSON value in the Destination tab > Destination Mappings section. Figure 4.2 shows the proposed design of this channel, which includes the HTTP listener as a source of the channel, two destinations, and a post-processor script. Table 4.1 shows the configurations of the channel and the source section of this channel.

Table 4.1 New order channel configurations

Tab	Configurations
Summary	**Name**: new_order_req **Set Date Types**: ***Inbound**: HL7 v3.x* ***Outbound**: HL7 v3.x*
Source	**Connector Type**: HTTP Listener **Source Queue**: OFF (Respond after processing) **Response**: Postprocessor **HTTP URL**: http://localhost:8090/

The first destination of the channel is a Database Writer. This destination receives the values from the Destination Mappings section and inserts it into the designed table in the SQL Server. The configurations and the insert query are described in Table 4.2.

Table 4.2 Destination 1 configuration and an insert query sample

Destination 1	**Connector Type**: *Database Writer* **Database Writer Settings:** ***Driver**: net.sourceforge.jtds.jdbc.Driver* ***Url**: jdbc:jtds: sqlserver://127.0.0.1:1433/Mirth_db* ***Username**: sa* ***Password**: ****** **SQL**: *INSERT INTO patient_request* *(* *patientid, patientname, dob, patientsex, studydatetime, accessionno, age, modalitytype, studydescription, imagecentername, imagecenterguid, refphysician, studyinstanceuid, patientoccupation, emergencyflag* *)* *VALUES* *(* *${patientid}, ${ patientname}, ${dob}, ${patientsex}, ${studydatetime}, ${accessionno}, ${age}, ${modalitytype}, ${studydescription}, ${imagecentername}, ${imagecenterguid}, ${refphysician}, ${studyinstanceuid}, ${patientoccupation}, ${emergencyflag}* *)*

The second destination of the channel is a File Writer. In order to provide a list of new patient orders for a modality, Mirth first generates a specific file with a ".wl" extension format (Table 4.1) and stores it in the folder which is accessible for the server of DICOM Worklist. Then, a

modality uses these files to create a list of new orders. The process of fetching these files is explained in subsection 4.2.2.

Table 4.3 Destination 2 configurations and a sample of the worklist file content

Destination 2	**Connector Type**: *File Writer* **File Writer Settings:** *Method: file* *File Name: Worklist-${accessionno}.wl* *File Exits: Overwrite* Template: *# Dicom-File-Format* *# Dicom-Meta-Information-Header* *# Used TransferSyntax: Little Endian Explicit* (0002,0000) UL 202 *# 4, 1 FileMetaInformationGroupLength* (0002,0001) OB 00\01 *# 2, 1 FileMetaInformationVersion* (0002,0002) UI [1.2.276.0.7230010.3.1.0.1] *# 26, 1 MediaStorageSOPClassUID* (0002,0003) UI [1.2.276.0.7230010.3.1.4.2831176407.11154.1448031138.805061] *# 58, 1 MediaStorageSOPInstanceUID* (0002,0010) UI =LittleEndianExplicit *# 20, 1 TransferSyntaxUID* (0002,0012) UI [1.2.276.0.7230010.3.0.3.6.0] *# 28, 1 ImplementationClassUID* (0002,0013) SH [OFFIS_DCMTK_360] *# 16, 1 ImplementationVersionName* *# Dicom-Data-Set* *# Used TransferSyntax: Little Endian Explicit* (0008,0005) CS [ISO_IR 100] *# 10, 1 SpecificCharacterSet* (0008,0050) SH [${accessionno}] *# 6, 1 AccessionNumber* (0010,0010) PN [${patientname}] *# 16, 1 PatientName* (0010,0020) LO [${patientid}] *# 8, 1 PatientID* (0010,0030) DA [${dob}] *# 8, 1 PatientBirthDate* (0010,0040) CS [${patientsex}] *# 2, 1 PatientSex* (0010,2000) LO [] *# 10, 1 MedicalAlerts* (0010,2110) LO [] *# 6, 1 Allergies* (0020,000d) UI [${studyinstanceuid}] *# 26, 1 StudyInstanceUID* (0032,1032) PN [${refphysician}] *# 6, 1 RequestingPhysician* (0032,1060) LO [${studydescription}] *# 6, 1 RequestedProcedureDescription* (0040,1001) SH [] *# 10, 1 RequestedProcedureID* (0040,1003) SH [${emergencyflag}] *# 4, 1 RequestedProcedurePriority*

When a POST request received from the HIS and processed through Destination 1 and Destination 2, it is essential to return a proper response. As it is shown in Table 4-1, the response configuration for the Source Setting section is sent as Postprocessor. It means that the channel generates the response after processing two destinations and generates a response with a postprocessor script and then returns a JSON similar to the response shown in Figure 4.1 to the received POST request.

eHospital application has a dedicated section to the dashboard; in this section, users can see the state of the patient request in the hospital or fetch the doctors' reports. Request to the Mirth could include a specific period of time such as current day, or month or a patient “accessionNo” to see the report. Figure 4.3 shows these transactions.

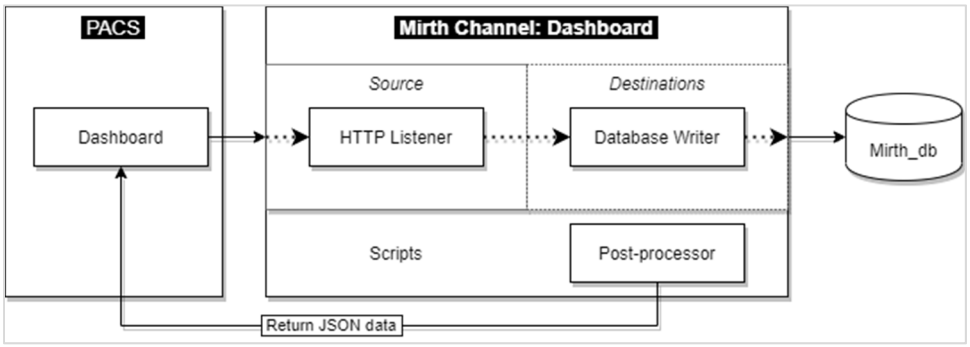


Figure 4.3 Transaction between the PACS dashboard and Mirth Connect

4.2.2 Modality and PACS dataflow

The workflow of creating a new worklist file is explained in the previous subsection. Mirth through “Destination 2” of the “new_patient_order” channel generates worklist files and saves them into a folder that is accessible by the DICOM worklist server. Whenever a modality sends a C-Find SCP request to the Orthanc, the DICOM worklist server checks the worklist folder and filter files related to the received request and return to the Modality (Figure 4.4).

For testing the process, the “findscu” command-line tool from the DCMTK utilities is used. The below sample shows the command that sends a request to Orthanc and returns all worklist for CT modality.

```
findscu –W –k "ScheduledProcedureStepSequence[0].Modality=CT" 127.0.0.1 4242
```

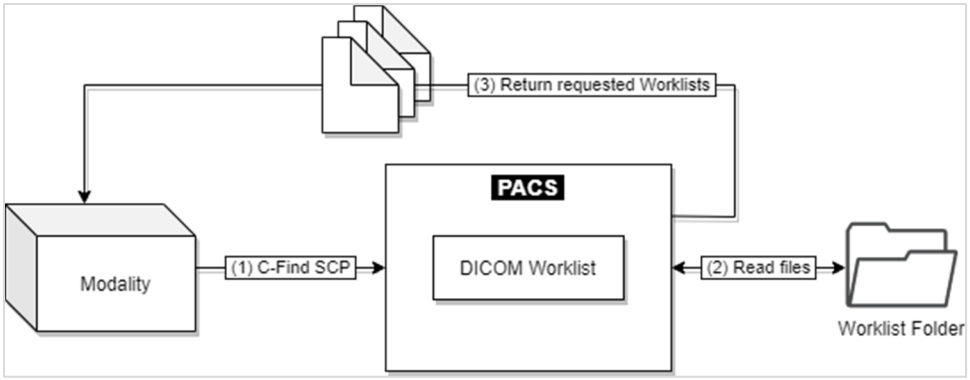


Figure 4.4 Updating Worklist files

After an imaging experiment, a modality should send the DICOM file to the PACS. DICOM to PACS channel (Figure 4.5) of Mirth receives DICOM files from the Modality and sends it to the PACS. Also, there is a filed in the "patient_request" table inside the database that holds the last status of a patient. The second destination of this channel updates the status that shows the imaging is done.

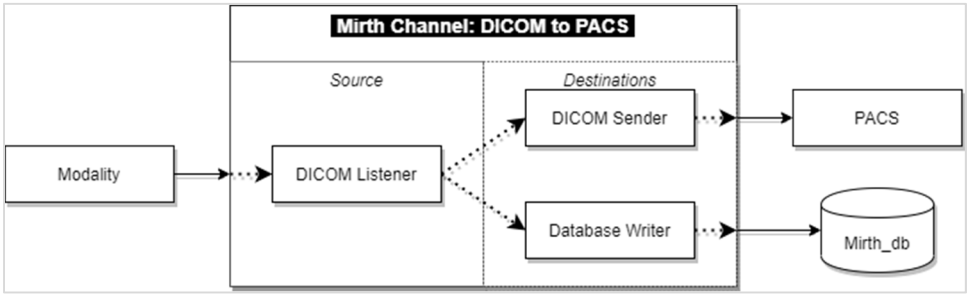


Figure 4.5 Returning the image from the Modality to the PACS

Table 4-4 shows the configurations for this channel. The connector type is selected as a DICOM listener and receives all the DICOM files from the modalities. Destination 1, as a DICOM sender, sends DICOM to PACS and Destination 2 to update the patient status in the database.

Table 4.4 DICOM to PACS channel configurations

Tab	Configurations
Summary	**Name**: DICOM_to_PACS **Set Date Types**: ***Inbound**: DICOM* ***Outbound**: DICOM*
Source	**Connector Type**: DICOM Listener **Source Queue**: OFF (Respond after processing) **Response**: None **HTTP URL**: http://localhost:104/
Destination 1	**Connector Type**: DICOM Sender **Remote Host:** 192.168.1.1 (PACS IP) **Remote Port**: 448
Destination 2	**Connector Type**: *Database Writer* **Database Writer Settings**: *Driver: net.sourceforge.jtds.jdbc.Driver* *Url: jdbc:jtds: sqlserver://127.0.0.1:1433/Mirth_db* *Username: sa* *Password: ****** ***SQL:*** *UPDATE TABLE patient_request* *SET status = 2 -- Represents that this patient image is sent to the PACS* *WHERE accessionNo = msg['tag00280030']*

4.2.3 PACS Interface dataflow

The PACS interface provides an opportunity for the technicians to query patients using the patient name, accession number, study description, and study date. Figure 4.6 shows a transaction of searching data in the Mirth_db and returns the response.

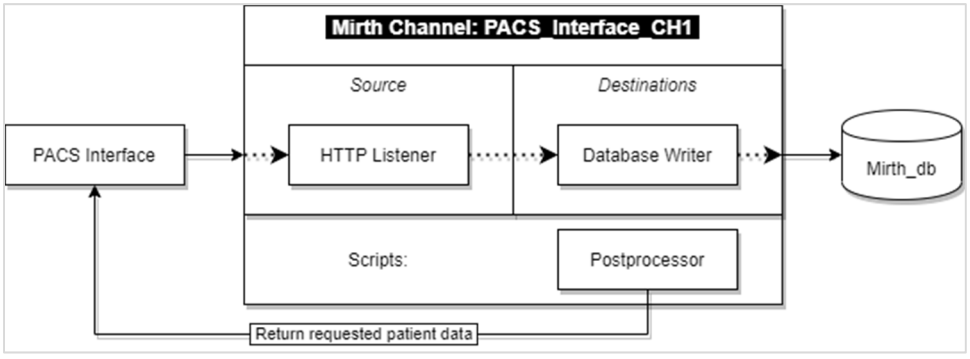


Figure 4.6 Searching data in the Mirth database

This channel has only one destination, which is a Database Writer that inserts a log record into the Mirth database. The postprocessor script will after the completion of the channel destination and return filter data to the PACS interface. Table 4.5 shows the configuration and SQL query code. Destination 1 inserts a log record into the database, and after that, fetch the patient record form the Mirth_db.

Table 4.5 PACS_Interface_CH1 channel configurations and SQL query code

Tab	**Configurations**
Summary	**Name**: PACS_Interface_CH1 **Set Date Types**: ***Inbound**: HL7 v3.x* ***Outbound**: HL7 v3.x*
Source	**Connector Type**: HTTP Listener **Source Queue**: OFF (Respond after processing) **Response**: Postprocessor **HTTP URL**: http://localhost:80/
Destination 1	**Connector Type***: Database Writer* **Database Writer Settings**: *Driver: net.sourceforge.jtds.jdbc.Driver* *Url: jdbc:jtds: sqlserver://127.0.0.1:1433/Mirth_db* *Username: sa* *Password: ****** **SQL***:* *INSERT INTO log([accessionNo], [action], [submited_user])* *VALUES(${accessionno}, 'return_patient_list', 'PACS_Interface_Admin')*
Scripts	*var dbConn = DatabaseConnectionFactory.createDatabaseConnection(* *'net.sourceforge.jtds.jdbc.Driver',* *'jdbc:jtds:sqlserver://127.0.0.1:1433/Mirth_db', 'sa', 'admin');* *var sql = "SELECT * from patient_request";* *var results = dbConn.executeCachedQuery(sql);* *dbConn.close();* *return results;*

When the received data is loaded into the web page, doctors could write the report with the help of the image viewer (see subsection 4.2.5) for the patient and save it in the Mirth_db. Figure 4.7 illustrates the diagram of this channel.

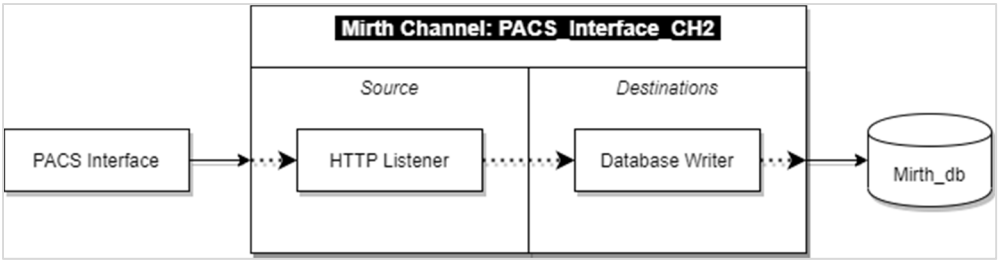


Figure 4.7 Receive reports from the PACS interface and insert into Mirth_db

Table 4.5 provides the configurations and SQL queries of the second channel of this section dataflow. The first query inserts the patient study text into the Mirth_db and then updates the patient state.

Table 4.6 PACS_Interface_CH2 channel configurations and SQL queries

Tab	**Configurations**
Summary	**Name**: PACS_Interface_CH2 **Set Date Types**: ***Inbound**: HL7 v3.x* ***Outbound**: HL7 v3.x*
Source	**Connector Type**: HTTP Listener **Source Queue**: OFF (Respond after processing) **Response**: Postprocessor **HTTP URL**: http://localhost:81/
Destination 1	**Connector Type***: Database Writer* **Database Writer Settings**: *Driver: net.sourceforge.jtds.jdbc.Driver* *Url: jdbc:jtds: sqlserver://127.0.0.1:1433/Mirth_db* *Username: sa Password: ****** **SQL:** *INSERT INTO patient_study([accessionNo], [study_result], [study_date])* *VALUES(${accessionno}, 'Dr study description', '2020-08-16 07:11 AM')* GO *UPDATE TABLE patient_request* *SET status = 3 -- Represents that this patient image study is finished* *WHERE accessionNo = msg['tag00280030']*

4.2.4 Image Viewer dataflow

Due to the fact that Orthanc has been selected as a PACS server in this research, it is important to explain the image viewer which are compatible with it. An image viewer plays a vital role

for the radiologists' accurate diagnostics. The Orthanc web viewer provides only basic functionality for users, which is not for diagnostic purposes. The modular structure of Orthanc provides an opportunity to use more comprehensive image viewer plugins and extend the basic functionality of Orthanc. The following plugin list proposes more sophisticated options than the basic Orthanc web viewer.

1. Osimis web viewer plugin: this plugin is the advanced version of the Orthanc web viewer. Some of the advance features are including:
    a. Annotations (linear, elliptic, rectangular, angle measurement, arrows, text);
    b. Integration to electronic medical report (EMR)/HIS/RIS through URL links;
    c. Support of DICOM video files (MPEG2);
    d. Hounsfield windowing presets;
    e. Measure Hounsfield units at a specific point of a CT study;
    f. Cine playback of multi-frame sequences;
    g. Series preview in thumbnails;
    h. Progressive image loading;
    i. Affero General Public License (AGPL).

2. ImageJ extension: it is a public domain Java image processing program with the following specifics:
    a. Runs as an online applet or downloadable application;
    b. Display, edit, analyze, process, save and print 8, 16, 32-bit images;
    c. Read TIFF, GIF, JPEG, BMP, DICOM, FITS, and raw;
    d. Multithread processing;
    e. Designed with an open architecture;
    f. GNU General Public License.

3. Stone of Orthanc:
    a. Lightweight;
    b. CPU hardware acceleration (SSE2, SSSE3 and NEON instruction sets);
    c. The highly versatile framework that can run even on low-performance platforms;

d. Can display a DICOM series without store entirely in the RAM;
e. Affero General Public License (AGPL).

Also, some compatible software with the Orthanc can be applied as an image viewer such as Open Health Imaging Foundation (OHIF), Horos, Gringko CADx, 3D Slicer, Medlnria, Aeskulp, and OsririX.

The image viewer should connect directly to the PACS server in order to fetch and load DICOM files to study patient images. It should be noted that all the plugins mentioned here and generally the open-source solutions are not recommended for diagnostic usage.

4.2.5 Dashboard dataflow

Managing well the data flow, in a hospital, is a decisive factor in facilitating the patients' requests. Monitoring the patients' states in the hospital will help managers to find and resolve their internal systems and procedures weaknesses.

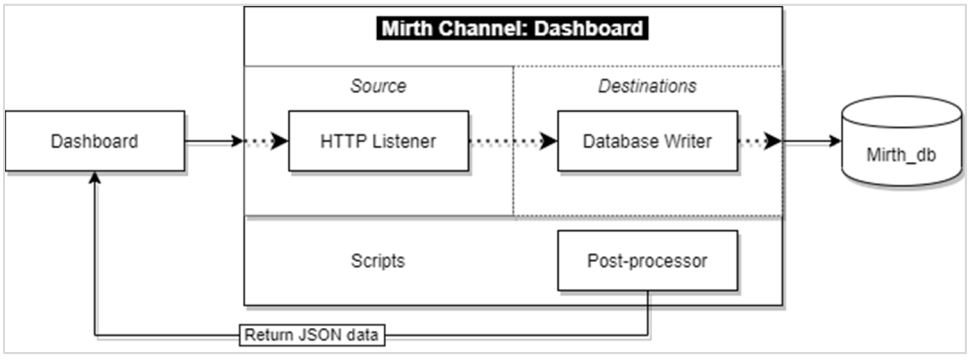


Figure 4.8 Dashboard transactions diagram

Through a dashboard, the manager can see not only how many new orders have been sent to the worklist and are waiting in the queue for service or have already been done, but also it can generate different types of reports to show the radiology workflow performance. The dashboard sends a 'GET request' to the Mirth software and receives a JSON transaction that

is displayed on the web page. Figure 4.8 shows the dataflow between the dashboard and Mirth Connect. The configurations and designed queries for this channel are demonstrated in Table 4.7.

Table 4.7 Dashboard Channel configurations and SQL queries

Tab	Configurations
Summary	**Name**: Dashboard **Set Date Types**: ***Inbound**: HL7 v3.x* ***Outbound**: HL7 v3.x*
Source	**Connector Type**: HTTP Listener **Source Queue**: OFF (Respond after processing) **Response**: Postprocessor **HTTP URL**: http://localhost:104/
Destination 1	**Connector Type***: Database Writer* **Database Writer Settings**: *Driver: net.sourceforge.jtds.jdbc.Driver* *Url: jdbc:jtds: sqlserver://127.0.0.1:1433/Mirth_db* *Username: sa Password: ****** **SQL:** *INSERT INTO log(* *[accessionNo], [action], [submited_user]* *)VALUES(* *${accessionno}, 'return_patient_list', 'PACS_Dashboard_Admin'* *)*
Scripts	*var dbConn = DatabaseConnectionFactory.createDatabaseConnection(* *'net.sourceforge.jtds.jdbc.Driver',* *'jdbc:jtds:sqlserver://127.0.0.1:1433/Mirth_db',* *'sa',* *'admin');* *var sql = "SELECT * from patient_request";* *var results = dbConn.executeCachedQuery(sql);* *dbConn.close();* *return results;*

4.2.6 TensorFlow Model dataflow

In chapter one, it was introduced how artificial intelligence, especially in computer vision, could impact medical imaging diagnostic in different ways. The deep learning algorithms that use neural network technology has recently become prominent in the area of computer vision and enables computers to detect objects in natural images. TensorFlow is a free and open-source library that is used for developing machine learning applications based on neural

networks. TensorFlow helps developers to create a model and train it with labeled images to categorized objects in its test data. Applying trained models in medical imaging is increasing due to a high level of accuracy in detecting patterns in images to help diagnostic. Thus, a well-designed and trained model could help the medical imaging specialists in their diagnosis.

After a prediction model for detecting specific patterns (for example pulmonary edema) in medical images is trained and deployed, doctors can send a request to the PACS to retrieve patient DICOM files with the help of the “accessionNo”. TensorFlow includes a library that already decodes DICOM files. After a patients’ DICOM files are received from the Mirth Connect, it can be decoded, and using the prediction model, it can be processed automatically, and the result with the highest probability sent to the doctors. Figure 4.9 shows an example of this transactions.

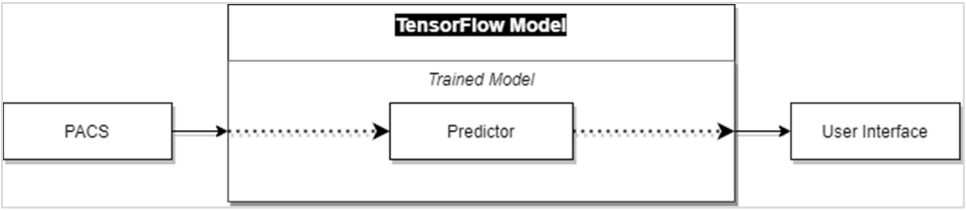


Figure 4.9 TensorFlow model

4.3 Laboratory Implementation

We implemented a test model in the lab to understand the transaction on Orthanc and modality in the chapter 3 of this study. To test the proposed model for Donka University Hospital, we implement the explained model in the laboratory. We used a virtual machine with Windows Server to install Microsoft SQL Server as a choice of database for mirth_db, Mirth Connect, Modality Emulator to simulate modality functionalities, and Postman to send request and receive request to the Mirth Connect. The detail of channel and dataflow scenarios are explained at the previous sections and same configurations are implemented in this laboratory

test. This laboratory test verified the proposed interoperability model feasibility that could be applied in the Donka hospital environment.

4.4 Conclusion

In this chapter, different data transaction scenarios were described. These transactions are the most important dataflow between the HIS, PACS, and the imaging modalities for university hospitals. Besides these popular scenarios, a dashboard, an image viewer, and PACS interface transactions were explained. Finally, TensorFlow, which has the potential to train diagnosis models, shows how deep learning could help doctors in automating some diagnostics using artificial intelligence directly on DICOM images. The open architecture of the proposed model could lead to time and cost reduction for the Donka hospital, as well as reduce technical barriers to exchange health information within the hospital.

CONCLUSION

5.1 Introduction

This chapter of the research concludes this thesis. First, a summary of the study and the purposes of the study are presented (see section 5.2). Then, the summary of the implemented model is provided (see section 5.3), and the significance of this implication is briefly explained (section 5.4). Recommendations for future research (see section 5.5) end the chapter.

5.2 Summary of research

The importance of the PACS server is explained in the first chapter of this research and how it became one of the vital IT systems of G8 countries over the last decade. It was found that PACS are necessary for all hospitals around the world but not always affordable for everyone, especially in the developing countries. Open-source PACS seems to be a viable alternative, but selecting among the many proposals requires creating evaluation criteria.

An objective of this research was to identify the needed functionality of "research PACS" to be useful for research and teaching hospitals in Africa. For this purpose, four selection criteria extracted from the article "Open Source in Imaging Informatics" by Nagy (2007) to evaluate open-source PACS are including:

- criteria 1: Community activities;
- criteria 2: Licensing models;
- criteria 3: Activity, support, and documentation;
- criteria 4: Enterprise functions and software characteristics.

We have found that open-source PACS with active community activities, good support, good documentation, rich enterprise functions, and a less strict licensing models are preferable.

A second objective of this research was to assess available open-source PACS regarding the fit of their functionalities with the specific requirements (designed criteria) of research and university hospitals in Africa. This aimed at choosing an open-source PACS candidate for experimentation. In this regard, a list of sixteen popular open-source PACS was identified then assessed using the defined criteria. The first criterion compares the retrieved data from the source code repositories and project forums, which shows that DCMTK, DCM4CHE, and Orthanc have the highest community activities among other open-source PACS. The second criterion investigates the licensing model and explains the restrictions and advantages of each model. Most of the open-source PACS have GPL licensing model. The third criteria assesses activity, support, and documentation. Orthanc has well designed and user-friendly website, easy to install (less than a minute), provide some research articles and an online book to support users and developers and it is a cross-platform application. The last criterion evaluates the selected PACS enterprise functions and their extensibilities. Orthanc provides some built-in functions and easily extensible by using API or developing new modules, which make it one of the best choice along with DCM4CHE. Totally, the combination of all results from the previous four criteria showed that Orthanc, DCM4CHE, DCMTK, Dicoogle, and MRIdb are the top rank open-source PACS. Thus, Orthanc is selected for the remaining of this research, and its modular architecture is reviewed in the second chapter.

In Chapter Three, the DICOM format was explained, then a laboratory test model designed to have a better understanding of the internal process, software, and hardware requirements was experimented. Finally, an integrated model was proposed for the university hospitals. The main issue in the hospital environments is that many heterogeneous systems medical devices are trying to exchange messages. In this respect, Mirth Connect was chosen and experimented as a communication bus between the HIS, PACS, and Modality.

During the case study Orthanc was experimented and a connectivity model for the Donka university hospital in Guinea, Africa case study was designed. The connectivity required the need for a common language used by hospitals. This problem was solved by FIHR/HL7 International in the form of establishing an FIHR/HL7 standard for communication and

interoperability of different healthcare systems, but to be more flexible in the future it needed a central interface that will facilitate the disparate systems trying to communicate and interoperate with eHospital in the future. One of the most popular and powerful solutions found in the literature to solve this healthcare problem is called the HL7 interface engine. One of the most popular options proposed by our networks of software engineers specialized in healthcare interoperability is "Mirth Connect HL7." This hospital had already acquired a HIS system (eHospital from Adroit), and we had design meetings with their developers' team to identify their type of request to a typical PACS. At the end of that chapter three, the connection model proposed and experimented for the Donka hospital with its PACS system is communicating through Mirth Connect messaging. In the proposed model, potential communications scenarios are briefly described and explained.

Finally, in chapter four, the data transaction scenarios are precisely defined and implemented for the Donka hospital case study. Moreover, using Mirth Connect HL7 has shown the potential of this approach for the Donka hospital to interconnect any future systems seamlessly to eHospital with minimum effort and modifications. The communication prototype case study between the Donka hospital information systems that have been tested successfully are:

1. Two scenarios between HIS (eHospital) and PACS (Orthanc);
2. Modality and PACS (two scenarios);
3. PACS interface and Mirth Connect;
4. Dashboard and Mirth Connect;
5. Image viewer and PACS;
6. TensorFlow model and PACS.

5.3 Discussion and Interpretation of the implemented model

The open-source PACS interoperability model for university hospitals experimented for use by the Donka hospital, in our case study, shows promise and could be implemented in other healthcare organizations across Africa and developing countries. It demonstrated how open-source software could help in interconnecting systems through a free middleware that supports

FIHR/HL7 interoperability protocol. In this proposed interoperability model, eHospital requests are considered as the HIS/RIS requests format to the PACS, which may be different from the other HIS system or may a Hospital use a separate RIS system for handling the radiology patient visits. Furthermore, Orthanc was selected for this case study as a choice of open-source PACS, but there are some other popular open-source PACS such as DMTCK, DCM4CHEE, and Dicoogle, which could have been selected. So replacing Orthanc with these other open-source PACS would lead to modification of some transactions due to the specific characteristics of Orthanc and its Plugins used here. However, both changes in HIS/RIS and PACS servers would not yield significant alterations to the interoperability model proposed as it has been designed to be an easily adaptable interoperability model using Mirth Connect.

The transaction scenarios experimented in the case study are the most important and critical communication transactions between a PACS and an HIS/RIS in a hospital. The first transaction is when a user sends a new request to the Mirth Connect. Then Mirth Connect creates a new file with ".wl" extension and storing it in a folder which is accessible for the DICOM server (in this model, by using Modality Worklists plugin turned Orthanc into a DICOM worklist server). When a Modality sends the C-FIND SCP request to the DICOM server, it checks the folder and returns related files to the Modality, and technicians select one of the listed patients for the experiment. After the experiment, generated DICOM files send to Mirth Connect, and then it is redirected to the PACS server to store. Doctors can access and view the patient radiology images using the Orthanc stone or other external plugins and write their diagnostic report using the PACS interface. This type of web application helps them to query patients using different filters. After writing the diagnostic report for the patient, it is submitted to the Mirth Connect and stored into the Mirth_db database. Externally designed dashboard and eHospital dashboard sync the state of patients with retrieving data from this database.

Finally, we have shown that this interoperability model allows easy access to medical images with the use of artificial intelligence algorithms that could help radiologists automate the detections of pneumonia, using X-Ray images, with acceptable accuracy (> 90%).

To sum up, this research has proposed four main contributions; firstly, comparative criteria ranked popular open-source PACS and could be useful for many healthcare institutes that could not afford commercial PACS. This part of the rersearch was published in a journal paper entitled “Assessing available open-source PACS options” and it is accepted to publish in the Journal of Digital Imaging. Secondly, the interoperability model demonstrated how to interconnect Orthanc (or any other open source PACS) and HIS/RIS seamlessly. Thirdly, we have demonstrated how a machine learning model could be used to obtain Xray images and automatically interpret these medical images to provide automated decision support for future modern radiologists. And lastly, we have shown the interoperability benefits of using a FIHR/HL7 SOA messaging system, like the open source project Mirth Connect, in a real case study for the Donka hospital.

5.4 Significance of the Study

The main objectives of this study were attained where small healthcare institutes that could not afford a commercial PACS can use and integrate an open source PACS with their HIS/RIS system. Open-source PACS demonstrate that they can be a good asset and provide basic radiology functionalities at very low cost. The case study shows promise as its experimentation showed that all the communications were possible and worked well once developed on our experimental prototype.

5.5 Recommendations for Future research

To implement this research at the Donka university hospital in 2021, some further steps should be considered. Firstly, we Mirth Connect has to be installed. Then, the Orthanc server needs to be installed. Next, the eHopital HIS/RIS needs to be properly connected to the network and add worklist plugins to the Orthanc. Then, create the channels and tables in the Mirth Connect and Mirth_db. After this initial installation, the administrators can check the connections of

different systems with Mirth Connect and verify that the defined routines in the interconnectivity model is active using Postman. Administrators should issue a new test order and then check the Mirth channel (new_patient_order channel), Mirth_db, and worklist folder to ensure that the user request is successfully logged in the database and that a worklist file is created. Then, try to load the list of worklists from the modality. The radiologist should be able to select one of the new order requests among this list. After selecting a patient and doing an X-Ray, the modality should connect to the Mirth Connect channel (DICOM_to_PACS channel) to send the DICOM files to the PACS. A test example DICOM files could be sent to the Mirth Connect in order to test this routine beforehend.

The previous paragraph explained the main steps to integrate an open-source PACS and HIS, which include send a new request, create a worklist file, read the worklist file from a modality, and store DICOM files in the PACS. The Mirth Connect software acts as a middleware and manages most of these actions between the different hospital sub-systems. Moreover, Mirth Connect provides much more advanced interconnectivity features that were described in this research, such as SSL connectivity, role-based access, and advanced alert functionality. Also, the Orthanc server functionalities could be expanded by developing new plugins and customized applications or adding different/advanced plugins for the many university research needs in radiology.

APPENDIX I

MIRTH CHANNEL IMPLEMENTATION

As explained in chapter 4, we defined different channels in order to exchange health information between different hospital information systems. In this section, the implementation of a channel is demonstrated.

First, install Mirth Connect in a server, which can be downloaded from the Mirth Connect website (www.mirthcorp.com). Mirth Connect has four main components:

1. **Mirth Connect Server:** it includes the back-end for the integration engine component and the management interface, which performs message transmission, transformation, and filtering;
2. **Mirth Connect Administrator:** it is a graphical user interface that connects to the Mirth Connect Server and permits admins to configure interfaces, monitor interface activity, and browse the message store;
3. **Mirth Connect Server Manager:** it is a graphical user interface that displays log files, manages the Mirth Connect service, and includes configuration settings;
4. **Mirth Connect Command Line Interface:** is a command-line tool that allows connections to the Mirth Connect Server to deploy/import/export channels and perform other administrative tasks.

After installing of the latest version of the software, the admin should login to the Mirth Connect Administrator. Figure 1 shows the user interface of Mirth Connect. In the Dashboard section, users can see the transactions between the HIS and the Mirth channels and the history of these transactions. To create a new channel, the admin should select Channels from the Mirth Connect box, and then right-click and select “new channel”.

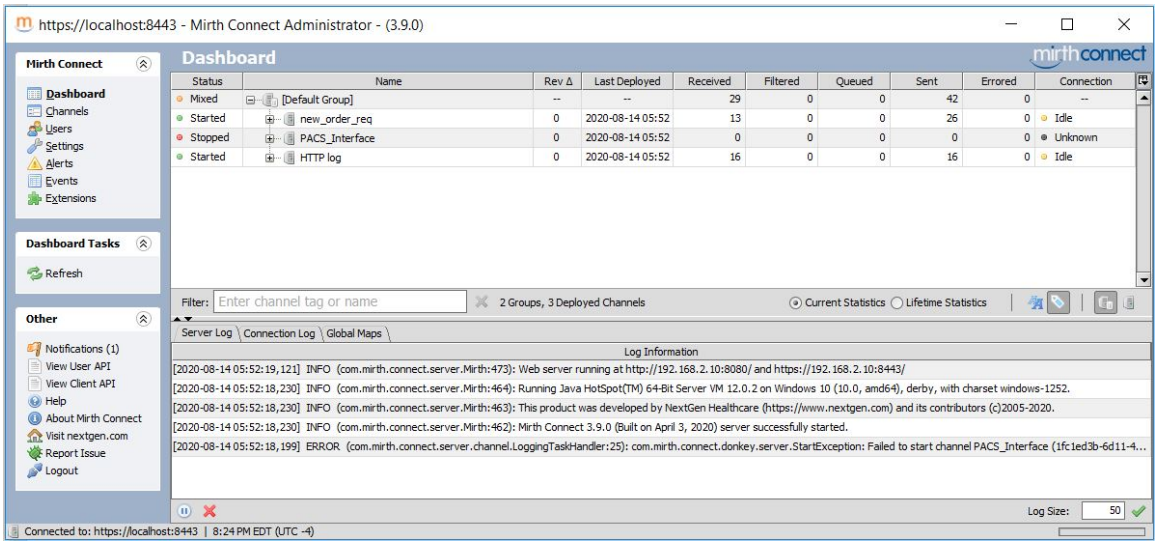


Figure-A I.1 Mirth Connect Administrator user interface

To create a new channel, first select Channels from the Mirth Connect panel at the top left and then right-click on the page and choose a new channel. There are four tabs in the Edit Channel page which is including:

1. Summary: includes channel properties, Message Storage, Message Pruning, Custom Metadata, and Channel Description;
2. Source: defines Connector Type, Source Settings, and Channel Reader Settings;
3. Destinations: includes Destination and Channel Writer Settings,
4. Scripts: includes developed scripts and defines when it is executed according to the state of the channel.

APPENDIX II

TENSORFLOW MODEL IMPLEMENTATION

In this research, we discuss that Artificial Intelligence, especially the Deep learning algorithm, which is a subset of machine learning, could help radiologists to increase their accuracy and speed of their detections by automating their diagnostic. Using our interoperability model, the imaging files can easily be accessed by a TensorFlow algorithm, developed by Google. In this appendix, we use a convolutional neural network model with TensorFlow to show how the Donka hospital radiology research lab could automatically detect pneumonia from X-Ray images. This model can predict pneumonia from X-Ray images with an acceptable accuracy (> 90%). This could easily be implemented as a automated detection program available to all the patients that go for a chest x-ray.

1- Importing necessary libraries

```
import os
import tensorflow as tf
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import seaborn as sns
import pandas as pd
from sklearn.utils import class_weight

%matplotlib inline
```

2- Defining a callback function that terminates the training of neural network if it reaches a specific accuracy (e.g., 95%).

```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('acc')>0.95):
            self.model.stop_training = True
            print('\n The model reached 95% accuracy on the training
set.')
```

3- Defining constants variables.

```
EPOCHS = 10
```

```
BATCH_SIZE = 16
INPUT_SHAPE = [200, 200, 3]
IMAGE_SIZE = [200, 200]
```

4- Reading dataset.

```
# Define the base directory of the dataset

base_dir =  os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray')

train_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/train/')

validation_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/val/')

test_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/test/')


# Training normal pictures directory
train_normal_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/train/NORMAL/')

# Training pneumonia pictures directory
train_pneumonia_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/train/PNEUMONIA/')

# Validation normal pictures directory
validation_normal_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/val/NORMAL/')

# Validation pneumonia pictures directory
validation_pneumonia_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/val/PNEUMONIA/')

# Test normal pictures directory
test_normal_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/test/NORMAL/')

# Test pneumonia pictures directory
test_pneumonia_dir = os.path.join('/kaggle/input/chest-xray-
pneumonia/chest_xray/test/PNEUMONIA/')

# Get files' name
train_normal_names = os.listdir(train_normal_dir)
train_pneumonia_names = os.listdir(train_pneumonia_dir)
val_normal_names = os.listdir(validation_normal_dir)
val_pneumonia_names = os.listdir(validation_pneumonia_dir)
test_normal_names = os.listdir(test_normal_dir)
```

```
test_pneumonia_names = os.listdir(test_pneumonia_dir)
```

5- Print training, validation, and test set sizes.

```
# Show the content of the base directory
print('Dataset folder content:')
print(os.listdir(base_dir))
print('')

# Print Dataset size
print('Train Normal Images:',  len(os.listdir(train_normal_dir)))
print('Train Pneumonia Images:',  len(os.listdir(train_pneumonia_dir)))
print('Total Train Images:',  len(os.listdir(train_normal_dir)) +
len(os.listdir(train_pneumonia_dir)))

print('')

print('Validation Normal Images:',
len(os.listdir(validation_normal_dir)))

print('Validation Pneumonia Images:',
len(os.listdir(validation_pneumonia_dir)))

print('Total Validation Images:',
len(os.listdir(validation_normal_dir)) +
len(os.listdir(validation_pneumonia_dir)))

print('')

print('Test Normal Images:',  len(os.listdir(test_normal_dir)))
print('Test Pneumonia Images:',  len(os.listdir(test_pneumonia_dir)))
print('Total test Images:',  len(os.listdir(test_normal_dir)) +
len(os.listdir(test_pneumonia_dir)))
print('')
```

Result of the previous cell.

```
Dataset folder content:
['test', 'val', '__MACOSX', 'train', 'chest_xray']

Train Normal Images: 1341
Train Pneumonia Images: 3875
Total Train Images: 5216

Validation Normal Images: 8
Validation Pneumonia Images: 8
Total Validation Images: 16
```

```
Test Normal Images: 234
Test Pneumonia Images: 390
```

6- Display some of the images from the dataset.

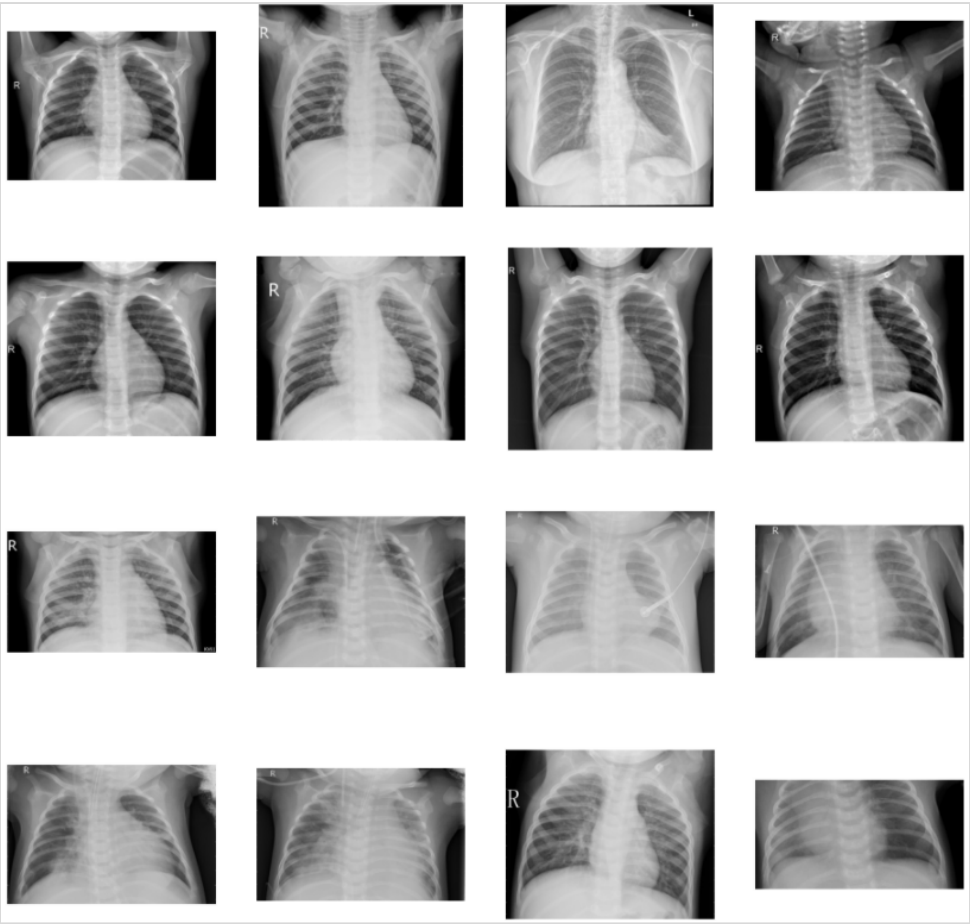
```
# Display some of the training images
fig = plt.gcf()
fig.set_size_inches(16, 16)

next_normal_pix = [os.path.join(train_normal_dir, fname)
                for fname in train_normal_names[0:8]]
next_pneumonia_pix = [os.path.join(train_pneumonia_dir, fname)
                for fname in train_pneumonia_names[0:8]]

for i, img_path in enumerate(next_normal_pix+next_pneumonia_pix):
  sp = plt.subplot(4, 4, i + 1)
  sp.axis('off')

  imgage = mpimg.imread(img_path)
  plt.imshow(imgage, cmap='gray')

plt.show()
```



7- Designing a CNN model

```
base_model = tf.keras.applications.InceptionV3(
                                    input_shape=INPUT_SHAPE,
                                    include_top = False,
                                    weights='imagenet'
                        )

for layers in base_model.layers[:200]:
    layers.trainable = False

model = tf.keras.Sequential([
            base_model,
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Dense(256,  ='relu'),
            tf.keras.layers.Dense(1, activation='sigmoid')
        ])

model.compile(
      loss='binary_crossentropy',
```

```
      optimizer=RMSprop(lr=0.001),
      metrics = ['acc']
)

model.summary()
```

```
Model: "sequential_27"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 4, 4, 2048)        21802784
_________________________________________________________________
flatten_25 (Flatten)         (None, 32768)             0
_________________________________________________________________
dropout_15 (Dropout)         (None, 32768)             0
_________________________________________________________________
dense_52 (Dense)             (None, 256)               8388864
_________________________________________________________________
dense_53 (Dense)             (None, 1)                 257
=================================================================


Total params: 30,191,905
Trainable params: 23,193,409
Non-trainable params: 6,998,496
```

8- Preparing the training and validation dataset. Using data augmentation expands the training dataset and helps to avoid overfitting.

```
# Using data augmentation to expand the dataset size
train_datagen = ImageDataGenerator(rescale=1.0/255,
                                   rotation_range=10,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=False,
                                   vertical_flip=False,
                                   fill_mode='nearest',
                                   validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
                                   train_dir,
                                   batch_size=BATCH_SIZE,
                                   class_mode='binary',
                                   target_size= IMAGE_SIZE,
                                   subset='training')
```

```
validation_datagen = ImageDataGenerator(rescale=1.0/255)

validation_generator = train_datagen.flow_from_directory(
                                    train_dir,
                                    batch_size=BATCH_SIZE,
                                    class_mode='binary',
                                    target_size= IMAGE_SIZE,
                                    subset='validation')
```

```
Found 4173 images belonging to 2 classes.
Found 1043 images belonging to 2 classes.
```

9- Start training model over training data and test the accuracy over validation data on each epoch.

```
callbacks = myCallback()

cw = class_weight.compute_class_weight('balanced',
                np.unique(train_generator.classes),
                train_generator.classes)

class_weights = {0: cw[0], 1: cw[1]}

history = model.fit_generator(train_generator,
                              class_weight=class_weights,
                              epochs=EPOCHS,
                              verbose=1,
                              validation_data=validation_generator,
                              callbacks=[callbacks])
```

Training result for each epoch

```
Epoch 1/10
261/261 [==============================] - 440s 2s/step - loss: 0.8799 -
acc: 0.9032 - val_loss: 7.7989 - val_acc: 0.9415

Epoch 2/10
261/261 [==============================] - 437s 2s/step - loss: 0.2365 -
acc: 0.9449 - val_loss: 7.9512 - val_acc: 0.9377

Epoch 3/10
261/261 [==============================] - ETA: 0s - loss: 0.1920 - acc:
0.9542

 The model reached to 95% accuracy on the training set.
261/261 [==============================] - 433s 2s/step - loss: 0.1920 -
acc: 0.9542 - val_loss: 0.2043 - val_acc: 0.9616
```

10- Test trained model accuracy on the test data

```
test_datagen = ImageDataGenerator(rescale=1.0/255)

test_generator = validation_datagen.flow_from_directory(
                                              test_dir,
```

```
                                                batch_size=BATCH_SIZE,
                                                class_mode='binary',
                                                target_size=IMAGE_SIZE)

# evaluate the model by Test data
scores = model.evaluate_generator(test_generator)
print("\n Test accuracy: %.2f%%" % (scores[1]*100))
```

```
Found 624 images belonging to 2 classes.
Test accuracy: 91.99%
```

11- Plot accuracy and loss for the training and validation images per epochs.

```
# Plot Model Accuracy and Loss
training_acc=history.history['acc']
validation_acc=history.history['val_acc']
training_loss=history.history['loss']
validation_loss=history.history['val_loss']
epochs=range(len(training_acc)) # Get number of epochs

# Plot accuracy of training and validation per epoch
plt.plot(epochs, training_acc, 'r', "Training Accuracy")
plt.plot(epochs, validation_acc, 'b', "Validation Accuracy")
plt.title('Training and validation accuracy')
plt.figure()

# Plot loss of training and validation per epoch
plt.plot(epochs, training_loss, 'r', "Training Loss")
plt.plot(epochs, validation_loss, 'b', "Valiation Loss")
```

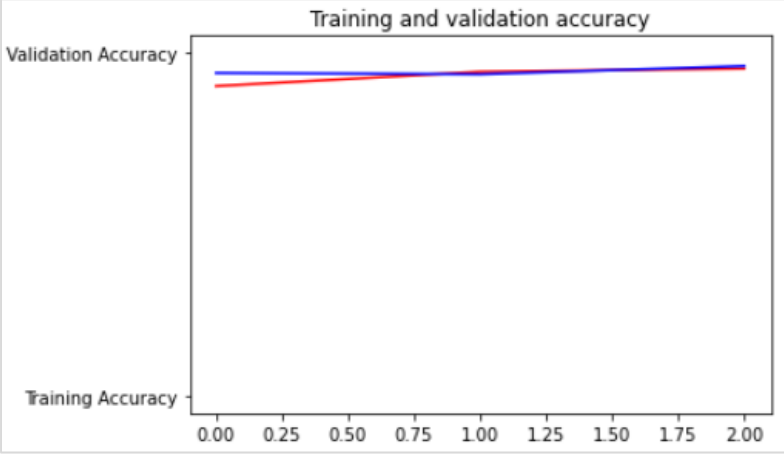


Figure-A II.1 Training and validation accuracy change during the training

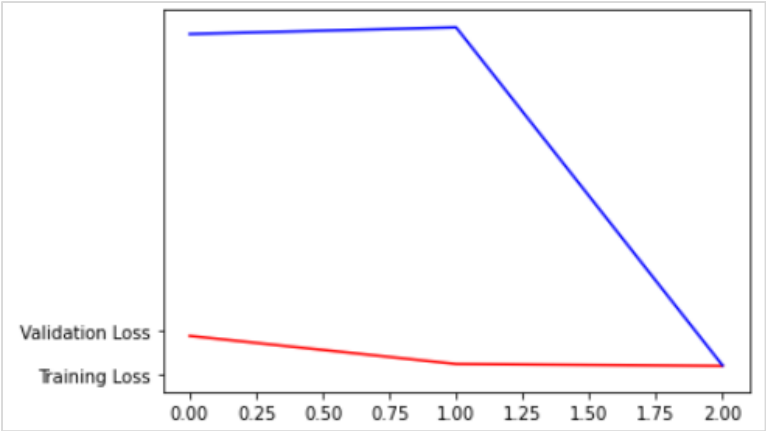


Figure-A II.2 Training and validation loss change during the training

BIBLIOGRAPHY

Arora, D., & Mehta, Y. (2014). Use of picture archiving and communication system for imaging of radiological films in cardiac surgical intensive care unit. Journal Anaesthesiology Clinical Pharmacology, 30(3):447–448. Available online at: https://doi.org/10.4103/0970-9185.137306

Bauman, R.A., Gell, G., & Dwyer, S.J. (1996). Large picture archiving and communication systems of the world—Part 1. Journal of Digital Imaging, 9(3):99–103. Available online at: https://doi.org/10.1007/BF03168603

Bui, A.A.T., Morioka, C., Dionisio, J.D.N. et al. (2007). openSourcePACS: An Extensible Infrastructure for Medical Image Management. IEEE Transactions on Information Technology in Biomedicine, 11(1):94–109. Available online at: https://doi.org/10.-1109/TITB.2006.879595

Capp, M. P., Nudelman, S., Fisher et al. (1981). Photoelectronic Radiology Department. Proceedings of the Conference on Digital Radiography, SPIE, vol. 314, Palo Alto, USA, pp. 1-8. Available online at: https://doi.org/10.1117/12.933008

Cheng, P. H., Chen, H. S., Lai, F., & Lai, J. S. (2010). The strategic use of standardized information exchange technology in a university health system. Telemedicine and e-Health, 16(3), 314-326.

Christensen, C.M., & Raynor, M.E. (2003). The Innovator's Solution: Creating and Sustaining Successful Growth. Harvard Business School Press. 304 p.

Costa, C., Ferreira, C., Bastião, L. et al. (2011). Dicoogle - an Open Source Peer-to-Peer PACS. Journal of Digital Imaging, 24(5):848–856. Available online at: https://doi.org/10.1-007/s10278-010-9347-9

Costa, C., Freitas, F., Pereira, M. et al. (2009). Indexing and retrieving DICOM data in disperse and unstructured archives. International Journal of Computer Assisted Radiology and Surgery, 4(1):71–77. https://doi.org/10.1007/s11548-008-0269-7

Cyr, T., Agarwal, A., & Furht, B. (2013). Brief overview of various healthcare tools, methods, framework and standards. In Handbook of Medical and Healthcare Technologies (pp. 285-295). Springer, New York, NY.

Doi, K., & Huang, H. K. (2007). Computer-aided diagnosis (CAD) and image-guided decision support. Computerized Medical Imaging and Graphics, 31(4):195–197. Available online at: https://doi.org/10.1016/j.compmedimag.2007.02.001

Doran, S. J., d'Arcy, J., Collins, D. J. et al. (2012). Informatics in Radiology: Development of a Research PACS for Analysis of Functional Imaging Data in Clinical Research and

Clinical Trials. RadioGraphics, 32(7):2135–2150. Available online at: https://doi.org/-10.1148/rg.327115138

Duerinckx, A. J. (2003). Introduction to two PACS ’82 Panel Discussions edited by Andre J. Duerinckx, M.D., Ph.D.: ‘‘Equipment Manufacturers’ View on PACS’’and ‘‘The Medical Community’s View on PACS’’. Journal of Digital Imaging, 16(1):29–31. https://doi.org/10.1007/s10278-002-6009-6

Dugar, N. (2018, January 8). PACS with AI can replace PACS without AI. Retrieved from https://www.auntminnieeurope.com/index.aspx?sec=sup&sub=aic&pag=dis&ItemID=615267

Dwyer, S. J., Templeton, A. W., Martin, N. L. et al. (1982). The cost of managing digital diagnostic images. Radiology, 144(2): 313–318. Available online at: https://doi.org-/10.1148/radiology.144.2.6806852

Erickson, B. J., Langer, S., & Nagy, P. (2005). The Role of Open-Source Software in Innovation and Standardization in Radiology. Journal of the American College of Radiology, 2(11): 927–931. https://doi.org/10.1016/j.jacr.2005.05.004

Farahani, N., & Pantanowitz, L. (2016). Overview of Telepathology. Clinics in Laboratory Medicine, 36(1): 101–112. https://doi.org/10.1016/j.cll.2015.09.010

Grant, R. M. (1996). Toward a knowledge-based theory of the firm. Strategic Management Journal, 17(S2): 109–122. Available online at:https://doi.org/10.1002/smj.4250171110

Gulati, R., Puranam, P., & Tushman, M. (2012). Meta-organization design: Rethinking design in interorganizational and community contexts. Strategic Management Journal, 33(6): 571–586. Available online at: https://doi.org/10.1002/smj.1975

Haas, A., Rossberg, A., Schuff, D. et al. (2017). Bringing the Web Up to Speed with WebAssembly. Dans Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (pp. 185–200). New York, NY, USA: ACM. Available online at: https://doi.org/10.1145/3062341.3062363

Hann, I.-H., Roberts, J. A., & Slaughter, S. A. (2013). All Are Not Equal: An Examination of the Economic Returns to Different Forms of Participation in Open Source Software Communities. Information Systems Research, 24(3), 520-538. Available online at: https://doi.org/10.1287/isre.2013.0474

Haque, W., Reed, A., & McCann, A. (2013, June). A framework for secure integration of distributed point-of-care testing results into Electronic Medical Records. In International Conference on Information Society (i-Society 2013) (pp. 73-78). IEEE.

Henderson, J. C., & Venkatraman, H. (1999). Strategic alignment: Leveraging information technology for transforming organizations. IBM systems journal, 38(2.3), 472-484.

Huang, H. K. (2003). Enterprise PACS and image distribution. Computerized Medical Imaging and Graphics: The Official Journal of the Computerized Medical Imaging Society, 27(2-3): 241–253.

Huang, H. K. (2010). PACS and Imaging Informatics: Basic Principles and Applications (2 edition). Hoboken, N.J: Wiley-Blackwell.

Huang, H. K. (2011). Short history of PACS. Part I: USA. European Journal of Radiology, 78(2): 163–176. Available online at: https://doi.org/10.1016/j.ejrad.2010.05.007

Huang, H. K., Andriole, K., Bazzill, et al. (1996). Design and implementation of a picture archiving and communication system: The second time. Journal of Digital Imaging, 9(2): 47. Available online at: https://doi.org/10.1007/BF03168857

Inamura, K., & Kim, J. H. (2011). History of PACS in Asia. European Journal of Radiology, 78(2): 184–189. Available online at: https://doi.org/10.1016/j.ejrad.2010.09.022

Inamura, K., Kousaka, S., Yamamoto, et al. (2003). PACS development in Asia. Computerized Medical Imaging and Graphics: The Official Journal of the Computerized Medical Imaging Society, 27(2-3): 121–128.

Jodogne, S. (2018). The Orthanc Ecosystem for Medical Imaging. Journal of Digital Imaging, 31(3): 341–352. Available online at: https://doi.org/10.1007/s10278-018-0082-y

Jodogne, S., Lenaerts, É., Marquet, L. et al. (2017). Open Implementation of DICOM for Whole-Slide Microscopic Imaging: Dans Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (pp. 81–87). Porto, Portugal: SCITEPRESS - Science and Technology Publications. Available online at: https://doi.org/10.5220/0006155100810087

Kagadis, G. C., Alexakos, C., Langer, S. G., & French, T. (2012). Using an Open-Source PACS Virtual Machine for a Digital Angiography Unit: Methods and Initial Impressions. Journal of Digital Imaging, 25(1): 81–90. Available online at: https://doi.org/10.1007/-s10278-011-9401-2

Katirai, H., & Sax, U. (2005, April). Unlocking the value of clinical information: what you need to do now to enjoy the benefits in the future. In Biennial Conference on Professional Knowledge Management/Wissensmanagement (pp. 330-338). Springer, Berlin, Heidelberg.

Kenwood, C. A. (2001). A business case study of open source software (No. MP-01B0000048). MITRE CORP BEDFORD MA.

Krogh, G. V., & Hippel, E. V. (2006). The Promise of Research on Open Source Software. Management Science, 52(7): 975–983. Available online at: https://doi.org/10.1287/m-nsc.1060.0560

Law, M. Y. Y., & Zhou, Z. (2003). New direction in PACS education and training. Computerized Medical Imaging and Graphics, 27(2-3): 147−156. Available online at: https://doi.org/10.1016/S0895-6111(02)00088-5

Le, A. H. T., Liu, B., & Huang, H. K. (2009). Integration of computer-aided diagnosis/detection (CAD) results in a PACS environment using CAD–PACS toolkit and DICOM SR. International Journal of Computer Assisted Radiology and Surgery, 4(4): 317–329. Available online at: https://doi.org/10.1007/s11548-009-0297-y

Lebre, R., Bastião, L., & Costa, C. (2019). An Accounting Mechanism for Standard Medical Imaging Services. Dans 2019 IEEE 6th Portuguese Meeting on Bioengineering (ENBENG) (pp. 1–4). Available online at: https://doi.org/10.1109/ENBENG.2019.8692545

Liu, B. J., & Huang, H. K. (2008). 13 - PACS and Medical Imaging Informatics for Filmless Hospitals. Dans D. D. Feng (Éd.), Biomedical Information Technology (pp. 279–305). Burlington: Academic Press. Available online at: https://doi.org/10.1016/B978-012373583-6.50017-7

Maniadi, E., Spanakis, E. G., Karantanas, A., & Marias, K. (2015). A supportive environment for the long term management of knee osteoarthritis condition. Proceedings of the 5th EAI International Conference on Wireless Mobile Communication and Healthcare - "Transforming Healthcare through Innovations in Mobile and Wireless Technologies". doi: 10.4108/eai.22-12-2015.151108

Marée, R., Rollus, L., Stévens, B., Hoyoux, R., Louppe, G., Vandaele, R., … Wehenkel, L. (2016). Collaborative analysis of multi-gigapixel imaging data using Cytomine. Bioinformatics (Oxford, England), 32(9): 1395–1401. Available online at: https://doi.-org/10.1093/bioinformatics/btw013

Mendel, J., & Schweitzer, A. (2015). PACS for the Developing World. Journal of Global Radiology, 1(2). Available online at: https://doi.org/10.7191/jgr.2015.1012

Morgan, L., & Finnegan, P. (2014). Beyond free software: An exploration of the business value of strategic open-source. The Journal of Strategic Information Systems, 23(3): 226–238. Available online at: https://doi.org/10.1016/j.jsis.2014.07.001

Nagy, P. (2007). Open Source in Imaging Informatics. Journal of Digital Imaging, 20(1): 1–10. Available online at: https://doi.org/10.1007/s10278-007-9056-1

Nagy, P. G. (2007). The future of PACS. Medical Physics, 34(7): 2676–2682. Available online at: https://doi.org/10.1118/1.2743097@10.1002/(ISSN)2473-4209.Vision2020

Ocasio, W. (1997). Towards an Attention-Based View of the Firm. Strategic Management Journal, 18, 187–206.

Openhub. (2019). dcm4chee. Retrieved from https://www.openhub.net/p/d_14065

Orthanc-server. (2019). Retrieved from https://www.orthanc-server.com/

Pinho, E., & Costa, C. (2016). Extensible Architecture for Multimodal Information Retrieval in Medical Imaging Archives. Dans 2016 12th International Conference on Signal-Image Technology Internet-Based Systems (SITIS) (pp. 316–322). Available online at: https://doi.org/10.1109/SITIS.2016.58

Ribback, S., Flessa, S., Gromoll-Bergmann, K. et al. (2014). Virtual slide telepathology with scanner systems for intraoperative frozen-section consultation. Pathology - Research and Practice, 210(6): 377–382. Available online at: https://doi.org/10.1016/j.prp.2014-.02.007

Roberge, D., MacLeod, B., Hartsock, B., & Asangansi, I. (2011, October). Integrating mobile collection software with health applications. In 2011 IEEE Global Humanitarian Technology Conference (pp. 122-126). IEEE.

Richardson, L., & Ruby, S. (2008). RESTful Web Services. (S.l.): O'Reilly Media. Retrieved from http://shop.oreilly.com/product/9780596529260.do

Salvador, P., Nogueira, A., & Goncalves, F. (2014). DICOM interception system for independent image backup. 2014 IEEE Network Operations and Management Symposium (NOMS), 1–4. Available online at: https://doi.org/10.1109/noms.2014.68-38338

Santos, M., Bastião, L., Costa, C. et al. (2011). DICOM and Clinical Data Mining in a Small Hospital PACS: A Pilot Study. Dans M. M. Cruz-Cunha, J. Varajão, P. Powell, & R. Martinho (Éds), ENTERprise Information Systems (pp. 254–263). Springer Berlin Heidelberg.

Seidel, M.-D. L., & Stewart, K. J. (2011, 23 novembre). An Initial Description of the C-Form. [book-part]. Available online at: https://doi.org/10.1108/S0733558X(2011)0000033-005

Silva, J. M., Pinho, E., Monteiro, E. et al. (2018). Controlled searching in reversibly de-identified medical imaging archives. Journal of Biomedical Informatics, 77:81–90. Available online at: https://doi.org/10.1016/j.jbi.2017.12.002

Singh, P. V., Tan, Y., & Mookerjee, V. (2011). Network Effects: The Influence of Structural Capital on Open Source Project Success. MIS Quarterly, 35(4): 813–829. Available online at: https://doi.org/10.2307/41409962

Teng, C., Mitchell, J., Walker, C. et al. (2010). A medical image archive solution in the cloud. Dans 2010 IEEE International Conference on Software Engineering and Service Sciences (pp. 431–434). Available online at: https://doi.org/10.1109/ICSESS.2010.55-52343

Top, M. (2012). Physicians' Views and Assessments on Picture Archiving and Communication Systems (PACS) in Two Turkish Public Hospitals. Journal of Medical Systems, 36(6): 3555–3562. Available online at: https://doi.org/10.1007/s10916-012-9831-5

Valente, F., Costa, C., & Silva, A. (2013). Dicoogle, a Pacs Featuring Profiled Content Based Image Retrieval. PLOS ONE, 8(5), e61888. Available online at: https://doi.org/10.13-71/journal.pone.0061888

Valente, F., Silva, L. A. B., Godinho, T. M., & Costa, C. (2016). Anatomy of an Extensible Open Source PACS. Journal of Digital Imaging, 29(3), 284-296. Available online at: https://doi.org/10.1007/s10278-015-9834-0

Valeri, G., Zuccaccia, M., Badaloni, A. et al. (2015). Implementation, reliability, and feasibility test of an Open-Source PACS. La radiologia medica, 120(12): 1138–1145. Available online at: https://doi.org/10.1007/s11547-015-0560-y

Vossberg, M., Tolxdorff, T., & Krefting, D. (2008). DICOM image communication in globus-based medical grids. IEEE transactions on information technology in biomedicine: a publication of the IEEE Engineering in Medicine and Biology Society, 12(2): 145–153. Available online at: https://doi.org/10.1109/TITB.2007.905862

Warnock, M. J., Toland, C., Evans, D. et al. (2007). Benefits of Using the DCM4CHE DICOM Archive. Journal of Digital Imaging, 20(1): 125–129. Available online at: https://doi.-org/10.1007/s10278-007-9064-1

Wetering, R. V. D., & Batenburg, R. (2014). Towards a Theory of PACS Deployment: An Integrative PACS Maturity Framework. Journal of Digital Imaging, 27(3): 337–350. Available online at: https://doi.org/10.1007/s10278-013-9671-y

Webster, J. D., & Dunstan, R. W. (2014). Whole-Slide Imaging and Automated Image Analysis: Considerations and Opportunities in the Practice of Pathology. Veterinary Pathology, 51(1): 211–223. Available online at: https://doi.org/10.1177/03009858135-03570

Woodbridge, M., Fagiolo, G., & O'Regan, D. P. (2013). MRIdb: Medical Image Management for Biobank Research. Journal of Digital Imaging, 26(5): 886–890. Available online at: https://doi.org/10.1007/s10278-013-9604-9

Xiong, W., Du, J., Nie, B. et al. (2017). Research on DICOM file compression and offline storage platform. Dans 2017 6th International Conference on Computer Science and Network Technology (ICCSNT) (pp. 520–524). Available online at: https://doi.org/10-.1109/ICCSNT.2017.8343753

Xue, Y., & Liang, H. (2007). Understanding PACS Development in Context: The Case of China. IEEE Transactions on Information Technology in Biomedicine, 11(1): 14–16. Available online at: https://doi.org/10.1109/TITB.2006.879580

Yang, C.-T., Chen, C.-H., & Yang, M.-F. (2010). Implementation of a medical image file accessing system in co-allocation data grids. Future Generation Computer Systems, 26(8): 1127–1140. Available online at: https://doi.org/10.1016/j.future.2010.05.013

Ye, Y., & Kishida, K. (2003). Toward an Understanding of the Motivation Open Source Software Developers. 25th International Conference on Software Engineering, 2003 (pp. 419–429). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dl.acm.org/citation.cfm?id=776816.776867

Zhang, S., Wu, H., Wang, X. et al. (2018). The application of deep learning for diabetic retinopathy prescreening in research eye-PACS. Dans J. Zhang & P.-H. Chen (Éds), Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications (p. 38). Houston, United States: SPIE. Available online at: https://doi.org/10.1117/12.2296673