



Miguel Gomes
Lage Valério

Dicoogle Analytics for Business Intelligence

Plataforma de Análise de Dados para o Dicoogle



**Miguel Gomes
Lage Valério**

Dicoogle Analytics for Business Intelligence

Plataforma de Análise de Dados para o Dicoogle

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

o júri / the jury

presidente / president

Prof. Doutor Augusto Marques Ferreira da Silva

Prof. Auxiliar do Dep. Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Joel P. Arrais

Prof. Auxiliar Convidado Dep. Informática da Universidade de Coimbra

Prof. Doutor Carlos Manuel Azevedo Costa

Prof. Auxiliar do Dep. Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

**agradecimentos /
acknowledgements**

Agradeço toda a ajuda a todos os meus colegas e companheiros.

Palavras Chave

PACS, DICOM, BioInformática, Big Data, Inteligência Empresarial, Análise de Dados

Resumo

As últimas décadas têm sido caracterizadas pelo aumento do número de estudos imagiológicos produzidos, elementos fundamentais no diagnóstico e tratamento médico. Estes são armazenado em repositórios dedicados e são consumidos em estações de visualização que utilizam processos de comunicação normalizados. Os repositórios de imagem médica armazenam não só imagem médica, mas também uma grande variedade de metadados que têm bastante interesse em cenários de investigação clínica e em processos de auditoria que visam melhorar a qualidade de serviço prestado. Tendo em atenção a tremenda quantidade de estudos produzidos atualmente nas instituições de saúde, verificamos que os métodos convencionais são ineficientes na exploração desses dados, obrigando as instituições a recorrer a plataformas de Inteligência Empresarial e técnicas analíticas aplicadas. Neste contexto, esta dissertação teve como objetivo desenvolver uma plataforma que permite explorar todos os dados armazenados num repositório de imagem médica. A solução permite trabalhar em tempo real sobre os repositórios e não perturba os fluxos de trabalho instituídos. Em termos funcionais, oferece um conjunto de técnicas de análise estatística e de inteligência empresarial que estão acessíveis ao utilizador através de uma aplicação Web. Esta disponibiliza um extenso painel de visualização, gráficos e relatórios, que podem ser complementados com componentes de mineração de dados. O sistema permite ainda definir uma multitude de consultas, filtros e operandos através do uso de uma interface gráfica intuitiva.

Keywords

PACS, DICOM, BioInformatics, Big Data, Business Intelligence, Data Analytics

Abstract

In the last decades, the amount of medical imaging studies and associated metadata available has been rapidly increasing. These are mostly used to support medical diagnosis and treatment. Nonetheless, recent initiatives claim the usefulness of these studies to support research scenarios and to improve the medical institutions business practices. However, their continuous production, as well as the tremendous amount of associated data, make their analysis difficult by conventional workflows devised up until this point. Current medical imaging repositories contain not only the images themselves, but also a wide-range of valuable metadata. This creates an opportunity for the development of Business Intelligence and analytics techniques applied to this Big Data scenario. The exploration of such technologies has the potential of further increasing the efficiency and quality of the medical practice.

This thesis developed a novel automated methodology to derive knowledge from multimodal medical imaging repositories that does not disrupt the regular medical practice. The developed methods enable the application of statistical analysis and business intelligence techniques directly on top of live institutional repositories. The resulting application is a Web-based solution that provides an extensive dashboard, including complete charting and reporting options, combined with data mining components. Furthermore, the system enables the operator to set a multitude of queries, filters and operands through the use of an intuitive graphical interface.

CONTENTS

| | |
|---|-----|
| CONTENTS | i |
| LIST OF FIGURES | v |
| LIST OF TABLES | vii |
| GLOSSARY | ix |
| 1 INTRODUCTION | 1 |
| 1.1 Overview | 1 |
| 1.2 Goals | 2 |
| 1.3 Outlines | 2 |
| 2 MEDICAL IMAGING LABORATORIES | 3 |
| 2.1 Overview | 3 |
| 2.2 Picture Archive and Communications System | 4 |
| 2.3 Digital Image Communications in Medicine | 5 |
| 2.3.1 DICOM Data Structure | 6 |
| 2.3.2 Digital Image Communications in Medicine (DICOM) Objects and Hierarchy | 7 |
| 2.3.3 DICOM Communications | 9 |
| 3 TECHNOLOGIES | 15 |
| 3.1 Technical Overview | 15 |
| 3.1.1 Big Data | 15 |
| 3.1.2 Data Cleansing | 16 |
| 3.1.3 Data Mining | 19 |
| 3.2 KDD and Data / Business Analytics | 21 |
| 3.3 Technologies Evaluation | 22 |
| 3.3.1 WEKA | 22 |
| 3.3.2 RapidMiner | 24 |
| 3.3.3 Orange | 25 |
| 3.3.4 OpenRefine | 25 |
| 3.3.5 R | 26 |
| 3.3.6 Python Data Analysis Ecosystem | 27 |
| 3.3.7 Project Adequacy | 28 |
| 3.3.8 Quantitative Evaluation | 28 |

| | | |
|-------|--|----|
| 4 | BUSINESS INTELLIGENCE IN MEDICAL IMAGING | 31 |
| 4.1 | Related Work | 31 |
| 4.2 | Dicooogle Project | 32 |
| 4.3 | Requirements | 34 |
| 4.3.1 | Functional Requirements | 34 |
| 4.3.2 | Non Functional Requirements | 35 |
| 5 | ARCHITECTURE | 37 |
| 5.1 | Client-Server model | 37 |
| 5.2 | Single Page Application | 37 |
| 5.3 | Methods | 38 |
| 5.3.1 | Rule Control | 39 |
| 5.3.2 | Anomaly Detection | 41 |
| 5.3.3 | View Control | 42 |
| 5.3.4 | Dashboard | 43 |
| 5.3.5 | Charts | 44 |
| 5.3.6 | Real-Time Analytics | 44 |
| 6 | IMPLEMENTATION | 45 |
| 6.1 | Model Overview | 45 |
| 6.2 | Client Side Specification - Presentation Layer | 46 |
| 6.2.1 | React | 46 |
| 6.2.2 | Redux | 47 |
| 6.2.3 | Bootstrap | 48 |
| 6.2.4 | Plotly.js | 48 |
| 6.2.5 | React-Router | 49 |
| 6.3 | Server Side Specification - Business Layer | 49 |
| 6.3.1 | Django | 51 |
| 6.3.2 | SciPy Stack and scikit-learn | 52 |
| 6.3.3 | Celery | 53 |
| 6.3.4 | Redis | 53 |
| 6.4 | Server Side Specification - Persistence Layer | 54 |
| 6.4.1 | Data Model | 54 |
| 6.4.2 | Database | 56 |
| 6.5 | Pagination | 56 |
| 7 | RESULTS | 61 |
| 7.1 | Interface Components | 61 |
| 7.1.1 | Forms | 61 |
| 7.1.2 | Tabular Data | 63 |
| 7.1.3 | Charts | 64 |
| 7.2 | Methods | 64 |
| 7.2.1 | Rule Control | 65 |
| 7.2.2 | Anomaly Detection | 67 |
| 7.2.3 | View Control | 68 |
| 7.2.4 | Dashboard | 71 |
| 7.2.5 | Charts | 71 |
| 7.3 | Deployment | 72 |
| 7.3.1 | Docker Images | 72 |
| 7.3.2 | Client | 72 |

| | | |
|-------|--------------------------|----|
| 7.3.3 | Server | 73 |
| 7.3.4 | Docker Compose | 73 |
| 8 | CONCLUSION | 75 |
| 8.1 | Conclusion | 75 |
| 8.2 | Future Work | 76 |
| 8.3 | Contributions | 76 |
| | REFERENCES | 77 |
| | APPENDIX | 81 |

LIST OF FIGURES

| | | |
|------|--|----|
| 2.1 | PACS Workflow | 4 |
| 2.2 | DICOM Data Elements [14] | 6 |
| 2.3 | DICOM Information Entity Hierarchy | 7 |
| 2.4 | DICOM Storage Service | 10 |
| 2.5 | DICOM Query Service | 11 |
| 2.6 | DICOM Retrieve Service | 11 |
| 2.7 | DICOM WADO | 12 |
| 3.1 | 5-fold Cross-Validation | 21 |
| 3.2 | Knowledge Discovery in Databases [25] | 22 |
| 3.3 | Weka Explorer view | 23 |
| 3.4 | Weka Knowledge Flow view [28] | 24 |
| 3.5 | RapidMiner process: Creating a Decision Tree Model | 25 |
| 3.6 | OpenRefine | 26 |
| 4.1 | Use Case Diagram | 35 |
| 5.1 | DicooogleBI Modules | 39 |
| 6.1 | DicooogleBI technologies | 45 |
| 6.2 | WebSockets Basic Communication | 51 |
| 6.3 | Publish Subscribe Messaging Pattern | 53 |
| 6.4 | Data Model | 55 |
| 6.5 | Filter Models | 55 |
| 6.6 | Action Models | 56 |
| 6.7 | Pagination Process | 59 |
| 7.1 | Example Form | 62 |
| 7.2 | Multiple Inputs | 62 |
| 7.3 | Field Suggestions | 62 |
| 7.4 | Table Loading Data | 63 |
| 7.5 | Data Sorted by StudyDate | 64 |
| 7.6 | Chart with Class 3 Disabled | 64 |
| 7.7 | Filters Menu | 65 |
| 7.8 | Filter All Entries with PatientBirthDate > 01-01-1950 | 65 |
| 7.9 | Actions Menu | 66 |
| 7.10 | Replace All CT Occurrences in the Modality Field with Tomography | 66 |
| 7.11 | Setting a Rule's Priority | 67 |

| | | |
|------|---|----|
| 7.12 | Example Ordinal Description | 67 |
| 7.13 | Example Nominal Description | 68 |
| 7.14 | Example Numerical Description as Ordinal | 68 |
| 7.15 | Aggregation Performed on Specific Columns | 69 |
| 7.16 | Example Subset | 70 |
| 7.17 | Updated Description | 70 |
| 7.18 | Select View Menu | 70 |
| 7.19 | Example Dashboard | 71 |
| 7.20 | Example Chart Input | 71 |
| 7.21 | Example Chart | 72 |

LIST OF TABLES

| | | |
|-----|---|----|
| 2.1 | CR Image IOD Module Table. Adapted and Abridged from [16] | 8 |
| 2.2 | DICOM Message Service Element (DIMSE) Services [18] | 9 |
| 2.3 | WADO-RS Actions [19] | 12 |
| 2.4 | QIDO-RS Actions [19] | 13 |
| 2.5 | STOW-RS Actions [19] | 14 |
| 3.1 | Data Cleansing and Data Mining Tools Overview | 29 |
| 6.1 | REST Hypertext Transfer Protocol (HTTP) methods [41] | 50 |
| 1 | Available DICOM tags | 85 |

GLOSSARY

| | |
|----------------|---|
| BI | Business Intelligence |
| DM | Data Mining |
| DC | Data Cleansing |
| KDD | Knowledge Discovery in Databases |
| ML | Machine Learning |
| PACS | Picture Archive and Communications System |
| DICOM | Digital Image Communications in Medicine |
| CT | Computer Tomography |
| CR | X-Ray |
| US | Ultrasounds |
| VR | Value Representation |
| VL | Value Length |
| PN | Person Name |
| UI | User Interface |
| SQ | Sequence of Items |
| OOP | Object-Oriented Programming |
| UID | Unique Identifier |
| IOD | Information Object Definition |
| IE | Information Entity |
| SOP | Service Object Pair |
| AE | Application Entity |
| SCU | Service Class User |
| SCP | Service Class Provider |
| DIMSE | DICOM Message Service Element |
| WADO | Web Access to DICOM Objects |
| WADO-RS | Web Access to DICOM Objects (WADO)-Restful Service |
| QIDO-RS | Query based on ID for DICOM Objects - Restful Service |
| STOW-RS | Store Over the Web - Restful Service |

| | |
|--------------|---|
| CRAN | Comprehensive R Archive Network |
| GREL | General Refine Expression Language |
| P2P | Peer to Peer |
| URL | Uniform Resource Locator |
| AI | Artificial Intelligence |
| HIS | Hospital Information System |
| RIS | Radiology Information System |
| REST | Representational State Transfer |
| HTTP | Hypertext Transfer Protocol |
| HTML | HyperText Markup Language |
| CSS | Cascading Style Sheets |
| JS | JavaScript |
| API | Application Interface |
| XML | Extensible Markup Language |
| JSON | JavaScript Object Notation |
| URI | Uniform Resource Identifier |
| TCP | Transmission Control Protocol |
| ORM | Object-relational mapping |
| SQL | Structured Query Language |
| RDBMS | Relational Database Management System |
| DRF | Django Representational State Transfer (REST) Framework |
| HDFS | Hierarchical Data Format Store |
| DOM | Document Object Model |
| HL7 | Health Level 7 |
| DBMS | DataBase Management System |
| IO | Input/Output |
| CSV | Comma-Separated Values |
| GUI | Graphical User Interface |
| SPA | Single Page Application |
| UI | User Interface |
| MVC | Model-View-Controller |
| OLAP | Online Analytical Processing |
| ETL | Extract, Transform and Load |

INTRODUCTION

This chapter makes an introduction about the thesis' scenario and problem. It outlines some of the main issues related medical imaging data analytics that the proposed system intends to correct, while also summarizing the work's main goals, and providing an overview of the remaining chapters.

1.1 OVERVIEW

The production of medical images in digital format has been growing in the last decades, representing an important and indispensable element in supporting medical decisions. However, it also imposes new challenges in managing resources and efficiency, where cost reduction and service quality improvement are two key points to evaluate.

Due to the amount of data conceived in the form of health exams, medical images, clinical trials, among others, un-automated data analysis and maintenance has become impractical, due to the size and complexity of the repositories, but also due to budget and personnel constraints. However, this scenario provides a great opportunity for the development of Business Intelligence (BI) systems that must be able to consume data provided by a Picture Archive and Communications System (PACS) archive, which is responsible for managing the digital image workflow in medical institutions, by providing medical imaging storage, distribution and visualization among different systems [1].

PACS repositories contain not only the medical images themselves but also metadata stored alongside them, referring to the circumstances under which the image was captured, such as the modality, equipment details, medical report, among others. Analysis of this metadata has already been the subject of several studies in the last years that demonstrate the benefits of systematic data analysis, such as in radiation dosage surveillance processes, efficiency analysis of professional practices, study of cost-effective diagnosis, and many others [2]–[5].

Due to the permissive constraints applied at the image repositories, as well as the very distinct conditions in which the data is captured, a large amount of the data stored is inconsistent, or even incorrect. A BI system offers a way of applying constraints to the data, providing an automated way for connecting inaccurate records or even remove them from the record set, as well as providing a set of tools that allow its posterior analysis.

Another potential benefit of BI, the automated detection of relationships and patterns “hidden” among vast amounts of data, aspect of huge interest in the medical field. Application of these techniques

has yet to be severely explored, mainly due to unreliable data constraints. However, it allows the detection of interesting and even groundbreaking patterns that would otherwise be impossible to perform by hand.

1.2 GOALS

This document proposes a new Business Intelligence system, Dicooogle Business Intelligence, that is to complement the Dicooogle PACS repository, by providing a robust system capable of maintaining consistent data across the whole repository, as well as making available simple, yet powerful data analysis tools. The developed system should support, among other features, Real-Time Analytics capabilities, as well as making available all of its features through a consistent and unified interface.

1.3 OUTLINES

- **Chapter 2:** This chapter provides a description of the state-of-the-art in medical imaging laboratories, by overviewing PACS systems, as well as the DICOM standard.
- **Chapter 3:** This chapter provides an overview of the technical concepts that are necessary in order to implement a Business Intelligence system, as well as an analysis of the different tools that allow its implementation.
- **Chapter 4:** This chapter provides an analysis of the related and previous work on which the proposed system was based, as well as the requirements the intended system must fulfil.
- **Chapter 5:** This chapter describes the proposed system's architecture, as well as some of the reasoning behind the implemented choices.
- **Chapter 6:** This chapter describes the methods developed throughout the system, focusing on both their implementation, as well as presentation.
- **Chapter 7:** This chapter discusses the results of the developed system.
- **Chapter 8:** This chapter presents some final considerations, as well as guidelines for future work.

MEDICAL IMAGING LABORATORIES

This chapter describes the current state of art in systems and technologies related to medical imaging, such as PACS and DICOM. This description is crucial in order to understand the current issues that created the need for the development of the proposed system.

2.1 OVERVIEW

Medical imaging is defined as the technique and process used to acquire a visual representation of parts of the human body. This enables the gathering of information about target organs, empowering the support of the clinical decision-making process related to diagnosis. There are several types of modalities set according to the distinct analysis needs, all based in different imaging technologies, like Computer Tomography (CT), X-Ray (CR) or Ultrasounds (US).

Historically, the first report of medical imaging happened over a century ago, with a primordial version of today's x-rays. Since then, although the equipment has evolved into safer and more reliable devices, it was still mostly analogical, relying on mostly physical storage, such as tapes[6]. Obviously, this is highly impractical, as archiving and distributing information of the captured images becomes a long and expensive task, especially as the amount of data starts to increase. With the advent of computational systems, the production and management of images has shifted to the digital form, which led to the development of PACS.

The adoption of this digital equipment has been steadily increasing, even more since the price gap is almost insignificant when compared to their analog counterparts. It is important to note that these systems do not entirely fix the analog systems' constraints. As a matter of fact, they have some added constraints, mainly related to patient information confidentiality [7]. Despite this, the advantages of having a digital archiving and distribution system clearly outweigh the drawbacks [8].

As such, PACSs are capable of greatly improving patient care quality, by reducing medical workflow delays, as well as providing controlled costs to organizations, making them an essential reference in the field of medical imaging analysis.

2.2 PICTURE ARCHIVE AND COMMUNICATIONS SYSTEM

With the recent growth in medical imaging technology, the ability of film-based systems to satisfy the needs of radiology departments has decreased. Besides, the price discrepancy between analog and digital no longer justifies opting for the first, particularly when taken into consideration the added benefits of implementing digital equipment into the medical practice. Thus, the need for a system that could bring digital medical imaging archives and digital distribution networks together arose.

PACS stands for Picture Archive and Communications System. It is a workflow-integrated imaging system designed to streamline acquisition, storage, distribution and visualization operations throughout the entire patient care delivery process, by defining a set of hardware, software and communications technologies [1][9].

PACS can be split into three main steps: Acquisition, Distribution and Visualization [9].

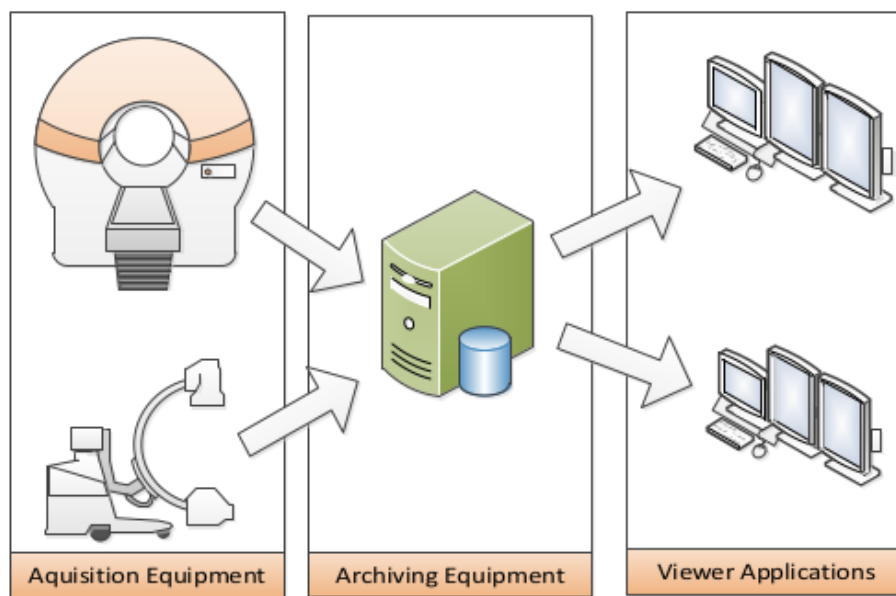


Figure 2.1: PACS Workflow

Acquisition is the process of producing medical images. These images can be obtained using two distinct methods, depending on the type of equipment used. The first one occurs when the image is produced by digital equipment. In this case, the images are acquired directly during the examination procedure. The second one occurs when the image is produced by analog equipment. In this case, the image has to be converted to a digital format, using scanning techniques. This method allows backwards compatibility with older equipment, thus making it widely used even today [8].

Distribution is the process responsible for delivering electronic images and related information between different PACS nodes for timely patient care. The main advantage of the distribution process is that, when considering an institution that is composed of several hospitals and clinics, it removes the necessity of having similar specialist services in every institution. A particular clinical service like radiology, for instance, can be centralized, and therefore shared between all entities that are part of the institution [1]. Another key point of the distribution process is that it allows the modalities and archive to have separate locations. Finally, distribution also allows sharing studies among different institutions, providing greater investigation opportunities.

Visualization is the presentation layer of a PACS. Image visualization is extremely important, since physicians must have an intuitive and reliable way of analysing studies.

A typical PACS workflow cycle is divided into several stages, starting with the patient's registration in the Hospital Information System (HIS) and Radiology Information System (RIS), passing through the examination itself, and ending with the images' analysis and storage. There are currently three main PACS architectures that allow the implementation of the described workflow [10].

In the **stand-alone** architecture, when studies are stored in the central repository, they are automatically sent to the registered workstations. Due to this, there is very little study loss. However, since the studies are sent to all workstations without being enquired to do so, they can be visualized by anyone in the system, eventually leading to security violations.

The **client-server** architecture relies on having one central repository, responsible for holding all of the submitted studies. When needed, workstations retrieve the desired studies, and discard them after the analysis. This architecture, although being limited by transfer times from the repository to the workstations, provides better access control.

Finally, there is the **web-based** architecture. As the name implies, in this architecture the review process is done inside a web application. Since this application is accessed through a web browser, there is no platform constraint, therefore providing incredible portability. This is also this architecture's main downfall, as the browser's features might be rather limiting. However, since this architecture is currently the most widely adopted, these limits are starting to become non-existent.

It is possible to see the importance of PACS since it provides a set of repositories and information systems which allow us to greatly increase the efficiency of the healthcare delivery process [1]. However, they continue to have some issues. The first issue is that medical images tend to generate huge amounts of data, that influence both the acquisition step, since all the data has to be stored, and the distribution step, since adequate transmission times between the different nodes have to be maintained. The second issue is related to patient confidentiality, since patient information exposure could severely compromise both the patient and the institution responsible [7].

2.3 DIGITAL IMAGE COMMUNICATIONS IN MEDICINE

In the early years of digital medical imaging, there was no unifying medical image standard. Because of this, every imaging equipment manufacturer had their own implementation, leading to incompatibility issues when exchanging images between different types of equipment. Thus, the development of a vendor-independent standard that allowed the integration of different digital products, modalities, archives and information systems was of primary interest [11].

DICOM is the international leading standard for storage and transfer of image data in medical applications. Its main objective is to allow connectivity, compatibility and workflow optimizations between different medical imaging equipment [12]. Therefore, DICOM is responsible for defining not only the image file format itself, but also the communications protocol required for different equipments to interact among them.

The development and proliferation of the DICOM standard was fundamental for the success and expansion of the PACS concept, since it guarantees interoperability between systems that process, store and transmit digital medical images [13].

2.3.1 DICOM DATA STRUCTURE

All information related to patients, studies, images and others is represented by DICOM Objects. These objects (also called Data Sets) are a mix of Data Elements (or Data Attributes), which are defined by DICOM according to the selected modality.

Every DICOM Element is composed of three mandatory fields and one optional that follow a TLV (Tag / Length / Value) structure, as represented in Figure 2.2.

The first field is a **Tag** that uniquely identifies the element, as well as its properties. Every tag is comprised of two numbers called Group and Element. The Group field is used to identify Elements related to one another, usually implying a relation between Data Elements and real world Object, while the Element field is used to identify the element itself. For instance, the tag (0x0100, 0x0010) matches the Patient's group (0x0100) and the Patient Name (0x0010) as the identifier within the group.

The second (and optional) field is called **Value Representation (VR)**. This field identifies the element data type, like for instance the Person Name (PN) data type. The need for this field is set according to the agreed transfer syntax (this will be further developed in 2.3.3). This field is optional because the VR can be regularly inferred from the remaining mandatory fields. For instance, in the tag (0x0100, 0x0010) (that, once again, identifies a Patient Name element), the VR field will always be PN, making this field redundant. The entity responsible for matching a given Data Element (by its Tag) to its respective VR (as well as the Element's name, its data type and multiplicity, among others), and thus allowing the VR field's omission, is the DICOM Data Dictionary. Despite very extensive, with around 2000 elements, it is impossible for the DICOM Dictionary to have an entry for every possible element. Therefore, this dictionary allows the insertion of private Data Elements, allowing the equipment vendors not to be limited by the existing fields [14][15].

The third field is the **Value Length (VL)** field. Since DICOM is a binary protocol (which means there is no way to denote the ending of a particular field, unlike text-based protocols), the length is required in order to decode the data correctly.

At last, there is the **Value Field**. This field is responsible for holding the actual binary data of the element, and it must have even length. This is achieved by using different padding methods according to the respective VR field. It is important to note that this field might eventually contain other Data Sets if the VR field is Sequence of Items (SQ) [14].

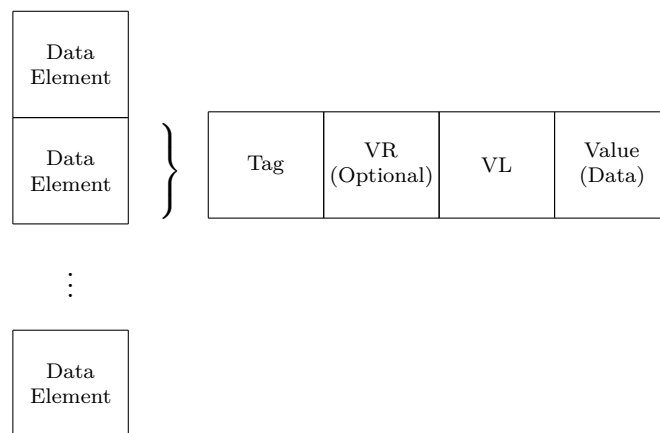


Figure 2.2: DICOM Data Elements [14]

2.3.2 DICOM OBJECTS AND HIERARCHY

DICOM Objects' organization follows an hierarchical approach similar to Object-Oriented Programming, reflecting a model of real world objects. In order to identify different objects, DICOM uses Unique Identifiers (UIDs).

These UIDs, as the name implies, are unique identification keys, used to unequivocally identify a given instance of an object. Each UID is composed of two parts: an `<org_root>` and a `<suffix>`, usually represented in the following format: "`<org_root>.<suffix>`". The first part uniquely identifies an organization, such as a manufacturer or a research organization, while the second identifies the object itself, which is usually a transfer syntax or a Service Object Pair (SOP) class, and it must be unique in the scope of the organization.

It is now important to define how DICOM organizes the real world objects. It does so by establishing a hierarchy of Information Entitys (IEs): Patient, Study, Series and Instance. This hierarchy is meant to provide a real world representation, where a Patient is a part of one or more Studies, a Study belongs to one or more Series, and a Series can have a number of images associated with it, as represented in Figure 2.3.

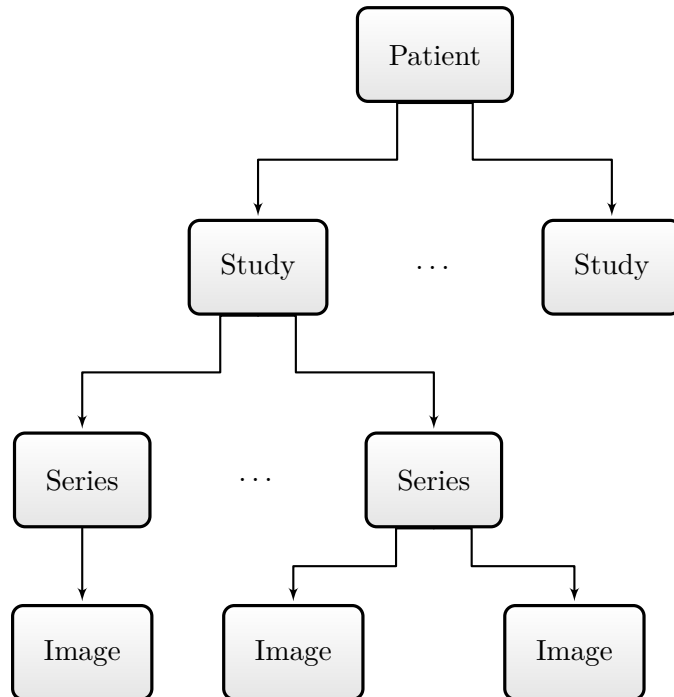


Figure 2.3: DICOM Information Entity Hierarchy

DICOM defines a data model using SOP classes and Information Object Definitions (IODs) in order to handle the IEs (this will be further developed in 2.3.3). DICOM specifies both Composite IODs (composed of only one IE) and Normalized IODs (composed of multiple IEs). In practice, the purpose of an IOD is to map one or more IEs to a set of Modules.

DICOM Modules are used to group elements into the same logical unit, and each IEs in a given IOD has a defined set of modules, which can be Mandatory, Conditional (this field becomes mandatory if a particular condition is met, else it should not be used) or User-Option (it is up to the user to make this field mandatory or not). Table 2.1 represents the CR image IOD.

| Information Entity | Module | Attributes |
|--------------------|------------------------|--|
| Patient | Patient | Patient's Name Patient ID Patient's Birth Date Patient's Sex ... |
| | Clinical Trial Subject | ... |
| Study | General Study | ... |
| | PatientStudy | ... |
| | Clinical Trial Study | ... |
| Series | General Series | ... |
| | Clinical Trial Series | ... |
| Image | General Image | ... |
| | Image Plane | ... |
| | Image Pixel | ... |
| | CT Image | ... |
| | ... | ... |

Table 2.1: CR Image IOD Module Table. Adapted and Abridged from [16]

Given this whole hierarchy, it can be established that an IOD allows building an Entity-Relationship Model between different IEs, the so-called IOD Entity-Relationship Model.

2.3.3 DICOM COMMUNICATIONS

In order to model the intercommunication of medical imaging equipment, DICOM makes use of the already existing TCP/IP protocol.

In a given network, every service or application present is identified by a triple, which consists of an Application Entity (AE) Title, the host's IP address, and the Port where the application is running [17][18].

DICOM's AEs provide services for each other. This leads to a sort of client-server architecture, where the service-requesting AE (client) is called Service Class User (SCU), and the service-granting AE (server) is called Service Class Provider (SCP).

In order to establish communication, peer AEs must first go through an Association phase. During this phase, the AEs negotiate the transfer syntax (transfer parameters such as byte encoding, compression to be used, among others) and the services supported. These services, as well as their associated rules, are defined as SOP Classes, which in turn are defined by the union of an IOD and a DIMSE Service Group.

Like IODs, SOP Classes can also be either Composite, if they are composed of a Composite IOD and a set of DIMSE-C Services; or Normalized, if they are composed of a Normalized IOD and a set of DIMSE-N Services. These DIMSE Services categories contain a series of services, each one representing an operation like **STORE**, **GET**, among others, that can be applied to the selected IOD [18]. The most relevant DIMSE services are represented in Table 2.2.

| Service | Description |
|---------|--|
| C-STORE | Request the storage of a Composite SOP Instance's information |
| C-FIND | Obtain a series of Attributes' values from Composite SOP Instances |
| C-GET | Fetch the information from one or more Composite SOP Instances |
| C-MOVE | Move the information from one or more Composite SOP Instances |

Table 2.2: DIMSE Services [18]

Only after all these parameters are agreed upon by both the SCP and the SCU, the transfer phase is initiated, in which a series of commands are exchanged between the SCU and the SCP.

In the following sections, the main services made available by the DICOM standard will be described.

STORAGE SERVICE CLASS

The Storage Service's goal is to transfer images between two different DICOM nodes. Therefore, it is one of the most important services available in a PACS, since it is used, for instance, for storing a study in the repository.

This service represents a typical case of the SCP/SCU pattern: a Storage SCU sends a **C-STORE-RQ** (Request), containing the DICOM Object image to be transferred to a stand-by Storage SCP, to which the SCP replies with a **C-STORE-RSP** (Response), acknowledging the reception of the data. This exchange is replicated for every image that the SCU intends to send. This process is described in Figure 2.4.

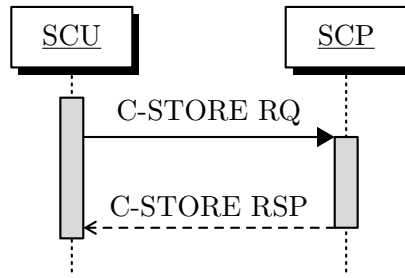


Figure 2.4: DICOM Storage Service

QUERY / RETRIEVE SERVICE CLASS

The Query / Retrieve Service is usually performed at the workstations' level in a PACS, and it is used by PACS nodes to query a DICOM archive (like a PACS repository) about a particular object's content, eventually retrieving that content.

The Query is usually performed using DICOM Attributes as parameters, such as Patient Name, Modality, among others. After the Query's completion, the Retrieve service might be executed, retrieving the Query's results from that node.

In order to use this Service, two commands are required: **C-FIND** and **C-MOVE**.

The SCU queries the listening SCP with a **C-FIND-RQ**, which contains a particular query, like for instance "PatientSex = F", in order to get all the Female Patients. The queried SCP will then respond with a **C-FIND-RSP** with a Pending status for each found object, as well as a final one with a Success status, indicating the end of the list of results. This pipeline is described in Figure 2.5.

The retrieval of the Query's response objects is done via the **C-MOVE** command. A **C-MOVE-RQ** command with a list of the desired object's UIDs is sent by the SCU, to which the SCP responds with a **C-STORE-RQ** for every object requested. Like in the Storage Service Class, every **C-STORE-RQ** is replied with a **C-STORE-RSP** by the SCU. However, in order for this to happen, the SCU needs to act as an SCP with a Storage Service. When the **C-STORE-RSP** corresponding to the last object is received by the SCP, it is sent a **C-MOVE-RSP** to the SCU, ending the transfer. This exchange is described in Figure 2.6.

It is also important to note that, by using the **C-MOVE-RQ**, the SCU can also order the query results to be sent to a machine other than its own.

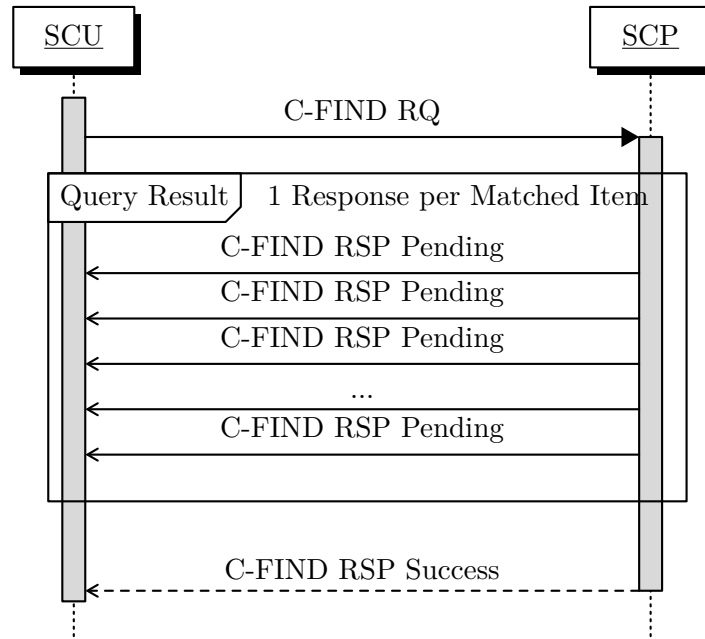


Figure 2.5: DICOM Query Service

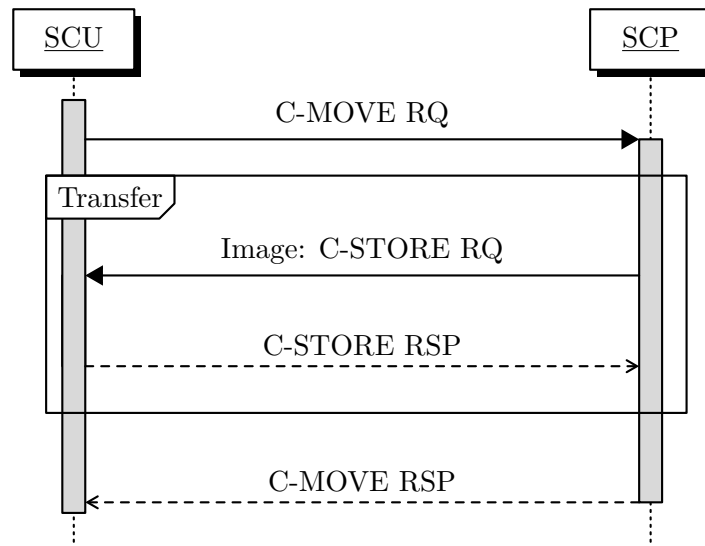


Figure 2.6: DICOM Retrieve Service

WADO

All the services previously described operate in the Application Layer of the OSI Model. However, the DICOM Upper Layer uses TCP which, despite being a very reliable protocol, has some limitations, the main one being that TCP usually has very strict security restrictions. The DICOM Upper Layer, being a subset of TCP, is often not allowed at the firewall level, and its access ends up being blocked in private networks.

In order to solve some of these issues, DICOM defines WADO, that uses the HTTP Protocol, which is usually not blocked in most networks. It makes use of the standard HTTP methods, such as GET and POST, and by refers to given objects by their UIDs, as represented in Figure 2.7. Despite being a very useful service, WADO does not provide some standard DICOM services, such as the Query

Service [19].

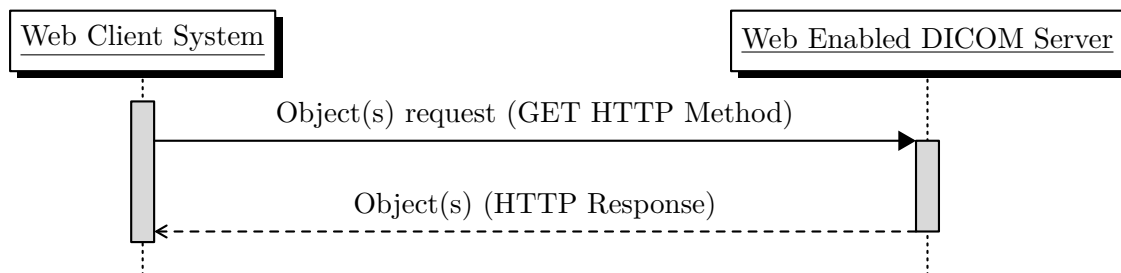


Figure 2.7: DICOM WADO

WADO-RS

WADO-Restful Service (WADO-RS), being a particular implementation of WADO, extends its usage through allowing access to different DICOM objects following a typical REST organization, while still proving all the same advantages. It does so by defining several action types represented in Table 2.3, which can only be accessed through HTTP GET requests.

| Action | Resource | Description |
|------------------|---|--|
| RetrieveStudy | <service>/studies/ <study> | Retrieve the set of DICOM instances associated with the study UID |
| RetrieveSeries | <service>/studies/ <study>/series/ <series> | Retrieve the set of DICOM instances associated with the study and series UID |
| RetrieveInstance | <service>/studies/ <study>/series/ <series>/instances/ <instance> | Retrieve the set of DICOM instances associated with the study, series and SOP Instance UID |
| RetrieveFrames | <service>/studies/ <study>/series/ <series>/instances/ <instance>/frames/ <framelist> | Retrieve the set of DICOM frames from a given study, series, SOP Instance UID, and frame numbers |
| RetrieveBulkData | <BulkDataURL> | Retrieve the bulk data for a given bulk data Uniform Resource Locator (URL) |
| RetrieveMetadata | <service>/studies/ <study>/metadata | Retrieve the DICOM instances presented as the full study metadata with the bulk data removed |

Table 2.3: WADO-RS Actions [19]

service represents the base URL for the service. **study** represents the study instance UID for a single study. **series** represents the series instance UID for a single series. **instance** represents the SOP instance UID for a single SOP instance. **framelist** is a list of one or more non duplicate frame numbers.

QIDO-RS

On the other hand, Query based on ID for DICOM Objects - Restful Service (QIDO-RS) is responsible for exposing Querying capabilities through a similar set of actions, described in Table 2.4

| Action | Resource | Description |
|--------------------|--|--|
| SearchForStudies | <service> /studies/<?query*, fuzzymatching,limit, offset> | Retrieve all studies matching the provided query |
| SearchForSeries | <service> /studies/<study> /series<?query*, fuzzymatching,limit, offset> | Retrieve all series matching the provided query |
| SearchForInstances | <service> /studies/<study> /series<series> /instances<?query*, fuzzymatching,limit, offset> <service> /studies/<study> /instances<?query*, fuzzymatching,limit, offset> <service> /instances<?query*, fuzzymatching,limit, offset> | Retrieve all instances matching the provided query |

Table 2.4: QIDO-RS Actions [19]

query represents the query to perform, using mostly the <attributeID>=<value> and includefield=<attributeID> syntaxes. **fuzzymatching** is a boolean value that indicates whether or not to apply fuzzy matching techniques in the query. **limit** represents the maximum number of results. **offset** represents the number of skipped results.

STOW-RS

Finally, Store Over the Web - Restful Service (STOW-RS) is responsible for exposing Storage services through a single action, described in Table 2.5

| Action | Resource | Description |
|-----------------|---------------------------------|-----------------------------|
| Store Instances | <service> /studies[/<study>] | Store the provided instance |

Table 2.5: STOW-RS Actions [19]

study represents the optional Study Instance UID associated with the provided instances.

CHAPTER 3

TECHNOLOGIES

This chapter provides an overview about the technical concepts that needed to grasp in order to implement a Business Intelligence system, as well as an analysis of the different evaluated tools that allow its implementation, finishing with a quantitative / comparative evaluation of each tool in the given category.

3.1 TECHNICAL OVERVIEW

Business Intelligence (and Analytics) is a set of techniques, technologies, systems, practices, methodologies, and applications that analyse critical data in order to hopefully create a better understanding of that data, transforming it into valuable information that can improve the business practice [20]. A typical Business Intelligence system encompasses capabilities such as reporting, dashboards, *ad hoc* query, search-based BI, Online Analytical Processing (OLAP), interactive visualizations, scorecards, predictive modelling and data mining [20]. Over the next sections, some required background concepts will be developed, enabling a further understanding of some of these enumerated capabilities.

3.1.1 BIG DATA

The amount of biomedical data being digitally collected and stored is huge, and the rate at which it is being generated is increasing every day. While storing all this data is a challenge by itself, analysing it is an even more difficult task. This, of course, happens in a wide variety of areas, not exclusively in the medical imaging field. For these reasons, standard storage and analytics tools can no longer keep up with the sheer volume and complexity of all this data. Therefore, new data processing applications and architectures are required. This lead to the appearance of the term Big Data.

The term Big Data is used to define the challenges that arise when dealing with huge amounts of data. It is usually considered to be a three-dimensional problem, the so-called 3 Vs [21]: Volume, Variety and Velocity. Volume represents the raw amount of data, Variety represents the complexity and disorganization of data, and Velocity represents the rate at which the data is being generated.

During the next sections, some of the Big Data related problems consequent of the digital medical practice will be described.

3.1.2 DATA CLEANSING

Besides being generated at a very high rate, medical imaging data is also highly inconsistent, given that it is captured under a multitude of different conditions, such as different installation configurations, different operators, among others. Due to this, the constraints applied at the medical repositories' level have to be somewhat forgiving in order to have a repository that is capable of capturing all this data. This, however, leads to highly inconsistent, and sometimes even incorrect data. In order to allow proper analysis of this data, it must first be cleansed. But due to the high amount of data available, it is completely infeasible to perform without automating this process.

Data Cleansing, also known as Data Cleaning or Data Scrubbing, is the process that ensures the reliability of the data in a given repository, by allowing the development of workflows that allow automatic detection and correction of inaccurate records from a given data set [22].

DATA CATEGORIES

Considering the process of data analysis as being a pipeline of different stages, Data Cleansing can be considered a preprocessing stage, and it can be applied to three different types of data: Structured, Semi-structured and Unstructured data.

Unstructured data is currently the most used data source in Big Data scenarios, which usually consist of analysing videos or raw text, such as social networks' comments.

Semi-structured data represents the data sources that, despite still being far from ready for analysis, is already organized, for instance in JavaScript Object Notation (JSON) or Extensible Markup Language (XML) formats.

Lastly, there is **Structured** data, which covers all the data that is known to have a particular data type, as well as format and structure, usually this implying that the data is already stored in a database.

DATA QUALITY AND RELATED PROBLEMS

Data Quality is defined as the level of the data's quality. Therefore, having high Data Quality means that that data translates with accuracy the real-world information.

Unfortunately, even though Data Cleansing is an automated process, it still requires a relatively high amount of human effort to define good metrics that correctly translate the incorrect records into accurate data. In order to define those metrics, one must first evaluate the problems presented.

There are two main categories of problems when referring to Data Quality: **Single Source** and **Multi Source** problems, for scenarios when the data is retrieved from only one source or from multiple sources, respectively.

Both of these problems can be further divided into two categories: **Schema Level** and **Instance Level** problems.

Schema Level problems happen mostly when working with Unstructured or Semi-Structured data. In these situations, since there is no notion of schemas or integrity constraints, there is no way of knowing beforehand what to expect from the data.

On the other hand, Instance Level problems occur after the Schema Level, whether it's because the applied schema is very permissive, or simply when there is no way to apply constraints in a given field [23].

DATA ANOMALIES

Data Anomalies are the data's errors that decrease its quality. These can be classified into [24]: **Syntactical**, **Semantic**, and **Coverage** anomalies.

Syntactical Anomalies represent concerns related to the format and values of the entries, and they can be either **Lexical Errors**, **Domain Format Errors**, or **Irregularities**.

Lexical Errors occur when a given field does not follow its specified format. For instance, this happens when a given PatientAge value is mistakenly inserted into the PatientSex field.

Domain Format Errors happen when a particular field's format is not respected. For instance, PatientAge should be in the format of <Patient_Age_In_Years>Y, or <Patient_Age_In_Months>M, but never just <Patient_Age_In_Years> (even though the value is technically correct, it does not follow the specified format).

Irregularities represent anomalies that lead to inconsistent data, such as using both metric and imperial systems, or different currencies. Once again, even though the field's values are correct, it might lead to incorrect information, especially if the applied metric / unit is not specified.

On the other hand, Semantic Anomalies are responsible for violating the real world constraints. They can be further split into **Integrity Constraint Violations** or **Contradictions**.

Integrity Constraint Violations represent the standard integrity violations, such as having a negative PatientAge.

Contradictions are anomalies detected when two (or more) different fields that allow the same value to be inferred generate different deductions. One example would be having an entry whose PatientAge field would be 79Y, but whose DateOfBirth field would be 01-01-2010.

Finally, **Coverage Anomalies** decrease the quantity of entities represented in the data. They can be represented by **Missing Values** or **Missing Tuples** anomalies.

Missing Values are detected when one (or more) fields are missing, while Missing Tuples anomalies happen when entire entries are missing.

QUALITY CRITERIA

In order to assure Data Quality, a series of criteria must be fulfilled [24]. Together, these criteria measure the level of **Accuracy** of the working dataset, which represents the proximity level of the data to its real-world representation.

These criteria can be divided into the following categories:

Completeness represents the level to which all entities present in the real-world are known, and it is measured as the percentage of Missing Tuples' anomalies. Unfortunately, this criteria is impossible to improve in most situations, since the missing entry(ies) can rarely be inferred from the present data, usually forcing the data to be re-captured.

Validity is the evaluation of the percentage of tuples that respect the defined constraints. Anomalies that violate these rules can be Integrity Constraint Violations, Irregularities, and others.

Schema Conformance measures the ratio of tuples in the dataset that are validated by the syntactical structure defined by the schema. Lexical Errors anomalies are the most common violation of this criteria.

Uniformity concerns Irregularities, therefore it measures the ratio of different metrics used for the same field under the same class.

Density refers mainly to the Missing Values anomaly, so it measures the quotient of values from the real-world entity that should be represented in the data. Note that this does not necessarily mean empty fields, since an empty field can sometimes be the expected representation of the intended value.

DATA CLEANSING PROCESS

Data Cleansing Process is defined as being the pipeline of operations that have to be performed in a given data collection in order to make it ready for analysis [24].

Data Auditing is the first step. In order to apply Data Cleansing techniques, it is necessary to analyse the data for defining all the anomalies it contains. This step relies on using some automated statistical methods in order to detect inconsistencies, usually by creating metadata such as cardinality, max and min values, variance, percentiles, and by applying Data Mining techniques, such as Clustering, Sequence Discovery, among many others (developed in Section 3.1.3) [23]. These methods are sometimes referred to as Exploratory Data Analysis techniques. However, this step does still rely heavily on human analysis, whether it is to analyse the generated metadata, or even to analyse a series of individual entries and fields.

After all the anomalies are defined, the **Workflow Specification** step is initiated. This step concerns the definition a series of operations, sometimes called Data Transformations, that have to be performed in order to detect and correct all the previously defined anomalies. Despite having a rather simple definition, it is the most complex steps, given that it represents the core of the whole Data Cleansing process. The defined transformations can belong to one of the following categories, that are set according to the type of anomaly detected:

Parsing is a technique used to detect syntax errors, based on a defined grammar. For instance, a parsing technique implemented using a string similarity algorithm would be capable of translating a tuple in the StudyDescription field from TRAX to TORAX.

Data Transformation maps data in a given format to a schema compliant with the application, for instance by translating a given date in the DD-MM-YYYY format to YYYYMMDD, or by normalizing a certain value to a particular range;

Integrity Constraint Enforcement is responsible for applying a given constraint to the respective field. This enforcement can simply remove the entries that violate the defined constraints, or try to fix these entries. However, this last mode of operation usually relies on user interaction, since it is very hard to automate it.

Finally, **Statistical Methods** allow the usage of the previously defined techniques in Data Auditing, like metadata creation and Data Mining techniques, in order to once again detect data anomalies, before querying the operator about the correct transformation if necessary.

After completing the Workflow Specification step, it is time for its execution in the **Workflow Execution** step. This step might eventually require some human interaction, since it is during its execution that all the previously defined user queries are performed.

At last, the **Post-Processing and Controlling** step is performed. During this step, the results from the Workflow Execution process must be verified, in order to confirm the correctness of the applied Data Transformations. Also, the operator is responsible for performing the transformations that were impossible to execute at the Workflow Execution stage, possibly even feeding new transformations to the Workflow Specifications based on what it was learned at this final stage.

3.1.3 DATA MINING

In the healthcare industry, it is common to have specialists that periodically analyse current trends and changes in health-care data. This analysis later allows them to support future decision making and planning, possibly improving the quality of the provided service [25]. However, manual analysis of this data is slow and expensive, and even impossible to perform in some situations, given the amount of data available. Therefore, the application of Data Mining (DM) to the BioInformatics field is of utmost interest.

Data Mining is a subfield of Artificial Intelligence (AI), and it is usually described as the computational process of extracting potentially interesting patterns in large sets of data that would otherwise be very hard to find [26].

DATA MINING METHODS

The extraction of patterns in Data Mining is done with the use of Machine Learning (ML) algorithms, which are responsible for enabling the withdrawal of relevant information from a dataset. These algorithms can be split into two main categories: Supervised Learning and Unsupervised Learning algorithms.

SUPERVISED LEARNING ALGORITHMS

Supervised Learning algorithms consist in generating a function that attempts to translate a given input (a question) to a correct output (an answer). This is done by providing the algorithm with a set of labelled data: a data set with several example entries with a known “correct answer” (the so-called label), this answer being either a class or a value.

Supervised Learning problems can either be divided into **Regression** or **Classification** problems.

In Classification problems, the classifier tries to learn a function that maps (or classifies) a certain entry to one of several previously defined classes as the label attribute of the input data.

Regression problems are actually quite similar to Classification problems, only instead of having a set of discrete values as the target, the predictor is fed a continuous variable as the label [26].

UNSUPERVISED LEARNING ALGORITHMS

Unlike Supervised Learning, which works with labelled data, **Unsupervised Learning** works with unlabelled data. Therefore, for the algorithm, there is no notion of a “correct” or “incorrect” result. In Unsupervised Learning, algorithms usually work by building representations of the data, based on the similarity between entries, that can be used for decision making. In a way, Unsupervised Learning algorithms allow the discovery of the label data, by presenting data in a human readable fashion.

Unsupervised Learning can be further divided into categories of problems: **Clustering**, **Dimensionality Reduction**, **Anomaly Detection** and **Density** problems [25].

The most common Unsupervised Learning problems are **Clustering** problems, where a given input dataset is divided into several different clusters (or classes), based on how similar the entries are to each other.

Other very popular Unsupervised Learning problems are the **Density** problems, also known as Association problems, which present a series of metadata that allows the definition of strong relations between different variables of the input dataset [27].

Anomaly Detection is a category of problems whose algorithms allow the detection of unexpected values, for instance by checking distances between neighbours (similar to clustering).

Finally, **Summarization** is the process that was previously talked about in the Data Auditing step of the Data Cleansing pipeline. It allows the creation of metadata about the input dataset, such as variance, percentiles, and so on, that allow a general interpretation of the information that is available. Summarization also includes the **Dimensionality Reduction** process, which allows simplified visualization of multi-dimensional data, usually by reducing it to two or three dimensions, thus making it human interpretable.

VALIDATION

Most Supervised Learning algorithms are implemented through creating models. These models, when fed with some input data, are usually not capable of successfully predicting every output correctly. This might be because the dataset is too small, or not generalized enough, among other reasons. Validation is the process of measuring how well a particular model fits a given dataset.

In order to evaluate a model, several metrics can be used, the most common one being accuracy. Accuracy is determined as being

$$\frac{\#TruePositives + \#TrueNegatives}{\#TruePositives + \#FalsePositives + \#FalseNegatives + \#TrueNegatives}$$

, therefore determining the percentage of true results in the whole test. Obviously, a test that outputs no False Positives and no False Negatives will yield an Accuracy of 100%.

A commonly wrong practice is training a given model with an entire dataset, and using that exact same dataset as the test for our model. When this is done, it might lead to the model having 100% accuracy. However, this value is misleading, since it usually means that our model has over-fitting issues: in this case, it means that our model memorized the previous dataset, instead of trying to generalize an answer from it. One of the most used techniques to fix this issue is **Cross-Validation**.

Cross-Validation aims to split the whole dataset into training and testing datasets. The most common Cross-Validation method is the **K-Fold Cross-Validation**. This technique consists in splitting the entire dataset into K smaller datasets (or folds). It then performs an iterative process of selecting one of the folds as being the testing dataset, and the others as the training dataset, and it does this until every fold has been used once as the training set, as described in Figure 3.1. This way, in every iteration, there is no equal data in both the training and the testing datasets. The results of all the iterations can then be combined (usually averaged), providing a model that should no longer over-fit.

Another factor that highly influences the evaluation of a model's accuracy is the complexity of the model itself.

Considering a somewhat complex dataset: by fitting a rather basic model with this data, the model might not be “smart enough”, and it might generalize the input data too much, usually leading to very poor prediction results. In this situation, it is said that the model became biased, which means that it **under-fits** the data.

On the other hand, by fitting an extremely complex model with that same data, that model will learn the noise (the variances in the data that are supposed to be considered as outliers by the

| | | | | | |
|---------------|----------|----------|----------|----------|----------|
| 1st Iteration | test | training | training | training | training |
| 2nd Iteration | training | test | training | training | training |
| | | | | | |
| | | | | | |
| | | | | | |
| 5th Iteration | training | training | training | training | test |

Figure 3.1: 5-fold Cross-Validation

algorithm) instead of trying to generalize an answer, therefore becoming a high-variance model. In this case, the model **over-fits** the data.

The process of finding the sweet spot that provides a fine-tuned model that neither over-fits or under-fits our data is called the **Bias-Variance Tradeoff**.

3.2 KDD AND DATA / BUSINESS ANALYTICS

Very commonly Knowledge Discovery in Databases (KDD) is used as an alternative term to the Data Mining process. However, KDD has a much wider scope than Data Mining, thus making the connotation given to the term KDD usually wrong.

KDD, also know as the Data Analysis process, or Data / Business Analytics, refers to the whole process that has to be executed in order to extract information from data, while Data Mining simply refers to the application of algorithms in order to discover hidden patterns in data [25]. Although both of these definitions might seem rather similar, which leads to the attribution of the same meaning to the KDD and Data Mining terms, hopefully the distinction between them will be made clear by the end of this section.

The KDD process can be divided into eight steps [21], [25]: Data Requirements, Data Collection, Data Processing, Data Cleansing, Data Projection, Modelling, Data Mining and Communication. Although some of these processes were already described in previous sections, they will be further organized into a pipeline.

The first step is the **Data Requirements** step, where it is developed an understanding of the application domain, by specifying the data that is going to be analysed, as well as the main goal of the whole KDD process, usually based on a given client's specifications and requirements.

Next, comes the **Data Collection** step, where the data is gathered from the specified sources. Depending on the amount of sources required, this process can be somewhat time-consuming.

The third step is **Data Processing**, where the data previously gathered during the Data Collection stage is organized into a specific structure, in order to ease its analysis in posterior phases.

The fourth step is **Data Cleansing**, which was extensively described in Section 3.1.2. As it stated, this is the process that ensures the reliability of the data in a given repository, by detecting and correcting inaccurate records from a given data set [22]. This step ends with the storage of the cleansed data in a Data Warehouse, thus making it ready for the analysis process.

These four tasks are also commonly referred to as being an Extract, Transform and Load (ETL) process.

After, the **Data Projection** step is executed. In this step, series of Exploratory Data Analysis techniques are applied in order to further understand the working dataset. These techniques allow mostly visual representations of metadata about the working dataset, like variance and standard deviation plots, or even Dimensionality Reduction representations, therefore allowing interpretations about the data, in order to ease the transition to the next steps.

The fifth step is the **Modelling** step. Given the conclusions taken from the working dataset in the previous step, it is now possible to choose a (set of) proper Data Mining algorithm(s) from the categories described in Section 3.1.3.

After the Modelling step, it is only natural to execute the chosen algorithm(s) in the **Data Mining** step. This step's output is then analysed, in order to evaluate the success or failure of the performed Data Analysis.

The final step is the **Communication** step, where the obtained results are communicated, in order to translate the acquired information into actual actions.

The whole process can be summarized in Figure 3.2.



Figure 3.2: Knowledge Discovery in Databases [25]

3.3 TECHNOLOGIES EVALUATION

This section describes and evaluates a set of available technologies for the implementation of Data Cleansing and Data Mining workflows.

3.3.1 WEKA

WEKA ¹ is an open-source suite of Machine Learning software written in Java. It is widely recognized in the Data Mining and Machine Learning fields, ever since its first public version in 1996.

In order to access all of its functionalities, WEKA provides both a series of Graphical User Interface (GUI) interfaces, as well as a complete and well documented Java library.

Most of the work developed in WEKA's GUI will either be done in the Explorer or the Knowledge Flow views.

The Explorer view (see Figure 3.4) provides a tab-based interface, where each tab is supposed to represent a different phase of a pipeline. In this view it is possible to add what WEKA calls "filters" to the data in the Preprocess stage, thus making WEKA a decent Data Cleansing tool; apply Classification (which includes the Classification and Regression described in the previous section), Clustering and Association algorithms, as well as interact with the algorithms' visualization tools;

¹<http://www.cs.waikato.ac.nz/ml/weka/>

Select relevant Attributes, in order to generate more reliable modes by removing attributes that do not influence the result; and Visualize our dataset, with some basic scatter plot controls.

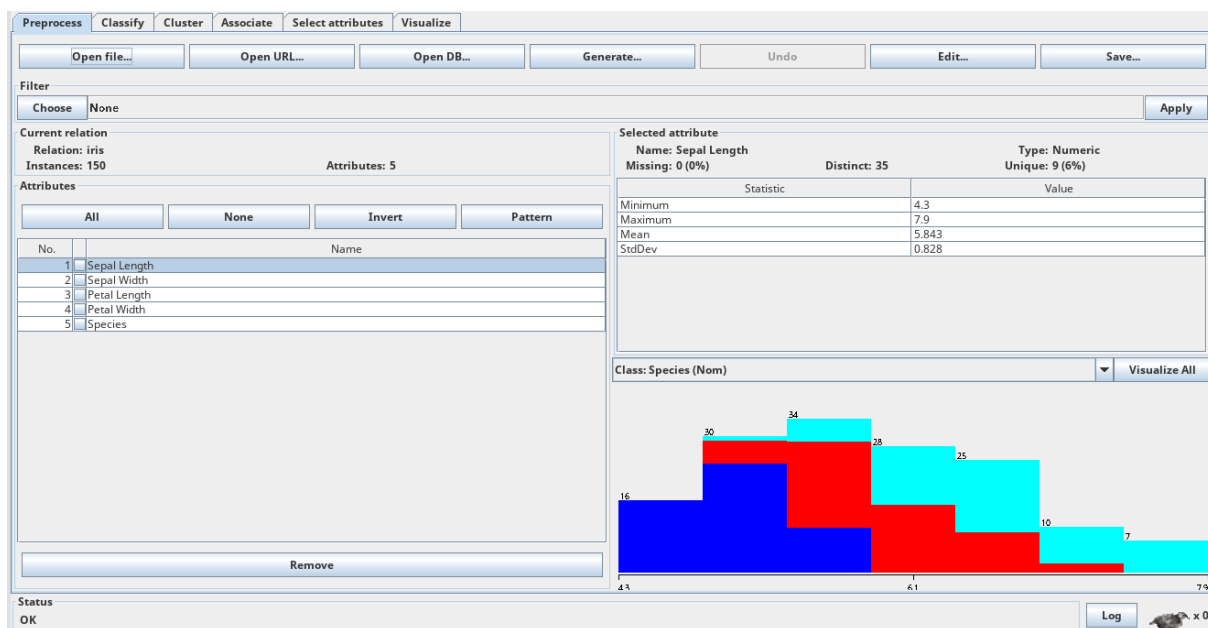


Figure 3.3: Weka Explorer view

WEKA also presents the Knowledge Flow view. In this view, a given pipeline can be represented as a diagram of drag and drop blocks connected between each other. These blocks can represent almost all of the components described before, such as data sources, filters, classifiers, and others. Although it does not implement all of the Explorer's functionality, it is probably the most pleasant view to work with, given how easy it is to update a given component, as well as being extremely scalable, capable of creating complex pipelines.

Besides these, WEKA also has an Experimenter interface that allows testing the performance of different ML algorithms in multiple datasets. The main feature of this interface is to allow the distribution of these computations across different nodes executing experimenter servers.

Despite having this many features, WEKA has extremely limited visualization methods.

Even though it has such powerful GUI tools, WEKA also contains the most extensive and comprehensive Java Machine Learning library. This library is extremely intuitive to use, since it uses a programming style that makes translating workflows developed in the GUI into code an easy task.

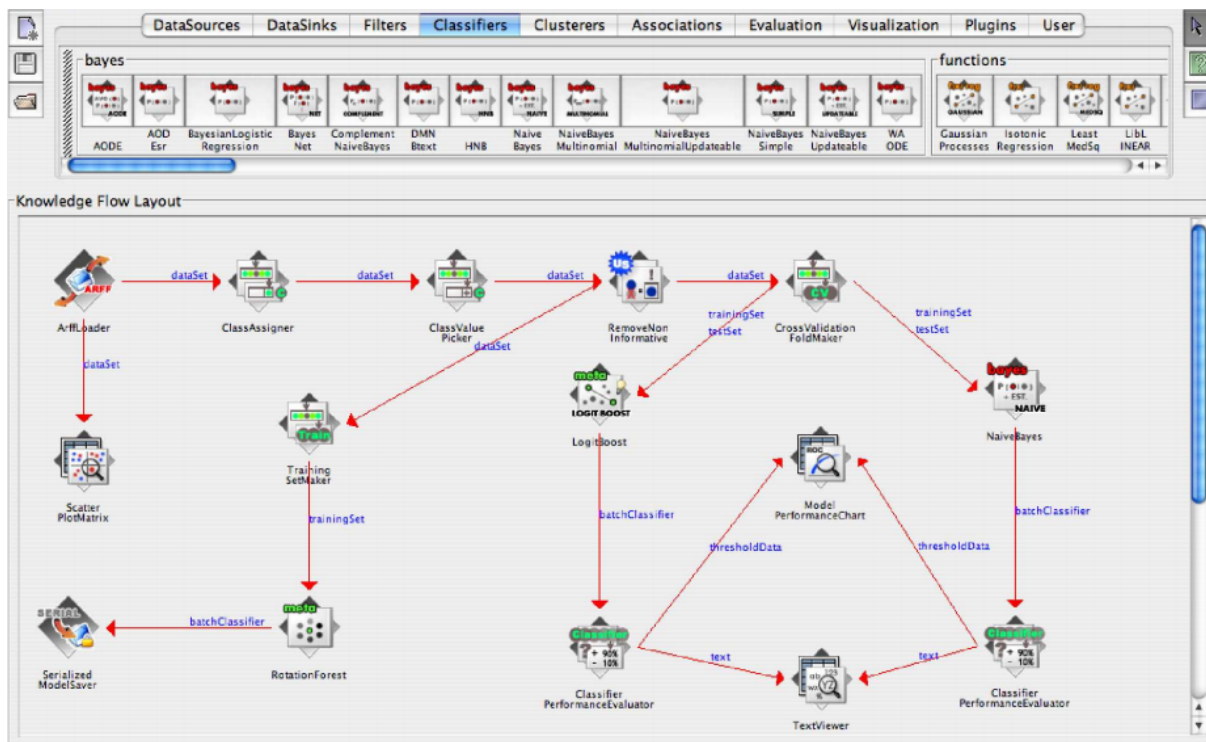


Figure 3.4: Weka Knowledge Flow view [28]

3.3.2 RAPIDMINER

RapidMiner ² is currently the most used integrated environment for Data Mining and Business Analytics. Up until version 5, RapidMiner is open-source, however, parts of it are under closed-source licences in preceding versions, despite most of it still being open-source. It is written in Java.

Given the previous section, RapidMiner can be thought of as being a kind of simpler WEKA, as it only provides its own implementation of the Knowledge Flow interface (represented in Figure 3.5).

RapidMiner’s mode of operation is all about creating processes. These processes are created by connecting a series of operators together, just like in WEKA, and it is even possible to have nested processes, allowing great modularity and workflow reuse. Therefore, all of WEKA’s Knowledge Flow advantages can be applied: the ability to create very scalable, complex and revisable workflows, while keeping amazing visibility. However, the main difference is that RapidMiner presents a much cleaner and user-friendly interface. Besides that, it makes available all of WEKA’s features (it even uses some of WEKA’s libraries for certain algorithms), while adding much better visualization tools (3D graphs, highly customizable plots, and so on) and a complete set of Data Cleansing tools [29].

Also, since RapidMiner is a more recent tool, it allows the development of a series of workflows involving a set of trending technologies. It provides access to a vast amount of dataset formats / databases, provides R and Python extensions out of the box, as well as a series of Cloud services, and it even allows the usage of the designed processes in Big Data scenarios, by translating them into Hadoop workflows for instance.

The main point of failure, when compared to WEKA, is the Java Application Interface (API). Despite following the same programming style as WEKA, and clearly being complete, it does suffer from lack of documentation. Although, the created processes can also be directly imported into Java,

but since RapidMiner uses a rather complex XML structure in order to define them, it makes it rather hard to perform any changes directly in the code.

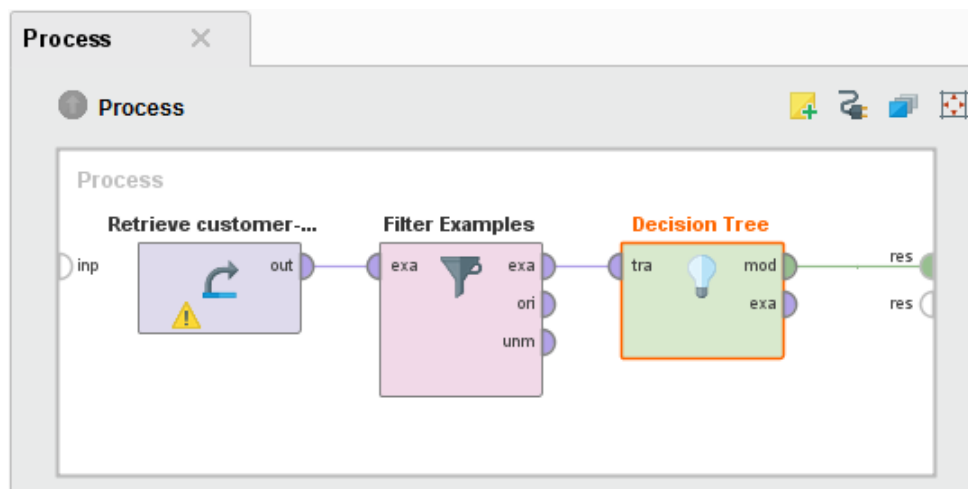


Figure 3.5: RapidMiner process: Creating a Decision Tree Model

3.3.3 ORANGE

Orange ³ is yet another open-source data visualization and data analysis tool. It is written in Python and C/C++.

It can be used with the same drag and drop based interface that was described in both RapidMiner and WEKA, or as a standalone library in Python code.

Given its similarity to RapidMiner's, there is no need to go into much detail about the GUI. It follows the same pipeline / drag and drop based interaction. It is only important to note that it lacks a lot of operators when compared to both RapidMiner and WEKA, as well as rather limited visualization and data analysis tools.

Used as a library, Orange is very intuitive: it uses a programming style that makes the translation from the developed workflows into Python code a rather easy task. However, once again, it is severely limited when compared to other tools.

3.3.4 OPENREFINE

OpenRefine ⁴ (previously known as GoogleRefine) is an open-source tool for turning messy data into usable and analysable data.

Unlike previous tools, OpenRefine uses a client-server architecture, where the OpenRefine instance runs in a server, and its interface can be accessed through a web browser. It is important to note that even though all the computations are being performed in the server, the data actually never leaves the client.

³<http://orange.biolab.si/>

⁴<http://openrefine.org/>

Like the previous tools, OpenRefine allows the usage of most standard file formats.

Its interface is rather similar to that of Excel's, where the users are presented with a table view of the imported data. This view also provides a set of tools needed for cleaning data: removing duplicate or empty cells, elements' search, removal of irrelevant items, among many others. In order to accomplish this, OpenRefine uses the concept of facets. These facets are typically applied to a given column, and they start by presenting an overview of the data present in that column, such as all the values that appear in that column, along with the number of times each one of them is referred, among others. After that, one of these values can be selected, thus removing all other values until this facet is removed. Of course, much more complex facets can be created, such as text-based facets, as well as multi-column facets. In order to accomplish this, OpenRefine makes use of General Refine Expression Language (GREL).

GREL is a simple expression language that provides a multitude of operands, allowing the creation of rather complex facets. Given its simplicity, it is extremely powerful. For instance, a facet like `and(cells["PatientSex"].value == "F", cells["AcquisitionDate"].value > 20100901)` would return all entries with Female patients with an Acquisition Date latter than 01-09-2010, as represented in Figure 3.6. GREL also makes available a series of functions, like string manipulation functions, such as `length()` and `contains()`, array functions, like `sort()` and `join()`, as well as many others. These expressions may also be written in Clojure and Python [30].

Even though it is a very powerful tool, it is not intuitive and user-friendly, even when performing relatively simple tasks. Besides, despite working in a client-server architecture, there is very little documentation about how to use its API, making it extremely hard to work without the GUI.

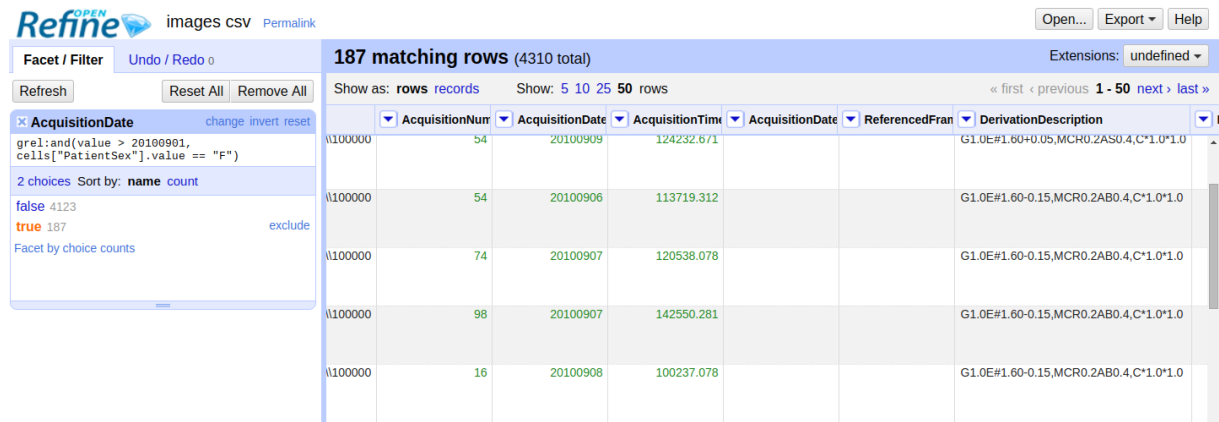


Figure 3.6: OpenRefine

3.3.5 R

R⁵ is a fully fledged open-source multi-paradigm programming language, developed specifically for Statistical Computing and Data Mining. It shows the fastest increase in popularity among all of the available Data Mining tools [31].

⁵<https://www.r-project.org/>

R shares some similarities with MATLAB, given that it is a highly mathematical influenced language, being heavily based on matrix arithmetic, by using data structures such as vectors, matrices, arrays, and others.

Despite this, R is not strictly a matrix arithmetic tool (although it has similar performance when compared to MATLAB or Octave), as it is highly expandable, mainly thanks to Comprehensive R Archive Network (CRAN), a repository of user-created packages. Thus, it is possible to extend R's Data Mining capabilities by importing even more Machine Learning algorithms, by using different plotting libraries, and even adding a Web Framework to the project, among many other options.

Besides, given that R makes available a huge amount of matrix based operands (which, once again, can be extended with CRAN packages), R can be also used as a Data Cleansing tool [32].

3.3.6 PYTHON DATA ANALYSIS ECOSYSTEM

Python is yet another fully fledged, open-source, multi-paradigm language. However, unlike R, it is a general purpose language, although it does not pale in comparison when used as a DM tool, as it will be shown later.

Given its high-level interactive nature, Python was an interesting target for the development of scientific libraries by the community. This lead to the creation of a set of libraries that are widely used for algorithmic development and data analysis.

NumPy ⁶ is a low-level library that adds support for n-dimensional arrays, as well as a set of functions to operate them, thus allowing Python to be used as a matrix arithmetic tool.

For more advanced operations, SciPy ⁷ can be used. It adds support for much more complex operations, such as image, signal and Fast Fourier transform manipulation, interpolation operations, among others.

Unfortunately, the previously enumerated tools are rather low-level when considering the types of data manipulation that are most often performed in typical Data Analysis workflows. In order to fill in this gap, pandas ⁸ was created. The main feature of this library is the addition of the DataFrame object, which acts mainly as a wrapper for NumPy arrays, and acts as a two-dimensional tabular data structure, adding slicing, grouping and reshaping operations, as well as many others.

For fulfilling the Data Mining requirements, the most relevant option in the Python ecosystem is the scikit-learn library ⁹, that provides implementations of almost every relevant Machine Learning algorithm, while always keeping a consistent interface, making adjustments to the code a breeze.

Plotting wise, currently the most adapted tool is matplotlib ¹⁰. Although it provides a huge variety of plotting tools, given that it is so highly customizable, it is not easy to work with out of the box in order to get non-basic plots, since the API is extremely overwhelming.

One of the main criticisms to the Python language is its low performance. However, all of the previously described libraries' implementations are extended with C bindings. These bindings are seemingly accessed from Python, therefore increasing the performance of these libraries to nearly compiled languages' level [33].

⁶<http://www.numpy.org/>

⁷<https://scipy.org/scipylib/>

⁸pandas

⁹<http://scikit-learn.org/stable/>

¹⁰<http://matplotlib.org/>

3.3.7 PROJECT ADEQUACY

Data Mining and Data Cleansing are core elements in this work, and the adopted tools must fulfil a set of requirements.

The first requirement is that the selected tool must have a consistent and well-documented interface. Since the tool's features will have to be exposed through the use of Web Services, it is very important not to be limited by unnecessarily complex structures, instead being able to focus on implementing a greater amount of features.

The second requirement is that it should be a complete tool (or set of tools). This would avoid the issue of having to use and learn different tools that would play different functions, like for instance having a particular Data Cleansing tool and a completely separate Data Mining tool working together.

3.3.8 QUANTITATIVE EVALUATION

In Table 3.1 a quantitative overview of all the evaluated Data Mining and Data Cleansing tools is presented.

Algorithms represents the variety and relevance of Data Mining algorithms available.

Operators represents the amount of Data Cleansing techniques available.

Features represents some extra available tools that might be useful, although not essential.

Visualization represents the amount of visualization tools made available by the tool. In Data Mining, this usually means the amount of graphic types available, and the amount of customization allowed. In Data Cleansing, this means how easy it is to get information about the operations that can be performed. For instance, by selecting a given column, how easy it is to detect similar fields, get basic information like percentiles, among others.

Wrapping represents the simplicity and flexibility of wrapping the presented tool with a layer that allows the exposure of Web Services. In this category, two factors are evaluated: how simple and well-documented the tool's programming interface is, as well as the amount and quality of Web Frameworks available.

Finally, Documentation represents how well documented the tool is, whether it's the tool's documentation itself, or documentation made available by other users.

Note that NA stands for Not Applicable, meaning that a given category does not apply to the respective tool, thus it should not be evaluated.

Taking into consideration the technologies' description and the quantitative evaluation of their features in the previous sections, the tool that best fits the identified requirements was Python and its NumPy library stack. It has a set of well documented and easy to use libraries that makes wrapping them and exposing Web Services with any of Python's Web Frameworks a rather simple task, as well as providing the highest amount of functionality and versatility.

| | | WEKA | RapidMiner | Orange | OpenRefine | R | Python |
|----------------|---------------|------|------------|--------|------------|----|--------|
| Data Mining | Algorithms | 4 | 4 | 2 | NA | 5 | 4 |
| | Interface | 4 | 5 | 4 | | NA | NA |
| | Features | 3 | 5 | 3 | | 4 | 4 |
| | Visualization | 2 | 4 | 3 | | 5 | 4 |
| | Wrapping | 4 | 2 | 4 | | 3 | 5 |
| | Documentation | 4 | 3 | 3 | | 4 | 4 |
| Data Cleansing | Operators | 3 | 4 | 3 | 5 | 5 | 5 |
| | Interface | 2 | 5 | 4 | 3 | NA | NA |
| | Features | 3 | 4 | 2 | 4 | 4 | 4 |
| | Visualization | 3 | 4 | 3 | 4 | 5 | 4 |
| | Wrapping | 4 | 2 | 5 | 3 | 3 | 5 |
| | Documentation | 4 | 5 | 3 | 3 | 4 | 4 |

Table 3.1: Data Cleansing and Data Mining Tools Overview

BUSINESS INTELLIGENCE IN MEDICAL IMAGING

This chapter provides an analysis of the related and previous work on which the proposed system was based, as well as the requirements the intended system must fulfil.

4.1 RELATED WORK

This section presents some previously developed tools related with the field of Business Intelligence in a medical imaging context.

Nagy et al. [34] developed a tool that implements a fully fledged Business Intelligence stack. It starts by aggregating data from all of the institution's systems, including PACS archives, from which it extracted DICOM files, and RIS, from which it extracted Health Level 7 (HL7) data, among others. These results are extracted on a timely basis, and stored in a MySQL database that acts as the central repository for the dashboard.

The dashboard itself features the most relevant chart types, such as histograms, bubble charts, among others. All of these charts feature live updating, meaning that whenever a database's entry is added/updated, the corresponding chart is automatically rendered in order to reflect these changes, as well as detailed reports of the currently selected data.

In turn, the framework developed by Kallman et al. [35] provides a similar set of functionalities. First, it separates the DICOM metadata header from the image's pixel data, storing the first in a separate repository. For this reason, both this and the previous framework share a common advantage: since they both use separate repositories from the existing PACS archives, this enables the developed repositories to be used for statistical purposes with no risk of interfering with the clinical workflow.

The main difference comes in the presentation capabilities. In order to query the data, this framework requires the usage of Structured Query Language (SQL) queries. Although it is a more powerful and flexible procedure, it makes it harder to perform data analysis for an less experienced user.

Wang et al. [3] developed a DICOM database for quality monitoring. Its main feature is its extensive module based system.

The first module is the *DICOM Receiver and Parser*. As the name implies, this module is responsible for communicating with the different PACS, as well as medical devices. For this, it uses the standard DICOM transport protocol. As the previous tools, this framework only uses the DICOM metadata, discarding the pixel data, achieved by the parser. This parser is also responsible for translating the DICOM metadata into the more generic XML format.

The second (and arguably, the most interesting) module is the *Knowledge Base* module. This module is capable of unifying distinct vendor data into the same categories, as well as enforcing the same measures to use the same units, among other features.

The third module is the *Database* itself, that follows a schema similar to the classic DICOM IE, only in this case with an added Dose Related Info entry, therefore implementing a Patient — Exam — Series — Image — Dose Related Info hierarchy. This entry was added because the developed tool revolves around using dosage information, and having that information in a specific database table would greatly improve the application's performance.

In the *Dosimetry Analyser* module it is possible to analyse information related to a given study, patient, and others, allowing the detection of irregular radiation dosage application through the usage of several filters.

There is also the *WebReporter* module, which as the name implies, is responsible for generating reports given a series of parameters. These reports can then be exported in Excel format; only then can charts be generated, using external tools.

The final module is the *Alert Mechanism*. This module enables the creation of notifications for customizable time periods, as well as thresholds. These notifications can be sent through email, cell phone text messages, among others.

4.2 DICOOGLE PROJECT

Dicoogle is a new open-source PACS archive, developed at Universidade de Aveiro/Bioinformatics Group, that proposes an alternative to the classic centralized database architecture, by implementing a Peer to Peer (P2P) solution that uses document-based indexing techniques [36], [37].

One of the main differentiating aspects about Dicoogle when compared to other PACS systems is that, given its document-based indexing capabilities, it allows any document type to be indexed besides the standard DICOM information model fields, all without having to create new database fields, tables and relations. Thus, it allows the extraction of much more information from medical repositories, as well as much greater flexibility. Besides that, given its P2P architecture, Dicoogle can easily be used as a network of repositories.

In order to allow easier integration with developers, Dicoogle provides a Plugin framework. This allows the development of a set of functionalities that can easily be integrated into the Dicoogle ecosystem, all without requiring extensive knowledge of its internals.

For the Plugins' implementation, Dicoogle provides a set of Java interfaces, one for each Plugin category. There is a total of five Plugin categories: Index, Query, Storage, Graphical and Rest plugins.

Storage Plugins are responsible for the persistence of the DICOM objects. Typically, storage is performed locally. However, given the most recent trends, it is also possible to perform this storage at the Cloud level.

The implemented Storage Plugin must return an URL (such as `file:///var/repo/1.2.12345.dcm`)

that uniquely identifies the resource it was asked to store. Obviously, the Storage Plugin is also responsible for retrieving the previously stored resource when presented with its URL.

Indexing Plugins' job is to extract and store information (mainly the DICOM objects' metadata). In order to accomplish this, when asked to index a particular object the plugin, instead of performing the indexing process himself, it returns a task. This task is then executed asynchronously, therefore minimizing the issues associated with a given indexing process taking too long to execute. On the other hand, **Query Plugins** retrieve that indexed information, by translating the stored information back to its original syntax. As the name implies, the retrieved information is selected based on the query that is submitted to the plugin.

Rest Plugins are Dicoogle's implementation of WADO (see Section 2.3.3). Since these Plugins implement the REST paradigm, they allow access to the information through standard HTTP methods, therefore not enforcing an application to use the DICOM protocol.

Finally, **Graphical Plugins** allow the customization of the standard desktop application by allowing the addition of new graphical components, like panels. There is also a recent project called Dicoogle Webcore, that allows the development of a new type of plugins: Web UI plugins. This project aims to extend Dicoogle's web interface. It achieves that by allowing the development of small Javascript modules that can be inserted on a given page that contains a previously defined slot.

From the developed application's standpoint, the most relevant Dicoogle's Plugins are the previously developed Query/Indexing Plugin, as well as the REST Plugin that allows remote access to the medical imaging studies.

Even though they can be developed separately, REST services should be written taking into consideration the underlying Plugins in order to take full advantage of their capabilities.

The current Indexing/Query Plugin was written using the Lucene ¹ database. Lucene is a full-featured text search engine written in Java, that provides extensive full-text indexing and searching capabilities. Given its many query options, the REST Plugin exports Lucene's Query Syntax. For instance, by providing the REST Service with the query `Modality:"CR"`, only DICOM objects of which the Modality field contains "CR" will be returned. However, more complex queries can also be performed, such as Range Queries (`StudyDate:[20050329 TO 20050429]` that will return DICOM objects whose StudyDate field is between 29-03-2005 and 29-04-2005) or Wildcard Queries (`Modality:"C?"` that will return DICOM objects whose Modality field contains the letter "C" followed by one other character, such as "CT" or "CR"). These Query capabilities allow Dicoogle to be used as a Data Mining tool, allowing the extraction of previously indexed imaging studies.

Dicoogle contains several key features regarding new ways of looking into metadata information for retrospective assessments. These may be useful in statistically oriented management and reporting tasks or wide-ranging clinical studies. By allowing the construction of multiple views over the data repository, in a flexible and efficient way, and with the possibility of exporting data for further statistical analysis, Dicoogle allows identification of data and process inconsistencies. This tool can be used to audit PACS information data and contribute to the improvement of radiology department practices. The system has been in use in several hospitals as DICOM Data Mining tool [38] and more than 22 million of DICOM images metadata have already been indexed. However, although this process has been conducted by clinical researchers, it deeply relies on engineers to support the processes, due to the fact that the data discovery and extraction processes can only interact through a command line based interface. Moreover, as it stands, Dicoogle does not provide analytics tools for supporting Business Intelligence. This thesis work aims to solve those issues.

¹<http://lucene.apache.org/core/>

4.3 REQUIREMENTS

As the amount of acquired imaging studies increases, so does the need of analysing them in order to improve the provided healthcare quality. As it stands, the current state of the art in Business Intelligence / Analytics applications applied to the field of medical imaging analysis present several downfalls. The clearest one being the fact that they were developed taking into consideration a specific PACS architecture, rendering them unusable in other environments. For these reasons, the need for the development of a new Business Intelligence platform that would complement the existing Dicoogle PACS arose.

This thesis aims to develop a new web-based Business Intelligence platform that uses the underlying Dicoogle PACS. It will take into consideration the pros and cons of previously developed Business Intelligence applications applied to the field of medical imaging, as well as being based around full-blown generic Business Intelligence applications not developed specifically for the analysis of biomedical data.

Furthermore, this system is to support Real-Time BI capabilities, meaning it should be capable of analysing medical studies as they are made available by the Dicoogle PACS.

Finally, the system should also make available all of its functionalities through the developed Web-Interface.

4.3.1 FUNCTIONAL REQUIREMENTS

A typical information system has to attend to is the authentication requirement. However, since the new BI system is to be added into the existing Dicoogle ecosystem, there is no need for such requirements, given that Dicoogle already handles all authentication related functionalities. Also, there was no requirement for the development of a role-based access control system.

Figure 4.1 represents the developed use case diagram.

These use cases can be described as following:

- **Create Rule** — A rule can be created by filling a form. If the rule is successfully submitted, it is then added to a table that contains all rules created so far. This table provides tools to delete or update a corresponding rule.
- **Update Rule** — Update a previously created rule. A rule can be updated by re-filling its corresponding form, or by changing its position in the table, which will trigger an update of the rule's priority (explained later).
- **Delete Rule** — Delete a rule from the system.
- **Analyse Working Dataset** — Display the working dataset in a tabular format.
- **Create Charts** — Create chart visualizations of the working dataset. These charts should all provide the basic expected tools, such as zooming in and out, class based selection, as well as image export.
- **Manage Dashboard** — The operator should be capable of freely adding and removing visualizations to its customizable dashboard, as well as resizing and reordering those same panels.
- **Create View** — A new view can be created by successfully submitting its respective form. After that, that view is immediately activated.

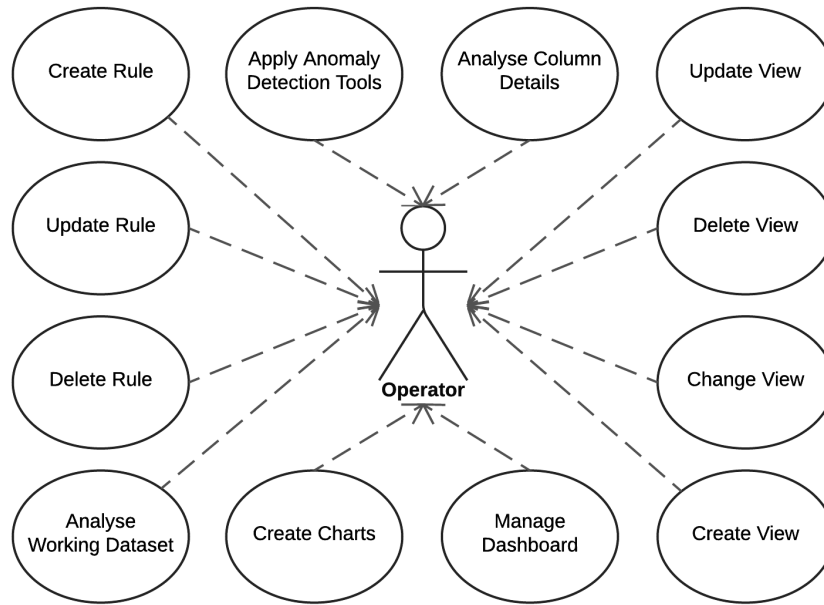


Figure 4.1: Use Case Diagram

- **Change View** — Change to the selected view. This automatically updates the whole client for supporting the selected view.
- **Delete View** — Deletes the selected view, and changes to the closest parent view.
- **Update View** — Change some settings from the selected view.
- **Analyse Column Details** — Obtain information about the selected field.
- **Apply Anomaly Detection Tools** — Apply tools that enable autonomous anomaly detection through rendering charts.
- **Export Comma-Separated Values (CSV)** — Download a CSV coreresponding to the current working dataset.

4.3.2 NON FUNCTIONAL REQUIREMENTS

The most commonly evaluated non-functional requirements are the Usability, Reliability, Performance and Supportability requirements.

The web application should provide a functional, yet appealing interface. However, given that it is a somewhat advanced tool, balance has to be found in order to provide an intuitive interface, while still keeping all of the system's advanced functionality usable. This means that the interface cannot be simplified too much without losing some of its functionality. This also means that it might not be completely intuitive when presented to a user that has absolutely no background related to Data Analysis tools.

Given that the system works with arbitrary types and amounts of data, it is extremely hard to achieve complete stability. However, in case some errors do occur, the operator should be presented

with a useful message that will allow the rectification of the flaw. Also, none of these errors can cause critical system failure; the only errors acceptable are the ones caused by the application's misuse by the user. Besides these points, the application should apply all the typical reliability constraints, such as ensuring atomic access to critical points.

Performance is somewhat hard to manage in a Business Intelligence application such as this one. This is not a typical Information System, in which most operations take little time to execute. In a BI system, most of the operations executed will take time if the operator is not careful with the amount of data it wants to work with. In order to minimize these issues, the operator is presented with a series of tools that enable shrinkage of the working dataset, therefore speeding up the operations. Also, the client application performs asynchronous requests to the server, meaning that the client application will never block while waiting for the server to execute. Finally, the system also performs automatic pagination, meaning that only small subsets of the working dataset are processed each time, allowing the execution of workflows that make use of big amounts of data over time.

Lastly, given that it is a web application, it can be accessed by every standard browser.

ARCHITECTURE

This chapter presents DicoogleBI's architecture, a new Business Intelligence platform that complements the Dicoogle PACS by providing a set of data manipulation capabilities in order to easily maintain and evaluate medical imaging data. This architecture developed fulfils the previously defined requirements, which implied developing an interface that is both fluid and usable, as well as gathering a reliable set of components that allow the creation of a stable and swift system. These components are developed in the following sections, as well as some of the reasoning that lead to their implementation.

5.1 CLIENT-SERVER MODEL

The proposed architecture follows a classic Client-Server model. Specifically, this model can be further segmented into three distinct layers: the Presentation, the Business, and the Persistence Layer.

In this model, the Client acts only as the Presentation Layer of the developed system. Its only role is to display the developed interface to the user, as well as resorting to the Server when required.

In turn, the Server is responsible for implementing both the Business Layer and Persistence Layer of the application.

The Business Layer encompasses most of the application's logic. It is responsible for reliably handling all of the client's requests, and for providing an adequate response.

Finally, as the name implies, the Persistence Layer is responsible for storing and maintaining data across multiple sessions.

5.2 SINGLE PAGE APPLICATION

When developing Web-based applications (or any type of application that relies on a GUI for that matter), the main concern to take into consideration is its usability. Previously developed tools in the field of Business Intelligence applied to medical informatics have proven to be somewhat lacking in this field. However, these systems were developed taking into consideration the target users, which consisted of mostly specialized personnel. In such cases, there is no need to focus heavily on the developed interface.

On the other hand, the proposed system has a wider range of target users. This means that it should be usable by both experienced and inexperienced users alike, all while keeping practical its core functionality.

In this specific system, another very important requirement is the interface’s performance, in particular for its data representation features. When taking into consideration the amount of data that has to be represented, some extra concerns have to be attained to in order to not affect the application’s performance when displaying this data.

In order to further improve the overall performance of the client, we chose to follow the Single Page Application (SPA) pattern.

Classic web applications work by following an URL based flow. This means that, whenever the user accesses a particular URL, the page associated with that URL is loaded. Following this architecture, the server is responsible for keeping all of the application’s state, as well as associating each request with a page, and returning its contents. This implementation’s main downfall is the delay generated between subsequent page loads.

On the other hand, following an SPA pattern, all of the necessary client code – HyperText Markup Language (HTML), JavaScript (JS) and Cascading Style Sheets (CSS) – is loaded in an initial page load, and the page is never reloaded again (unless specifically requested). This means that the client now holds some of the application’s state.

Although following an SPA pattern causes a slight initial overhead on the first page load, when compared to other approaches, it enables the development of a client that combines both a Desktop App-like performance with the accessibility and reliability of a web application that can be accessed on a wide variety of web browsers.

In the scope of the developed system, however, since the server is already responsible for handling resource-heavy computations, this pattern allows offloading the client’s rendering and processing to the client’s machine.

5.3 METHODS

The proposed system was developed on top of Dicoogle Open Source PACS. The application’s pipeline intends to follow the common Business Analytics lifecycle, starting with the data acquisition stage and ending with the analysis of the obtained results. While Dicoogle already handles the acquisition and metadata extraction of medical studies, all other Analytics components have to be implemented.

The following sections describe the developed components represented in Figure 5.1, that enabled the development of the system’s methods.

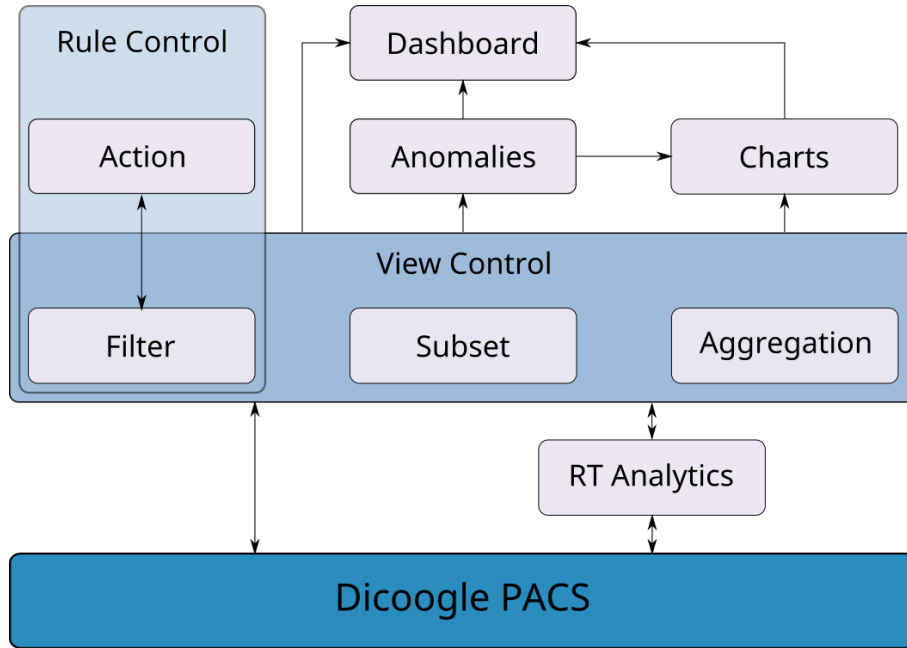


Figure 5.1: DicoogleBI Modules

5.3.1 RULE CONTROL

As said previously, the Data Cleansing stage is one of the most important steps in the Business Intelligence life cycle. Since it handles the detection and correction of inaccurate records, it is crucial in order to generate data on which proper analysis can be performed, allowing reliable conclusions to be taken. The main concern in developing this component was to provide the operator with a tool that allowed manipulation of multiple irregular records simultaneously, since more often than not a detected inaccuracy occurs more than once over the dataset, and manually correcting every one of those records would be highly inconvenient.

Because of this, a Rule-based control system was implemented. This system works in two phases: a Filter phase, and an Action phase. A Filter is responsible for detecting the inaccurate records in the working dataset, while an Action performs the correction itself.

In order to keep this component powerful and yet not too overwhelming with a huge amount of operators that would not be used, a small set of filters and actions was developed that is capable of covering all previously described use cases. When combined, these filters and actions are capable of providing reliable data.

So far, the developed system has five available filters:

- **Empty Field** — This filter allows the detection of empty values on a set of fields. Empty fields are one of the most common anomalies when working with data, and not very often does an empty field actually represent information, hence violating the validity of the dataset. Therefore, detecting these fields is usually the first and simplest step when it comes to Data Cleansing (DC). Unfortunately, most of these fields cannot be inferred, usually leading to their removal from the dataset in later stages.
- **Expected Value** — This filter allows the detection of one (or more) values on a given group of fields (or columns, as referred to in the Data Model). It works by defining a set of expected_values, and a set of columns on which to detect those values.

- **Regular Expression** — A more generic, yet slightly more advanced version of the Expected Value filter, since it requires Regular Expression’s knowledge; otherwise, it works exactly the same way as the Expected Value filter.
- **FilterDate** — Allow filtering the working dataset based on a date interval applied to a set of fields. If both the `start_date` and the `end_date` are specified, then that interval is used. If, however, only the `start_date` is specified, then all dates larger than `start_date` will be matched.
- **Expression** — It should only be used as fallback, since it uses pandas’ `query()` operator, making this operator very dangerous to use. However, it enables the execution of statements or even small scripts, allowing detection of more complex cases, that would otherwise be very hard, over even impossible, to perform with the previous filters.

All filters (with the exception of the Expression filter) feature a **reverse** flag. This flag indicates that the records matched by the filter are to be the ones excluded from the dataset. For instance, if an Empty Field filter is applied to the `PatientAge` column, only records that contain a non-empty `PatientAge` field are considered.

Furthermore, the Regex and Expected Value filters also feature a **match_columns** flag. If active, this flag tries to match every column to every value. It does so by padding both the fields and the replacement value sets to the smallest length of both sets; each of the set’s values is then matched to each other based on their index. For instance, if an Expected Value filter is specified with the [`PatientAge`, `StudyDescription`] columns, the [`12Y`, `CHEST`] values, and the **match_columns** flag active, then the `12Y` value will be matched against the `PatientAge` field, and the `CHEST` value will be matched against the `StudyDescription` field. Otherwise, if the **match_columns** flag is not active, then each value is matched against each column.

When executed, these filters take a given input dataset and generate a subset whose rows match the applied parameters.

In order to create a Rule, an associated Action as to be created as well. These can be one of the following:

- **Fill Empty Field** — Replaces the empty values in the defined fields with a chosen value.
- **Replace by Value** — Replace a defined set of values with another. However, it is also possible to match a column with its corresponding value to be replaced and value to replace by activating the Match Columns flag.
- **Replace by Regex** — Same action as the previous, with one particular characteristic: if the value to be replaced is a Regular Expression that contains a capture group, that same capture group can be used as the value to be replaced. For instance, setting the value to be replaced to `(\d+)Y` and the replacement value to `\g<1>` in the `PatientAge` field would basically remove the `Y` from every value in the `PatientAge` field.
- **Date To Age** — Convert selected columns containing date values to the corresponding age.
- **Normalize Age** — DICOM’s Age fields are usually stored in a `<Patient_Age_In_Years>Y`, `<Patient_Age_In_Months>M`, `<Patient_Age_In_Weeks>W`, `<Patient_Age_In_Days>D`. This action normalizes all these values under the same unit: Years, Months, Weeks or Days.
- **Field To Date** — Convert selected columns composed of values that represent dates to actual date types. For instance, the DICOM standard’s date specification defines that a date field should be represented by an integer value in the `YYMMDD` (Y-Year, M-Month, D-Day)

format. Applying this action using a format such as `%Y%m%d` would allow the defined column to be interpreted as a Date field, enabling further applications for this field.

- **Expression** — Makes use of pandas' `eval()` operator in order to allow the creation of more complex scenarios that can not be covered by previous actions.

All actions (with the exception on the Expression action) also feature a Target Columns attribute, therefore allowing the creation of new columns containing the applied transformations, instead of employing them directly to existing columns.

In order to keep the system as modular as possible, the Filters and the Actions handlers are implemented separately. The Filters module applies a particular Filter and returns the filtered dataset, still containing all of its original fields, while the Actions module takes this same dataset and applies the correct transformation to its fields.

This logic is enforced in all of the implemented Filters and Actions, meaning that every created Filter will only work as a selector, only returning a subset of the original dataset, and never actually performing any transformations, while every Action will never contract or expand the number of rows or fields in its receiving dataset, instead only performing the appropriate transformations.

However, from the development of this system, one slight problem arose. The devised rules might not be mutually exclusive, that is, a rule's triggered action might change the values that match another rule. For this reason, every rule has a priority attribute. This is an integer value that represents each rule's priority: the smaller the value, the larger the priority of the respective rule. Therefore, a high priority rule will be executed before a lower priority one.

5.3.2 ANOMALY DETECTION

The first (and simplest) component that allows the operator to perform anomaly detection is the Description component. In order to accomplish this, this component associates a given field to one of two categories: ordinal or nominal data, based on the field's datatype. As the names imply, if the selected field's data type is numerical (either integer or float), then it is provided an ordinal description; otherwise, it is provided a nominal description.

An ordinal description returns a count attribute, with the total number of non-empty entries in the field, as well as some of the most common statistics for a numerical set of values, such as mean, standard deviation, and minimum and maximum values. On the other hand, a nominal description returns the number of entries in which a given value appears. Also, both descriptions feature an empty values' count, if applied.

Unfortunately, this automatic detection of a field's description based on its data type is sometimes flawed. More often than not, a particular field is composed of an integer set of values; however, those values do not represent a continuous variable, meaning that they are actually supposed to be interpreted as categorical. Therefore, all numerical fields expose the option of manually enforcing the description type.

5.3.3 VIEW CONTROL

In order to ease the process of working with different datasets, the concept of Views was implemented. It is very rare that one would want to work with the PACS repository in its entirety, particularly when the size of the repository might severely impact the performance of even the most basic operations. Besides, very often is data analysis performed in a particular subset of the repository, such as analysing patient data related to a particular Modality, or analysing reports on a given date range. Because of this, views are intended to represent smaller, more specific repositories, inherited from the original PACS repository.

Initially, the system provides a **default** view. This view represents the entire dataset, from which all other views are inherited. On this view, all previously referred data manipulation tools can still be applied. However, the view itself cannot be changed.

In order to create other views, three transformations that allow the definition of different types of subsets are provided:

- **Aggregation** — This transformation is meant to mimic the SQL **groupby** operation. First, a set of fields on which to perform the aggregation itself has to be provided. Then, the aggregation functions to perform are defined.

It is possible to provide only one function as a parameter; in this case, that same function will be applied to all the fields that are not part of the aggregation fields.

However, the previously described workflow is rarely the desired output. For this reason, it is also possible to provide a column to function mapping. This works by providing a column, and a series of functions associated with that same column it. This way, only the defined columns and functions will be determined. The transformation is particularly useful when taking into consideration the previously defined Information Entity Hierarchy (Patient, Study, Series, Image).

Whether or not to perform this column-to-function mapping is decided by the value of the **All** field. If this flag is active, then only the **Functions** field will be taken into consideration. Otherwise, then the **Functions** field is ignored, only taking into consideration the **Columns** fields.

On either case, the allowed functions are: **maximum**, **minimum**, **standard deviation**, **mean**, **size**, and **sum**.

- **Subset** — Enables the user to define a static subset: in this case, an interval of either rows and/or fields on which to work with. Each on these also has an **exclude** flag associated that, when active, indicates that the inserted fields are not to be included.
- **Filter** — In this transformation, it is possible to use the previously developed Filters from the Rule-based system as transformations. The exact same filters are available in this module, therefore working in the same way.

It is important to note that each one of these transformations can only be used once in a given view: meaning that a view can have only one aggregation, subset and/or filter transformation.

Given these transformations, it is now possible to make use of the previously enumerated Dicoogle Data Mining capabilities in various situations.

The first one occurs when an Expected Value or a Filter Date filter are specified, in which case this filter can be exactly mapped to the exported service's query capabilities. For instance, a filter whose expected values are **CR** and **F**, and whose fields are **Modality** and **PatientSex** would map to the **Modality:"CR" AND PatientSex:"F"** query, which would return all entries with the CR modality

and female patients from Dicoogle's export service. On the other hand, a Filter Date filter with `start_date=01-01-2010` and `end_date=01-01-2015` applied to the `PatientBirthDate` field can be translated to the `PatientBirthDate=[01-01-2010 to 01-01-2015]` query.

The second instance happens when a series of fields are defined in the subset transformation, which once again, can be directly mapped to a query. If for instance, a subset with the fields `Modality` and `PatientAge` was defined, then this is equivalent to the query `fields=[Modality,PatientAge]`. The same happens when a row interval is defined in this transformation, given that every row is identified by its `uri` field, which can also be used in the provided query.

After being created, a view can then be activated. This means that changing to a particular view will automatically update the whole client, given that only one view can be active at any given time. This behaviour also implies that, whenever a given rule or analysis is created/applied, it only affects / corresponds to the currently active view.

In order to further extend the concept of views, the developed system allows the creation of nested views. It is, therefore, possible for a created view not to be inherited directly from the `default` view, instead being inherited from other created views. For supporting this, two alternatives surfaced.

The first one would be to determine a given view's dataset by applying its transformations to its parent view's dataset directly using (for the most part) pandas' operators. This would translate in two downsides. First, only the views inherited directly from the parent view could make use of Dicoogle's Data Mining capabilities, since these would be the only views that do not rely on the parent view, because the `default` view cannot have any transformations. Second, this would force a given view to always have access to the updated parent view's dataset, which could cause unnecessary processing times if the parent view was never directly activated. However, this would be the simplest process to implement.

The second option was to develop a system that allowed merging a view's transformations with its parent's. This would allow maximizing Dicoogle's capabilities, which would allow to offload some of the system's processing, as well as not relying on the parent's dataset, since it the only requirement is its transformations. However, this is the most complex option, since merging the transformations has a series of caveats that need to be attended.

The second alternative's advantages justified the extra effort required to implement this system, so that was the employed option.

For the development of this component, we make use of Dicoogle's export service. This service allows exporting imaging studies through simple HTTP methods. However, we quickly arrived at the conclusion that exporting the whole PACS repository to the platform through this service was simply too time and resource-consuming. For this reason, we decided to implement a Pagination system that will also be described throughout the following sections.

5.3.4 DASHBOARD

The developed solution provides extensive dashboard capabilities. The dashboard enables the development of a fully customizable page by allowing the inclusion of any of the visualization components present throughout the Client. Each one of the selected visualizations is then associated with a fully resizable and sortable panel, allowing the page's organization to be tuned to the operator's desire.

In order to implement the dashboard's persistence, two aspects have to be taken into consideration.

The first aspect is the persistence of the dashboard's layout. Every time a panel is changed (resized, reordered or deleted), the dashboard's layout is saved, similar to the presented layout in Listing 1

```
[
  {i: 'a', x: 0, y: 0, w: 1, h: 2},
  {i: 'b', x: 1, y: 0, w: 3, h: 2, minW: 2, maxW: 4},
  {i: 'c', x: 4, y: 0, w: 1, h: 2}
]
```

Listing 1: Example Dashboard Layout

The `i` attribute identifies the respective layout element. `x` and `y` represent the panel's coordinates, while `w` and `h` represent its width and height, respectively. Finally, a panel might also have a minimum and maximum width and height, represented by the `minW`, `minH`, `maxW` and `maxH` attributes.

The second aspect is related to the panel's data. In order to actually render information, every panel relies on data sent by the server in response to a given request. Besides, as it was previously said, every operation performed in the client relies on the currently active view. For these reasons, every panel has an associated request and view, which are saved and executed every time the dashboard is reloaded.

5.3.5 CHARTS

The system also makes available standard chart visualization tools. These enable a simpler analysis of the working dataset, which is particularly useful when combined with the Dashboard's capabilities.

These visualizations take three parameters. The first two parameters are the `X` and `Y` values. `X` represents the dataset's fields to be used, while `Y` represents their corresponding values. It is possible not to refer an `Y` value however. In that case, the size of the `X` fields will be used as the `Y` value. The last parameter is what type of chart to be rendered. Currently, the supported chart types are Scatter, Pie, Bar and Time-Series Charts.

5.3.6 REAL-TIME ANALYTICS

The system was developed to support continuous operation, which means that data will be analysed and made available for visualization as soon as it arrives at the PACS archive, instead of relying on repository snapshots acquired on a timely basis. Thanks to Dicoogle's integration, whenever new images are received, the system is notified of their arrival. The system then acquires these images and parses them, meaning studies are automatically added to the corresponding views, as well as being matched against all the defined Rules.

CHAPTER 6

IMPLEMENTATION

This chapter discusses all aspects related to the implementation of the DicoogleBI system.

6.1 MODEL OVERVIEW

As said previously, the system follows a Client-Server architecture. Figure 6.1 represents some of the technologies used in order to implement the system's architecture.

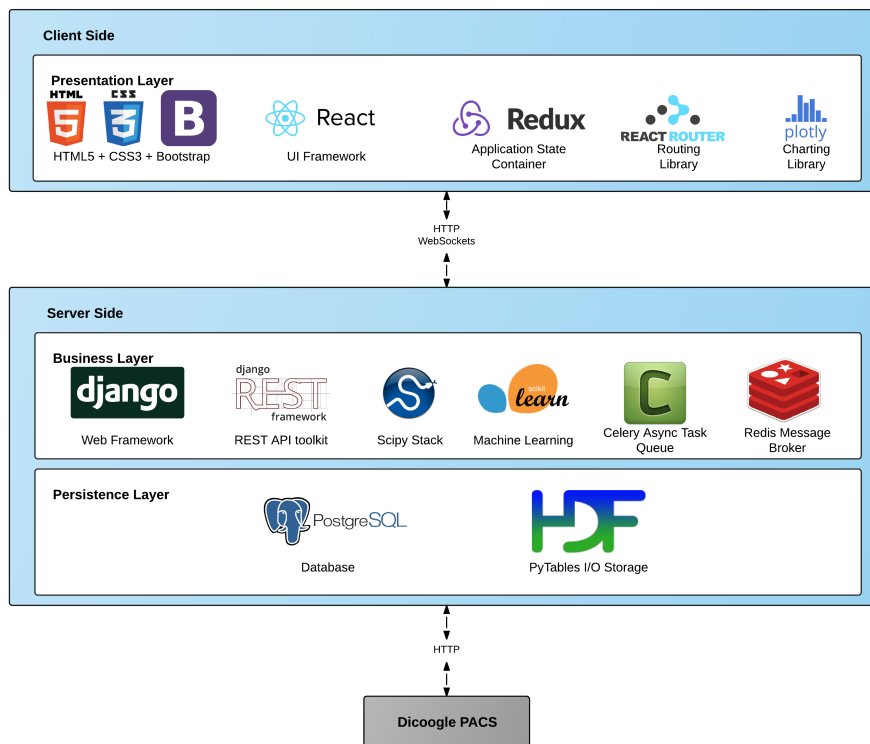


Figure 6.1: DicoogleBI technologies

The developed components will be further explained in the following sections.

6.2 CLIENT SIDE SPECIFICATION - PRESENTATION LAYER

As it is expected, following the previously specified SPA pattern leads to moving some of the logic typically implemented in the server to the client. However, having a client that has to manage the User Interface (UI), the server requests and application state can quickly lead to an unmaintainable codebase as the application starts to scale. Besides, having to translate this state into interface changes can also cause some severe performance bottleneck issues. Therefore, choosing the right framework (as well as the tools that surrounding it) is of utmost importance. The following sections will describe the technologies used for DicoogleBI's Client-side development.

6.2.1 REACT

The application was developed using the React JavaScript framework¹ for rendering UIs. Nowadays the most used UI pattern is the Model-View-Controller (MVC) pattern, which splits an application into three components [39]:

- **Model** represents the actual application state, meaning it encapsulates the behaviour and data of the application domain.
- **View** translates the application's state to the actual interface itself.
- **Controller** is responsible for handling the user input and translating into changes in the application's state.

In an MVC pattern, React can then be thought of as enabling the development of the application's View layer, leaving the implementation of the other components up to the user.

React's core building blocks are called components. Ideally, these components should be modular and composable, and represent a small UI element. The reason for this is that components can be chained together, enabling the development of highly scalable and manageable applications (codebase wise) by creating component hierarchies.

Each React component has two main sources of data: **props** and **state**. **props** are objects passed from a parent React component to its child components, while **state** is only managed by the component itself. This reflects one of React's characteristics: its unidirectional data flow, meaning data can be passed from parent to child (through the use of **props**), but not the other way around. It is important to note that whenever a component receives new **props** or changes its **state**, it triggers the re-rendering process, in which the latest component's representation is flushed to the Document Object Model (DOM).

In order to actually render data, a React component has to return a React element. These elements are the same HTML elements present in a normal DOM, such as `divs`, or `lis`. However, since React is JS based, it provides a series of methods that allow the generation of these HTML elements in JS, the most relevant being JSX. What JSX does is to allow the creation of React elements using (mostly)

¹<https://facebook.github.io/react/>

standard HTML syntax. Unfortunately, using HTML syntax in a JavaScript file is not valid JavaScript syntax. For this reason, every JSX file has to first be transpiled (more on this later), as exemplified in Listing 2.

```
// Input JS (JSX):
var app = <div color="blue">
    <h1>Hello</h1>
</div>;
// Output after transpiled (JS):
var app = React.createElement(
  div,
  {color:"blue"},
  React.createElement(h1, null, "Hello")
);
```

Listing 2: Example JSX transpiled code

It is still possible to use the `React.createElement` directive directly instead of relying on JSX; however, it is much simpler and reliable to work with JSX.

Another key point of React is its Virtual DOM. Virtual DOM, as the name implies, is a virtual representation of the DOM. One can think of the virtual DOM as being an in-memory object that contains the representation of the actually rendered DOM element (a given component’s virtual DOM is, in fact, its returned React element). Therefore, whenever a given change is performed in the component that triggers a re-render, React is capable of performing a **diff** between the new element and the previous, only changing the new items, not re-rendering the ones that remain unchanged, thus making React a highly performant framework.

6.2.2 REDUX

As said previously, most React components have an internal state. However, as an application starts to scale, having a series of components managing their own internal state becomes very hard to manage. In order to solve this issue, Redux ² is used. Redux acts as a state container for JS applications. When applied to React, Redux works by managing a **store**, a standard object that holds all of the components’ **state**. Therefore, when using Redux, components (should) no longer have a **state** that only they manage internally.

A typical Redux flow is implemented using its three basic components:

- **Actions** — Actions are payloads of information that send data from the React components to the store. They are represented through plain JavaScript objects, each one containing one attribute that represents the action **type**, as well as the information itself. However, in order to reduce the amount of code generated, these objects are usually created through the use of Action Creators. These Action Creators are JS functions that simply return an object that represents the desired action. Actions are then “committed” to the Store through Redux’s `dispatch()` function.

²<http://redux.js.org/index.html>

- **Reducers** — Reducers are functions responsible for translating a particular action into a state change. They accomplish this by acting upon the previous state and the received action. Based on the action type, the reducer (which can be thought of as being a `switch-case` statement) is capable of identifying what it should change in the previous state tree, therefore generating the next application's state.
- **Store** — The Store component is a standard JS object that holds all of the Application's state. However, holding the whole state in a single object can soon become a confusing task as the application starts to scale. For this reason, Redux's Reducers each correspond to one of the Store's keys. Each of these reducers can be further combined under the same key by using Redux's `combineReducers()`, allowing the Store to be organized in virtually any configuration.

In order to receive their `state` from the Store, React components now `connect()` to the Store, receiving their state through `props`. Although it is possible to subscribe to the entire Store, the most common option is to subscribe to smaller parts of it. This is accomplished by corresponding a given Store entry, such as a Reducer, to one of its `props`' keys.

One of the greatest advantages of Redux is that the Store component can also be used as a central repository of information. This means that whenever a particular component updates the Store, all components associated with the corresponding state will receive the new state. This allows having incredible flexibility when developing component hierarchies, allowing having components associated with same states in completely different sections of the application.

6.2.3 BOOTSTRAP

In order to develop the GUI, Bootstrap was used. It is currently the most popular HTML, CSS, and JavaScript framework ³. Its main purpose is to facilitate the process of developing responsive web sites, while also providing a wide range of useful and aesthetically pleasant HTML and CSS components, all of them automatically supported in all of the most widely available browsers.

6.2.4 PLOTLY.JS

Finally, Plotly.js ⁴ was used as the charting library. It provides a wide variety of customizable charts, as well as interesting chart visualization tools such as viewport control, export functionality, categorical selectors, among others, all out-of-the-box. One of its greatest features is the fact that it is built on top of d3.js ⁵. d3 is a low-level data visualization library, and given its integration with the plotly.js library, it is possible to make use of d3's lower level primitives in order to fine-tune some components.

³<http://getbootstrap.com/>

⁴<https://plot.ly/javascript/>

⁵<https://d3js.org/>

6.2.5 REACT-ROUTER

React-Router ⁶ was used in order to ease the process of implementing the feel of a typical Web Application that follows a URL implementation. In order to simulate this implementation, React-Router associates a given URL (the **path**) with a React Component (the **component**), as exemplified in Listing 3

```
<Provider store={store}>
  <Router history={history}>
    <Route path="/" component={RootApp}>
      <IndexRoute component={AnomalyDetectionRoot}/>
      <Route path="/dashboard" component={DashboardRoot}/>
      <Route path="/rulecontrol" component={RuleControlRoot}/>
      <Route path="/statistics" component={StatisticsRoot}/>
      <Route path="/viewmanager" component={ViewManagerRoot}/>
      <Route path="/visualization" component={VisualizationRoot}/>
    </Route>
  </Router>
</Provider>
```

Listing 3: React-Router Configuration

Provider is not part of the React-Router spec itself. It is a react-router-redux ⁷ component that allows the currently active route to be exported to the Redux Store.

Router is the main Router component. It handles all the URLs, as well as the browser history, in order to allow the user to go back and forth in the application.

Route, as the name implies, is the component that associates a URL to the corresponding React element. The root Route "/" defines the component that is to be loaded in all of the underlying routes. In this case, the **RootApp** component renders the top **Navbar**.

Next, the **IndexRoute** defines the parent Route's "Home Page". In this case, **AnomalyDetectionRoot** is the component rendered when the user only specifies the Application's base URL.

All following Routes define the remaining Application's entry points.

6.3 SERVER SIDE SPECIFICATION - BUSINESS LAYER

The server's architecture was developed taking into consideration the best option in order to provide the client application with its needed services.

Currently, the most widely implemented client-server architectural style follow the REST pattern. In particular, for SPA client applications, RESTful Web Services are the most viable option, when considering their acceptance and reliability. For this reason, the Server's implementation exposes a REST API.

⁶<https://github.com/reactjs/react-router>

⁷<https://github.com/reactjs/react-router-redux>

REST is a software architectural style whose main purpose is to provide separation of concerns by enabling the development of a Client-Server model [40]. Given that it is an architectural style, it does not enforce a particular communication protocol. However, in the developed application, REST is used in combination with the HTTP protocol, which will be the only scenario considered from now on.

REST relies on some simple concepts. First, it starts by defining a Resource, which is anything that the client can access through the REST services. It can be a file, a web page, information, as well as a set of other resources. These resources are represented as an Internet media type (usually JSON, but sometimes XML, and others), and are identified by an Uniform Resource Identifier (URI). However, in order to interact with them, an action has to be specified together with the URI. This action is represented in the form of a verb; since REST uses the standard HTTP methods, the methods themselves represent the action to be performed on the Resource, as it is represented in Table 6.1. For instance, by performing a PUT in the resource `http://api.com/resources/resource1`, a new `resource1` will be created (or a replacement of the existing one will occur); or by performing a GET in the resource `http://api.com/resources/`, the response will include all the URIs the resource contains.

| Method | Description |
|--------|---|
| GET | Read a resource. |
| PUT | Insert a new resource or update if the resource already exists. |
| POST | Insert a new resource. |
| DELETE | Delete a resource. |

Table 6.1: REST HTTP methods [41]

One other requirement is the implementation of a notification system. Since using polling / streaming techniques from the client is not the best practice when taking into account the most recent alternatives [42], [43], we opted for implementing WebSockets communication.

WebSockets [44] is a protocol that allows full-duplex communication over a standard Transmission Control Protocol (TCP) connection, after an initial HTTP handshake, as represented in Figure 6.2. Despite being mostly used in order to provide low-latency interactive communications between a browser and a server, in the scope of the developed system, its main purpose is to allow the server to push notifications to the client, given that HTTP only allows client to server communication establishment.

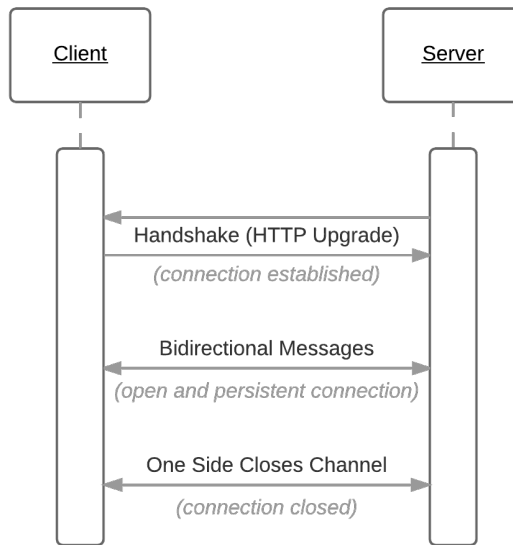


Figure 6.2: WebSockets Basic Communication

6.3.1 DJANGO

When taking into consideration that the previously chosen data mining libraries are all part of the Python ecosystem, it makes sense to expose the developed methods using a Python Web Framework, in this case being the Django Web Framework ⁸. Besides being the most reliable and mature Python Web Framework, it can also be extended with a powerful Web API toolkit: the Django REST Framework (DRF) ⁹.

In Django, one of the most important components is the Object-relational mapping (ORM). This component is responsible for mapping the relations from a Relational Database into a series of Objects, enabling querying and manipulating the database's data using an object-oriented paradigm. This provides several advantages over working with SQL directly: the most important one being that a lot of the standard database management tasks are automated, such as creating, changing or deleting tables; besides, since it is an abstraction layer over the database system, ORM allows changing the underlying Relational Database Management System (RDBMS) without having to change any of the already created code.

In order to specify the database relationships using ORM, Django uses Models. A model is the single, definitive source of information about the data. It specifies the essential fields and behaviours of the data stored in the database [45]. These models are specified using standard Python classes, each class representing a database table, whose fields are defined by the class' attributes. For instance, the models specified in Listing 4 would translate to the SQL table represented in Listing 5.

DRF also provides Serializers. Their purpose is to allow complex data, such as model instances or network requests, to be converted to/from native Python datatypes (usually dictionaries) in a process called serialization/deserialization, respectively. These datatypes can be later rendered to/from the previously enumerated Internet media types, in this case being the JSON format. Serializers are often used when bound to a given model definition. In this scenario, they dictate how to map an incoming

⁸<https://www.djangoproject.com/>

⁹<http://www.django-rest-framework.org/>

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Listing 4: Example Django model [45]

```
CREATE TABLE person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

Listing 5: Example SQL table [45]

request to a model definition, given that their format often differs. However, these serializers can also be used by themselves, in which case they can be used to validate incoming requests, besides providing the standard serialization operations.

Next, for handling incoming requests, Django defines Views. In short, views are simple Python classes (or functions) that take a given Web request and return a corresponding Web response. Each view has an associated Route, which acts as the previously defined REST URIs, and defines different handlers according to the incoming HTTP request type (GET, POST, and others).

6.3.2 SCIPY STACK AND SCIKIT-LEARN

As disclosed previously in Section 3.3.6, between all of the data manipulation libraries / frameworks, the SciPy stack ¹⁰ and scikit-learn libraries proved to be the best option.

SciPy stack's most basic component is the NumPy library. NumPy is designed to efficiently manipulate multi-dimensional arrays of arbitrary records, allowing not only the implementation of mathematical processes, but also general purpose applications that rely on data types other than numbers. It is a somewhat low-level tool for proper data manipulation. However, given its arbitrary data type capabilities, it is possible to use it to implement higher level libraries. And such is the case of pandas, which is a library that provides high-performance, easy-to-use data structures and data analysis tools. It grants a series of higher level data manipulation functionalities that (for the most part) use the underlying NumPy library. The most important one is the DataFrame object. This object represents a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labelled axes. It supports operations such as indexing (both column and row-wise), filtering missing data, Input/Output (IO) operations, merging, group by, among many others. Given this, very rarely is necessary to resort to NumPy directly.

Furthermore, all of these libraries can be further extended using the scikit-learn library, which exposes a series of ML algorithms through a simple and intuitive interface.

¹⁰<https://www.scipy.org/stackspec.html>

6.3.3 CELERY

Celery ¹¹ is an asynchronous task queue based on distributed message passing. These queues represent a mechanism to distribute work (tasks) across threads or machines (workers). For accomplishing this, Celery relies on communicating through messages, usually relying on a broker to mediate interactions between clients and workers. The most common brokers used with Celery are message brokers such as Redis ¹² or RabbitMQ ¹³, given their asynchronous capabilities. However, some Database engines are also partially supported.

As it stands, the developed system makes use of Celery's capabilities by launching tasks that are triggered by a client request, but that take too long to process and do not influence the response, as it will be further explained.

6.3.4 REDIS

Redis ¹⁴ is an in-memory data structure store, used as database, cache and message broker. Its main purpose is to serve as a reliable and fast key-value database, which is a simple yet very powerful model, particularly when considering that Redis supports data types such as strings, lists, sets, and many others.

This component's main feature is to establish communication between the different Celery tasks, mostly through the use of its implementation of the Publish/Subscribe messaging paradigm. In this messaging pattern, a Publisher sends messages to a series of channels, and Subscribers of those channels receive the sent messages [46], as described in Figure 6.3. Neither one of these entities is aware of the other, meaning that a Publisher is not aware of Subscribers associated with a channel to which it sent messages, and that a Subscriber is not aware of how many Publishers sent messages to its subscribed channels.

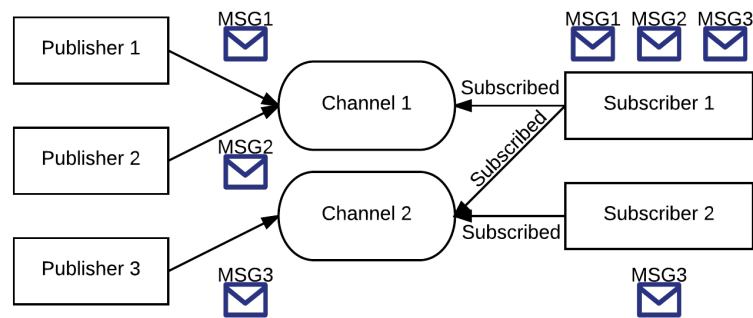


Figure 6.3: Publish Subscribe Messaging Pattern

Besides that, Redis is also used as a standard key-value in-memory database whenever there is a need to save data that should not be persisted throughout sessions.

¹¹<http://www.celeryproject.org/>

¹²<http://redis.io/>

¹³<https://www.rabbitmq.com/>

¹⁴<http://redis.io/>

Finally, it is also used as a distributed synchronization mechanism in order to mediate access to pandas' Hierarchical Data Format Store (HDFS) from the different Celery tasks, since this component is not thread-safe for writing.

6.4 SERVER SIDE SPECIFICATION - PERSISTENCE LAYER

6.4.1 DATA MODEL

The application's Data Model is represented in Figure 6.4.

As it is represented, a **View** is the centre point of the Data Model. Each View has a unique name, as well as a dirty and loaded state. Besides that, even though it is not represented, each View also has associated a dataset, which does not actually belong to the Model, given that it is stored in the corresponding pandas' HDFS entry. A View's dirty state indicates that, the next time this view is loaded, its `uri_list` has to be re-calculated, instead of loading it directly from the HDFS, as explained in Section 5.3.3. The loaded state indicates whether all of a view's pages have been downloaded or not, as described in the previous section.

Next, a View might have some **Rules** associated. Each Rule has a name, as well as a priority, and a **Filter**, represented in Figure 6.5. Each Filter can be one of: **FilterDate**, **Regex**, **Expression**, **ExpectedValue** or **EmptyField**. Each one of these fields was further explained in Section 5.3.1.

These Rules also have an associated **Action** that, as seen in Figure 6.6, can be either a **FillEmptyValues**, **DateToAge**, **ReplaceByRegex**, **ReplaceByValue**, **NormalizeAge**, **FieldToDate**, or **Expression**. Their respective fields reflect the descriptions made in Section 5.3.1.

Next, a View can have multiple associated **Layouts**. These are used in the system's Dashboard. Each one of these Layouts has a unique name that identifies it. The panel, value and layout fields are only used by the client. The panel is used by the client to know which type of component it should render in the panel. The value is the request performed by the client in order to get the correct information to pass on to the panel. Finally, the layout contains the panel's settings, such as coordinates and size.

Finally, a user-created View can also have three transformations associated (one of each type).

The **SubsetTransformation** contains the columns and rows to include (or exclude, if their respective exclude flags are active), as well as possibly an `uri_list` that reliably identifies the rows.

The **FilterTransformation** only refers to a Filter.

Lastly, the **AggregationTransformation** contains the mapping, that indicates the aggregations to perform on each column. This mapping is only considered if the `all` flag is activated. Also, the `groupby` value is the set of columns on which to perform aggregation, as developed in Section 5.3.3. The PostgreSQL relational database was chosen for data persistence. Besides being currently the most used, as well as one of the most reliable Relational DataBase Management System (DBMS), its main feature in the context of the developed system is its usage as a JSON database, which is particularly useful in order to store data in the database that does not need to comply to a given schema. Furthermore, it also provides native support for Array fields, that are used extensively throughout the system.

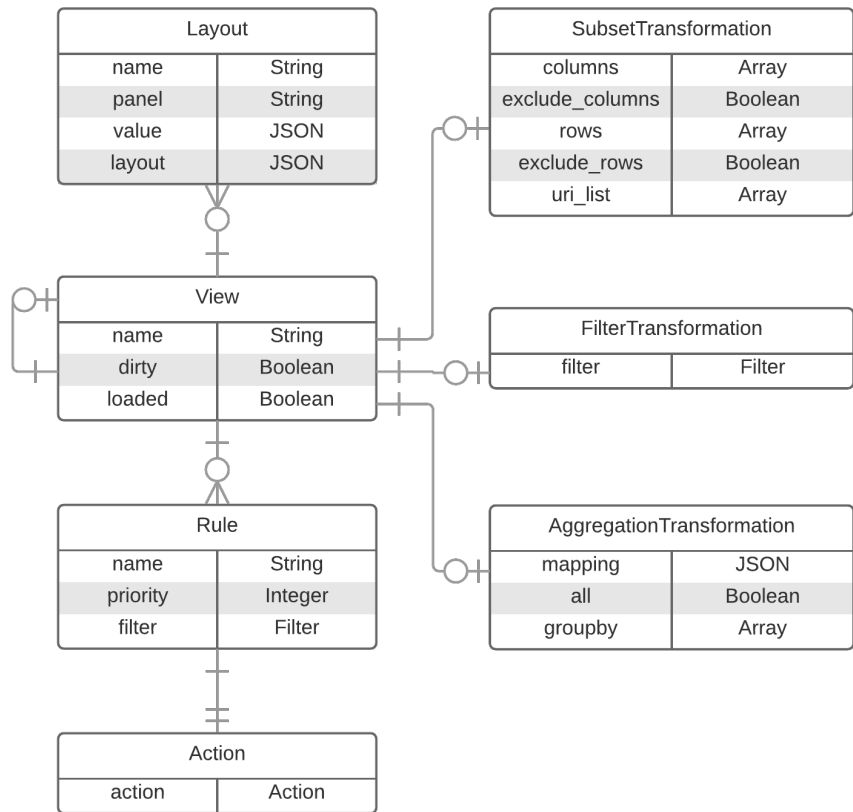


Figure 6.4: Data Model

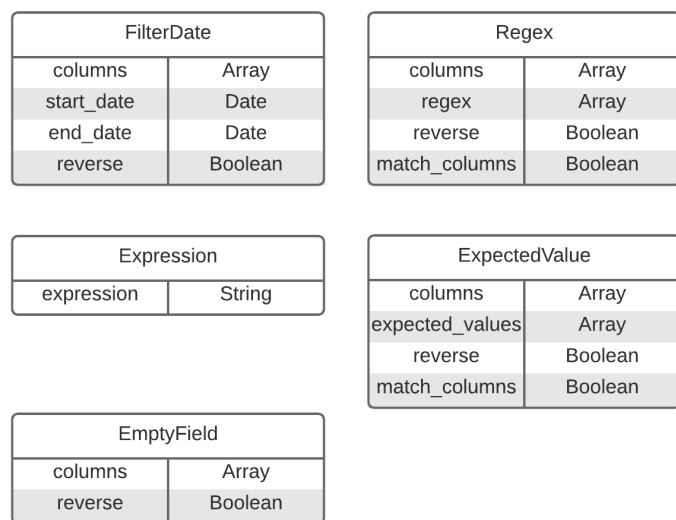


Figure 6.5: Filter Models

| FillEmptyValues | |
|-----------------|-------|
| source_columns | Array |
| target_columns | Array |
| values | Array |

| DateToAge | |
|----------------|-------|
| source_columns | Array |
| target_columns | Array |

| ReplaceByRegex | |
|----------------|---------|
| source_columns | Array |
| target_columns | Array |
| regex | Array |
| values | Array |
| match_columns | Boolean |

| ReplaceByValue | |
|-----------------|---------|
| source_columns | Array |
| target_columns | Array |
| expected_values | Array |
| values | Array |
| match_columns | Boolean |

| NormalizeAge | |
|----------------|--------|
| source_columns | Array |
| target_columns | Array |
| normalize_to | String |

| FieldToDate | |
|----------------|--------|
| source_columns | Array |
| target_columns | Array |
| date_format | String |

| Expression | |
|------------|--------|
| expression | String |

Figure 6.6: Action Models

6.4.2 DATABASE

Besides PostgreSQL, pandas' HDFS is also used to provide persistence. Its purpose is to act as a high-performance IO tool responsible for storing large Python objects, given that using PostgreSQL to achieve that would be too penalizing for its performance. This tool can be considered a simple key-value storage system, in which each key corresponds to a Python object. In this case, the objects stored are all pandas' DataFrames, each one representing a view (as it will be explained later on). HDFS can also be used as a sort of DBMS, given that uses the underlying PyTables library ¹⁵. In this case, HDFS supports querying capabilities that allow only the required fields to be loaded from storage to memory, which can greatly improve the application's performance.

6.5 PAGINATION

Given that views' datasets are not derived directly from the parent's dataset, the first caveat that had to be taken care of was the situation when a view's change affects its child views. Unfortunately, implementing an algorithm that detected when this happened would be too much of a task, without yielding almost no benefits. For this reason, whenever a given view's dataset changes, it sets all of its (direct and indirect) children's **dirty** and **loaded** flag to **true**. Whenever a given view's **dirty** flag is active, it signals that the next time it is activated, its query has to be recalculated and performed to

¹⁵<http://www.pytables.org/>

Dicoogle's export service. On the other hand, the `loaded` flag indicates whether or not the dataset is finished loading.

When merging views, the first step is to determine the query to perform to Dicoogle's export service, since this sets the view's base dataset. As said previously, the transformations that can be translated to the query are the Expected Value and Filter Date filters from the Filter transformation, as well as the Subset transformation. Therefore, these are the first ones to be merged. Merging the Subset transformation is a simple task: all it has to be done is to check which of the child's fields and/or row `uris` are contained in the parent views. For the Expected Values filter, it is enough to concatenate every view's queries based on their aggregated fields, separating them by an "AND". For the Filter Date filter, these can also be simply translated to Dicoogle's query syntax, such as `[20020101 to 20030101]`, as long as these dates are comprised of the parent views.

This initial query only returns the `uri` field, not the entire dataset, since this field is required to implement the Pagination system. This system works by only downloading a certain number of records (50,000 by default) from the PACS repository by using these `uris` as unique record identifiers. For this reason, these `uris` are sliced into sets of 50,000 in order to download the pages on which the following processes are applied. This is the only way to achieve the pagination behaviour, since the current Lucene indexing plugin used by Dicoogle does not support pagination. Furthermore, the `uri` field is the only field that is absolutely guaranteed to be present in every record.

Following the page's acquisition, all of the filter transformations that are not the Expected Value and the Filter Date filters are applied. In this phase, no proper merging is performed: instead, all of the parent's filter transformations are performed, chaining their filtered results, up until the current view.

Finally, it is applied the aggregation transformation.

After all of the transformations are performed, the view's rules from the rule-based system are applied to the current page.

This process is triggered whenever there is a new request performed by the client that is related to a `dirty` view. However, the whole point of the pagination process is to not keep the client waiting for too long for a view's processing. Therefore, whenever the first page reaches its final transformation, the view's `dirty` state is deactivated, and the requested method is applied as developed in the next section.

After the initial page load, all following pages are acquired and transformed using an asynchronous Celery task launched after sending the client's response. This task is a loop that performs the same operations as the previous workflow, meaning it acquires and transforms all remaining pages. This time, however, whenever a page reaches its final transformation, its result is appended to the view's HDFS entry. Besides, it also sends a message to the view's Publish/Subscribe channel in order to notify other running Celery tasks about the availability of more data, as described in the next section.

When the loop finishes, meaning there are no more available pages, another message is sent to the view's Publish/Subscribe channel, indicating that Celery tasks relying on this view can terminate gracefully.

Every view also features a `loaded` flag, which is set to `false` when the fetching task is initiated, and set to `true` when it finishes. This flag is to allow a view's downloading process to be resumed in the future in case its Celery task is shut down. This usually happens when the server is manually turned off before finishing the download.

The entire pipeline (with the exception of the previous paragraph) is represented in the left-most branch of the diagram represented in Figure 6.7.

METHOD APPLICATION

When a requested is performed on a view with its **dirty** flag activated, it first triggers the process described previously, which disables the view's **dirty** state, meaning a method is only applied when a view is not **dirty**. This implies that the dataset used in the method's execution is loaded from the corresponding view's HDFS entry, this process being paginated as well.

As previously, the client's HTTP request is immediately answered with the information related to the first page, and all remaining pages are handled with a Celery task. However, more often than not, this transformation does not have all available pages for its execution. For this reason, whenever there are no more available pages, the running task subscribes to the view's Publish/Subscribe channel, waiting for a notification. This notification either indicates the task that more pages are available, thus it should execute one more iteration, or that the pagination process is over, in which case the task should terminate.

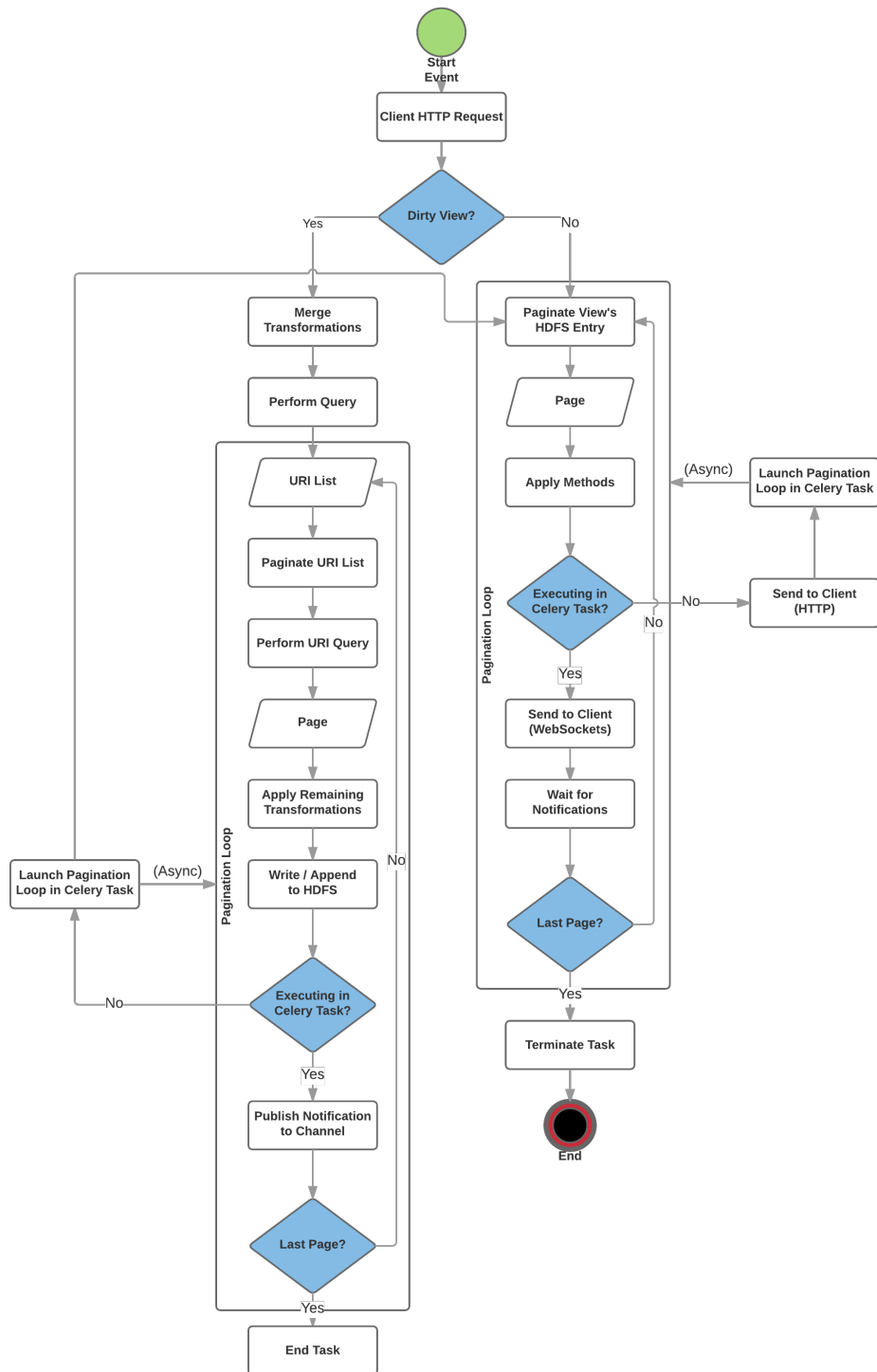


Figure 6.7: Pagination Process

RESULTS

This chapter will discuss all aspects related to the results of the developed system. These include both details related to the systems deployment, as well as the validation of some previously defined use cases.

7.1 INTERFACE COMPONENTS

The interface plays a big role in the developed system. It has to be reliable and stable enough, as well as being intuitive even to inexperienced users, in order to provide a pleasant experience.

Given that the developed system is a Business Intelligence application, some interface components had to be developed in order to cover some specific use cases, that are usually not required in typical web-based applications.

7.1.1 FORMS

Forms are the most used component throughout the entire system. They are responsible for providing the establishment of most of the developed tools' parameters. For this reason, it is of paramount importance to devise forms that provide reliable feedback to the user, as well as a simple, yet powerful set of tools. In Figure 7.1 we can see the example of one of many of the interface's forms.

When necessary, some of the form's fields required multiple inputs; those are handled as seen in Figure 7.2.

Furthermore, when possible, the form's fields will also provide suggestions, as exemplified in 7.3.

Expected Value

Filter the dataset based on a set of values.

Columns: Fields on which to search for the values.
 Expected Values: Values to search for.
 Reverse: Filter out the matching entries.

Name

rule_1

Columns

x

Modality

x

Expected Values

x

CR

x

CT

x

☐ Reverse

Figure 7.1: Example Form

Expected Values

x

CT

CR

Add "CR"?

Expected Values

x

CT

x

CR

NEXT

Figure 7.2: Multiple Inputs

Columns

x

Modality

x

InstitutionName

PatientID

PatientName

PatientSex

SOPInstanceUID

SeriesDescription

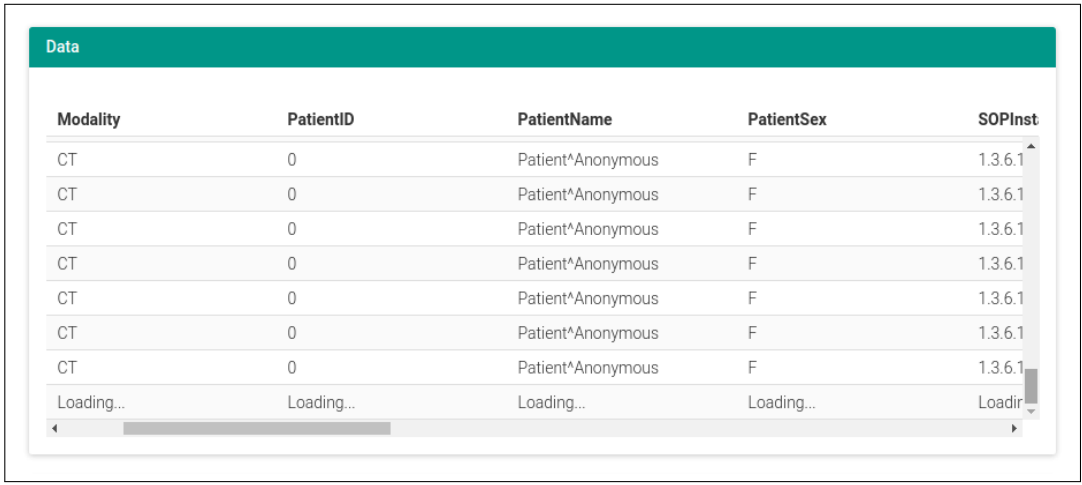
Figure 7.3: Field Suggestions

7.1.2 TABULAR DATA

Usually, displaying tabular data is considered to be a somewhat trivial task. However, when taking into consideration the amount of data the system has to deal with, displaying it in a tabular fashion can severely impact the client's performance.

There are two performance bottlenecks when considering this scenario.

The first one is related to actually loading the data itself. Obviously, loading possibly tens of thousands of rows at once would severely impact the performance of the application, possibly even crashing the application. For this reason, this component implements a pagination system that, by default, starts by loading 1000 rows when it is first rendered. Then, as the user scrolls through the data, whenever it is close to reaching the last available element, 1000 more rows are loaded asynchronously, as represented in Figure 7.4.



| Modality | PatientID | PatientName | PatientSex | SOPInst |
|------------|------------|-------------------|------------|---------|
| CT | 0 | Patient*Anonymous | F | 1.3.6.1 |
| CT | 0 | Patient*Anonymous | F | 1.3.6.1 |
| CT | 0 | Patient*Anonymous | F | 1.3.6.1 |
| CT | 0 | Patient*Anonymous | F | 1.3.6.1 |
| CT | 0 | Patient*Anonymous | F | 1.3.6.1 |
| CT | 0 | Patient*Anonymous | F | 1.3.6.1 |
| CT | 0 | Patient*Anonymous | F | 1.3.6.1 |
| Loading... | Loading... | Loading... | Loading... | Loadir |

Figure 7.4: Table Loading Data

The other performance bottleneck is related to the actual rendering of the data itself. Having a DOM with thousands of elements can usually lead to stuttering in the application's interface.

Given that this component is managed with React, it is capable of rendering only the observable rows (plus some defined threshold) based on the scroll position.

Overall, both of these techniques enable efficient rendering of high amounts of data.

Furthermore, the dataset can also be sorted according to a given field, as displayed in Figure 7.5.

7.2.1 RULE CONTROL

As said previously in Section 5.3.1, a Rule is composed of a Filter and an Action.

In order to choose a Filter, the user is first presented with a menu containing all available filters, as seen in Figure 7.7.

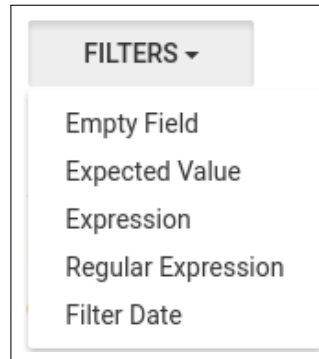


Figure 7.7: Filters Menu

Then, the corresponding filter's form is displayed, such as in Figure 7.8.

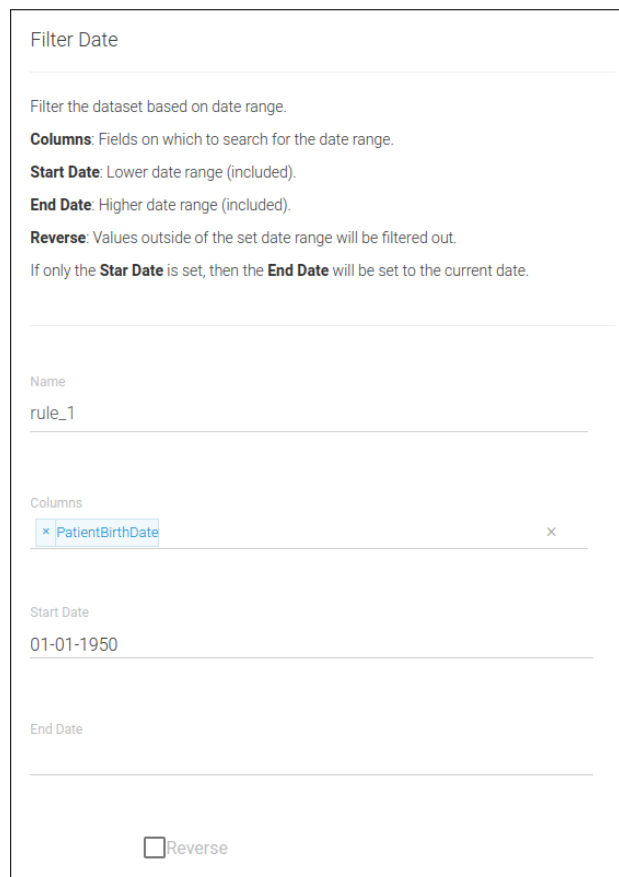


Figure 7.8: Filter All Entries with PatientBirthDate > 01-01-1950

Same as the Filters, Actions are selected through a similar menu (Figure 7.9), and their corresponding form can then be filled (Figure 7.10).

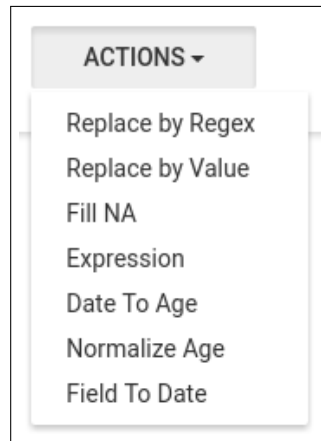


Figure 7.9: Actions Menu

Replace by Value

Replace matched values by a fixed value.

Columns: Fields on which to search for **Expected Values**.

Expected Values: List of values to match.

Value: Values to replace.

Target Columns: Columns to write results to.

Match Columns: Map **Columns**, **Expected Values** and **Value** sets, instead of permutating them.

Value

Columns

Expected Values

Target Columns

☐ Match Columns

Figure 7.10: Replace All CT Occurrences in the Modality Field with Tomography

Each Rule is then added to the Rule List, where it can be deleted and modified. A given Rule's priority can be tuned by dragging the corresponding rule in the list of active rules in the client's interface or by using the two buttons beside it, as seen in Figure 7.11.

| Rules | | | | | |
|------------|--------------------|------------------|------------|------------|----------|
| Name | Filter | Action | | | |
| regex_1 | Regular Expression | Replace by Value | ↑ PRIORITY | ↓ PRIORITY | ✕ REMOVE |
| expected_1 | Expected Value | Replace by Value | ↑ PRIORITY | ↓ PRIORITY | ✕ REMOVE |

(a) regex_1 with higher priority than expected_1

| Rules | | | | | |
|------------|--------------------|------------------|------------|------------|----------|
| Name | Filter | Action | | | |
| expected_1 | Expected Value | Replace by Value | ↑ PRIORITY | ↓ PRIORITY | ✕ REMOVE |
| regex_1 | Regular Expression | Replace by Value | ↑ PRIORITY | ↓ PRIORITY | ✕ REMOVE |

(b) regex_1 with lower priority than expected_1

Figure 7.11: Setting a Rule's Priority

7.2.2 ANOMALY DETECTION

This component relies on displaying useful metadata related to the currently selected field, as explained in Section 5.3.2. If the selected field's data type is numerical (either integer or float), then it is provided an ordinal description (Figure 7.12); otherwise, it is provided a nominal description (Figure 7.13).

| Column Details | | COLUMN ▾ |
|--|----------|----------|
| PatientBirthDate | | |
| <input type="checkbox"/> Force Nominal | | |
| count | 59309 | |
| mean | 19531224 | |
| std | 1712357 | |
| min | 19220101 | |
| max | 19980726 | |
| null | 0 | |

Figure 7.12: Example Ordinal Description

However, a numerical field's interpretation can be manually enforced, resulting in an output similar to Figure 7.14.

| Column Details | | COLUMN ▾ |
|----------------|-----------|----------|
| PatientAge | | |
| unique ▾ | frequency | |
| 94Y | 127 | |
| 90Y | 645 | |
| 89Y | 135 | |
| 88Y | 120 | |
| 87Y | 364 | |
| 85Y | 111 | |
| 84Y | 214 | |

Figure 7.13: Example Nominal Description

| Column Details | | COLUMN ▾ |
|---|-------------|----------|
| PatientBirthDate | | |
| <input checked="" type="checkbox"/> Force Nominal | | |
| unique | frequency ▾ | |
| 19730607 | 601 | |
| 19880426 | 598 | |
| 19880904 | 480 | |
| 19891225 | 451 | |
| 19550923 | 385 | |

Figure 7.14: Example Numerical Description as Ordinal

7.2.3 VIEW CONTROL

Views allow the user to work with smaller and more controlled subsets belonging to the underlying PACS repository, as described in Section 5.3.3. For creating views, the user is presented with three Transformations: the Aggregation, the Subset, and the Filter Transformations. The Aggregation represented in Figure 7.15 represents an aggregation performed on the Modality, PatientSex and StudyDescription fields, with the `max` and `min` functions applied to the StudyDate field. On the other hand, the View created from the Subset transformation displayed in Figure 7.16 will only contain the parent view's rows 0 to 200 and 400, and it will not contain the SeriesDescription field. Finally, the Filter transformation allows the usage of the previously represented rules in Section 7.2.1.

Create View

View

test_view

×

AGGREGATION

SUBSET

FILTER

Aggregation

Aggregate the dataset

Groupby: Fields under which to perform aggregation.

All: Whether to apply **Functions** to the remaining fields or not.

Functions: Functions to apply to the remaining fields.

Specific fields can be added, and functions applied to them specifically by using Add Column.

Groupby

×

Modality

×

PatientSex

×

StudyDescription

×

☐ All

Functions

Column 0

PatientBirthDate

×

×

max

×

min

×

ADD COLUMN

SUBMIT

Figure 7.15: Aggregation Performed on Specific Columns

After being activated, a view then updates the whole client. For instance, if a View such as the one represented in Figure 7.15 is activated, the client's forms would be changed to something as Figure 7.17, among all other components.

A view can then be selected on the menu shown in Figure

Create View

View

test_view

×

AGGREGATION

SUBSET

FILTER

Subset

Slice the dataset

Rows: List of rows to filter. A range can be defined, such as 100-200.
 Columns: Set of columns.

Rows

×

0-200

×

400

×

☐ Exclude

Columns

×

StudyDescription

×

☒ Exclude

SUBMIT

Figure 7.16: Example Subset

Columns

| |
|------------------------|
| Modality |
| PatientSex |
| StudyDescription |
| PatientBirthDate(amax) |
| PatientBirthDate(amin) |

Figure 7.17: Updated Description

Select View

View

test_view

×

SUBMIT

DELETE

Figure 7.18: Select View Menu

7.2.4 DASHBOARD

As said previously in Section 5.3.4, the developed solution provides extensive dashboard capabilities, allowing the inclusion of any of the system’s visualization components in an organizable environment, allowing the development of workflows such as the one represented in Figure 7.19.



Figure 7.19: Example Dashboard

7.2.5 CHARTS

The system also allows the creation of multiple charting types, such as Scatter, Pie, Bar and Time-Series Charts, as developed in Section 5.3.5.

For instance, selecting the parameters presented in Figure 7.20 would result in a chart such as the one presented in Figure 7.21.

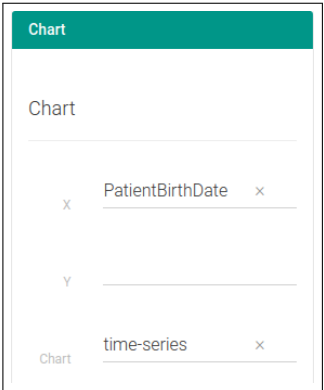


Figure 7.20: Example Chart Input

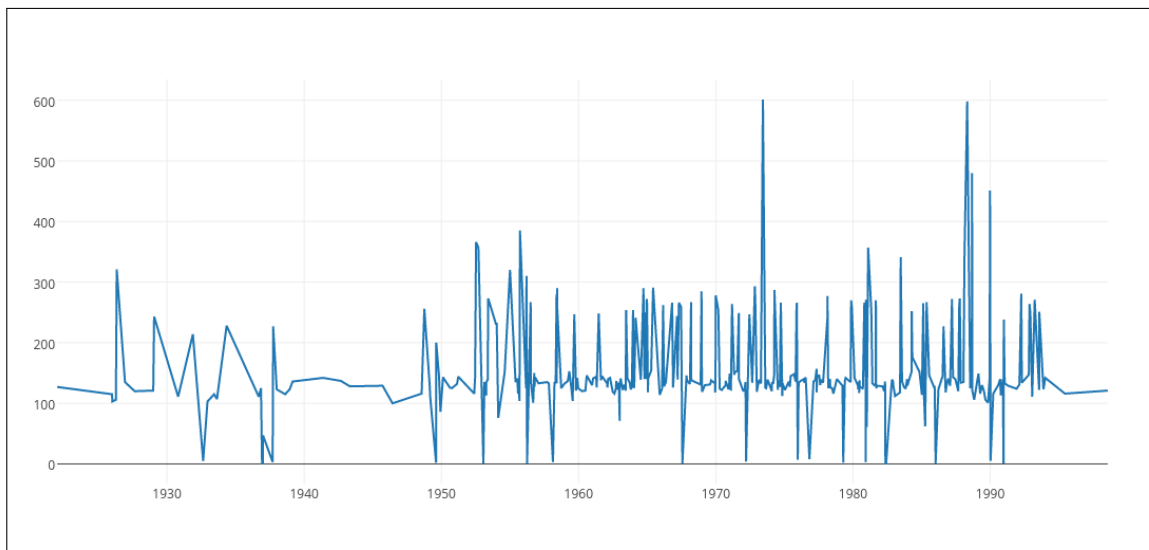


Figure 7.21: Example Chart

7.3 DEPLOYMENT

The application's deployment is managed using Docker ¹. Docker's main focus is to automate the deployment of applications inside software containers, including everything the application needs to run, such as code, binaries, libraries, among others. These applications will run isolated from the host's system.

All developed containers rely on previously developed base images, which can be considered as basic Operative Systems (usually Linux distributions). Therefore, containers are running instances of a given image. A given image can be nested with another image, causing Docker to be highly modular.

7.3.1 DOCKER IMAGES

Instead of simply creating an image and then publishing it for other users to download, which leads to huge file transfers (especially whenever that image requires an update), Docker allows the definition of Dockerfiles. These Dockerfiles represent a series of instructions that enable the creation of an exact copy of the image on which the system is running, without having to actually transfer the image itself.

In order to follow the modular practices that Docker advertises, both the Client application and the Server application are deployed using separate Dockerfiles.

7.3.2 CLIENT

The application was written using ES2015(ES6) standard, and uses the CommonJS module format. ES2015 is a newer version of the ECMAScript standard, which is a language specification that JavaScript implements, while CommonJS specifies a module format for JS. However, neither one of these features is yet (fully) supported by browsers, therefore both a module bundler and a compiler

¹<https://www.docker.com/>

are needed. The module bundler is responsible for analysing all of the project's dependencies and concatenating them into one big file that can be used in a browser. In this case, the module bundler is Webpack ². Although it is mainly used as a module bundler, Webpack's behaviour can be further extended. In this case, it is also being used to compile the written code to valid JavaScript. This step is necessary because not only because browsers do not yet fully support the ES2015 syntax, but also because, as said previously, the JSX syntax used to develop React components is not valid JS. This is accomplished by using the Babel ³ compiler. The whole compilation process results in a JavaScript bundle with 2.8 MB.

The client's docker uses the base Docker `nginx` image, which serves the static HTML, CSS and JS code (Listing 6).

7.3.3 SERVER

The Server application is deployed in the `jupyter/scipy-notebook` image, which is based of a base Debian image (Listing 7). It already contains all of the necessary scientific packages, such as pandas and scikit-learn, all managed using the Conda manager ⁴. Therefore, all there is to install is django, DRF, Redis, among others.

The Django server is deployed through a `uwsgi` ⁵ application server container, that interacts with an NGINX ⁶ Reverse Proxy.

This particular image also has an entry-point that is responsible for launching every process necessary to the server's execution, such as the Redis, NGINX and `uwsgi` daemons, as well as automatically perform database migrations that have not yet been applied (Listing 8).

7.3.4 DOCKER COMPOSE

As seen previously, the developed system's components need to be executed in different containers. In order to manage this, we are using Docker Compose ⁷, which allows the definition of multi-container applications.

In this case, Docker Compose launches three separate containers: the `db`, the `web` and the `client` containers (Listing 9).

The `db` image was not previously referred to because its configuration needs nothing besides its base `postgres` image. Its configuration relies simply in the `POSTGRES_USER` and `POSTGRES_DB` environment variables, which are set at execution time. This container is executed in port 5432, and is mapped to the host's 5432 port.

The `web` container is executed in port 8000, and is mapped to the host's 8000 port.

Finally, the client container is executed in port 8888, and is mapped to the host's 8888 port.

²<https://webpack.github.io/>

³<https://babeljs.io>

⁴<http://conda.pydata.org/docs/>

⁵<http://uwsgi-docs.readthedocs.io/en/latest/>

⁶<https://www.nginx.com/>

⁷<https://docs.docker.com/compose/>

CONCLUSION

This chapter will summarize the results obtained from the developed system, as well as some of the scenarios it enables. Finally, some future work to perform in order to improve the current product will be listed.

8.1 CONCLUSION

Nowadays, the exploitation of medical imaging laboratories has become a crucial part of healthcare institutions business models. Medical imaging repositories hold a tremendous amount of data. Since this data is derived directly from the medical practice, it has extreme accuracy for analytics purposes. Timely exploitation of this data has promised to improve the efficiency of medical institutions' business practices, as well as the quality of healthcare services. Their analysis has already been subject of several studies in the last years that demonstrate the potential benefits from systematic data analysis, namely in radiation dosage surveillance processes [3], efficiency analysis of professional practices [2], [4], study of cost-effective diagnosis [5], among others [5], [34].

Even though this importance has already been acknowledged by the community, the volume and production rate of medical imaging data makes manual analysis impractical. Moreover, due to some limitations of the DICOM standard that only enforces the storage of a few number of mandatory attributes in the databases, data analysis processes are limited in traditional PACS repositories. As a result, the exploring of imaging metadata, for extraction of relevant knowledge, is usually performed by third party applications that require pre-installation and integration with image archives and RIS. Moreover, state-of-the-art tools that focus on the analysis of the metadata are either not open-source, or too tightly associated with a proprietary PACS, making their integration into other repositories an extremely difficult task. Besides, most of these systems are using deprecated and unreliable tools.

This thesis reports the development of an innovator real-time Business Intelligence framework for medical imaging repositories that makes possible to use the metadata stored in a DICOM based repository, without requiring predefined data models or imposing rigid data flows. Specifically, the developed system takes advantage of Dicoogle's data mining features for extracting data from existing PACS. The system provides a series of exploratory techniques and visualization tools, which enable further understanding of the working dataset, allowing the extraction of valuable information.

The developed system exposes its methods through the use of a Web interface, hence making the system accessible to any user, which only requires a simple Web Browser. This interface was carefully designed in order to provide the most possible functionality, while still keeping its basic usability in mind, making it accessible to even a relatively inexperienced user. In order to complement its extensive data manipulation components, DicooogleBI also exposes a series of data visualization tools, such as functional and appealing charts, as well as a greatly customizable dashboard.

8.2 FUTURE WORK

Even though this thesis fulfils all specified requirements, there are many fields to explore that allow its further improvement.

One of the first concerns in the current state is the system's performance. Currently, the factor that most impacts the system's execution time are IO operations, specifically the HTTP transfers on which the system relies in order to access the PACS' studies. Improving on this factor could severely improve the overall application behaviour.

The second field that would be of great interest for the application would be the inclusion of Machine Learning algorithms. Even though some workflows that included these algorithms were developed, there was no time to integrate them into the platform, even more so because exploring useful algorithms for this specific scenario proved to be a very time-consuming task.

8.3 CONTRIBUTIONS

- Improvements in Dicooogle's Platform (<https://github.com/bioinformatics-ua/dicooogle>).
- M. Valerio, T. Marques, and C. Costa, "User oriented platform for data analytics in medical imaging repositories." (accepted for poster presentation).
- M. Valerio, T. Marques, and C. Costa, "Real-Time Business Intelligence Framework for Medical Imaging Repositories." (to be submitted).

REFERENCES

- [1] H. K. Huang, “Enterprise pacs and image distribution”, vol. 27, no. 2-3, pp. 241–253, 2003, ISSN: 08956111. DOI: 10.1016/S0895-6111(02)00078-2.
- [2] M. Hu, W. Pavlicek, P. T. Liu, M. Zhang, S. G. Langer, S. Wang, V. Place, R. Miranda, and T. T. Wu, “Informatics in radiology: efficiency metrics for imaging device productivity”, *RadioGraphics*, vol. 31, no. 2, pp. 603–616, Mar. 2011, ISSN: 0271-5333. DOI: 10.1148/rg.312105714.
- [3] S. Wang, W. Pavlicek, C. C. Roberts, S. G. Langer, M. Zhang, M. Hu, R. L. Morin, B. A. Schueler, C. V. Wellnitz, and T. Wu, “An automated dicom database capable of arbitrary data mining (including radiation dose indicators) for quality monitoring”, *Journal of Digital Imaging*, vol. 24, no. 2, pp. 223–233, Apr. 2011, ISSN: 08971889. DOI: 10.1007/s10278-010-9329-y.
- [4] S. Ondategui-Parra, S. M. Erturk, and P. R. Ros, “Survey of the use of quality indicators in academic radiology departments”, *American Journal of Roentgenology*, vol. 187, no. 5, W451–W455, Nov. 2006, ISSN: 0361-803X. DOI: 10.2214/AJR.05.1064.
- [5] W. Raghupathi and V. Raghupathi, “Big data analytics in healthcare: promise and potential”, *Health Information Science and Systems*, vol. 2, p. 3, 2014, ISSN: 2047-2501. DOI: 10.1186/2047-2501-2-3.
- [6] R. G. Stern, “Diagnostic imaging: powerful, indispensable, and out of control”, vol. 125, no. 2, pp. 113–114, 2012, ISSN: 00029343. DOI: 10.1016/j.amjmed.2011.07.037.
- [7] W. Pan, G. Coatrieux, D. Bouslimi, and N. Prigent, “Secure public cloud platform for medical images sharing.”, *Studies in health technology and informatics*, vol. 210, pp. 251–5, 2015, ISSN: 0926-9630.
- [8] R. H. Choplin, J. M. Boehme, and C. D. Maynard, “Picture archiving and communication systems: an overview.”, *Radiographics : A review publication of the Radiological Society of North America, Inc*, vol. 12, no. 1, pp. 127–9, Jan. 1992, ISSN: 0271-5333. DOI: 10.1148/radiographics.12.1.1734458.
- [9] T. Godinho, *Computer methods for performance optimization in medical imaging networks*, Aveiro, Portugal, 2015.
- [10] H. Huang, *PACS and Imaging informations: basic principles and applications*. Wiley, 2004, p. 637, ISBN: 0471251232.
- [11] P. Mildenerger, M. Eichelberg, and E. Martin, *Introduction to the DICOM standard*, 4. Springer-Verlag, Apr. 2002, vol. 12, pp. 920–927, ISBN: 0938-7994. DOI: 10.1007/s003300101100.
- [12] D. Haak, C. E. Page, S. Reinartz, T. Krüger, and T. M. Deserno, “Dicom for clinical research: pacs-integrated electronic data capture in multi-center trials”, *Journal of Digital Imaging*, vol. 28, no. 5, pp. 558–566, Oct. 2015, ISSN: 1618727X. DOI: 10.1007/s10278-015-9802-8.

- [13] T. Soares, *Plataforma web de monitorização de dose de radiação em imagem clínica*, Aveiro, Portugal, 2015.
- [14] S. Rosslyn, “Digital imaging and communications in medicine (dicom) part 5 : data structures and encoding”, Tech. Rep., 2015.
- [15] —, “Digital imaging and communications in medicine (dicom) part 6 : data dictionary”, Tech. Rep., 2015.
- [16] —, “Digital imaging and communications in medicine (dicom) part 3 : information object definitions”, Tech. Rep., 2015.
- [17] —, “Digital imaging and communications in medicine (dicom) part 17 : explanatory information”, *Medicine*, vol. 10, no. 1, p. 1040, 2011, ISSN: 15355667. DOI: 10.1007/978-3-642-10850-1.
- [18] —, “Digital imaging and communications in medicine (dicom) part 7 : message exchange”, Tech. Rep., 2015.
- [19] —, “Digital imaging and communications in medicine (dicom) part 18 : web services”, Tech. Rep., 2015.
- [20] H. Chen, R. H. L. Chiang, and V. C. Storey, “Business intelligence and analytics: from big data to big impact”, *Mis Quarterly*, vol. 36, no. 4, pp. 1165–1188, 2012, ISSN: 0276-7783. DOI: 10.1145/2463676.2463712.
- [21] E. S. EMC, *Data Science & Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data*. 2015, ISBN: 111887613X.
- [22] J. Wang, M. Zaki, H. Toivonen, and D. Shasha, “Introduction to data mining in bioinformatics”, *Data Mining in Bioinformatics*, pp. 3–8, 2005. DOI: 10.1007/1-84628-059-1_1.
- [23] E. Rahm and H. Do, “Data cleaning: problems and current approaches”, *IEEE Data Engineering Bulletin*, vol. 23, no. 4, pp. 3–13, 2000, ISSN: 03064379. DOI: 10.1145/1317331.1317341.
- [24] H. Müller and J.-C. Freytag, “Problems, methods, and challenges in comprehensive data cleansing”, *Challenges*, no. HUB-IB-164, pp. 1–23, 2003.
- [25] R. Pitre, U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, M. V. Chavan, and P. R. N. Phursule, “A survey paper on data mining with big data”, *AI magazine*, vol. 5, no. 3, pp. 37–53, 2014, ISSN: 0738-4602. DOI: 10.1609/aimag.v17i3.1230.
- [26] R. Burget, J. Karasek, Z. Smekal, V. Uher, and O. Dostal, “Rapidminer image processing extension: a platform for collaborative research”, *International Conference on Telecommunications and Signal Processing*, no. November 2015, 2010.
- [27] Z. Ghahramani, “Unsupervised learning bt - advanced lectures on machine learning”, *Advanced Lectures on Machine Learning*, vol. 3176, no. Chapter 5, pp. 72–112, 2004, ISSN: 1745-1337. DOI: 10.1007/978-3-540-28650-9_5.
- [28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software”, *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, p. 10, Nov. 2009, ISSN: 19310145. DOI: 10.1145/1656274.1656278.
- [29] F. Akthar and C. Hahne, “Rapidminer operator reference”, Tech. Rep., Aug. 2012. [On-line]. Available: https://rapidminer.com/wp-content/uploads/2013/10/RapidMiner_OperatorReference_en.pdf.
- [30] R. Verborgh and M. D. Wilde, *Using OpenRefine*. 2013.
- [31] KDnuggets, *Analytics/data mining tools used for a real project*, <http://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-software-used.html>, Accessed: 2016-01-27, 2014.

- [32] W. N. Venables and D. M. Smith, “An introduction to r”, Tech. Rep., Dec. 2015. [Online]. Available: <https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>.
- [33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: machine learning in python”, *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2012, ISSN: 15324435. DOI: 10.1007/s13398-014-0173-7.2. arXiv: 1201.0490.
- [34] P. G. Nagy, M. J. Warnock, M. Daly, C. Toland, C. D. Meenan, and R. S. Mezrich, “Informatics in radiology: automated web-based graphical dashboard for radiology operational business intelligence.”, *Radiographics : A review publication of the Radiological Society of North America, Inc*, vol. 29, no. 7, pp. 1897–1906, Nov. 2009, ISSN: 0271-5333. DOI: 10.1148/rg.297095701.
- [35] H.-E. Källman, E. Halsius, M. Olsson, and M. Stenström, “Dicom metadata repository for technical information in digital medical images.”, *Acta oncologica (Stockholm, Sweden)*, vol. 48, no. 2, pp. 285–288, 2009, ISSN: 1651-226X. DOI: 10.1080/02841860802258786.
- [36] C. Costa, C. Ferreira, L. Bastião, L. Ribeiro, A. Silva, and J. L. Oliveira, “Dicoogle - an open source peer-to-peer pacs”, *Journal of Digital Imaging*, vol. 24, no. 5, pp. 848–856, Oct. 2011, ISSN: 08971889. DOI: 10.1007/s10278-010-9347-9.
- [37] F. Valente, L. A. B. Silva, T. M. Godinho, and C. Costa, “Anatomy of an extensible open source pacs”, *Journal of Digital Imaging*, 2015, ISSN: 1618727X. DOI: 10.1007/s10278-015-9834-0.
- [38] M. Santos, L. Bastião, C. Costa, A. Silva, and N. Rocha, “Dicom and clinical data mining in a small hospital pacs: a pilot study”, in *Communications in Computer and Information Science*, vol. 221 CCIS, Springer Berlin Heidelberg, 2011, pp. 254–263, ISBN: 9783642243516. DOI: 10.1007/978-3-642-24352-3_27. [Online]. Available: http://link.springer.com/10.1007/978-3-642-24352-3%7B%5C_%7D27.
- [39] S. Burbeck, “Applications programming in smalltalk-80 (tm): how to use model-view-controller (mvc)”, *Smalltalk-80 v2*, vol. 80, no. Mvc, pp. 1–11, 1992.
- [40] R. T. Fielding, “Architectural styles and the design of network-based software architectures”, *Building*, vol. 54, p. 162, 2000, ISSN: 1098-6596. DOI: 10.1.1.91.2433. arXiv: arXiv:1011.1669v3.
- [41] R. Battle and E. Benson, “Bridging the semantic web and web 2.0 with representational state transfer (rest): semantic web and web 2.0”, *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 1, pp. 61–69, 2008, ISSN: 1570-8268. DOI: 10.1016/j.websem.2007.11.002.
- [42] P. Lubbers and F. Greco, *Html5 websocket: a quantum leap in scalability for the web*, 2016. [Online]. Available: <https://www.websocket.org/quantum.html>.
- [43] V. Pimentel and B. G. Nickerson, “Communicating and displaying real-time data with websocket”, *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, Jul. 2012, ISSN: 1089-7801. DOI: 10.1109/MIC.2012.64.
- [44] I. Fette and A. Melnikov, *The websocket protocol*, RFC 6455 (Proposed Standard), Internet Engineering Task Force, Dec. 2011. [Online]. Available: <http://www.ietf.org/rfc/rfc6455.txt>.
- [45] Django, *Models*, 2016. [Online]. Available: <https://docs.djangoproject.com/en/1.9/topics/db/models/>.
- [46] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The many faces of publish/-subscribe”, *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, Jun. 2003, ISSN: 03600300. DOI: 10.1145/857076.857078.

APPENDIX

```
FROM nginx
```

```
COPY index.html /usr/share/nginx/html/index.html
```

```
COPY static /usr/share/nginx/html/static
```

```
COPY bower_components /usr/share/nginx/html/bower_components
```

Listing 6: Client Dockerfile

FROM jupyter/scipy-notebook

```
USER root
RUN apt-get update
RUN apt-get install -y libpq-dev python-dev redis-server nginx curl
RUN echo "deb http://www.rabbitmq.com/debian/ testing main" >> /etc/apt/sources.list
RUN curl http://www.rabbitmq.com/rabbitmq-signing-key-public.asc | apt-key add -
RUN apt-get update
RUN apt-get install -y rabbitmq-server --force-yes
RUN conda install -y h5py pytables

USER jovyan
WORKDIR /home/jovyan/work
COPY DicoogleBI_django/ /home/jovyan/work/DicoogleBI_django
COPY DicoogleBI_django/DicoogleBI_django/prod-settings.py /home/jovyan/work/DicoogleBI_django/prod-settings.py
RUN pip install -r DicoogleBI_django/requirements.txt

COPY dicoogle-python/ /home/jovyan/work/dicoogle-python
WORKDIR /home/jovyan/work/dicoogle-python
USER root
RUN python setup.py install
USER jovyan

WORKDIR /home/jovyan/work/DicoogleBI_django
USER root
RUN rm -v /etc/nginx/nginx.conf
RUN ln -s /home/jovyan/work/DicoogleBI_django/nginx.conf /etc/nginx/nginx.conf
ENTRYPOINT ["/home/jovyan/work/DicoogleBI_django/docker-entrypoint.sh"]
```

Listing 7: Server Dockerfile

```
#!/bin/bash

export PYTHONUNBUFFERED=0
service redis-server start
service nginx start
service rabbitmq-server start
python manage.py makemigrations DicoogleBI
python manage.py migrate
uwsgi --socket 127.0.0.1:29000 --module DicoogleBI_django.wsgi --workers 3
```

Listing 8: Server Entrypoint

```

version: '2'
services:
  db:
    image: postgres
    environment:
      POSTGRES_USER: miguel
      POSTGRES_DB: dicoogle
    ports:
      - "5432:5432"
  client:
    dns_search: wireless.ua.pt
    dns:
      - 193.136.172.20
      - 193.136.172.21
    build:
      context: .
      dockerfile: client-Dockerfile
    environment:
      - NGINX_PORT=80
    ports:
      - "8888:80"
  web:
    dns_search: wireless.ua.pt
    dns:
      - 193.136.172.20
      - 193.136.172.21
    build:
      context: .
      dockerfile: server-Dockerfile
    ports:
      - "8000:8000"
    depends_on:
      - db

```

Listing 9: Docker Compose

| OverlayRows | Columns |
|----------------------------|-------------------------------|
| NumberOfPhaseEncodingSteps | OverlayBitsAllocated |
| SeriesInstanceUID | uri |
| FrameOfReferenceUID | RequestedProcedureDescription |
| BitsAllocated | AccessionNumber |
| Modality | ContentTime |
| ImagingFrequency | PatientWeight |
| RequestingPhysician | ReferringPhysicianName |
| dBdt | NumberOfAverages |

| | |
|-------------------------------|-----------------------------------|
| AngioFlag | SliceThickness |
| InPlanePhaseEncodingDirection | OverlayColumns |
| PatientBirthDate | SmallestImagePixelValue |
| OverlayBitPosition | PerformedProcedureStepStartTime |
| SamplesPerPixel | WindowWidth |
| MRAcquisitionType | DateOfLastCalibration |
| SAR | PerformingPhysicianName |
| EchoTime | SliceLocation |
| BitsStored | PerformedProcedureStepDescription |
| StudyDescription | WindowCenter |
| StudyInstanceUID | PerformedProcedureStepStartDate |
| PixelRepresentation | WindowCenterWidthExplanation |
| DerivationDescription | PerformedProcedureStepID |
| DeviceSerialNumber | LargestImagePixelValue |
| ScanningSequence | TimeOfLastCalibration |
| ProcedureCodeSequence | RescaleSlope |
| StationName | ImageComments |
| ImplementationClassUID | RescaleIntercept |
| ProtocolName | Manufacturer |
| InstanceNumber | EchoTrainLength |
| ContrastBolusVolume | SourceApplicationEntityTitle |
| SeriesDescription | SOPClassUID |
| ReferencedPatientSequence | PatientPosition |
| MediaStorageSOPClassUID | SoftwareVersions |
| VariableFlipAngleFlag | OverlayType |
| ImagedNucleus | HighBit |
| ContentDate | PhysiciansOfRecord |
| RequestAttributesSequence | OperatorName |
| TransmitCoilName | MagneticFieldStrength |
| PixelBandwidth | FlipAngle |
| ImplementationVersionName | StudyDate |
| PhotometricInterpretation | RescaleType |
| ReferencedImageSequence | InstanceCreationTime |
| MediaStorageSOPInstanceUID | SourceImageSequence |
| AcquisitionDate | InstanceCreationDate |
| RequestedProcedureID | InstitutionName |
| SeriesNumber | SeriesDate |
| ReferencedStudySequence | ManufacturerModelName |
| SeriesTime | PatientID |
| Rows | InstanceCreationDate |
| SequenceName | TransferSyntaxUID |
| RepetitionTime | PercentPhaseFieldOfView |
| PercentSampling | ContrastBolusAgent |
| AcquisitionNumber | InstitutionAddress |
| AcquisitionTime | RequestedCodeSequence |

| | |
|-------------|------------|
| PatientName | StudyID |
| PatientAge | PatientSex |

Table 1: Available DICOM tags