Thesis, COLLÉGIALITÉ

Auteur : Leclercq, Gérôme
Promoteur(s) : Phillips, Christophe
Faculté : Faculté de Médecine

Diplôme : Master en sciences biomédicales, à finalité spécialisée en biomédical data management
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/17910

Automatic tool for pre-processing acquired data at the Cyclotron Research Centre

LIÈGE université
GIGA
CRC In vivo Imaging

Gérôme Leclercq

Degree Candidate for a Master of Science in Biomedical Sciences, specialising in biomedical data management

Faculty Advisors: PHILLIPS Christophe, BELIY Nikita

Liège University School of Medicine and Biomedical sciences

Liège

June 2023

Contents

Abstract

Medical data management, particularly in the field of neuroimaging, is essential for neurological research. Technological advances have enabled the massive collection of complex and voluminous data in a multitude of formats. Effective management of this data is necessary to facilitate research in the case of this thesis, but also accurate diagnosis and personalised treatment if looking a little further afield.

The aim of this thesis is to develop an automatic tool that addresses the challenges of collecting, storing, pre-processing and distributing data acquired in a neurology institution, specifically the Cyclotron Research Centre. The objective is to streamline and optimise the data management process, ensuring efficient and reliable processing of complex and voluminous neuroimaging data. By automating these tasks, the tool aims to improve the overall workflow, enhance data accessibility, and promote reproducible research within the institution.

To achieve the objective, the methodology employed consisted of setting up a data processing pipeline controlled by a database. This pipeline not only facilitated data processing but also acted as a storage system for a multitude of information, including the history of data pre-processing, associated metadata and the complete path to the data.

The results obtained provide solid proof of the effectiveness of using a data processing pipeline to achieve the objectives set. The implemented pipeline prototype demonstrated its ability to successfully manage the various stages of data processing, enabling 2 types of real data to be processed automatically. The pipeline approach has enabled effective automation of data pre-processing tasks, improving the efficiency and accuracy of the process. In addition, the ability to store and access a detailed history of the data pre-processing, facilitated the reproducibility of the results and the traceability of the steps performed.

In summary, the utilisation of a data processing pipeline proved to be an effective and robust solution for achieving the project's objectives.

Acknowledgements

I would like to express my gratitude to everyone who contributed to the completion of this thesis.

First, I would like to sincerely thank my thesis co-promoter, Nikita Beliy, for his guidance and sound advice. His mentorship and expertise were essential throughout this project, helping me to develop skills in the field of data management.

I would also like to thank my promoter, Christophe Phillips, for his administrative guidance and advice on this manuscript.

My thanks also go to all the members of the Cyclotron Research Centre for their collaboration and support. Their contributions, whether in the form of datasets provided to test the tool or feedback for the development of functionalities, have enriched this work and improved its quality.

I would like to express my sincere gratitude to my Uncle, Kouassi Ezane, for contacting Erica Smith who gave me invaluable proofreading support and advice on academic writing. I am extremely grateful for her time, expertise, and availability to help me enhance the academic writing of this dissertation.

Finally, I would like to express my gratitude to my parents and in particular my mother, Elisabeth Ezane, for her constant support and encouragement throughout this adventure and in all my projects. Her presence and moral support have been a precious source of motivation and comfort.

In conclusion, I would like to thank all the people who contributed to the production of this thesis on the creation of an IT tool. Their support and contributions were essential to the success of this project and greatly enriched my data management and programming experience.

List of Figures

Chapter 1

Data and Data Management

This chapter will briefly define the concept of data and then transfer it to the field of neuroimaging. This chapter will also discuss aspects of data management and introduce the concept of a pipeline.

1.1 Data in neuroimagery

According to the Cambridge Dictionary, data is "Information, especially facts or numbers, collected to be examined and considered and used to help decision-making, or information in an electronic form that can be stored and used by a computer"[26], but this definition is too generic to be applicable in this thesis. Therefore, an adaptation of this definition is necessary to conform to the scientific world, which wants to be precise, rigorous, and systematic in its approach to studying a particular field of knowledge. This definition will be essential to facilitate the understanding of this introduction and the manuscript.

Data in biomedical studies are information coming from a subject/patient related to his/her health, and/or physical and mental state. They are helping advance neuroscience research by proving concepts and verifying hypothesis but also to predict diseases at earlier stages, for example. These data are sensitive due to the fact that they come from patients. Additional security rules therefore apply and will be discussed subsequently.

Data alone are not relevant; their full meaning is only accessible when they are analysed and interpreted. This is why data management is a discipline in its own right, as it brings data to life by transforming it into actionable knowledge. Therefore, the time has come to move into this essential area of activity to maximise the potential of data in neuroimaging.

1.2 Data management in neuroimagery

According to Oracle, the general definition applicable to all areas is: “Data management is the practice of collecting, keeping and using data securely, efficiently, and cost effectively.”[7]. Data management may be a little-known discipline, but in fact everyone does data management. For example, the simple act of copying data to a hard drive is data management. Putting holiday photos on a USB stick or on a cloud, so they don’t get lost is also data management.

Data management duties range from data extraction, data backup, and data transformation. Even data analysis can be considered as data management. This range is not exhaustive as the concept is so vast, but in general, data management can be limited to the transfer, transformation, security, and processing of data.

As long as the volume of data and its complicity remains small and simple, respectively, the data owner, which is the person in charge of his/her data in this thesis, can do all the data management himself without any complex concepts. However, as the volume and complexity of the data increases, the management task becomes increasingly complex. Therefore, the data owner needs a solid data management strategy to take full advantage of the data.

1.2.1 Importance of data management in biomedical research

Biomedical data come from patients and therefore relate to health; therefore, they must be of high quality. Health data incorrectness due to inaccurate or biased acquisition could lead to the deterioration of the quality of research and consequently of health care. For example, if researchers analyse noisy, poorly reconstructed or poorly managed data, this may dilute the effect they are trying to prove or even simulate an effect where there is none and therefore draw fundamentally wrong conclusions. Biased data analysed can quickly have serious consequences in the medical research world.

Knowing the complexity of data management and the risk factors associated with its poor practice, more and more teams realise the need for a data manager with the task of overseeing data management and ensuring the quality of data within the research unit.

1.2.2 Main components of data management

The list of duties included in data management may vary from one field of application to another. Data management in the biomedical field will consist of different challenges compared to, for example, financial field. Within biomedical research world itself, data management can contain numerous tasks according to the biomedical research field. In neurology, data management will be oriented towards tasks that will improve the quality of research in general. To achieve this, data management consists of a number of guidelines, which will be described in the remainder of this subsection.

Identify user needs

Understanding user needs is a key task in data management to identify the issues. To do this, the data manager remains in contact with users to identify the issues, the gaps and improvements to realise. Thus, the data manager regularly receives feedback and requests from users and can react to demand promptly. With this approach, researchers can perform their analysis without thinking about technical elements.

Ensure consistency, homogeneity, and organisation of acquired data

Ensuring the homogeneity, consistency, and organisation of the acquired data are crucial tasks in data management as well. Indeed, it is important to maintain a good structure in order to know where to find the data at any time and to know that these data are homogeneous and consistent. To do this, the data manager defines reference data and the architecture of data.

Reference data will serve as a common base from which the new data will be compared and validated. Thus, the comparability of data collected at different times or in different contexts will be ensured, thus facilitating data quality control.

Data architecture will serve to maintain the same way to organise data in folders and subfolders. This way of data organisation will facilitate collaboration and data sharing between researchers.

Ensure data availability and discoverability

Data availability means that data is accessible and available to those who need it, this is an important task assigned to data managers as non-available data cannot be analysed by researchers.

Data discoverability means that users can find the data without knowing where exactly it is stored. This is possible due to the storage of associated metadata, making it easy to identify and find data.

Monitor resources used to manage data

Data management, as any form of data manipulation, uses computer resources. Monitoring these resources as well as ensuring that there are enough of them to process data is a part of data management duties. For example, it is important to reserve enough free space not only for the data currently acquired, but also for the data that will be acquired in the future.

Software and version control

Data management can use a variety of tools, they can be internal but most of the time they contain external software regularly updated to improve performance or to correct errors in them. On one hand, using updated software can compromise data homogeneity, but on the other hand, the use of non-updated software can introduce biases into data due to software issues. Identifying the need for updates coupled with keeping track of software versions used is a part of data management.

Ensure the quality of data

As mentioned in Section 1.2.1, data quality and reliability is essential in the research world and this task is part of data management. To do this, data managers can put in place procedures to estimate validity and reliability of data.

Ensure the security of data

Being sure of data security is an important task. Indeed, biomedical research uses highly sensitive data, and the European Union imposes a regulation, named General Data Protection Regulation (GDPR)[9], to all treatment of such data. For data management, it means that data is stored securely, with access provided only for authorised people, and processed locally without using cloud services.

In addition, data management must take into account CIA Triad which is Confidentiality, Integrity, and Availability. Confidentiality involves ensuring that data is protected from unauthorised access. Integrity, Guaranteeing the integrity of data means that it is not altered, corrupted or modified in an unauthorised way. Availability, ensure that the data are available and tools for their management are operational when required.

Most of the data management tasks described above can be automated using scripts, and these scripts can be organised into a pipeline, but before going any further, it is important to understand what a data processing pipeline is.

1.2.3 Data management using processing pipeline

In the context of this thesis, a pipeline consists of collection of scripts allowing data to be processed on its output. The main purpose of a processing pipeline is therefore to successively apply steps to each data consistently, to enable a standardised processing of data.

The steps mentioned above can range from data acquisition, data pre-processing, data anonymisation, data fusion and integration, data storage and management, and data sharing and collaboration. In other words, the data management steps included in a processing chain can be anything that can be automated.

Having acquired this knowledge about data, data management and processing pipelines, the next step is to focus on an institution where these concepts can be applied keeping in mind the aim to propose a prototype of such a pipeline, which would process biomedical data in a consistent and rigorous manner.

Chapter 2

Cyclotron Research Centre

After exploring the concepts and theoretical principles of data management, it is now essential to put this knowledge into practice. In this new chapter, the conceptual framework will be replaced by an experimental approach within an institution, implementing the methods and techniques acquired to achieve data management in neurological biomedical research. Indeed, in this chapter, the Cyclotron Research Centre (CRC) will be presented in two different layers. The first layer will be the general presentation of CRC and the second layer will talk about research at CRC with its goals and studies to understand the environment. Subsequently, data acquisition and manipulation will be covered.

2.1 Overview of GIGA-Cyclotron Research Centre

The GIGA-CRC *in vivo* imaging is a research unit part of the GIGA Institute[10]; it also encompasses GIGA-Cyclotron technological platform[11]. This centre is specialized in the field of *in vivo* imaging for clinical and preclinical research. A wide range of diseases is covered by research at the GIGA-CRC-ivi, including neurological disorders such as Alzheimer's disease, Parkinson's disease, consciousness disorders, epilepsy, cancer, and cardiovascular disease. Human studies aim to better understand biological processes related to sleep, emotions, addiction, movement, cognition (memory, attention, executive functions) and consciousness. Preclinical studies are conducted in small animals for the development of new therapeutic molecules in these areas of research.

Covering these research topics is possible thanks to various neuroimaging modalities that can be acquired at the CRC, including Magnetic Resonance Imaging (MRI), Positron Emission Tomography (PET), Electroencephalography (EEG), as well as sleep chambers, Transcranial Magnetic Stimulation (TMS) and neuropsychological tests. The CRC also contains a cyclotron and associated radiopharmaceutical laboratory where radioligands are developed and produced.

2.2 Acquired image modalities at CRC

In this section, the different acquisition methods, machines, and data within the CRC will be described in detail from acquisition to converted format.

To accomplish this, all of the steps from acquisition machines to raw data formats for each multimodal technological infrastructure available at the CRC will be covered to promote the understanding of future challenges related to the management of this data.

Positron Emission Tomography

Positron Emission Tomography (PET) is a non-invasive molecular imaging technique that allows visualisation of metabolic and molecular activity due to radioactive tracer injected into the subject body. The radioactive element decays with an emission of a positron, which in turn annihilates with an electron of the matter in the brain. A pair of photons emitted during annihilation are detected by the PET scanner, allowing to reconstruct the location of the initial decayed radioactive element[2].

The CRC utilises an Ecat Extract HR+ PET camera, manufactured by Siemens, for human PET equipment. This camera has the capability to capture whole-body PET scans with a 15.5 cm axial field of view, as well as to perform 3D volumetric acquisitions of the entire brain, and both 2D and 3D dynamic acquisitions[12]. The preclinical equipment is composed of the Siemens Concorde FOCUS 120 micro-PET[12].

The acquisition data obtained from these equipment are saved using the ECAT proprietary data format with ".c7" as file extension[35].

Magnetic Resonance Imaging

Magnetic Resonance Imaging (MRI) is a non-invasive medical imaging technique that allows visualisation of the soft tissues in humans or animals. To do this, first, a strong static magnetic field B0 (from 0,5 up to 9 Tesla) is applied to the subject, which orients the hydrogen protons of the matter alongside the B0 axis. Then, a radio frequency magnetic field is applied in order to perturb, i.e., excite, the protons. Excited protons induce a magnetic field which is detected by the MRI reception coil. Then after excitation, protons start to relax, i.e., return to their equilibrium state. With the help of linear gradient coils, the collected signals can be reconstructed into 3D images, for example of a brain, distinguishing different tissues, measuring blood oxygenation level, or water diffusion (determining the orientation of white matter fibers), depending on the applied radio frequencies pulse and gradient sequence[14].

The human MRI equipment at the CRC includes 2 Siemens scanners, The 3T MAGNETOM Prisma scanner is equipped with an array receiver coils for parallel and multi-band imaging. The 7T MAGNETOM Terra scanner is equipped with 1TX and pTX

Nova Coils (single and parallel transmit coils respectively). The preclinical equipment is an Agilent MicroMRI 9,4T 310 ASR[12].

The acquired data is saved using the DICOM format, which stands for Digital Imaging and Communications in Medicine, with .DCM or .IMA file extension[27].

Electroencephalography

Electroencephalography (EEG) is a noninvasive brain imaging technique able to measure electrical activity of brain. It is based on the measurement of fluctuations in the electrical activity of brain cells, called action potentials, which are generated by neural activity. The measurement is done via electrodes placed on the scalp of a subject. These signals are recorded as a set of signals named EEG trace and are used to study cognitive and emotional processes, as well as to diagnose neurological disorders such as epilepsy[22].

The human EEG equipment at CRC is composed of two different EEG systems. One is manufactured by Electrical Geodesics, Inc., Eugene, (OR, USA) with 256 channels. The other system with 64 channels is manufactured by BrainAmp DC[12]. They have respectively 5000 Hz and 1450 Hz as sampling rate.

The acquisition data are saved using the EMBLA proprietary data format[16] with .EBM file extension.

Actigraphy

Actigraphy is a noninvasive method that measures physical activity. This technique is based on wearing a small device called an actimeter, which records body movements and vibrations over time[19]. This technique is used to study the general behaviour of humans, their daily sleep patterns and disorders such as apnoea and insomnia. Its main advantage is the possibility of recording over a long period of time (month, year).

At the CRC, sleep units possess several actimeters produced by Axivity. The acquisition data are saved using the Continuous Wave Accelerometer (CWA) format with .CWA file extension[15].

Transcranial magnetic stimulation

Transcranial magnetic stimulation (TMS) is a noninvasive method of stimulating specific areas of the brain using localised magnetic field. TMS can be used to study the connections between different regions of the brain by applying a magnetic impulse to one region, and measuring the propagation of magnetic perturbation through the brain using EEG for example. This technique is used to study brain function but may also have therapeutic applications for the treatment of certain neurological and psychiatric diseases[6].

The human TMS equipment at the CRC includes high-precision Transcranial Magnetic Stimulation system equipped with a neuronavigation system produced by NBS,

Modality	Source format	Converted format
PET	ECAT	NIFTI
MRI	DICOM	NIFTI
EEG	EMBLA	EDF, BVCDF, MEEG
Actigraphy	CWA	CSV
TMS	NEXSTIM	BVCDF, EDF

Table 2.1: A summary of different acquired and converted data formats acquired at CRC.

Nexstim in Finland. Different functioning modes are available, including single pulse TMS, coupled pulse TMS and rTMS. Two 8-shape coils are also available, one is a standard coil and other is an air-cooled coil which is ideal for rTMS usage[12].

The acquisition data are saved using Nexstim proprietary data format which comprises two files with .nbe and .nxe file extensions[25].

2.3 Converted data formats

The acquired data format is dictated by acquisition machines, which makes source data format not always convenient. They therefore need to be converted to be compatible with the used analysis and visualisation software. The most common converted data formats used in the CRC are listed below and summarised in Table 2.1.

NIFTI

Neuroimaging Informatics Technology initiative (NIFTI) format[8] is one of the most popular formats for MRI and PET images. A single NIFTI file can contain several DICOM or PET images, which makes it useful for storing a temporal series of images. The majority, if not all, MRI visualisation and analysis softwares support NIFTI format, assuring interoperability of software. Moreover, there are many available DICOM to NIFTI converter tools.

At the CRC, all DICOM images from MRI and PET machines are converted to NIFTI format.

EDF

European Data Format is a popular format for EEG, MEG, and TMS data. The EDF is a versatile and easy-to-use file format for sharing and storing recordings of biological and physical signals that are recorded on multiple channels[18].

At the CRC, data from EEG and TMS machines are converted to EDF format.

MEEG

MEEG is a useful data structure provided by the SPM toolbox to facilitate working with MEG data. This is a container for the actual MEG data itself. These live on disk in two files: a .dat file, which contains the time series data, and a .mat file that contains header information[32].

At the CRC, data from EEG machines are also converted to MEEG format to facilitate the use of the SPM toolbox[33].

BVCDF

BrainVision Core Data Format[4] is a proprietary format for EEG and TMS, it is similar to MEEG through its data structure with accessible metadata. It is therefore a good alternative to EDF as it is easy to read and does not require much effort to be used with the SPM package.

BVCDF is therefore a another file format used to store neurophysiological data with easy access to them, making it also useful for sharing. Furthermore, BVCD files can be imported and analyzed directly using various software packages, including BrainVision Analyzer, EEGLAB, MNE-Python, and with minor modifications, the SPM package.

CSV

Comma Separated Values (CSV) is an open format, that stores any tabular data in table form where the fields are separated by a particular symbol (semicolon, comma, tab). It is a simple, readable text file format. CSV files can be imported into many data processing programs, such as Microsoft Excel or OpenOffice Calc, for easy viewing and analysis[17].

At the CRC, Comma Separated Values is the converted format for actigraphy and psychological test data.

2.4 CRC data management

The CRC does not possess a centralised data management system, which means that the processing of data depends on both the acquisition machine and the group of experimenters. Practices can therefore range from everyone doing their own procedure, to an automatic pipeline that pre-processes and stores the data.

For data acquired on dedicated MRI, PET and TMS machines, the data is stored on the acquisition machine and a copy is sent to their server, where it is then retrieved, converted and placed in research group space. The retrieval, conversion, and relocation to the project space is manual. Next, the files are organised following the Brain Imaging Data Structure (BIDS)[24] which is a standardized format for organizing and sharing neuroimaging data. BIDS is used at the CRC on a dedicated data storage server managed by the SEGI and GIGA (MassStorage). The organisation scripts are executed manually by the user.

For data acquired on dedicated EEG machines, the CRC has a fully automated pipeline capable of retrieving the data from the server, converting it, organising it, performing quality control, and storing it on MassStorage.

For data from actimeter and other psychological tests. Currently, the CRC is not able to automatically send data to the server. The data are therefore transferred, converted and organised manually, using procedures that vary from project to project. A project here refers to a group of researchers working on the same theme and analysing data from the same set. This can range from manual encoding of questionnaires to direct analysis of the data without going through any system.

2.4.1 Issues and hazards of current practices

The above practices leave room for possible problems and dangers in data management. These will be described in detail in order to capture the functioning of the CRC data management.

Lack of standardisation and centralisation in data management

The CRC leaves room for each team and sometimes even each researcher to use their own set of tools, such as scripts or software. This makes it possible for these tools not to be shared and therefore generates a loss of continuity and time, as scripts often have to be rewritten or created to process the data. This lack impacts one of the most crucial steps, which is the pre-processing of data.

No ready analysis data

Much of the data collected by the CRC is not ready for analysis as soon as the researcher wishes to use it. The researcher often has to do their own pre-processing of the data shortly before analysis, which can lead to errors and inconsistencies in the analyses due to the rush.

Unknown data quality

The quality of the data is often unknown until it is analysed. Since most of the data acquired at the CRC is intended for future use, the discovery of problems in the data years later can sometimes rule out any possibility of remediation.

Lack of information

There is a lack of information about the history and organisation of the data within the CRC. Researchers often have little information about how the data was pre-processed, including the software, scripts and parameters used. This can lead to a loss of continuity and context in subsequent analyses.

The fact that there are no clear guidelines on data management at the CRC leave rooms for issues, these do not significantly impact the first analysis of acquired data. However, it significantly delays the re-analysis of the data and reduces the repeatability of results. Therefore, non-uniformity of system present can reduce the quality of research produced at CRC.

2.4.2 Benefits of data pipeline in CRC

As mentioned earlier in Section 1.2.3, a processing pipeline can ensure the proper functioning of different tasks and thus address different CRC issues. In particular, such pipelines can provide:

Ability to standardise and/or centralise procedures

With a processing pipeline approach, it will be easier to encourage users working with the same type of data to use the same tools, which will homogenise pre-processing of data at all levels. In other words, between data from the same researcher, between data from researchers in the same project, and between data from different projects.

Ability to process data in a data type independent way

With a processing pipeline approach, it is possible to have different processing with the same global infrastructure. Thus, all modalities will benefit from the same level of protection and control.

Ability to centralise storage

With a processing pipeline approach, it is possible to organise a centralise storage, where pre-processed data will be available. This storage will be accessible only for authorised people, and will ensure data safety.

Ability to book-keep

With a processing pipeline approach, it will be possible to keep track of all processed data. For example, individual files could be found based on metadata, ensuring data discovery. Additionally, the full history of the processing of the file, used tools and their versions, will be stored and could be used to improve reproducibility of results.

A processing pipeline can offer many benefits to the CRC by saving valuable time through the systematisation of data management. This will allow researchers to spend more time on their work rather than on data management.

Chapter 3

Pipeline Proposition

In the previous chapter, a pre-processing pipeline was introduced as a way to automatise data managing tasks described in Section 1.2 and as a response to current issues in data management at the CRC, described in Section 2.4.1.

This chapter will discuss some existing pipelines and why they do not fully satisfy the needs of data management at the CRC, thus motivating the implementation of a new specifically tailored pipeline. Then, the specific CRC data management needs will be reviewed, and from these, the design of the pipeline will be derived.

3.1 Review of certain existing pipelines

Two major categories of pipelines can be defined: generic pipelines, not specifically defined for any area, and medical imaging processing pipelines, specifically defined for neuroimaging data.

As an example of generic pipeline, Luigi[3] can be cited, as an open-source framework developed in Python for creating and executing data pipelines. It makes it easy to manage tasks and the dependencies between them, providing a user-friendly interface for defining workflows. Luigi can be a powerful tool as a data processing pipeline; however, it is not designed to run automatically when data arrives; also, Luigi does not provide bookkeeping system which conserves the history of processed data.

As an example of a neuroimagery pipeline, Nipype[13] can be cited; it is an open-source library developed in Python that facilitates the integration and construction of neuroimaging data analysis pipelines. It provides a unified interface for interacting with various tools and software commonly used in the field of brain imaging, such as FSL, FreeSurfer and SPM. Nipype can be a powerful tool as a data processing pipeline; however, it is not designed to run automatically when data arrives; also, Luigi does not provide bookkeeping system, and it is oriented specifically to MRI images processing.

Porcupine[34] is another example of a neuroimagery pipeline, it is an open-source software library written in Python for creating data processing pipelines. It provides a user-friendly, modular interface for defining and executing complex workflows. Porcupine can be a powerful tool for data processing pipeline; however, it has the same drawbacks as Nipype, which is used in Porcupine.

The common default of pipelines discussed above is a lack of bookkeeping, which is necessary for maintaining the traceability of processed data. Also, they are not centralised and thus cannot guarantee the homogeneity of processed data between different projects. To our knowledge, no existing pipelines fully satisfy the CRC data management requirements, motivating thus the development of a new pipeline from scratch.

3.2 Requested features of the pipeline

Before starting the development/implementation of the pipeline, the required features must be listed first, as they are a direct response to data management duties and challenges within CRC, respectively in Sections 1.2, 2.4.1. These features are listed below, with a short indication on how they could be realised.

Automatic, standardised, and centralised pre-processing procedures/elements

First, the pipeline should process the data without user interaction or with limited user interaction, as soon as the data is acquired. This can be ensured by running the pipeline within an infinite loop, constantly scanning folders where data to process arrive.

Second, the pipeline must process the data as uniformly as possible, using the same tools. To ensure this, the pipeline should provide a standard and validated set of data processing steps.

Finally, the pipeline should be able to run on one dedicated server, and support different projects.

The pipeline must therefore ensure that all data leaving the pipeline is stored on the MassStorage and also provide standardised output directories in order to have the same directory architectures, i.e., the same type of folders, subfolders and file names.

Having these features available within the CRC would eliminate the errors and biases associated with the different personal tools used by each researcher, improve collaboration between members of the CRC research team and make the search for information more efficient. Indeed, the discoverability of data will be enhanced. The reproducibility of research results will also be improved, since everyone will be using the same procedures in the pipeline.

Analysis-ready data as pipeline output

The pipeline should ensure that the data coming out of it are ready to be used, so that all the researcher has to do is enter the data they want to analyse into their favourite dedicated software. As definition of "analysis-ready" may differ from project to project, pipeline must support running additional processing steps for given projects.

This will save CRC members a considerable amount of time, since users of the tool will be able to access their pre-processed data as quickly as possible. This can be implemented by maintaining a library of processing steps, and using some of them based on a configuration file provided by project owners.

Quality metrics associated with data

The pipeline should provide users with assessment of the data quality to enable them to make a decision based on the metrics. This will require an implementation of the user feedback mechanism, which will allow users to communicate their decision to pipeline, and will allow the pipeline to react to user decision.

Having this feature available at the CRC would help researchers to check the quality of data as they leave the pipeline. The data will be sorted before analysis begins, so that poor-quality data will no longer be inadvertently analysed.

Data-type independent infrastructure

The pipeline should provide a data management method that is independent of data types, as a result, the list of supported data formats should be easily extendable, to reduce the needed effort of pipeline maintenance in the event that the CRC has a new data modality to acquire. The pipeline should also be able to take into account all data modalities acquired within the CRC and pre-processed them considering each specificity of the different modalities/formats and even projects. The range of data that can enter and exit the pipeline should still be limited to only the data that the CRC wants to pre-process.

Making this function available to CRC would facilitate the expansion of the data modalities, as the pipeline could quickly take into account their specificities.

Data security

The pipeline must guarantee the security of data, in particular that members of one project could not access the data from other projects. Additionally, user interfaces should not allow manipulations that can destroy or corrupt processed data.

The availability of this function at the CRC would ensure that the data belonging to a project are neither affected nor seen by anyone other than the members of the project. Users should therefore only have access to their data and not to others. Under no circumstances should a problem with data processing block the operation of the pipeline.

Data bookkeeping

The pipeline should be able to provide all the information relating to each stage of processing for each data item entered into the pipeline. This will enable the history of the data to be easily tracked and accessed, ensuring greater traceability and transparency of the data.

In addition to the processing history of data, used tools and versions should be part of the database, available to project members.

The availability of this function at the CRC will provide a complete record of changes and activities related to the data during the pipeline, enabling better traceability, reversibility and problem resolution. This helps to guarantee the integrity, transparency and reliability of the data throughout its life-cycle.

Visualisation of the progress of data processing

The pipeline should provide a user-friendly interface to follow the data flow. In other words, users should be able to visualise the state and stage of the data at a given time in the pipeline.

Having this service available at the CRC gives users a clear and accessible overview of the state of data processing. This gives them a better understanding of the state of their data, enabling them to plan the fate of data leaving the pipeline.

Sending of reports

The pipeline should include a tool to send reports to the users involved, in order to inform them about the status of their data on a weekly, monthly and emergency alert basis.

In this way, the user does not have to connect to the pipeline GUI to find out the status of their data and that of the pipeline. In addition, there will be archival written proof.

Pipeline operations monitoring

The pipeline should have an element able to ensure the monitoring of pipeline operations. This means having a tool that can monitor the activities of systems and applications, detect problems, and subsequently improve performance.

The availability of this service at the CRC will help to find out where the problem arises when a bug occurs.

Possibilities for the user to control and interact with

The pipeline should provide functionality for the user to interact with the pipeline. This includes immobilising data in a particular step, for example, removing corrupted data, resubmitting corrected data, choosing which steps to apply to the data.

The availability of these features at CRC will help users to maintain control of their data.

To summarise, the pre-processing pipeline should constantly run over a dedicated server, waiting for the data to process. Then, the pipeline should process data from different projects independently, but sharing common processing tools. The pipeline must support all data modalities used at the CRC and be able to support additional modalities if needed. It must integrate a GUI with a reporting tool allowing the user to monitor and interact with the pipeline. Last, but not least, the pipeline should integrate a database containing history for all processed data.

To my knowledge, there is as yet no pipeline that fully and optimally integrates all the requirements above-mentioned. In the next chapter, the implementation of the prototype for such pipeline will be discussed.

Chapter 4

Design and implementation

After providing an overview of data management at CRC in Chapter 2 and discussing the pipeline requirements in Chapter 3, it is now time to delve into the practical design and implementation.

The upcoming chapter will provide a comprehensive description of the pipeline's actual design, along with an explanation for the selection of tools employed for its realisation. Additionally, the algorithms utilised to implement each component of the design will be briefly explained, shedding light on their functionality.

4.1 Overall Pipeline Design

The overall design of the pipeline is summarised in Figure 4.1. One of the central elements of this project is the centralised database. This is a database that not only collects and conserves the information about processed or in-processing data, but also ensures the communication between the pipeline and associated graphical user interface (GUI). Furthermore, the database functions as a control element, storing information about the current processing status. Will be covered in more detail in Section 4.1.2. The rationale behind choosing a database as a central element is that it naturally satisfy several requirements imposed to the pipeline. Namely, the ability to organise data in a structured way using tables, relationships and predefined schemas. The ability to efficiently manage storage and resources to process large volumes of data, offering rapid access thanks to indexes and advanced search algorithms. Databases also provide essential security measures, including access permissions to protect sensitive information. Integrity constraints ensure data validity and consistency, preventing duplication or inconsistency. Databases also promote interoperability by using standardised query languages, making it easier to exchange data between different applications and systems. They support the management of simultaneous access to data, guaranteeing consistency and integrity even when users interact simultaneously. Finally, databases offer data backup and recovery

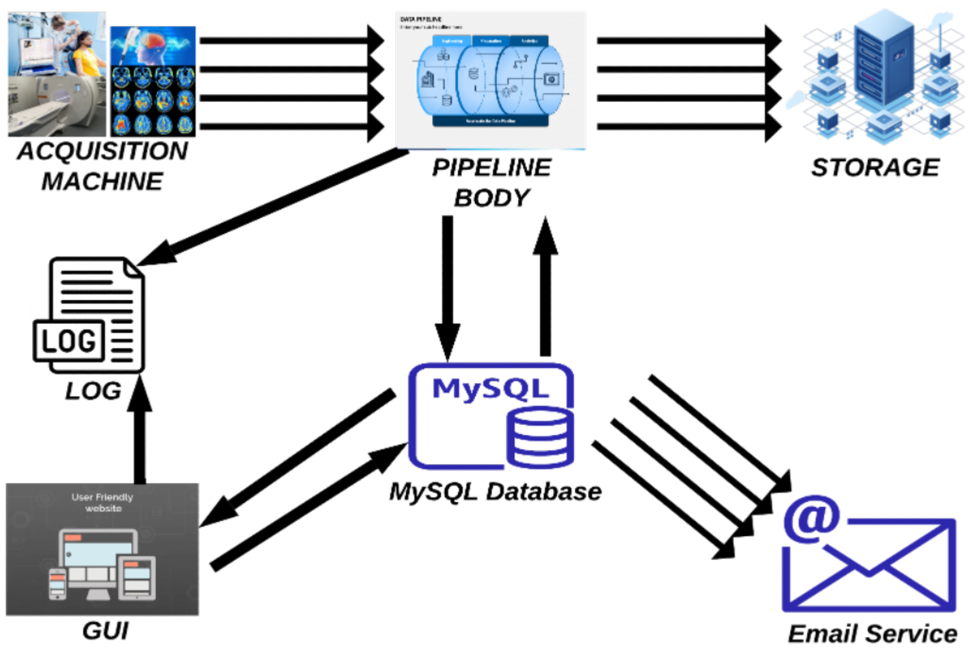


Figure 4.1: Overview of the processing pipeline design, composed of central elements which are the Pipeline body and Database. Other elements are secondary, they are composed of graphical user interface, email service and log files.

mechanisms, enabling data to be restored in the event of failure or accidental loss. The capabilities of this tool therefore cover functionalities relating to the CIA triad, storage of information required for data history, software versions, data monitoring, and user control and interaction.

The other central element of this project that really process data is the body of the processing pipeline. The pipeline body will constantly monitor the input data, and if found, will first register data into the database before processing it. Each individual processing step corresponds to an independent function/class, which will check in the database if there is data that can be processed by a given step. Also, processing steps are responsible for updating the status of processed data in a database. A more in-depth analysis of the pipeline structure will be presented in Section 4.1.3. The rationale behind choice of usage of a processing pipeline as a secondary central element is that it naturally satisfies several requirements imposed to the project. Namely, the ability to automate data flow, minimising the need for manual intervention and labour-intensive processes. By applying defined rules and procedures, pipelines ensure consistency and quality, providing reliable data for analysis. In addition, pipelines are designed with scalability in mind, enabling them to efficiently handle growing volumes of data, as well as new types of data. They can be easily adapted to meet performance, compatibility and capacity requirements as data needs increase. Activity monitoring ensures that data is constantly being processed. Another advantage of data pipelines is the ability to reuse components, which rationalises development and maintenance efforts. This reuse means that processing steps can be used for different data streams, saving valuable time and effort during the development process. The tool's capabilities therefore cover automation, standardisation and centralisation, the production of analysis-ready data with quality measurements, and the scalability of incoming data.

In addition to the core elements, there are auxiliary components which do not directly impact the pipeline's functionality but contribute to its smooth operation. These are GUI, email service and log files.

First, the mentioned database not only stores processed data but also controls the execution of the pipeline. Consequently, the GUI for the database can serve as a graphical interface to the pipeline. Section 4.1.4 provides a detailed discussion on the GUI. The decision to incorporate a graphical user interface as a secondary element in the data pipeline is driven by its ability to fulfil several project requirements. Namely, it allows users to interact with the system visually and interactively, simplifying their interaction with the data. The graphical interface makes it easy to navigate between the different functions and sections of the application, using intuitive navigation elements such as menus, buttons and icons. In addition, the GUI is designed for interoperability, making it compatible with a variety of operating systems and platforms, enabling seamless integration with other applications and promoting efficient data exchange. The CakePHP[5] interface has been selected for its web-based, facilitating access to databases without connecting to the server running the pipeline. Moreover, CakePHP also incorporates security measures that enable controlling access privileges for specific users.

Second, incorporating an email service, although not implemented in this work, would be a valuable addition. This service would enable sending messages to pipeline users regarding important notifications concerning their data, such as error notifications, reports, and system maintenance updates.

Third, implementing a logging system, albeit partially implemented, would serve as a valuable asset. This system would enable monitoring the pipeline's execution, identifying errors, and detecting inefficiencies. It would prove especially useful in the event of pipeline failure.

The interaction between two elements is indirect, it is the database that serves as a relay, which is why it is presented as the governing element of this project. Indeed, each element of the pipeline project does not communicate directly with each other but uses the database as an interpreter. Similarly for the GUI, each intervention of the user on the data does not directly affect the operation of the pipeline. The intervention is first recorded in the database before being operational on the pipeline body. The messaging service follows the same pattern. The only element that interacts independently of the database is the log file directory as it is there to run in parallel with the pipeline to record events to find out what happened if a problem occurs or if something goes wrong. The operation of this LOG directory is therefore independent of the database.

4.1.1 Overview of the pipeline functioning

Data acquired from one of the many acquisition machines at the CRC are copied into an input folder. A specific module, called grabber (see in Section 4.1.3) regularly scans this input folder for new files. If grabber finds a new data file, it creates a new entry into the database corresponding to the data received. This process is called registration. Then, grabber identifies the project which owns this data, and moves it to the processing folder of a dedicated project, in a subfolder according to the data type.

The processing folder is where the steps (as discussed in Section 4.1.3) perform their tasks. All steps of the processing operate in the same way, the backup step will be used as an example: Data that is ready to be processed by the backup stage can be identified by its “ready” status for the step in question in the database. The backup stage regularly scans the database to identify data ready for processing. At the database level, when the step has identified all the files eligible to be processed, it updates their status one by one to “running” when the backup is carried out, and then to “done” to announce the end and successful completion of the step with this specific data. In the processing folder, the step copies the eligible files one by one from the “process” folder to the “backup” folder. Each processing step in the pipeline acts in the same way, both on the database via status updates, and on the data itself at the folder and file level.

Another module, named status updater (see in Section 4.1.3) manages the operation and interaction of steps. To accomplish this, it scans the database to locate all the files that have successfully completed a step, indicated by a “done” status. It then adds new entries for these files in a table, marking them with a “ready” status. This signals to the subsequent step that it can proceed with its function.

4.1.2 Database

The database for this project consists of seven tables. Each table stores different information necessary for the proper functioning of the automatic data pre-processing tool. The Enhanced Entity-Relationship (EER) diagram represents the entities, attributes and relationships in the information system designed for the CRC in Figure 4.2.

Recordings

The Recordings table contains common information about each data file in the pipeline. It is composed of four columns, one for primary key, one for foreign key, and the other for information.

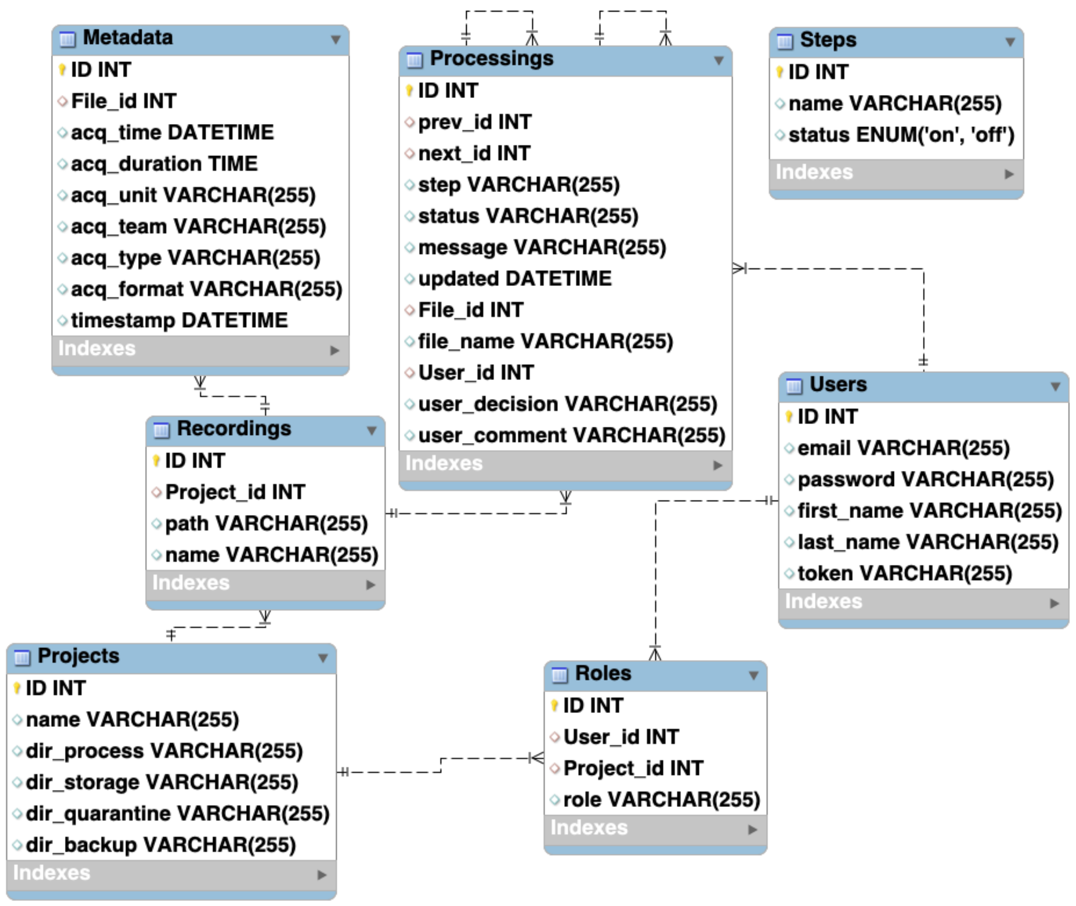


Figure 4.2: EER diagram of pre-processing pipeline database.

It has links to other tables (seen in Figure 4.3), the column Project_ID is the foreign key linking to Projects table and this relation allows each file to be associated with a project.

The Recordings table contains up-to-date information on the name of the file and its absolute path directory, which gathers all the information needed to reconstruct the absolute path of the file.

Metadata

The Metadata table contains information related to acquisition of data, namely time/date, duration, units, team, type, and formats of acquisition. It is composed of nine columns, one for primary key, one for foreign key, and the other for information.

It has links to one table (seen in Figure 4.4), the column File_ID is the foreign key linking to Recordings table and this relation allows each metadata to be associated with a file.

The Metadata table is used to quickly retrieve a given file based on its metadata and inversely quickly retrieve acquisition information metadata corresponding to a file.

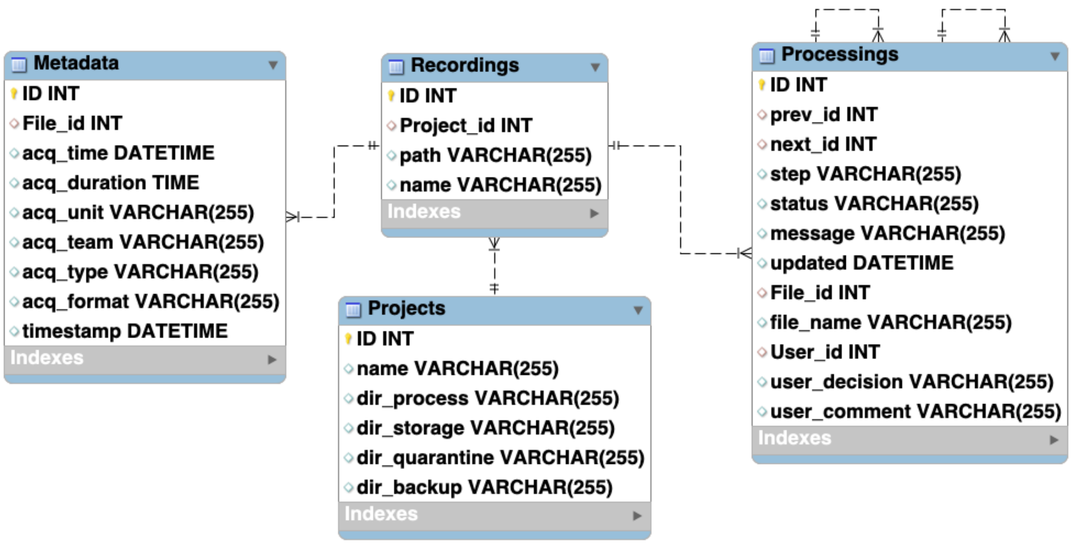


Figure 4.3: EER diagram focus on table Recordings and related tables.

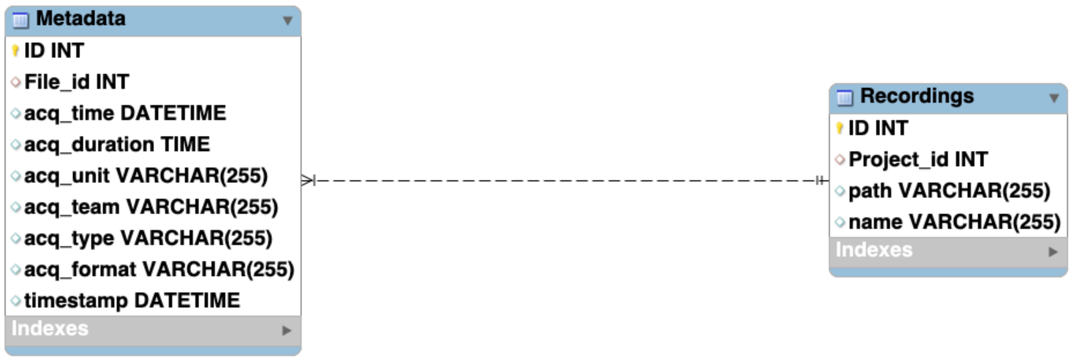


Figure 4.4: EER diagram focus on Metadata and related table.

Processings

The Processing table contains information about the execution of the various processing steps for data that has been or is being processed in the pipeline. Each step applied to a data file, as well as the next step to be applied, has an entry in this table, and this exists for each data file in the pipeline. There are two clusters of columns, the first cluster includes information to provide up-to-date information on the situation of a file through the pipeline. The second cluster includes information related to the user's interaction.

It has links to other tables as well as self-referential connections (seen in Figure 4.5), the column File_ID serves as the foreign key linking to the Recordings table, establishing a relationship that enables each processing step to be associated with a file. The User_ID is the foreign key linking to Users table and this relation allows each processing step to be associated with a user interaction if it exists. The prev_ID and next_ID are the foreign key linking the Processings table itself and these relations allow each processing step to be associated with the previous one and the next one if exists.

The Processings table is used for multiple reasons. On the one hand, the table conserves the history of data processing. On the other hand, this table, using "step" and "status" performs the control of the execution of steps. Status_updator will search the entries == "done" status and create a new entry for the next step with status == "ready" as explained briefly before in Section 4.1.1 and in better detail in Section 4.1.3. Individual Steps will search entries with status == "ready", to process corresponding files and update status to done.

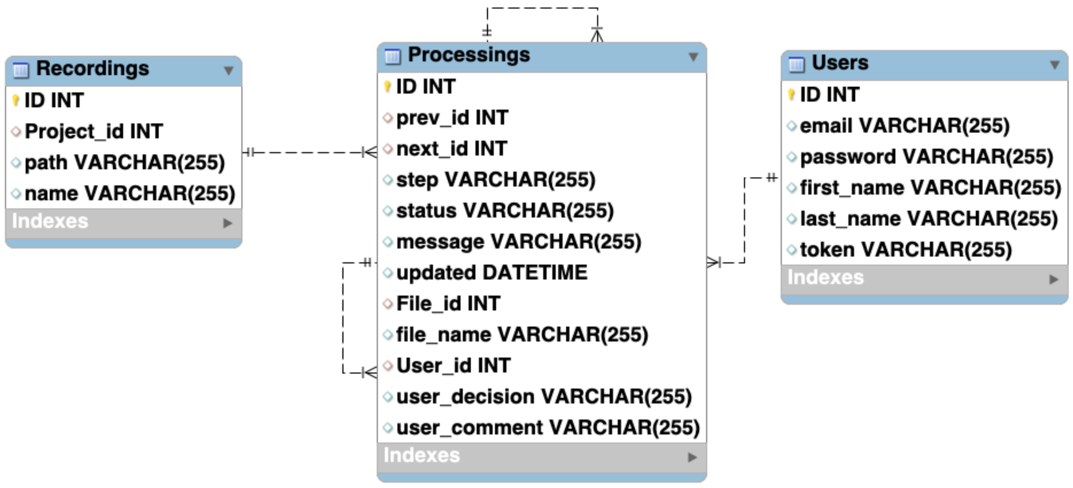


Figure 4.5: EER diagram focus on Processings and related table.

Projects

The Projects table contains information related to projects, specifically names and absolute path to each directory of the steps associated with these projects. It is composed of six columns, one for primary key, one for name, and others for directories needed to steps.

Projects table has links to two tables (seen in Figure 4.6) and is used to gather information on Projects using the pipeline to pre-process their data.

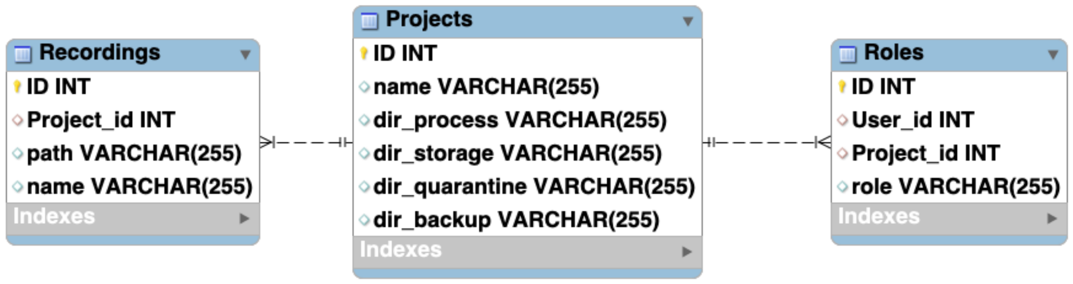


Figure 4.6: EER diagram focus on Projects and related tables.

Roles

The Roles table contains information related to users' roles in projects. It is composed of four columns, one for primary key, two for foreign keys, and the other for information on the role.

It has links to other tables (seen in Figure 4.7), the column User_ID is the foreign key linking to Users table and this relation allows each role to be associated with a project. The column Project_ID is the foreign key linking to Projects table and this relation allows each role to be associated with a project.

The Roles table contains information on the roles of users in projects, which gathers all the information needed to identify the role of a pipeline user in each project, and therefore grant them the various rights associated with it.

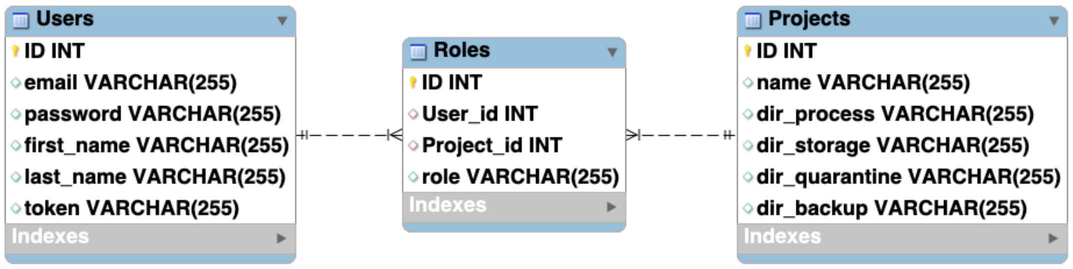


Figure 4.7: EER diagram focus on Roles and related tables

Users

The Users table contains information related to users in general, such as email, name and password. It is composed of six columns, one for primary key, and the other for information on users.

It has links to two tables (seen in Figure 4.8), and is used to gather information on users using the pipeline to pre-process their data.

Steps

The Steps table contains information related to steps with possibilities to turn "ON" or "OFF". It is composed of three columns, one for primary key, and the other for information on activity status of a step.

It has no links with other tables (seen in Figure 4.9), and is used to gather information on steps activation status to be check if they are "ON" before launching them during the pipeline operation.

To create this database, the SQL, MYSQL coupling was used, this choice has several advantages. First, SQL (Structured Query Language) is a standard query language used

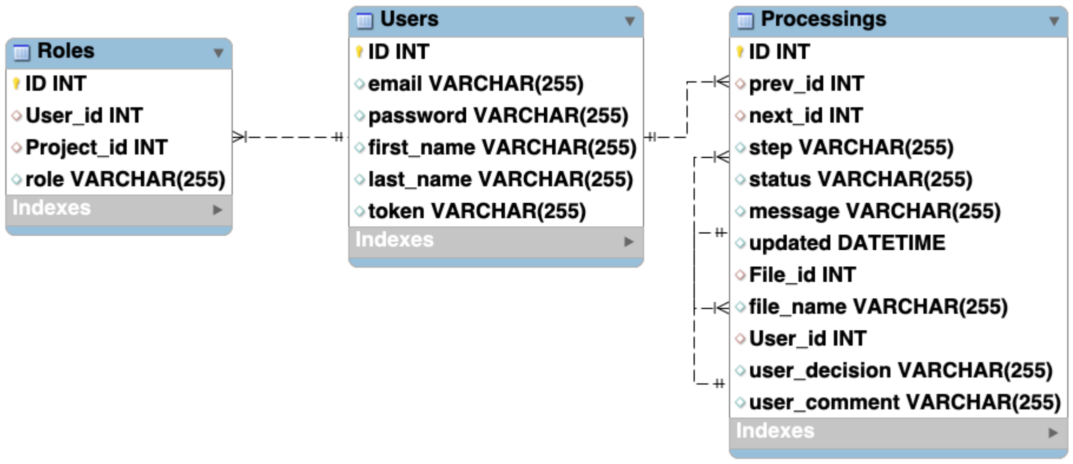


Figure 4.8: EER diagram focus on Users and related tables.

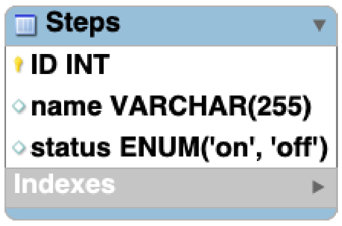


Figure 4.9: EER diagram focus on Steps

to interact with relational databases. MySQL is a relational database management system (RDBMS) that supports SQL as a query language. By using MySQL, one benefit from compatibility with the standard SQL language, which facilitates the development and portability of the automatic data management tool. Secondly, MySQL is known for its high performance and ability to handle large amounts of data. It uses advanced optimisation techniques and supports indexing and caching to improve query performance. This allows it to handle read and write operations efficiently, even in high load environments. Third, SQL provides clear and expressive syntax for formulating queries, making the code easier to write, read and understand. MySQL also provides user-friendly tools and a graphical user interface[1] for easy database management and administration. Fourth, MySQL provides advanced security features, such as user and privilege management, secure logins, data encryption and authentication mechanisms. This protects the sensitive data stored in the database and ensures its confidentiality and integrity. Fifth, MySQL has a large community of developers and users who provide technical support, training resources and regular updates. There are also extensive documentation and online forums where help and advice can be obtained to solve problems. Finally, MySQL has a library specifically designed for Python, called "mysql-connector-python". This library allows you to establish a connection between a Python application and a MySQL

[1]MySQLWORKBENCH

database, making it easy to integrate the database functionality into the Python code. It provides simple and intuitive methods for executing SQL queries, retrieving results and managing transactions. With this library, the power of MySQL can be leveraged directly into Python projects, simplifying development and interaction with the database.

4.1.3 Pipeline body

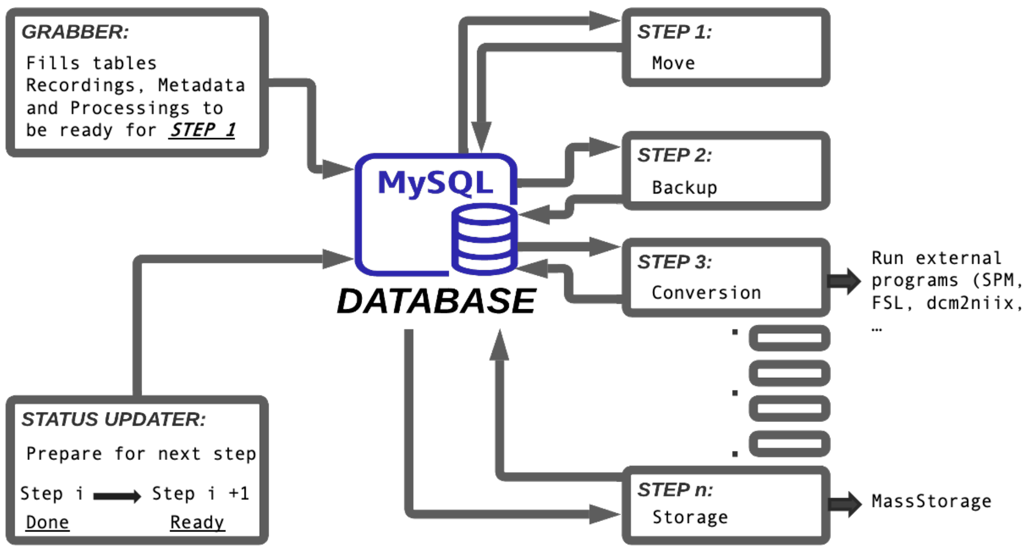


Figure 4.10: Overview of the processing pipeline body. The system is composed of two mandatory modules for the operation which are grabber and status updater, the others are optional steps but ensure data processing.

The processing pipeline body consists of several independent modules all organized in a governing script named "global_run_threads.py". These modules can be split into two categories, one is focused on the correct operation of the pipeline named "system" category, including grabber and status updater, and the other focused on the processing steps that the data will undergo in the pipeline is named "data processing" category. The role of grabber is to bring the data file into the pipeline to be ready for the first steps of data pre-processing, which corresponds to register data into the database. The purpose of the steps is to process data as it moves through the pipeline, performing tasks such as backup and conversion. Steps communicate only with the database, and can be executed independently of each other in no particular order. This design has been chosen in order to facilitate the parallelism. Indeed, without imposing order of execution, steps can run in parallel without blocking each other. The role of status updater is to manage the operation and interaction of the steps, which corresponds to identify within the database, the final steps of file processing complete and prepare them for the next ones. The governing script corresponds to an infinite loop in which all steps are executed for all data file entering the processing pipeline. An overview of the pipeline body is summarised in Figure 4.10.

All the elements and governing script are written/implemented using Python 3.10.9 computer language[30]. Python is an open-source programming language that is widely used for a variety of tasks including data analysis, data science, AI, automation and more. It is supported by a large community of developers and its wealth of specific modules and libraries for data processing and communication with databases. It is an easy language that is quick to write and deploy, allowing prototypes to be quickly created and tested. Between all provided libraries, the most used for this project is "mysql-connector-python", provided by Oracle[1], which implements for communication with MySQL database in Python scripts. Python is a cross-platform programming language, which means that it can be used on many different operating systems, including Windows, MacOS and Linux. This reduces the problems of operating system specificities. Overall, Python is a popular and powerful choice for developing automated data management tools, offering a unique combination of ease of use, performance and flexibility.

All the elements involved in pipeline functioning are discussed below.

Grabber

The grabber module part of system category handles the recording of files destined to be pre-processed by the pipeline. To do this, grabber extracts needed information from data files, and inputs it into the database. In particular, grabber creates an entry in table Recordings4.3 with the path to the data file to process within the pipeline. Corresponding entry in table Metadata4.4 is created, containing acquisition information. Finally, an entry is created in table Processings4.5, the table with the step name filled with value "register", and status with value "done". The overview of grabber module is presented in Figure 4.11 and the function algorithm doing the operation is given in Annexe 6.1, Figure A.1.

Status updater

The status updater module is part of the system category, which links the different steps in the pipeline. To do this, it interacts with the Processings table by identifying, modifying and adding entries. In particular, status updater, from the provided ordered list of processing steps to apply to the data files, will search in Processings table4.5 for entries for a given step with a status "done". If there are eligible file processes, status updater adds an entry for the next step with a "ready" status and the ID of the previous step. Status updater modifies the entry for the step it identified during its scan by specifying the ID of the step it has just created. The overview of status updater module is presented in Figure 4.12 and the function algorithm doing the operation is given in

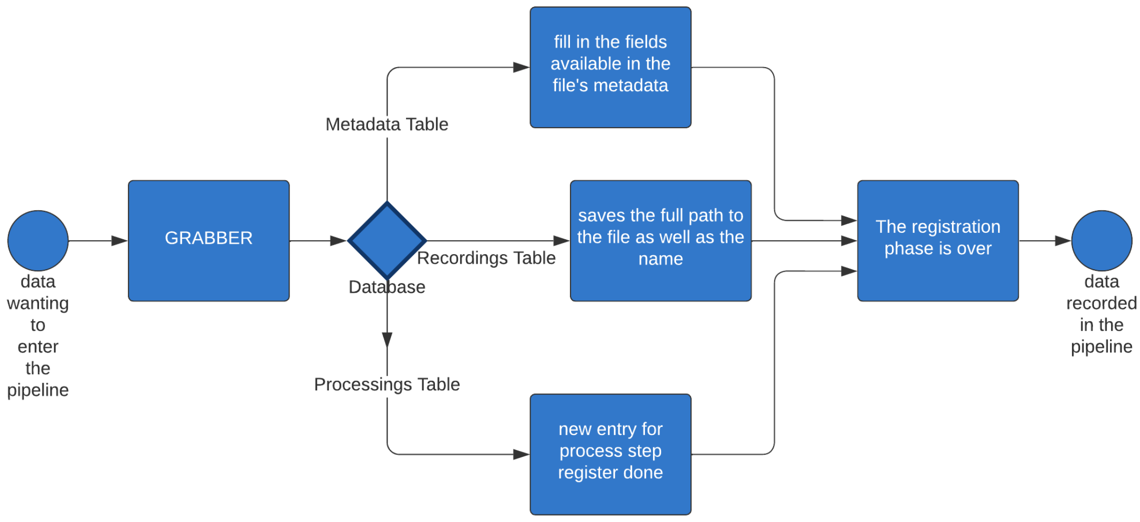

fill in the fields available in the file's metadata
Metadata Table
GRABBER
Recordings Table
Database
saves the full path to the file as well as the name
The registration phase is over
data wanting to enter the pipeline
data recorded in the pipeline
Processings Table
new entry for process step register done

Figure 4.11: Overview of grabber module operation, it handles data which has the property of wanting to enter the pipeline, and extracts from its metadata the information needed to populate the Metadata4.4, Recordings4.3 and Processings4.5 tables. Data leaving this module have the property of being recorded into the pipeline, meaning have the register step done.

Annexe 6.1, Figure A.2.

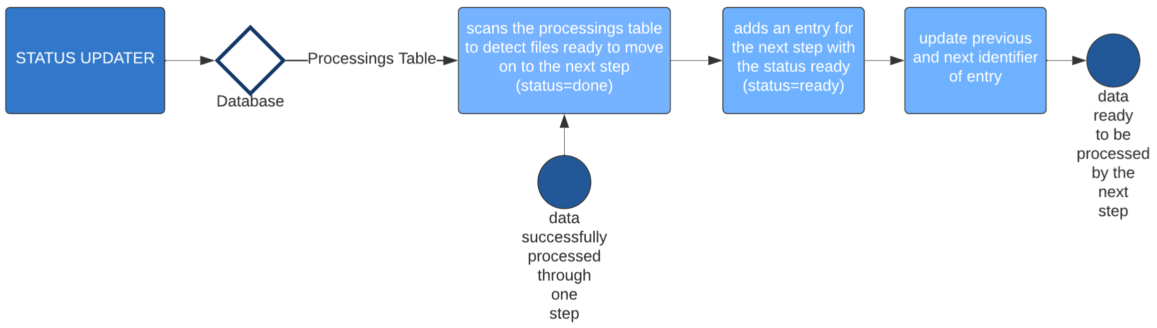

STATUS UPDATER
Database
Processings Table
scans the processings table to detect files ready to move on to the next step (status=done)
adds an entry for the next step with the status ready (status=ready)
update previous and next identifier of entry
data ready to be processed by the next step
data successfully processed through one step

Figure 4.12: Overview of status updater module operation, it handles data which has the property of having been successfully processed by a step of the pipeline, and prepare them for the next step in table Processings4.5. Data leaving this module have the property of being ready to be processed by a specific step (the step following the one in which they were identified).

Data processing Steps

The data processing steps module encompasses all the steps used to process data. This generality is possible due to the interaction that each step has with the database is identical, it is the activity that it performs on the data file that makes them different. To do this, each step interacts with the Processings table4.5 by identifying and updating entries according to status of the process. In particular, when step is ON, it first searches in the Processings table4.5 for entries with step fields corresponding to its name, and status “ready”. For found entries, step updates the status to “running” and executes the data manipulation function. Once data is processed, step updates the status to “done”.

The data processing steps are implemented utilising class inheritance principle[28]: the parent class BaseStep[2] defines the interactions with the database, while a daughter class of steps define only the name of step and a function that manipulates the data provided by path to the data file. Thus, defining a new processing step will require only an implementation of data manipulation function. For example, backup step is defined as follows:

```
class BackupStep(BaseStep):
def __init__(self, cnx, project):
    BaseStep.__init__(self, cnx, project)
    self.step_name = "backup"
    self.proj_pth = projects_functions.get_project_paths(
        self.cursor, project, "backup")

    if self.proj_pth is None:
        self.proj_pth = self._unclaimed_proj_pth_backup

def step(self, source, dest):
    dest = os.path.join(dest, '.')
    shutil.copy(source, dest)
    logging.debug("backup done to %s", dest)
```

The overview of a general steps operation is presented in Figure 4.13 and steps class inheritance implementation is given in Annex 6.1, Figure A.3.

4.1.4 Secondary elements

The processing pipeline also includes secondary elements not strictly necessary for pipeline functioning. These elements include GUI, email and log service.

Graphical User Interface

The pipeline provides a rudimentary GUI, created with CakePHP[5] as a proof of concept. This GUI is simply a tool for visualising each table in the database, with a few additional functionalities allowing modification of entries in tables. An example of a view of table Processings is presented in Figure 4.14. CakePHP has been selected for its capability

[2]implementation in Steps/BaseStep.py

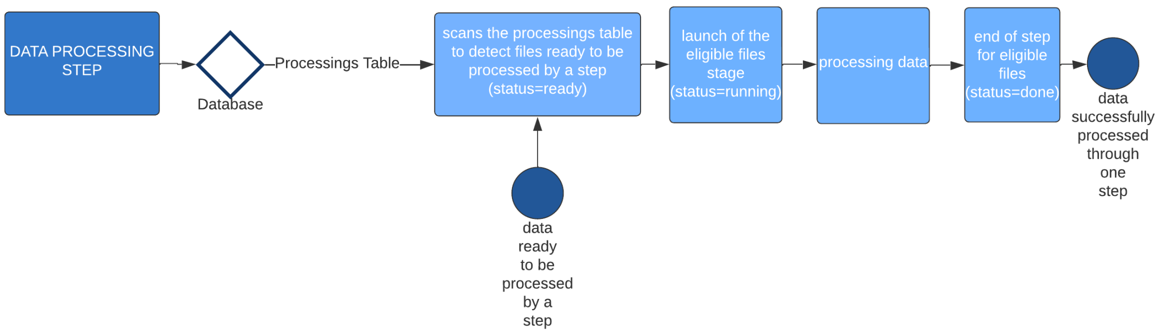
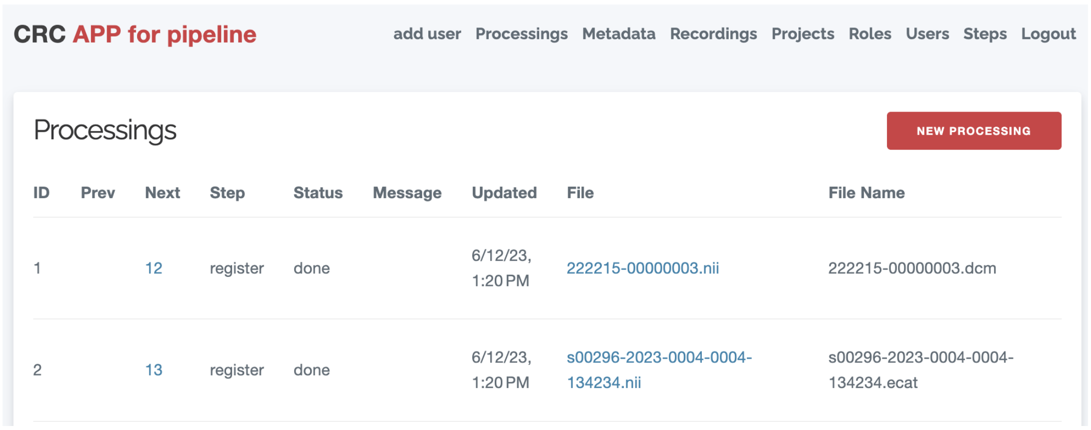

DATA PROCESSING STEP
Database
Processings Table
scans the processings table to detect files ready to be processed by a step (status=ready)
launch of the eligible files stage (status=running)
processing data
end of step for eligible files (status=done)
data successfully processed through one step
data ready to be processed by a step

Figure 4.13: Overview of a general step operation, it handles data which has the property of being ready to be processed by this step, and processed them, while updating information on the status of the process in table Processings4.5. Data leaving this step have the property of being successfully processed by the step involved.


CRC APP for pipeline

Processings

ID	Prev	Next	Step	Status	Message	Updated	File	File Name
1		12	register	done		6/12/23, 1:20 PM	222215-00000003.nii	222215-00000003.dcm
2		13	register	done		6/12/23, 1:20 PM	s00296-2023-0004-0004-134234.nii	s00296-2023-0004-0004-134234.ecat

Figure 4.14: Screenshot in GUI for table Processings

to quickly generate the representation of the processing pipeline's related database. The CMS of this rudimentary GUI can be found at [20].

Email service

The email service has been imagined, but never implemented, as a means of communication with users outside the GUI. A possible implementation could be a separate program, which regularly scans the database, and sends the user a resume of found information. For example, data that generated errors, or weekly report of processed data.

Log files

The log file for this project is a file containing a chronological record of the events, actions and errors that occur within the body of the pipeline.

To create the log files, the "logging" module[29] was used, a library built into Python that provides an integrated solution for managing logs in applications with several advantages listed below.

- It allows log levels (DEBUG, INFO, WARNING, ERROR, etc.) that can be easily configured to filter information according to severity.
- It allows additional information to be added to the logs, such as the timestamp, module name, line number, etc., to make it easier to track events.
- It facilitates the centralisation and aggregation of logs from different parts of the application.
- It allows controlling the verbosity of logs, which is useful when developing, debugging or deploying the application.
- It allows this library to be utilised for programs using threads and parallelism.

4.1.5 Folders and subfolders involved in the data file processing pipeline

To function correctly, the pipeline needs three folders in which the data files travel. These are the income_data, processings_info and conversion_dir folders, which will be described and explained briefly below.

income_data folder contains the files intended to be processed by the pipeline.

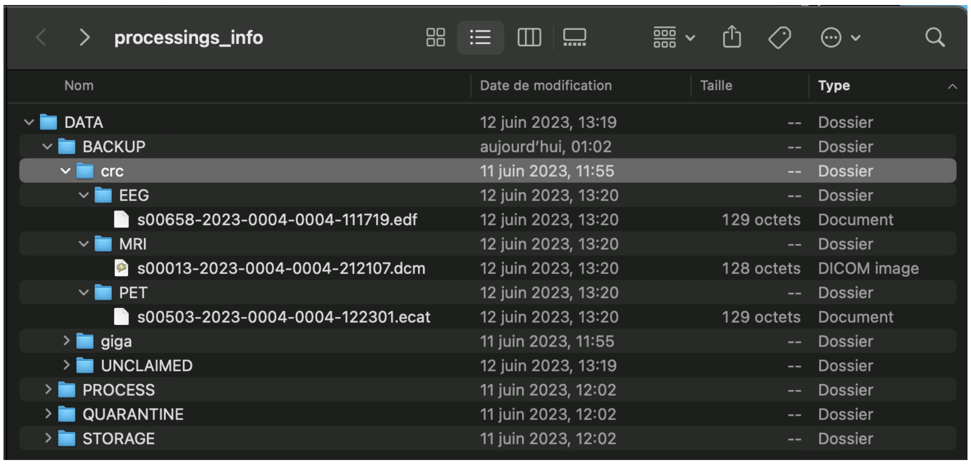
processings_info folder contains four subfolders: BACKUP, PROCESS, QUARANTINE and STORAGE. Each subfolder has a specific role, which can be deduced from its name. Within these subfolders, there are sub-sub-folders corresponding to the projects. Within these sub-sub-folders, there are sub-sub-sub-folders corresponding to the data modality. Within these sub-sub-sub-folders, there is finally the data file. This infrastructure is illustrated in Figure 4.15.

conversion_dir temporarily contains the transitory files needed for certain data conversion, as well as the converted file before it is moved.

All these directories path are hard-coded in a confidential_info folder containing a JSON file that stores confidential information, as directories path and connection information to the database, required for the system to function correctly.

4.2 Implementation of the pipeline system

The implementation of the pipeline can be found at [21] with state of pipeline corresponding to this thesis marked by tag "alpha_1.0".


processings_info

Nom	Date de modification	Taille	Type
DATA	12 juin 2023, 13:19	--	Dossier
BACKUP	aujourd'hui, 01:02	--	Dossier
crc	11 juin 2023, 11:55	--	Dossier
EEG	12 juin 2023, 13:20	--	Dossier
s00658-2023-0004-0004-111719.edf	12 juin 2023, 13:20	129 octets	Document
MRI	12 juin 2023, 13:20	--	Dossier
s00013-2023-0004-0004-212107.dcm	12 juin 2023, 13:20	128 octets	DICOM image
PET	12 juin 2023, 13:20	--	Dossier
s00503-2023-0004-0004-122301.ecat	12 juin 2023, 13:20	129 octets	Document
giga	11 juin 2023, 11:55	--	Dossier
UNCLAIMED	12 juin 2023, 13:19	--	Dossier
PROCESS	11 juin 2023, 12:02	--	Dossier
QUARANTINE	11 juin 2023, 12:02	--	Dossier
STORAGE	11 juin 2023, 12:02	--	Dossier

Figure 4.15: Screenshot of processings_info folder branched infrastructure

The infrastructure of the crc_preprocessing_pipeline project folder is organised as follows, in the following sub-folders:

admin contains Python scripts related to database management, project configuration, and user interaction with error files and processing steps. A pipeline reset script for testing is also included.

fileIO contains functions allowing to extract metadata from data to process.

pipeline_interaction contains functions implementing "grabber" and "status updater" modules.

sql_interaction contains utilities and reusable functions for interacting with each table in the database via SQL queries.

steps contains functions implementing each data pre-processing step available in the pipeline.

test contains unit test scripts that have been developed throughout the implementation of the project.

tools contains utilities for loading configuration files and generating fictitious data for testing and development purposes.

There are also root files that contain the functions that use all the functionalities present in the subfolders, enabling the pipeline to function in practice enabling the project objectives to be achieved.

4.2.1 Key data processing pipeline algorithms

In this subsection, the essential algorithms that play a central role in the data pipeline will be discussed. To achieve this, three major algorithms that are at the heart of the system will be presented, focusing on their operation and their specific features. Next, the way in which these algorithms fit together harmoniously in the main file will be examined, while exploring the overall algorithm that governs the entire process. This approach will provide better understanding of the interaction and contribution of each of the algorithms in achieving the objectives of the data pipeline. The algorithms involved are those behind the grabber, status updater and steps modules, as well as the main file can be seen in Appendix 6.1.

Chapter 5

Results and discussion by pipeline testing

After implementing as many features as possible, it is now time to demonstrate the actual functionality of the pipeline and assess whether the current implementation meets all expectations. This discussion will cover the pipeline's functionality, any defects or issues encountered, as well as its extensibility.

5.1 The initialisation of the pipeline

Before testing the functioning of the pipeline, the database must first be created, and few projects and toggle steps must be inserted into the database.

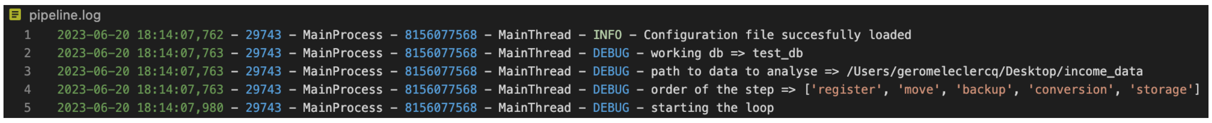
The database is initialised using script "db_create.py", which provide a command line interface to the "admin.db_management.create_tables" function. After execution of "create_tables" function, all necessary tables are properly initialised.

The toggle steps are initialised using script "step_add.py", which provide a command line interface to the "admin.step.insert_step" function. After execution of "insert_step" function, toggle step is properly initialised with status "ON". The same process needs to be performed with all toggle steps in order to ensure complete initialisation at this level.

The projects are initialised using script "project_add.py", which provide a command line interface to the "admin.project.create_project" function. After execution of "create_project" function, project is properly initialised with all the needed directories. The same process needs to be performed with all projects in order to ensure complete initialisation at this level. For the demonstration, two projects have been created namely "giga" and "crc".

5.2 Functioning of the pipeline

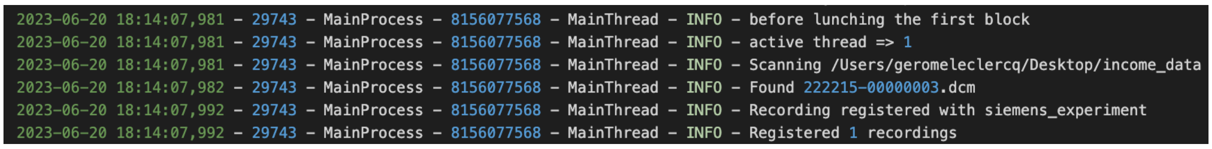
The initialised pipeline is started by executing controlling script “global_run_thread.py” which implements the main loops (Figure A.4) of the pipeline. The logging messages appearing in the dedicated log file (see in Figure 5.1) show that pipeline is indeed started.



```
pipeline.log
2023-06-20 18:14:07,762 - 29743 - MainProcess - 8156077568 - MainThread - INFO - Configuration file succesfully loaded
2023-06-20 18:14:07,763 - 29743 - MainProcess - 8156077568 - MainThread - DEBUG - working db => test_db
2023-06-20 18:14:07,763 - 29743 - MainProcess - 8156077568 - MainThread - DEBUG - path to data to analyse => /Users/geromeleclercq/Desktop/income_data
2023-06-20 18:14:07,763 - 29743 - MainProcess - 8156077568 - MainThread - DEBUG - order of the step => ['register', 'move', 'backup', 'conversion', 'storage']
2023-06-20 18:14:07,980 - 29743 - MainProcess - 8156077568 - MainThread - DEBUG - starting the loop
```

Figure 5.1: log messages associated to the start of pipeline operation

The pipeline scans the input folder waiting for files to pre-processed. As soon as files are found, they are registered in the database so that they can start to be processed by pipeline. This is shown by log messages (see in Figure 5.2).



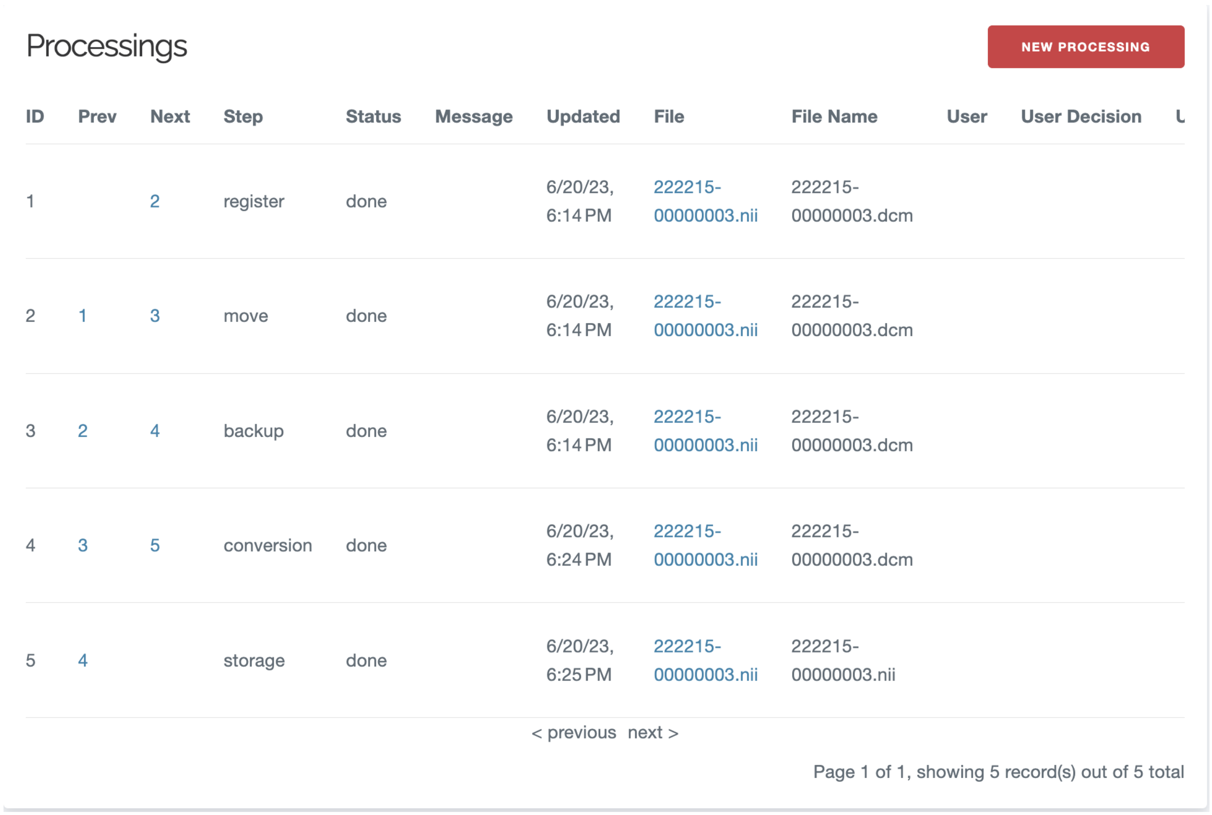
```
2023-06-20 18:14:07,981 - 29743 - MainProcess - 8156077568 - MainThread - INFO - before lunching the first block
2023-06-20 18:14:07,981 - 29743 - MainProcess - 8156077568 - MainThread - INFO - active thread => 1
2023-06-20 18:14:07,981 - 29743 - MainProcess - 8156077568 - MainThread - INFO - Scanning /Users/geromeleclercq/Desktop/income_data
2023-06-20 18:14:07,982 - 29743 - MainProcess - 8156077568 - MainThread - INFO - Found 222215-00000003.dcm
2023-06-20 18:14:07,992 - 29743 - MainProcess - 8156077568 - MainThread - INFO - Recording registered with siemens_experiment
2023-06-20 18:14:07,992 - 29743 - MainProcess - 8156077568 - MainThread - INFO - Registered 1 recordings
```

Figure 5.2: log messages associated to the grabber operation

In the current pipeline version, the processing steps consist of “move”, “backup”, “conversion” and “storage”. Each file that is processed by these steps have entries in “Processings” table corresponding to processes undergo by the file. Therefore, this table contains the full history of the file, as seen in Figure 5.3.

- MOVE step represents The movement of registered data from the IN folder to the PROCESSINGS folder.
- BACKUP step represents the copy of data in the PROCESSINGS folder to the BACKUP folder.
- CONVERSION step converts the data in the PROCESSINGS folder into the standard format used by CRC, depending on the type of data.
- STORAGE step saves (moves) the converted data in the PROCESSINGS folder to the STORAGE folder.

The pipeline offers some levels of interactivity. First, the execution of togged steps can be stopped by setting status to “OFF” in the Steps table. Once a step is “OFF” and when data arrives to the stopped step, they remain in status “ready” until the step is turned “ON”, when the step is activated, the data is processed normally.



Processings

NEW PROCESSING

ID	Prev	Next	Step	Status	Message	Updated	File	File Name	User	User Decision	U
1		2	register	done		6/20/23, 6:14 PM	222215-00000003.nii	222215-00000003.dcm			
2	1	3	move	done		6/20/23, 6:14 PM	222215-00000003.nii	222215-00000003.dcm			
3	2	4	backup	done		6/20/23, 6:14 PM	222215-00000003.nii	222215-00000003.dcm			
4	3	5	conversion	done		6/20/23, 6:24 PM	222215-00000003.nii	222215-00000003.dcm			
5	4		storage	done		6/20/23, 6:25 PM	222215-00000003.nii	222215-00000003.nii			

< previous next >

Page 1 of 1, showing 5 record(s) out of 5 total

Figure 5.3: GUI view on Processings table to demonstrate file history, and demonstrate pipeline works with MRI data

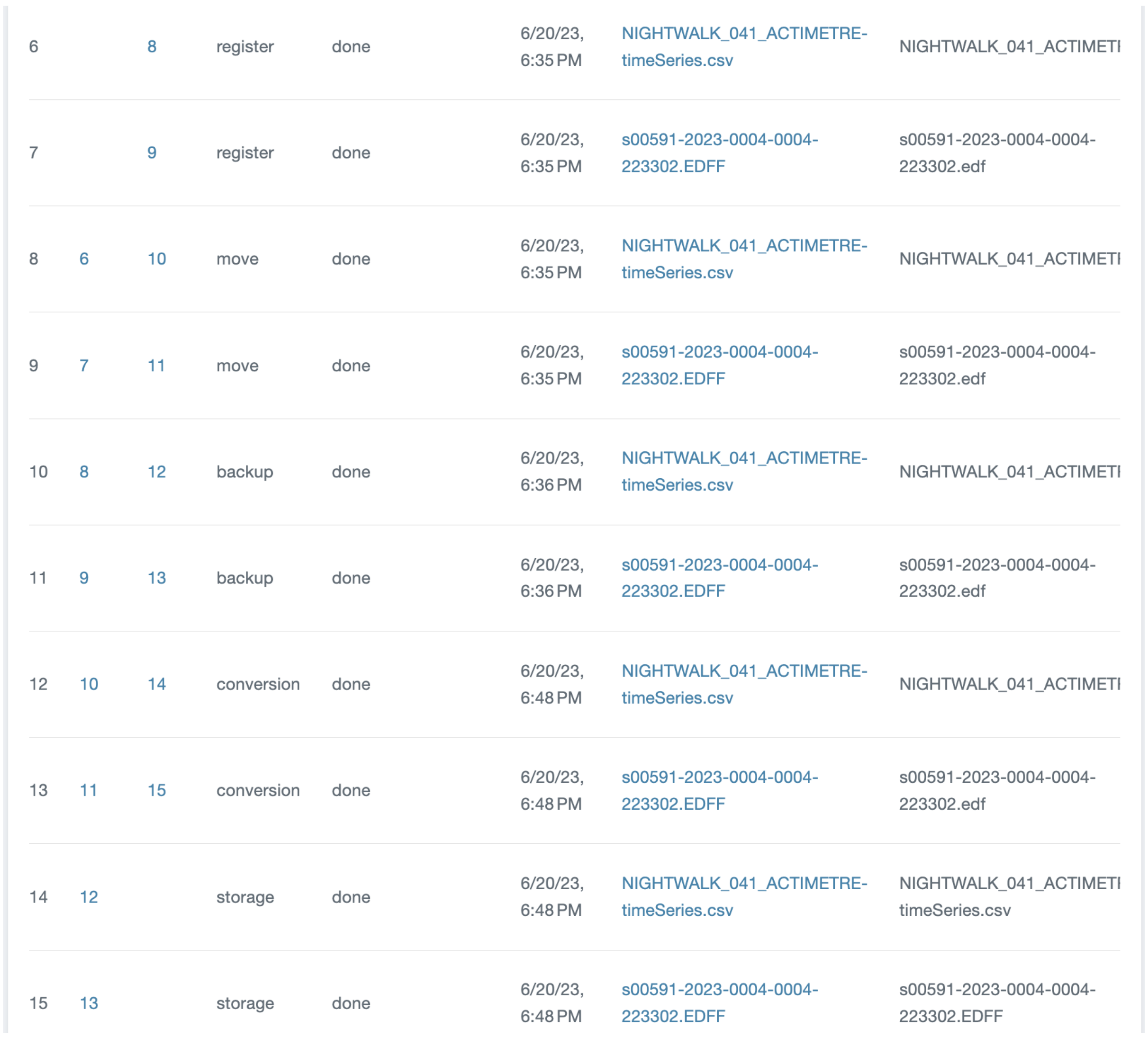
The second possible intervention is at the level of erroneous files. It represents an endeavour to operationalise user actions on individual data files, aiming to enable more effective user control and management. The anomaly which is an "error" status will be placed in quarantine, in other words, moved from the PROCESSINGS folder to the QUARANTINE folder.

As a result, the user can influence not only the operation of the pipeline stages, but also the processing of individual files.

The pipeline was implemented to process dummy data, DICOM images and actigraphy recordings[1]. Screenshots on Figures 5.3, 5.4 show proof that the pipeline process all of these types of files.

It can be concluded that the prototype for the pipeline performs as expected. It is reasonably data-type independent, constantly checks for new data to processes, and process it as soon as it arrives. The history of processed data is stored in the database, and ensures the traceability of the data.

[1]Actigraphy compatibility was implemented as a test of pipeline extensibility, and will be discussed in Section 5.5



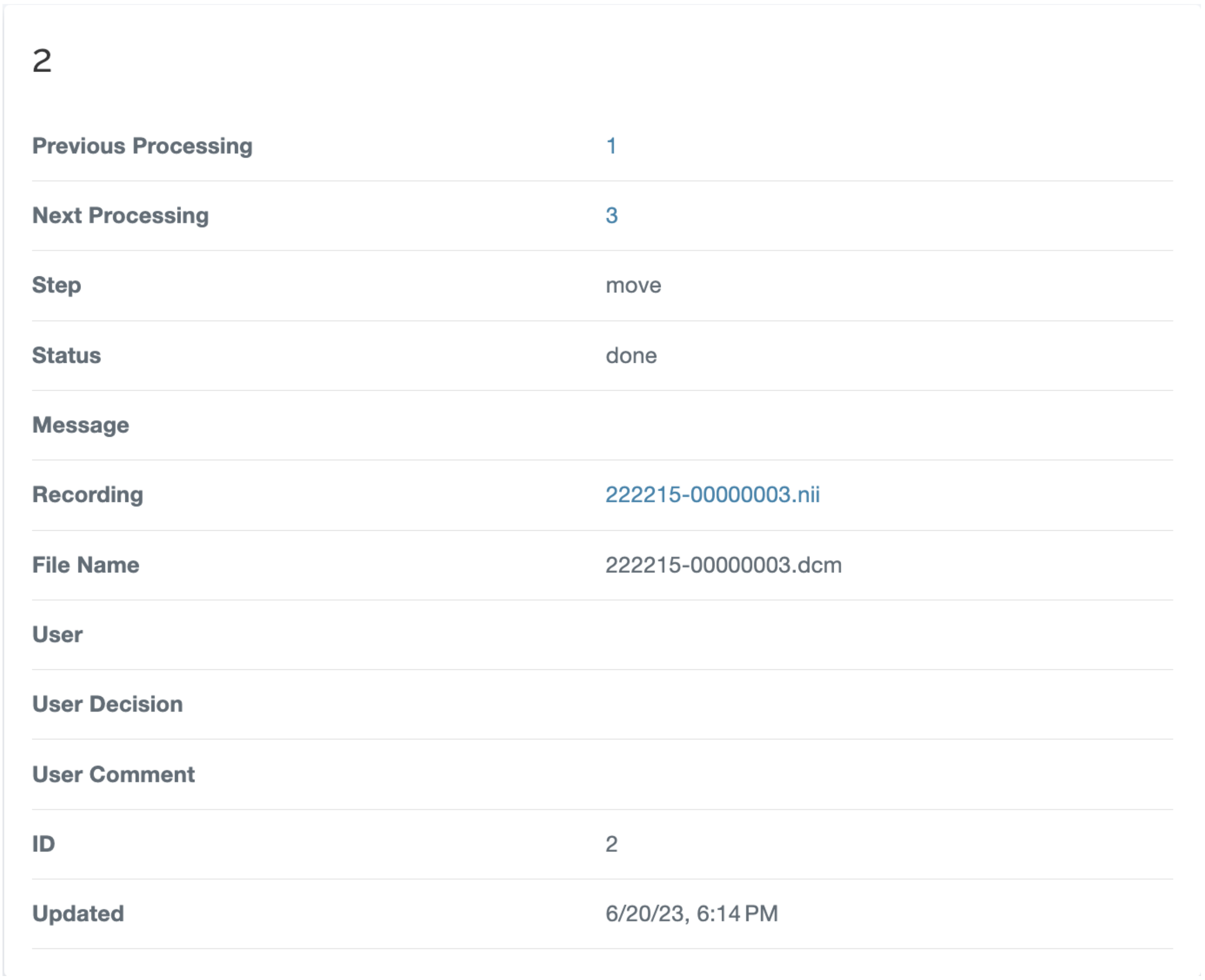
6		8	register	done	6/20/23, 6:35 PM	NIGHTWALK_041_ACTIMETRE-timeSeries.csv	NIGHTWALK_041_ACTIMETI
7		9	register	done	6/20/23, 6:35 PM	s00591-2023-0004-0004-223302.EDFF	s00591-2023-0004-0004-223302.edf
8	6	10	move	done	6/20/23, 6:35 PM	NIGHTWALK_041_ACTIMETRE-timeSeries.csv	NIGHTWALK_041_ACTIMETI
9	7	11	move	done	6/20/23, 6:35 PM	s00591-2023-0004-0004-223302.EDFF	s00591-2023-0004-0004-223302.edf
10	8	12	backup	done	6/20/23, 6:36 PM	NIGHTWALK_041_ACTIMETRE-timeSeries.csv	NIGHTWALK_041_ACTIMETI
11	9	13	backup	done	6/20/23, 6:36 PM	s00591-2023-0004-0004-223302.EDFF	s00591-2023-0004-0004-223302.edf
12	10	14	conversion	done	6/20/23, 6:48 PM	NIGHTWALK_041_ACTIMETRE-timeSeries.csv	NIGHTWALK_041_ACTIMETI
13	11	15	conversion	done	6/20/23, 6:48 PM	s00591-2023-0004-0004-223302.EDFF	s00591-2023-0004-0004-223302.edf
14	12		storage	done	6/20/23, 6:48 PM	NIGHTWALK_041_ACTIMETRE-timeSeries.csv	NIGHTWALK_041_ACTIMETI timeSeries.csv
15	13		storage	done	6/20/23, 6:48 PM	s00591-2023-0004-0004-223302.EDFF	s00591-2023-0004-0004-223302.EDFF

Figure 5.4: GUI view on Processings table to demonstrate file history, and demonstrate pipeline works with dummy and actigraphy data

5.3 Graphical user interface

Rudimentary Graphical User Interface (GUI) has been developed in parallel with the pipeline, allowing graphical representation of database tables. Furthermore, it allows all the information in a record to be grouped together using a view action. An example of table representation, as well as view action on one of the entries can be found in Figure 5.5.

The development of the GUI was not the main goal, but can be used as a future project at CRC.


2

Previous Processing	1
Next Processing	3
Step	move
Status	done
Message	
Recording	222215-00000003.nii
File Name	222215-00000003.dcm
User	
User Decision	
User Comment	
ID	2
Updated	6/20/23, 6:14 PM

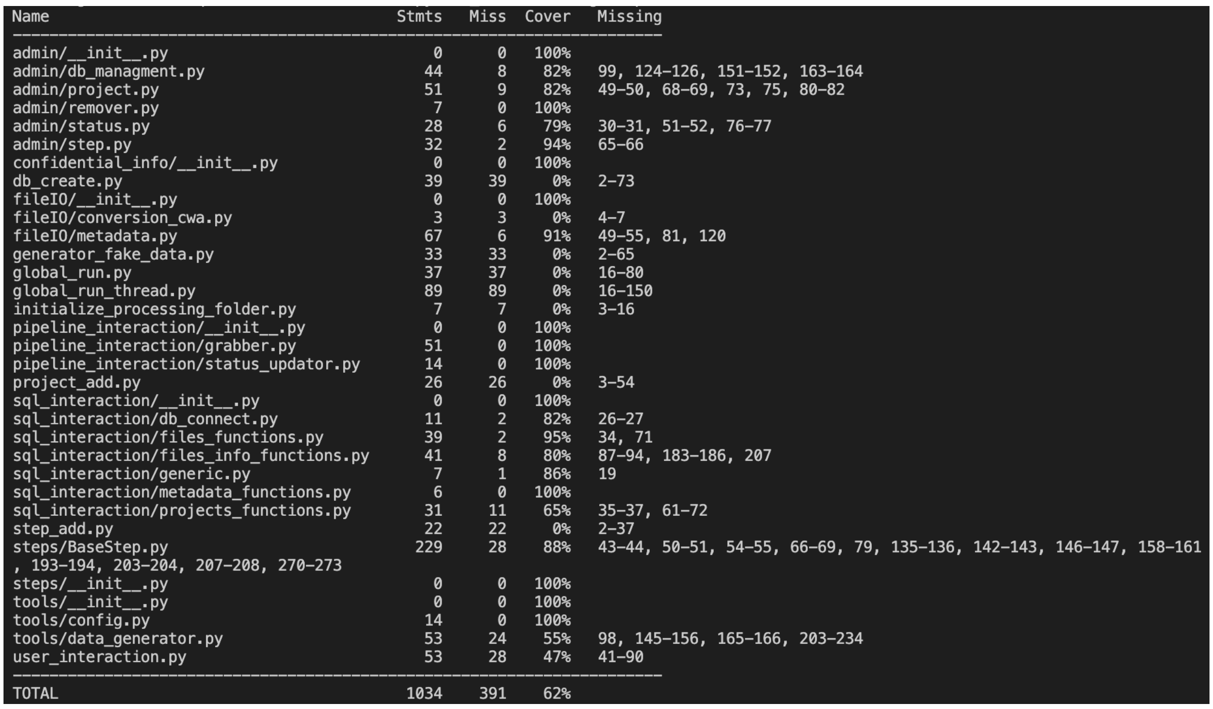
Figure 5.5: GUI view action on one processings record

5.4 Unit test of pipeline

Unit tests used in this project are small scripts that run specific pipeline functions with predefined attributes that return an expected value. In case of one unit test, if the value returned is not what was expected, this indicates that the function tested is no longer doing what it was supposed to do.

So, in parallel with the implementation of the pipeline, unit tests have been developed to ensure the quality, maintainability, scalability, and confidence of the code. They enable the reduction of errors and bugs in the code throughout development.

Code coverage has been quantified using the coverage.py tool which measures the code coverage of Python programs. The unit tests developed during the project ensure code coverage of 62%, which means that 62% of the source code in the pipeline is covered by unit tests. The coverage report is in Figure 5.6.



```
Name                                      Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------
admin/__init__.py                             0      0   100%
admin/db_managment.py                        44      8    82%   99, 124-126, 151-152, 163-164
admin/project.py                             51      9    82%   49-50, 68-69, 73, 75, 80-82
admin/remover.py                              7      0   100%
admin/status.py                              28      6    79%   30-31, 51-52, 76-77
admin/step.py                                32      2    94%   65-66
confidential_info/__init__.py                 0      0   100%
db_create.py                                 39     39     0%   2-73
fileIO/__init__.py                            0      0   100%
fileIO/conversion_cwa.py                      3      3     0%   4-7
fileIO/metadata.py                           67      6    91%   49-55, 81, 120
generator_fake_data.py                       33     33     0%   2-65
global_run.py                                37     37     0%   16-80
global_run_thread.py                         89     89     0%   16-150
initialize_processing_folder.py               7      7     0%   3-16
pipeline_interaction/__init__.py              0      0   100%
pipeline_interaction/grabber.py              51      0   100%
pipeline_interaction/status_updator.py       14      0   100%
project_add.py                               26     26     0%   3-54
sql_interaction/__init__.py                   0      0   100%
sql_interaction/db_connect.py                11      2    82%   26-27
sql_interaction/files_functions.py           39      2    95%   34, 71
sql_interaction/files_info_functions.py      41      8    80%   87-94, 183-186, 207
sql_interaction/generic.py                    7      1    86%   19
sql_interaction/metadata_functions.py         6      0   100%
sql_interaction/projects_functions.py        31     11    65%   35-37, 61-72
step_add.py                                  22     22     0%   2-37
steps/BaseStep.py                           229     28    88%   43-44, 50-51, 54-55, 66-69, 79, 135-136, 142-143, 146-147, 158-161
, 193-194, 203-204, 207-208, 270-273
steps/__init__.py                             0      0   100%
tools/__init__.py                             0      0   100%
tools/config.py                              14      0   100%
tools/data_generator.py                      53     24    55%   98, 145-156, 165-166, 203-234
user_interaction.py                          53     28    47%   41-90
-----------------------------------------------------------------------
TOTAL                                      1034    391    62%
```

Figure 5.6: source codes coverage report

5.5 Testing extensiveness of pipeline

The pipeline was designed to be easily extendable to new data formats and new processing steps.

Extensiveness to new steps is ensured by the implementation of the parent class BaseStep (see in Figure A.3). The implementation of a new step is easily performed by creating a new child class and implementing just the function that performs the data treatment, since all interaction with the pipeline has been implemented in the parent class. Once implemented, the step class must be included to "basesteps_2_run" dictionary, which serves as a library of steps. The insertion is therefore natural and requires no major adaptation to the pipeline source code, just code additions following the same implementation logic.

The pipeline was initially implemented and operated solely with fake data, which provided metadata and extensions of medical images common to the CRC. Thereafter, the pipeline was extended to DICOM (MRI) files in order to prove that it was possible to input real data. To achieve this, it was necessary to find a way of reading the metadata in a DICOM file and find a tool able to convert DICOM files into NIFTI files. The tool should be external as it is launched from a subprocess, to ensure any tool. The tool used in this prototype is dcmstack software[23].

Second, the pipeline was extended for CWA files (Actigraphy) to test how easy was to extend to new data formats. The process of extension is the same as used for extension to DICOM formats. The function "cwa_info" from "cwa_metadata" package[15] has been used to extract metadata, "accelerometer" software[36] has been used to convert to CSV files. In total, the extension took three days, with the majority of time used to search and learn the external tools.

5.6 Discussion of the current implementation

The pipeline, while useful and functional, has a few imperfections and is not without flaws.

5.6.1 Planned but not implemented elements

Several features have been planned but not implemented due to time restriction, the list below represents them.

- Email service
- More advanced GUI (search bar, web page filters, and tailored pages to display complete history of a file)
- True parallelism in main function. To implement true parallelisation, features such as processes or asynchronous tasks should have been considered. Processes allow code to be executed concurrently using multiple processor cores, while asynchronous tasks allow non-blocking operations to be managed concurrently. By using processes or asynchronous tasks, it would have been possible to start pipeline stages in parallel, allowing several stages to run simultaneously without waiting for the others to finish. This can significantly improve pipeline performance and efficiency. It should be noted that the only stage that takes time is the conversion stage, which is currently not due to the pipeline but to the conversion software. This delay is only observed for actigraphy data.
- Exit conditions for the main function (capture the interrupt signal (SIGINT) generated by Ctrl+C). At the moment, the only way to stop the main function, which is an infinite loop, is via Ctrl+C. The plan was to add an exit condition to the main loop and capture the interrupt signal (SIGINT) generated by Ctrl+C. When Ctrl+C is used to interrupt the program, an interrupt signal is sent to the running process. It is possible to capture this signal by defining a signal handler that will be called when the interrupt signal is received.

  In order to implement this concept, it would have been necessary to begin by defining a signal handler function that would be invoked upon capturing the interrupt signal. It would have been possible to include any clean-up or close operations required before exiting the program. Next, the signal handler had to be associated with the interrupt signal using the signal library function "signal.signal(signal.SIGINT, signal_handler)". This associates the signal handler with SIGINT. This implementation would therefore tell the program to capture the interrupt signal. Finally, an exit condition should have been added to the main loop. In other words, replace the True condition in the main loop with an appropriate condition that would determine when the program should end. For example, using an exit_requested flag that would have been set when stopping the pipeline was desired.

Using this approach, when the Ctrl+C command is pressed, the interrupt signal is captured, and the signal handling function is called. In this way, the operations required to clean up or close the program correctly can be carried out, and then exits by calling sys.exit(0) or returning from the main function. This would have resulted in a cleaner exit from the program, ensuring that the final operations were performed before exiting.

5.6.2 Limitations and issues of current implementation

Several issues with the implementation of the pipeline can be identified. These issues, partially due to time constraints, particularly due to lack of experience, which can serve as guidelines for future development and may serve as a basis for refactoring.

This critical assessment will provide a better understanding of the challenges the pipeline will face and identify areas where improvements are required prior to its official deployment. A discussion will be held covering issues with the pipeline code, such as lack of functionality in general. It will highlight limitations that may hinder its smooth operation and issues that may affect its maintainability and the tool's scalability. By examining these aspects in depth, recommendations and prospects for improvement will be formulated to improve the current implementation of the pipeline.

Code

- Consistency and readability, when examining the code, some coding conventions are not always respected as too long lines in "global_run_thread.py". In addition, the names of variables and functions are not always sufficiently explicit, making it difficult to understand the role of each element. For example, in "sql_interaction" modules, several functions do not have an explicit name as "update_next".

- Code redundancy, the source code of this pipeline will need to be reviewed in order to eliminate redundant parts of the code through object-oriented programming with class inheritance. For example, in "fileIO.metadata.py" there is a lot of code redundancy.

- Poorly documented code, making it difficult to understand specific functionality and the interactions between different parts of the code. For example "project_add.py" does not have any commentary within the code.

By addressing these different aspects, it will be possible to significantly improve the quality, maintainability and comprehensibility of the pipeline code, making it easier to evolve and adapt to future needs. Consequently, the source code of this pipeline will need to be reviewed in order to eliminate redundant parts of the code through object-oriented programming with class inheritance. The code will also need to be further documented and restructured to make it more streamlined before deployment.

5.6.3 Missing functionality

The in-depth assessment of the pipeline implementation revealed several major shortcomings that compromise its ability to fully meet the needs and requirements of the project. These shortcomings relate to critical functionality that significantly limits the overall effectiveness and efficiency of the pipeline. In this section, these major gaps will be highlighted and will be explained why resolving them is essential to improving the pipeline's performance and usefulness.

Tailoring the Pipeline: Addressing Project and Team-Specific Requirements

The current pipeline does not support several lists of steps to be run, meaning that all data, regardless of its project, undergoes the same steps in the same order. Each project should be able to define its own list of steps to be performed on the data and in the order in which they are performed. This is vitally important for data processing and analysis, as not every project/team needs the same things. This gap limits the ability of the pipeline to be adapted to all CRC teams.

Lack of concrete user interaction on the pipeline

Another major shortcoming of the pipeline is the lack of user interaction which results in limited control over the pipeline stages and the data within the pipeline. Although there are ways of interacting with the pipeline, such as quarantining an erroneous file or activating/deactivating a pipeline stage, they are one, insufficient and two, unavailable to the user as they are launched via terminal commands. Users should be able to perform these two interactions and more, such as quarantining selected files and restarting stages on certain files in a user-friendly way. This shortcoming prevents the full potential of the pipeline from being exploited.

Lack of parallelism

The pipeline currently implements parallelism partially. The steps within one project are executed in separate threads, however, the iteration over projects is still sequential, and can be paralysed. Additionally, it can be beneficial to use subprocesses instead of threads[2].

5.6.4 Graphical user interface

The pipeline GUI has several major gaps in terms of crucial functionality, which has a significant impact on the user experience. Currently, every user has access to every user's data, which raises confidentiality and security issues. It is essential to put in place appropriate security mechanisms to restrict access to data to only authorised personnel, ensuring that each user can only see their own data.

In addition, the ability to delete data should be restricted or prohibited, as this can lead to the irreversible loss of important information. Rigorous access control must be put in place to ensure that only authorised individuals can carry out modifications on the data.

Another limitation of the current GUI is the visibility of all the columns in the database, even if some of them are useless for a given user. This can lead to information overload and make it difficult to view and interpret relevant data. It is important to implement display customisation features, allowing each user to select the columns that are relevant to their work and to hide unnecessary information.

In addition, image viewing is an essential functionality that is lacking in the current GUI. It is crucial to enable users to view images directly on the interface, providing a better understanding and visual analysis of the data.

To address these shortcomings, it is imperative to develop and integrate functionality into the pipeline GUI that enables appropriate access control, customisation of data display, and user-friendly image visualisation. By filling these gaps, the pipeline GUI will be more functional, secure and user-friendly, providing users with an enhanced and more productive experience.

[2]In python3 only one thread can be executed a given time[31]

Chapter 6

Conclusions and perspectives

In this chapter, the objectives of this thesis will be assessed. The main objective was to create a tool for pre-processing data acquired at the CRC. From this main objective, a multitude of sub-objectives emerged representing the functionalities that this tool, being a data pipeline, should have. These functionalities were basically the coupling of the tasks of a data manager applicable to the CRC, as well as the benefits of the pipeline concept, covering problems linked to data management. Among the most important were the automation, standardisation and centralisation of procedures and elements; the provision of infrastructure independent of the type of data but specific to each type; the output of analysis-ready data. To achieve this, a pipeline was created from scratch, incorporating several elements such as the body of the data processing pipeline, a database, and a rudimentary graphical user interface.

Having implemented as many pipeline elements as possible, it is reasonable to say that most of the objectives of this thesis have been achieved. The resulting tool goes beyond the proof-of-concept stage and can therefore be likened to an advanced prototype. In fact, the pipeline works on fictitious data but also with two real types of data, MRI and actigraphy (seen in Section 5.2). It covers the majority of CRC problems linked to data management, since it takes all the files present in the database and uses them to manage the data:

- The pipeline takes all the files present in an input folder, if the CRC acquisition machines automatically save the data in this folder, the data will automatically enter the pipeline. Ensuring automation and centralisation.

- All files entering the pipeline will have some of their information stored in a database. Ensuring centralisation of information.

- Each file is then pre-processed by the same software and scripts, with its progress through the pipeline and its history recorded in the database. Ensuring standardisation of the software used.

- Each file migrates to the same type of output, i.e., the same folder and subfolder classification infrastructure. Ensuring standardisation of directories.
- A graphical interface is available for viewing information about the data and more, stored in the database. Ensuring visualisation.

However, it is important to underline certain limitations of the present study. First, due to time and resource constraints, the resulting pipeline is only an advanced prototype. Consequently, it cannot yet be deployed on a large scale within the CRC. Furthermore, although several data modalities have been considered, not all data acquired at the CRC are covered by the pipeline, such as PET, EEG, and TMS. The compatibility of these modalities within the pipeline warrants further investigation in future work. Fortunately, the scalability of this pipeline, which has been intentionally designed to be highly extensible in terms of data modalities, will facilitate such studies. In addition, although there are five effective stages, they do not cover all the needs of CRC users, and others are needed, such as a stage for estimating data quality, and a stage capable of bidsifying the data. As far as the GUI is concerned, it is still very rudimentary and will be a job in itself. The pipeline itself also needs to be optimised so that there is real parallelism, real output conditions and, above all, more user interaction.

6.1 Perspectives

In the light of our results, a number of avenues for improving the pre-processing tool for data acquired at CRC have emerged. Here are a few suggestions:

- Extend compatibility to all CRC data modalities, including GIGA neurological data, in order to reach more users and cover more sectors.
- Extend the number of pre-processing steps to make it even easier for users.
- Develop a better graphical user interface, with all the standard functions required for user-friendliness.

These work perspectives pave the way for new implementations and will contribute to enhancing the tool.

Bibliography

[1] Oracle and/or its affiliates. *mysql-connector-python*. Version 8.0.31. URL: https://dev.mysql.com/doc/connector-python/en/.

[2] Abi Berger. "Positron emission tomography". In: *BMJ* 326.7404 (2003), p. 1449. ISSN: 0959-8138. DOI: 10.1136/bmj.326.7404.1449.

[3] Eric Bernhardsson, Elias Freider, et al. *Luigi*. Version 3.3.0. 2022. URL: https://luigi.readthedocs.io/en/stable/index.html.

[4] Brain products. *Specification of BrainVision Core Data Format 1.0*. URL: https://www.brainproducts.com/download/specification-of-brainvision-core-data-format-1-0/.

[5] Cake Software Foundation, Inc. *CakePHP*. Version 4.4.0. 2023. URL: https://cakephp.org/.

[6] Amit Chail et al. "Transcranial magnetic stimulation: A review of its evolution and current applications". In: *Industrial Psychiatry Journal* 27.2 (2018), pp. 172–180. ISSN: 0972-6748. DOI: 10.4103/ipj.ipj_88_18.

[7] Oracle Corporation. *Oracle database. What is data managment.* 2023. URL: https://www.oracle.com/database/what-is-data-management/ (visited on 05/10/2023).

[8] Data Format Working Group. *Nifti-2 Data Format*. en. URL: https://nifti.nimh.nih.gov/nifti-2 (visited on 05/10/2023).

[9] Council of the European Union European Parliament. "The protection of natural persons with regard to the processing of personal data and on the free movement of such data". In: *Official Journal of the European Union* L.119 (Apr. 27, 2016), pp. 1–88. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[10] *GIGA CRC in vivo imaging - About us*. 2023. URL: https://www.gigacrc.uliege.be/cms/c_4288614/en/about-us (visited on 06/22/2023).

[11] *GIGA Cyclotron - About us*. 2023. URL: https://www.gigacyclotron.uliege.be/cms/c_4491113/en/gigacyclotron-about-us (visited on 05/10/2023).

[12] *GIGA Cyclotron - Equipment.* 2023. URL: https://www.gigacyclotron.uliege.be/cms/c_4467083/en/gigacyclotron-equipment (visited on 05/10/2023).

[13] Krzysztof Gorgolewski et al. "Nipype: A Flexible, Lightweight and Extensible Neuroimaging Data Processing Framework in Python". In: *Frontiers in Neuroinformatics* 5 (2011). ISSN: 1662-5196. DOI: 10.3389/fninf.2011.00013.

[14] Vijay P.B. Grover et al. "Magnetic Resonance Imaging: Principles and Techniques: Lessons for Clinicians". In: *Journal of clinical and experimental hepatology* 5.3 (2015), pp. 246–255. DOI: 10.1016/j.jceh.2015.08.001.

[15] Yu Guan et al. *Open Movement.* 2014. URL: https://github.com/digitalinteraction/openmovement.

[16] Natus Medical Incorporated. *Embla RemLogic PSG Software.* 2023. URL: https://natus.com/products-services/embla-remlogic-software (visited on 06/18/2023).

[17] Kashif Iqbal. *CSV File Format.* en. Dec. 2019. URL: https://docs.fileformat.com/spreadsheet/csv/ (visited on 05/10/2023).

[18] Bob Kemp and Jesus Olivan. "European data format 'plus' (EDF+), an EDF alike standard format for the exchange of physiological data". In: *Clinical Neurophysiology* 114.9 (2003), pp. 1755–1761. ISSN: 1388-2457. DOI: https://doi.org/10.1016/S1388-2457(03)00123-8.

[19] J. Krishna and S. Mashaqi. "Actigraphy". In: *Encyclopedia of the Neurological Sciences (Second Edition).* Ed. by Michael J. Aminoff and Robert B. Daroff. Second Edition. Oxford: Academic Press, 2014, pp. 36–40. ISBN: 978-0-12-385158-1. DOI: https://doi.org/10.1016/B978-0-12-385157-4.00547-9.

[20] Gerome Leclercq. *cake_php_interface.* URL: https://gitlab.uliege.be/CyclotronResearchCentre/LocalResources/acquisitionpipelinewg/cake_php_interface.

[21] Gerome Leclercq and Nikita Beliy. *crc_preprocessing_pipeline.* Version alpha_1.0. 2023. URL: https://gitlab.uliege.be/CyclotronResearchCentre/LocalResources/acquisitionpipelinewg/crc_preprocessing_pipeline/-/tree/alpha_1.0?ref_type=tags.

[22] Erik K. St Louis et al. *Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants.* American Epilepsy Society, 2016. URL: https://www.ncbi.nlm.nih.gov/books/NBK390346/.

[23] Brendan Moloney. *dcmstack*. Version v0.9. URL: https://github.com/moloney/dcmstack.

[24] "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments". In: 3.160044 (1 2016). Ed. by Nature. DOI: doi.org/10.1038/sdata.2016.44.

[25] *Nexstim - Technology*. URL: https://www.nexstim.com/healthcare-professionals/technology (visited on 05/10/2023).

[26] Cambridge University Press. *Cambridge dictionary online. Meaning of data in English*. 2023. URL: https://dictionary.cambridge.org/dictionary/english/data (visited on 05/10/2023).

[27] NEMA PS3. *Digital Imaging and Communications in Medicine (DICOM) Standard*. en. 2023. URL: https://www.dicomstandard.org/current (visited on 05/10/2023).

[28] Python Software Foundation. *Classes*. URL: https://docs.python.org/3/tutorial/classes.html.

[29] Python Software Foundation. *logging — Logging facility for Python*. URL: https://docs.python.org/3.10/library/logging.html?highlight=logging#module-logging.

[30] Python Software Foundation. *Python*. Version 3.10.9. URL: https://www.python.org/.

[31] Python Software Foundation. *threading — Thread-based parallelism*. URL: https://docs.python.org/3.10/library/threading.html.

[32] The FIL Methods Group. *SPM12 Manual*. Chap. 4. URL: https://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf.

[33] The Wellcome Centre for Human Neuroimaging. *SPM - Statistical Parametric Mapping*. en. URL: https://www.fil.ion.ucl.ac.uk/spm/.

[34] Tim Van Mourik et al. *Porcupine*. Version 1.4.0. 2018. URL: https://timvanmourik.github.io/Porcupine/.

[35] Oikonen Vesa. *TPC - ECAT format*. 2020. URL: http://www.turkupetcentre.net/petanalysis/format_image_ecat.html.

[36] Rosemary Walmsley et al. "Reallocation of time between device-measured movement behaviours and risk of incident cardiovascular disease". In: *British journal of sports medicine* 56.18 (2022), pp. 1008–1017.

Appendix A

Algorithms of main functions

A.1 Main algorithms in data processing pipeline

In this annex, four algorithms are schematised representing:

Grabber implementation can be found in "pipeline_interaction/grabber.py". The algorithm behind "real_register" function is presented in Figure A.1.

Status_updater implementation can be found in "pipeline_interaction/status_updator.py". The algorithm behind "update_status" function is presented in Figure A.2.

BaseStep implementation is based on class inheritance and can be found in "steps/BaseStep.py". The class inheritance logic behind BaseStep is presented in Figure A.3.

Global_run_thread implementation can be found in "root/global_run_thread.py". The algorithm behind "step_runner", the function that serves as a master script to execute all steps, is presented in Figure A.4.

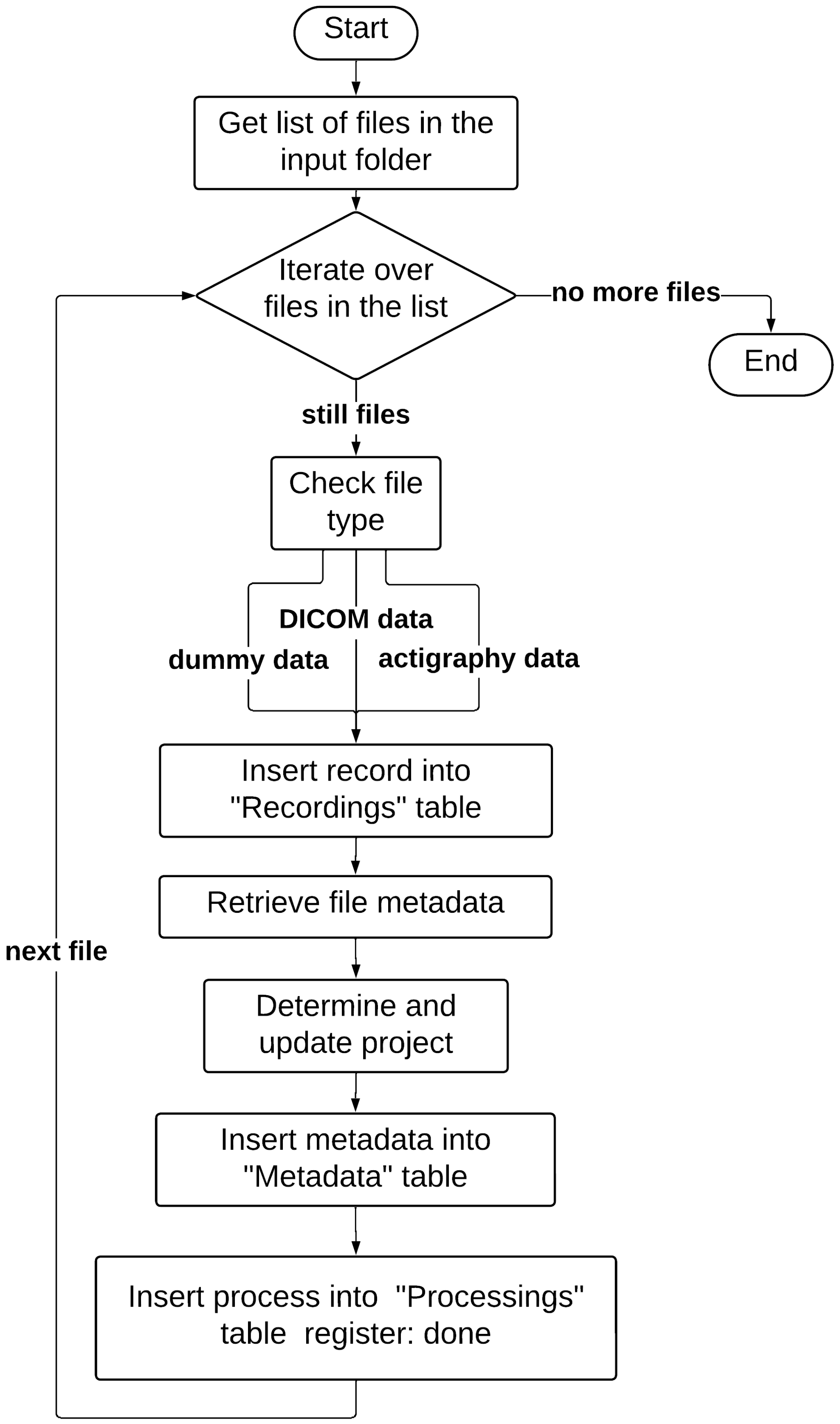

Start
Get list of files in the input folder
Iterate over files in the list
no more files
End
still files
Check file type
DICOM data
dummy data
actigraphy data
Insert record into "Recordings" table
Retrieve file metadata
next file
Determine and update project
Insert metadata into "Metadata" table
Insert process into "Processings" table register: done

Figure A.1: real_register function algorithm doing operation of grabber module

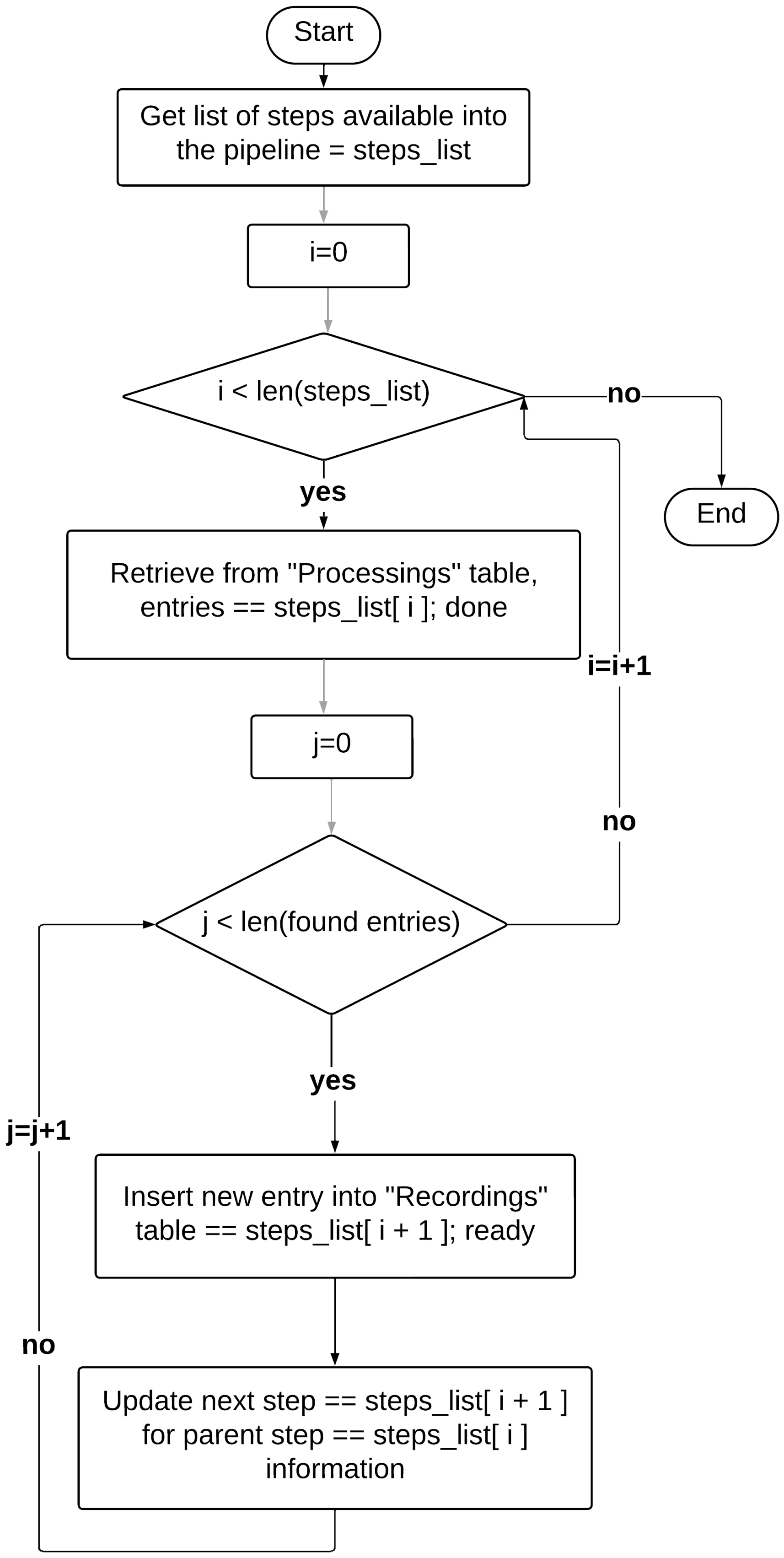

Start
Get list of steps available into the pipeline = steps_list
i=0
i < len(steps_list)
no
End
yes
Retrieve from "Processings" table, entries == steps_list[i]; done
i=i+1
j=0
no
j < len(found entries)
yes
j=j+1
Insert new entry into "Recordings" table == steps_list[i + 1]; ready
no
Update next step == steps_list[i + 1] for parent step == steps_list[i] information

Figure A.2: update_status function algorithm doing operation of status updator module

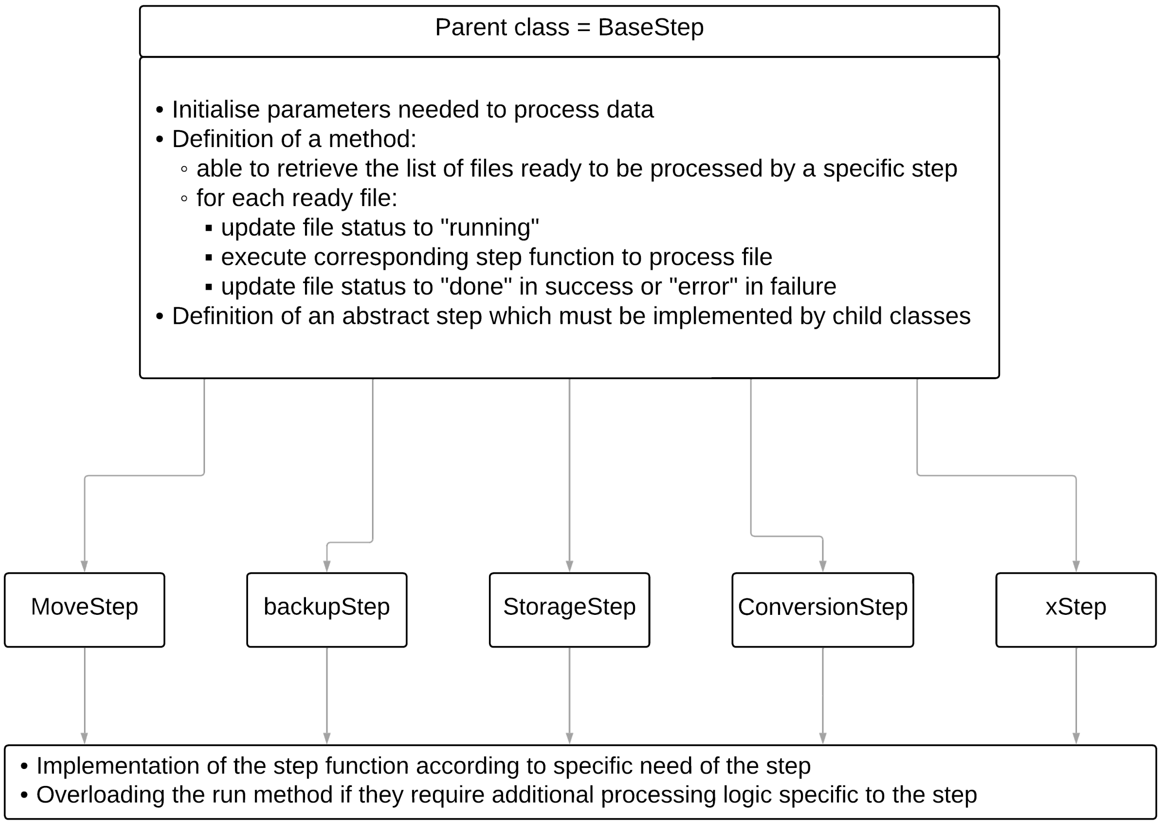
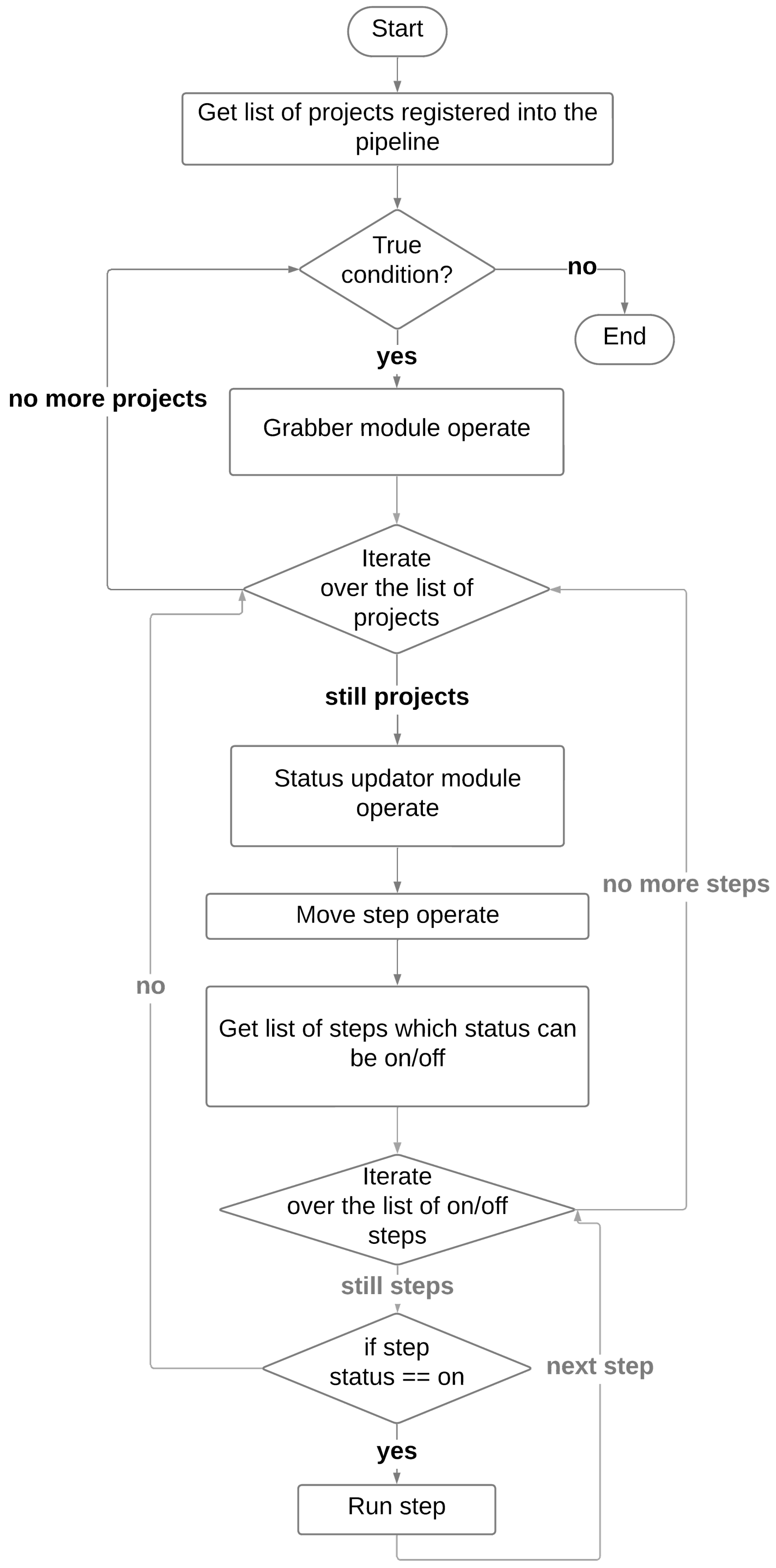

Parent class = BaseStep
• Initialise parameters needed to process data
• Definition of a method:
◦ able to retrieve the list of files ready to be processed by a specific step
◦ for each ready file:
▪ update file status to "running"
▪ execute corresponding step function to process file
▪ update file status to "done" in success or "error" in failure
• Definition of an abstract step which must be implemented by child classes
MoveStep
backupStep
StorageStep
ConversionStep
xStep
• Implementation of the step function according to specific need of the step
• Overloading the run method if they require additional processing logic specific to the step

Figure A.3: BaseStep implementation logic by class inheritance



Start
Get list of projects registered into the pipeline
True condition?
no
End
yes
no more projects
Grabber module operate
Iterate over the list of projects
still projects
Status updator module operate
no more steps
Move step operate
no
Get list of steps which status can be on/off
Iterate over the list of on/off steps
still steps
if step status == on
next step
yes
Run step

Figure A.4: step_runner function algorithm doing operation of master script