POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Generation of Synthetic Data From Digital Health Records

Tesi di Laurea Magistrale in
Computer Science and Engineering
Ingegneria Informatica

Author: **Tancredi Covioli**

Student ID: 944834
Advisor: Prof. Marco Gribaudo
Co-advisors: Dott. Ing. Tommaso Dolci
Academic Year: 2021-22

Abstract

The development of technologies such as cloud computing, IoT, and social networks has caused the amount of data generated daily to grow at an incredible rate. This rapid and unstoppable growth gave birth to the term Big Data. Big data has also arrived in the healthcare field, with the introduction of new tools producing massive amounts of structured and unstructured data. For this reason, medical institutions are moving towards a data-based healthcare, with the objective of leveraging this data to support clinical decision-making through suitable information systems. This has come with the need to evaluate their performance. One of the techniques commonly used for the performance assessment of such Systems is modeling, which consists in performing the evaluation on a model of the system under analysis, without the necessity of the implementation of such system to be completed. However, in order to make an adequate performance assessment of Big Data-centered systems, we need a diversity of volumes and speeds that, due to the sensitivity of data concerning healthcare, is not available. While in other fields this problem is usually solved through the use of synthetic data generators, in the field of healthcare these are few and not specialized in performance evaluation.

For this reason, in this work we focused on the creation of a synthetic data generator for the evaluation of the performance of a Big Data system model. The dataset used as a reference for the creation of the generator is MIMIC-III, which contains the digital health records of thousands of patients collected over a time span of multiple years. As a first step, an analysis of the dataset was performed, where multiple distribution fitting techniques (e.g., phase-type fitting) were adopted to model the temporal distribution of its data. After that, we developed a generator structured as a multi-module library to allow the customization of each component. Finally, we tested our generator by evaluating the performance of a simple model of a big data system in different scenarios. Through these experiments, we showed the granular control that the generator offers over the synthetic data produced, and the simplicity with which it can be adapted to different uses.

Keywords: Big Data, Performance Evaluation, Synthetic Data Generation, Healthcare

Sommario

Lo sviluppo di tecnologie come il cloud computing, l'IoT e i social network ha fatto sì che la quantità di dati generati quotidianamente crescesse a un ritmo incredibile, dando vita al fenomeno Big Data. Questi sono arrivati anche nel campo sanitario con i nuovi strumenti per il monitoraggio continuo dei pazienti. Per questo, le istituzioni mediche si stanno orientando verso un'assistenza sanitaria basata sui dati per supportare, tramite sistemi informatici adatti, il processo decisionale clinico. Ciò ha comportato la necessità di valutare le prestazioni di tali sistemi. Una delle tecniche utilizzate per la valutazione delle prestazioni è la modellazione, che consiste nell'eseguire la valutazione su un modello del sistema in analisi, senza la necessità che l'implementazione di tale sistema sia completa. Tuttavia, per effettuare un'adeguata valutazione di sistemi simili, è necessaria una diversità nella quantità e nella velocità d'arrivo dei dati che, a causa della sensibilità dei dati trattati nel dominio sanitario, non è disponibile. Mentre in altri settori questo problema viene solitamente risolto attraverso l'uso di generatori di dati sintetici, nel campo della sanità questi sono pochi e non specializzati nella valutazione delle prestazioni.

Per questo motivo, in questo lavoro ci siamo concentrati sulla creazione di un generatore di dati sintetici per la valutazione delle prestazioni di un modello di un sistema Big Data. Il dataset utilizzato come riferimento per la creazione del generatore è MIMIC-III, che contiene le cartelle cliniche digitali di migliaia di pazienti raccolte in un arco temporale di diversi anni. Come primo passo, è stata eseguita un'analisi del set di dati, in cui sono state utilizzate diverse tecniche di adattamento di distribuzione (ad esempio, il phase-type fitting) per modellare la distribuzione temporale dei dati. Successivamente, abbiamo sviluppato il generatore, strutturandolo come una libreria multi-modulo per consentire la personalizzazione di ogni componente. Infine, abbiamo testato il nostro generatore valutando le prestazioni di un semplice modello di sistema big data in diversi scenari. Attraverso questi esperimenti, abbiamo dimostrato il controllo granulare che il generatore offre sui dati sintetici prodotti e la semplicità con cui può essere adattato a diversi usi.

Parole chiave: Big Data, Valutazione delle Prestazioni, Generazione di Dati Sintetici, Sanità

Contents

1 | Introduction

The development of technologies such as cloud computing, IoT, and social networks has caused the amount of data generated daily to grow at an alarming rate. This rapid and unstoppable growth gave birth to the phenomenon known as Big Data.

The term Big Data is used to describe a massive volume of both structured and unstructured data, difficult to process using traditional techniques [24], but with the potential to help companies in any field to improve their operations and make faster, more intelligent decisions.

Big data has also entered the healthcare field, with the introduction of new tools for continuous patient monitoring producing massive amounts of structured and unstructured data every day. For this reason, medical facilities are moving towards data-based healthcare, with the objective of leveraging this data to support clinical decision-making, disease surveillance and public health management [2].

This new request to elaborate a multitude of heterogeneous data has resulted in the emergence of new systems capable of processing it as well as the need to evaluate their performance.

Techniques for evaluating the performance of a system generally fall into two categories: empirical techniques and modeling techniques.

Empirical techniques require the system or network to be evaluated to exists and direct measurements of the evaluation target to be taken, while modeling techniques only require a model of the system, and therefore enable the performance evaluation of the architecture of the system in question before its implementation has been completed [4].

Whichever technique is chosen to make such evaluation, to asses the performances of systems working with Big Data in a relevant way a diversity of data and workloads must be considered [11].

1.1. Scenario and Problem Statement

In many practical scenarios, obtaining a variety of real data is not trivial because many data owners are not willing to share their data due to confidential issues. This is particularly true in the healthcare scenario, where the subjects of such data are people and the research on them is greatly restrained by laws protecting patients' privacy, such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States and the General Data Protection Regulation (GDPR) in Europe [30]. Even assuming that we have access to such data, they may not be sufficient or representative enough to make a proper assessment of the performance of the system under analysis.

Consider, for example, a large hospital, with a multitude of patients, that has recently begun to use automated electronic instruments to take patients' vitals and record all their treatments, injections, X-ray reports etc. Let's assume, then, that the hospital is interested in storing this data for analysis purposes, with the goal of making predictions about the health of its patients.

If we suppose we were commissioned to build a system capable of achieving these goals, how could we size the architecture required by such a system? Since the tools would not have been used for long, the available data (assuming the hospital is willing to share it) would not be enough to conduct a comprehensive and thorough performance assessment.

For these reasons, while evaluating the performance of Big Data systems, the consensus is to *generate synthetic data* as inputs of our performance evaluation on the basis of the few real datasets available [11].

However, the techniques highlighted above for performance evaluation require the generation of such data to be done in different ways. For the performance evaluation by empirical techniques, since it is applied to implemented and running systems, it is necessary for the generated data to have the same characteristics as the original data in terms of features and values. On the other hand, for performance evaluation by modeling techniques, it is much more important, since it focuses on unimplemented models, to respect the temporal distribution of the data one wants to simulate and to have control over the volume of this data. In the healthcare field, there is a shortage of both types of generators highlighted.

In this thesis, we will focus on the creation of a generator for synthetic data to be used for the performance evaluation through *modeling techniques* of Big Data systems in the healthcare field.

1.2. Methodology

The dataset used as a reference to create the generator is MIMIC-III, which comprises de-identified health records associated with over forty thousand patients that stayed at the Beth Israel Deaconess Medical Center between 2001 and 2012 [15].

It contains structured data such as patient demographics, laboratory results etc., partially structured and unstructured data such as caregiver notes written in natural language and a multitude of recordings of various physiologic signals. Precisely because of the heterogeneity and the high volume of its data, we consider it the most appropriate for the performance evaluation of Big Data systems. Moreover, being it the only public digita health records dataset available, it is also the most widely used for research applications.

Since the goal is to create a synthetic data generator for performance evaluation using modeling techniques, our work will focus on keeping the same time distribution of the data contained in MIMIC-III.

To obtain the time intervals required, we will perform an analysis of the reference with a focus on the interaction process between the patients and the hospital that led to the storage of the data in its system, using various distribution fitting techniques and different distributions to model the duration of the stages that comprise this process and employing additional procedures to make the results adherent to reality. We will then use the same fitting techniques to model the various kind of data recorded in the system during each of the interaction stages identified.

Finally, the software development of the generator will focus on the possibility to choose different parameters for the identified distributions, on the ability to calibrate its output to provide the diversity needed to evaluate the performance of a Big Data system and on the possibility to change the procedures followed by the analysis without the need to change significantly the architecture of the generator.

1.3. Contributions

The main contribution of this work is a software component capable of reproducing the timings of the records contained in MIMIC-III, with the goal of adding to the current state of the art a new source of synthetic data for modeling-based performance evaluation of Big Data systems. While showing the structure of the generator in this paper, we have focused on its functional capabilities and the connection of these to the architectural choices made, so that it can be adapted to different datasets or use cases without losing

such capabilities.

Moreover, we also consider the procedures followed to perform the analysis on MIMIC-III to be a valuable addition to the state of the art, in the hope that they may be useful in replicating the work on other datasets.

1.4. Structure of Thesis

To show how the generator was developed and the ideas behind the chosen architecture, we decided to divide this work into two main chapters.

- One dedicated to the analysis of the MIMIC-III dataset, a necessary step to gather the required information for the creation of the generator.
- One dedicated to the software development of the generator, where we use the information obtained from the analysis as the foundation for the design of its architecture.

The thesis is therefore structured as follows:

- Chapter 1 presents the focus and main objective of the thesis, describing its context and the methodologies followed.
- Chapter 2 shows the state of the art, presenting other synthetic generators found in the literature and the other available datasets that were considered for the creation of the synthetic generator that is the subject of this paper.
- Chapter 3 introduces the background of the work, explaining the tools used as well as the theoretical background needed to understand their operation, and showing in detail the structure of MIMIC-III, which will be referred to in the other chapters of the paper.
- Chapter 4 shows the procedures followed during the analysis performed on MIMIC-III, specifying the outputs of this analysis.
- Chapter 5 shows the software architecture adopted for the generator, listing the features included and the design choices made.
- Chapter 6, presents some example use cases for the created generator, showing how it works and its usage.
- Chapter 7 summarizes the work that has been done while discussing potential future works.

2 | State of the Art

As highlighted in the previous chapter, the advent of Big Data in healthcare has led to the need to build new systems suitable for processing this new data load and, with it, the need to evaluate their performance.

When trying to evaluate the performance of a system working with Big Data, the need arises to consider large volumes of data, both structured and unstructured, and which have a fine enough temporal granularity to identify the peak usage of the system under consideration in order to make a proper and complete performance assessment.

The paucity of publicly available datasets that meet these characteristics causes the need to consider generating synthetic data to make this assessment.

In this chapter we are therefore going to highlight, in Section 2.2, the generators found in the literature for the field under consideration.

Since any synthetic data generator must be based on an existing dataset to ensure the veracity of the generated data, we will also go over the datasets available in the literature in Section 2.1. The features sought for these datasets are:

- Data heterogeneity, namely the need to contain structured, partially structured and unstructured data.
- A high level of temporal granularity, namely the need to cover events that occur even in fairly short time intervals.
- A volume of data large enough to make the simulator realistic.

2.1. Datasets

In the literature, publicly available datasets with the characteristics sought are very few.

One of the datasets considered was the New Zealand National Minimal dataset, a "collection of public and private hospital discharge information, including clinical information, for inpatients and day patients" [13]. Although it met the volume requirement, covering

the discharges of all patients in New Zealand public hospitals since 1993, it was not considered granular enough for the purposes presented (it does not contain records collected about patients during their stay but only information recorded during their discharge) and not heterogeneous enough (containing only tabular data).

Another dataset considered as a basis for the generator was the ChestX-ray8 dataset, a collection of X-ray images of over 30000 patients [29]. Unfortunately, the radiological reports associated with each image are not included in the dataset, severely limiting its heterogeneity and, therefore, its usefulness.

The only dataset deemed large, heterogeneous and granular enough to be used as a reference for the creation of the generator was MIMIC-III [15]. It contains data associated with more than 40000 patients collected over 10 years by the Beth Israel Hospital in Boston and includes, in addition to a large amount of structured tabular data collected and placed temporally with minute accuracy, a collection of medical notes (written in plain English) issued by the caregivers of the patients, which constitutes a big chunk of semi-structured data in addition to the structured data of the tables. Moreover, it is associated with a database containing a multitude of physiological signals recorded during the ICU stays of the patient, which further increases the heterogeneity of the dataset.

2.2. Synthetic Generators

A commonly used methodology related to machine learning for generating synthetic data in the healthcare field are Generative Adversarial Networks (commonly referred to as GANs).

A GAN is a kind of artificial intelligence algorithm based on an adversarial training system, where two competing models (a generator model and a discriminator model) are trained against each other [10].

They have been used for the development of medGAN [8], a generator that focuses on privacy preserving synthetic health data which, however, is only able to generate binary or, at most, integer values, severely limiting its usefulness.

An evolution of medGAN is provided by healthGAN [30], which is capable of the simulation of real-distributed values, while remaining limited in the heterogeneity of the simulated data.

In [21] is proposed SmoothGAN, a new approach for the generation of high quality synthetic data that maintains important relations and factors of the original data; it is not

focused on following the time distribution of the original data though and, moreover, it does not seem to have a functioning implementation.

GANs were also used previously in [7] to model events contained in the MIMIC-III dataset, but their focus was again not on the timing of the data, which were apparently omitted from the output of the generator.

The most notable synthetic data generator considered not based on GANs is provided by Synthea, a tool suite focusing on the generation of synthetic health records that cover the entire lifetime of the patients [28], but that lacks the granularity to be used for performance evaluation purposes.

2.3. Summary

In this chapter we have shown the lack of synthetic generators suitable for the performance evaluation of Big Data systems through modeling methods, further highlighting their necessity. We have also shown how the majority of the few available datasets cannot be taken as a reference for the development of a new generator, given their deficiency in one or more necessary features, and how MIMIC-III was, to the best of our knowledge, the only one available with enough heterogeneity and granularity to be used as a reference for the creation of a synthetic data generator.

3 | Backgrounds

This chapter outlines the theoretical basis needed to understand the contents of the thesis and presents the tools used in it.

Section 3.1 introduces MIMIC-III, specifying its structure and dwelling on the tables and attributes used during the development of the generator.

Section 3.2 presents the concept of Phase-Type distributions and the HyperStar tool, used to fit such distributions on the empirical data.

Section 3.3 presents the Kolmogorov-Smirnov test, used to compare two empirical samples and assess whether they come from the same distribution.

Finally, Section 3.4 provides the theoretical basis for understanding the performance evaluation methods used in this work, as well as presenting JMT, a tool for performance evaluation using modeling techniques.

3.1. Mimic-III

MIMIC-III (Medical Information Mart for Intensive Care, version 3) is "a large, freely-available database comprising de-identified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012" [15].

It is populated with information that have been collected during ordinary hospital treatment, and two distinct critical care information systems – iMDsoft MetaVision ICU and Philips CareVue Clinical Information System – were operational during the data collecting period.

Additional information stored throughout the stay of the patients were collected from the hospital and laboratory health record systems.

3.1.1. De-identification Procedures

Before being added to the MIMIC-III database, data was first de-identified in compliance with Health Insurance Portability and Accountability Act (HIPAA) requirements using structured data cleansing and date shifting; as such, fields like the patient's name, phone number, and address were removed.

To preserve intervals, dates were specifically relocated into the future by a random offset for each unique patient, resulting in stays that take place between the years 2100 and 2200.

When dates were changed, the current time, the day of the week, and the general seasonality were preserved.

This means that, if measurement A has been registered as if it was made on `Thursday, 2150-01-01, at 14:00:00`, and measurement B has been registered as if it was made at `Thursday, 2150-01-01 at 15:00:00`, *if the two measurements were performed on the same patient*, then measurement B was made 1 hour after measurement A, and both measurements were really made on a Thursday afternoon in winter.

Finally, patients aged over 89 years old had their dates of birth changed before the random offset to the future was applied to hide their genuine ages, making them appear to be over 300 years old in the database.

3.1.2. Contents

MIMIC-III is presented as a relational database. It is comprised by 26 tables, linked by identifiers with suffix 'id'.

For instance, `subject_id` denotes a specific patient, `hadm_id` denotes a specific hospitalization (often denoted as an **admission** to the hospital), and `icustay_id` denotes a specific stay in an intensive care unit.

Some of these tables are used to hold charted observations like notes, lab results, and fluid balance. The term **events** is used to denote these observations once they are registered into MIMIC. For example, the `OUTPUTEVENTS` table contains all measurements related to output associated with a patient, and the laboratory test results of a patient are contained in the `LABEVENTS` table.

The tables available in MIMIC-III can be split into four different categories:

- Tables used to define and track patient stays.

- Tables containing events collected in critical care units.
- Tables that contain events collected in general during hospital stay of the patients.
- Dictionary tables, used to describe and define the codes and items contained in the other tables.

Each and every table containing the events registered during the collection period of MIMIC-III comprises at least the following attributes:

- A `row_id`, a unique identifier for the row of the table.
- A `subject_id`, which uniquely identifies patient.
- A `hadm_id`, which uniquely identifies admission.
- If the table contains data collected in the critical care unit, an `icustay_id` identifies the ICU stay in which the event was collected.
- At least one time-related value that identifies the time at which the observation that led to the event being registered was taken.

In the following, we will describe some of the tables contained in MIMIC-III (specifically those that collect the events and those that define and track the patients stays), indicating their attributes of greatest interest.[1]

PATIENTS Contains the information about the patients who stayed in the hospital during the collection period. It defines the `subject_id` attribute, which is referenced by every event-related table. It provides the `gender` of the patient and their date of birth through the attribute `DOB`.

ADMISSIONS Contains the information about the admission of a patient to the hospital. It defines the `hadm_id` attribute, referenced by all event-related tables, and is associated to each patient by its `subject_id`. The information regarding the length of the stay of each patient is expressed through the attributes `admittime`, which represents the start of the hospital stay, and `dischtime`, which represents the end of the hospital stay. It also provides some patient demographic information, such as their `language`, their `marital_status` and their `ethnicity`.

ICUSTAYS Contains the informations about the single ICU stay of a patient. It defines the `icustay_id` attribute, referenced throughout the dataset, and is associated to

[1]For a complete list of all the tables contained in MIMIC-III and their structure, please refer to `https://mimic.mit.edu/docs/iii/tables`.

each patient by its `subject_id` ad to each admission by its `hadm_id`. The information regarding the duration of the stay is provided through the attributes `intime` and `outtime`.

DATE TIME EVENTS Contains all date-related measurements about a patient in the ICU, like the date of last dialysis. It is associated with each patient by its `subject_id`, to each admission by its `hadm_id` and to each ICU stay by its `icustay_id`. It also provides a `charttime`, a timestamp that marks the time at which the observation was charted, and a `storetime`, a timestamp that marks the time at which an observation was input in the hospital system.

CPT EVENTS Contains a list of which procedures were billed for which patients and is useful for determining whether certain procedures have been performed. It is associated with each patient by its `subject_id` and to each admission by its `hadm_id`. It also provides a `chartdate`, which shows the date at which the procedure occurred, without a specific time indication.

CHART EVENTS Contains the charted data registered during the ICU stay of a patient. It is associated with each patient by its `subject_id`, to each admission by its `hadm_id` and to each ICU stay by its `icustay_id`. It also provides a `charttime`, a timestamp that marks the time at which the observation was charted, and a `storetime`, a timestamp that marks the time at which an observation was input in the hospital system.

CALLOUT Contains the information about the ICU discharge plan for the patient. The process involves:

- A registration that the patient is ready to leave the ICU
- An acknowledge by a coordinator that the patient requires a bed outside the ward.
- A variable period of time in order to coordinate the transfer
- An outcome of the callout, which might result in the patient leaving the ICU or the callout to be revoked.

It is associated with each patient by its `subject_id` and to each admission by its `hadm_id`. It also provides a `createtime` that marks the time at which the callout was initiated, an `updatetime` that marks the time at which the callout event was updated the last time, an acknowledge time that records the time at which the callout was acknowledged by a coordinator, and an `outcometime` that records the time at which the outcome of the callout was obtained.

LAB EVENTS Contains the information about laboratory based measurements performed during the stay of the patients. It is associated with each patient by its `subject_id` and to each admission by its `hadm_id`. It also provides a `charttime`, a timestamp that records the time at which the data was measured.

MICROBIOLOGY EVENTS Contains microbiology information, like cultures acquired from the patients and associated sensitivities. It is associated with each patient by its `subject_id` and to each admission by its `hadm_id`. It also provides a `charttime`, a timestamp that records the time at which the data was measured.

NOTE EVENTS Contains all notes written by the medical staff of the medical institute for their patients. It is associated with each patient by its `subject_id` and to each admission by its `hadm_id`. It also provides a `category` indicating the type of note recorded, a field `text` containing the text of the note, a `charttime` that marks the time at which the observation was charted, and a `storetime` that marks the time at which an observation was input in the hospital system. A subset of the events registered in this table is missing the `charttime`, specifically the notes with a `category` of either "Discharge summary", "ECG" or "Echo".

OUTPUT EVENTS Contains information about all the fluids which have either been excreted by or extracted from the patient. It is associated with each patient by its `subject_id`, to each admission by its `hadm_id` and to each ICU stay by its `icustay_id`. It provides a `charttime`, a timestamp that records the time at which the observation was charted, and a `storetime`, a timestamp that records the time at which an observation was input in the hospital system.

INPUT EVENTS (MV) Contains the input events collected by the iMDsoft MetaVision sytem. Input events are related to the measurement of any fluid which have been administered to the patients. Each input event stored in this table is associated with each patient by its `subject_id`, to each admission by its `hadm_id` and to each ICU stay by its `icustay_id`. It provides a `starttime` and an `endtime` which records the period of the input, and a `storetime` that records the time at which an observation was input in the hospital system.

INPUT EVENTS (CV) Contains the input events collected by the CareVue system. Each event is associated with each patient by its `subject_id`, to each admission by its `hadm_id` and to each ICU stay by its `icustay_id`. It provides a `charttime` which records

the period of the input, and a `storetime` that records the time at which an observation was input in the hospital system.

3.1.3. MIMIC-III Waveforms database

Intensive care units' bedside patient monitors recorded a multitude of observations of various physiologic signals ("waveforms"), which were then compiled in the MIMIC-III Waveform Database [20].

The MIMIC-III Waveform Database is a companion to the MIMIC-III Clinical database, but matching the clinical data with the waveform data was a non-trivial task because each database's material was individually gathered. For this reason, only a subset of the Waveform Database records has been matched with the patients of the clinical database [20].

In almost all cases, the waveform records comprises multiple files containing an uninterrupted recording of a set of simultaneously observed signals. Each of this files is called a *segment*. Each record provides:

- A *Master Header File* that comprises the subject id, the surrogate time and date of the record, and other technical information.
- A *layout header file* that specifies the all of the signals observed in any segment belonging to the record.

Both header files are structured according to the format specified in `https://physionet.org/physiotools/wag/header-5.htm#sect5`.

3.2. Phase-Type Distributions and Hyperstar

Distribution fitting is the fitting of a probability distribution to a sample of data concerning the repeated measurement of a variable phenomenon. The aim of distribution fitting is to enable the creation of new samples that could come from the same phenomenon. The goodness of the fit will depend on the characteristics of the distribution chosen and of the phenomenon to model.

Phase-type distributions are probability distributions constructed as mixtures of Exponential distributions [12]. They are commonly used for distribution fitting purposes because they possess the welcome property of being able to approximate any distribution with arbitrary precision [5].

The main idea behind Phase-Type distributions is to produce a sample – that resembles a

sample generated by any non exponential distribution – through the use of exponentially distributed steps called *stages*.

In Figure 3.1 we can see a visualization of the set of stages that make up an example phase-type distribution: each of the arcs connecting one stage to another is associated with a parameter λ_i (used to define the exponential distribution that causes the jump from one stage to another), and each stage has its own probability of being the starting point of the generation procedure. To generate a sample of the distribution, we only need to choose the starting stage and traverse the stages, considering the exponentially distributed time at each step. The sample is generated when the *absorbing stage* (filled in black in Figure 3.1) is reached.

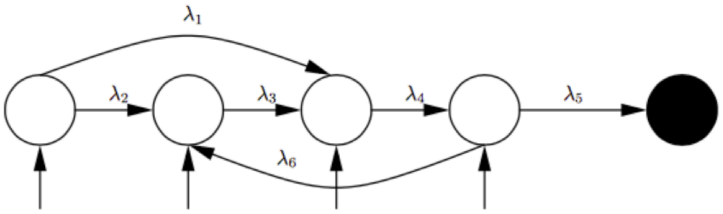


Figure 3.1: Stagrs of a generic Phase-Type distribution.

The fitting of a Phase-type distribution to a sample can be performed either by knowing the structure of the process that generated the sample (from which we might be able to define the stages of the Phase-Type distribution) or by using fitting algorithms based on parameter estimation [27].

The Phase-Type distribution family is divided into many sub-classes; one of the most important is the *Hyper-Erlang* distribution.

The Erlang distribution is the sum distribution of k exponential random variables (where k is called the *phase* of the Erlang distribution) characterized by the same rate λ. Hyper-Erlang distributions are themselves mixtures of mutually independent Erlang distributions characterized by different rates and number of phases [16]. Each Erlang distribution is called a *branch* of the Hyper-Erlang.

A representation of a Hyper-Erlang distribution by means of the *stages* previously defined for Phase-Type distributions is shown in Figure 3.2.

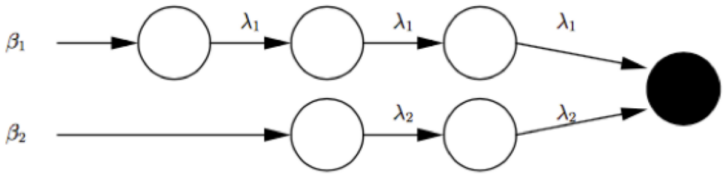


Figure 3.2: Stages of an Hyper-Erlang distribution.

3.2.1. HyperStar

To perform the parameter estimation of a Phase-type distribution based on a sample extracted from an unknown distribution and when the process responsible for the generation of the sample is unknown, we can rely on parametric fitting tools such as *HyperStar*.

HyperStar is an extensible tool with a graphical user-interface that enables simple and efficient fitting of phase-type distributions to datasets [22]. that allows exploration of the dataset and provides immediate feedback of the fitting results.

The fitting process adopted by HyperStar involves the interaction with the user (as the case is with many other tools used for fitting Phase-Type distributions) to indicate the important features of the sample by identifying the clusters of data to focus on during the parameter estimation. Such sample must be provided as a list of comma separated values.

The interface of the program during this interaction can be seen if Figure 3.3.

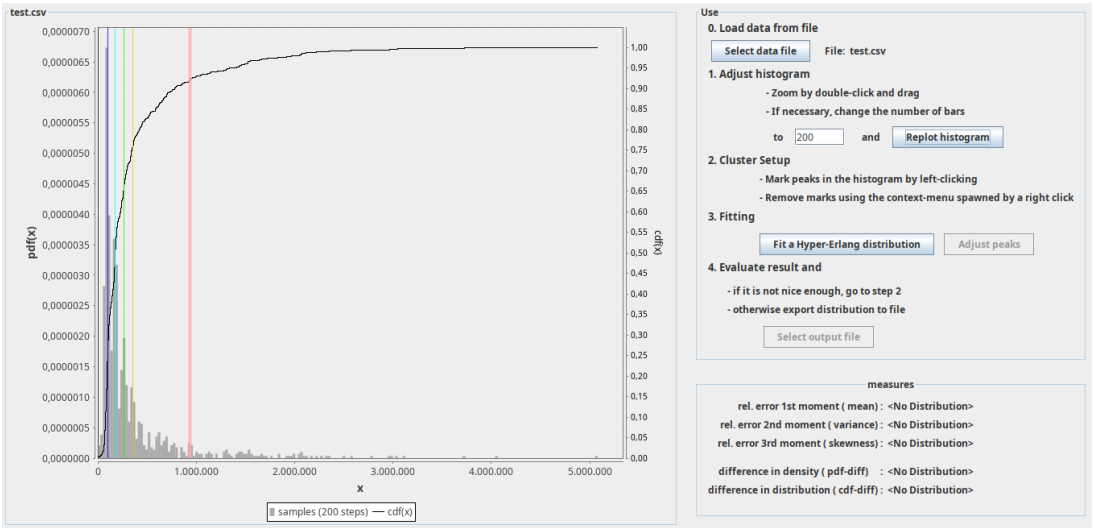


Figure 3.3: Interface of HyperStar during cluster definition.

By default, the tool fits an Erlang distribution to each cluster identified this way and composes them into a single Hyper-Erlang distribution. The result of the fitting is visualized

in the form of the cumulative distribution function of the fitted distribution, which can be compared with the empirical cumulative distribution function of the sample, as it can be seen in Figure 3.4. HyperStar also provides a measurement of the relative error between the empirical moments (mean, variance, skewness) of the samples and the moments of the fitted distribution.

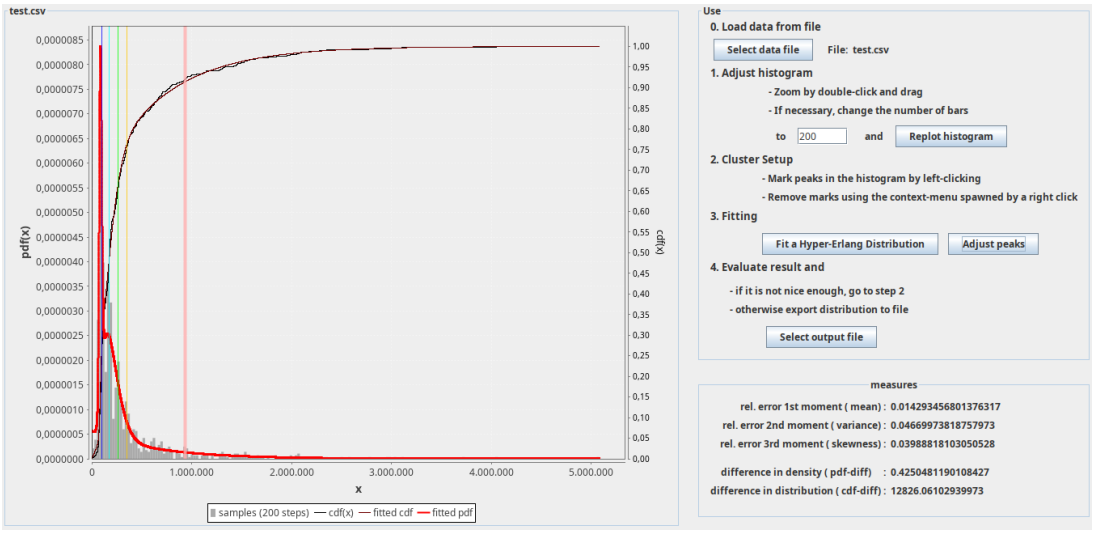


Figure 3.4: Results of the fitting process shown by HyperStar.

The fitted distribution is then saved on a file in a human readable format that encapsulates the number of branches, the probability of each branch and the rates and phases of each branch, as shown in Figure 3.5.

```
# Parameters of hyper-Erlang distribution:
# number of different Erlang branches, m
5

# number of phases of i-th Erlang branch, i = 1,...,m
47
7
8
1
1

# lambda values of i-th Erlang branch, i = 1,...,m
5.293006897306026E-4
4.29898428512436E-5
3.1751917574230804E-5
1.7246550274603846E-6
1.3532058730585751E-6

# initial probabilities of i-th Erlang branch, i = 1,...,m
0.22291853178155774
0.21933751119068934
0.19695613249776187
0.19247985675917637
0.1683079677708147

# Distribution fitted by HyperStar
```

Figure 3.5: Sample output of the fitting process performed by HyperStar.

3.3. Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test for two samples (also called simply the Simrnov test) is a non-parametric statistical test used to compare those samples and assess whether they were extracted from the same continuous distribution [3].

To do so, the test computes the maximal distance between the cumulative empirical distribution functions (abbreviated ECDF) of the two samples; Figure 3.6 shows an example of such maximal distance.[2].

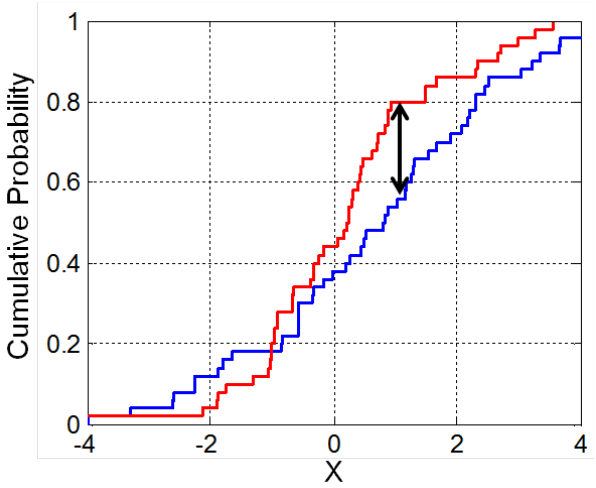


Figure 3.6: An example showing the the maximal distance (the black double sided arrow) between the two cumulative empirical distirbution functions of two samples (shown in red and blue).

If the maximal distance is bigger than a critical value D, the Kolmogorov-Smirnov test rejects the null hypothesis that the two samples are extracted from the same distribution. The critical value D is expressed as:

$$D = \sqrt{-ln(\frac{\alpha}{2}) \cdot \frac{1 + \frac{m}{n}}{2m}}$$

The parameter α that appears in the formula for the critical value D is the maximum probability that the test will result in a false negative (e.g. the two samples come from the same distribution, but the test fails to detect it). The parameters n and m represent the size of the two samples.

[2]Figure taken from `https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test`

3.4. Performance evaluation and JMT

Performance Evaluation is the quantitative and qualitative study of systems to evaluate, measure, predict and ensure target behaviors and performances.

To evaluate the performances of a system, we usually adopt or construct a *model* of it in order to make the evaluation easier.

A *Queuing network model* is a particular approach to modelling computer systems in which the system is presented as a *network of queuing stations* [17].

A **Queuing station** is a collection of *service centers*, which represents system resources, and *jobs*, which represent the unit of work that the service center has to perform.

When all service centers are busy processing a job and other jobs arrive at the queuing station, they are put into an ordered *queue* (if available) to be processed later.

To define the behavior of the jobs inside the model, like how they move between service centers or how long it takes for the service center to perform the job, we use a set of model parameters, of which some are listed here:

- The number of service centers in each queue.
- For each service center, its *service time*, or the time needed to perform the job.
- The *visits* of each queue, which represents the average number of times a job will visit that resource while inside the system.

When the jobs considered involve different service times or may follow a different path inside the system, they may be introduced as different *job classes*. In case multiple jobs classes have to be considered, the service time and the visits for each of the classes and each of the service centers have to be provided as parameters of the model.

3.4.1. Performance Indexes

Once the parameter of the model have been provided, we can use it to evaluate the performances of the system by computing its *performance measures*. Some of these performance measures are shown in the following.

Utilization Is computed for each queuing station and represents the proportion of time that the station has been occupied performing a job (instead of not doing anything). Once the utilization gets to 1 (or 100%), the queuing station is said to be *unstable*, since additional jobs arriving will have to wait more and more before they can be performed.

Response Time Is computed either for each queuing station or for the entire system and represents the average time spent in the queuing station by a job, both queuing and being performed.

Throughput Is computed either for each queuing station or for the entire system and represents the average rate (expressed in $\frac{jobs}{seconds}$) at which customers pass through the queuing station or the system.

3.4.2. JMT and JSimGraph

Java Modelling Tools (abbreviated *JMT*) is a free and open source suite consisting of multiple tools for performance evaluation and modelling of computer and communication systems [25]; it was founded in Politecnico di Milano, and is used nowadays to teach about performance evaluation in some computer engineering courses.

One of its commonly used tools is *JSimGraph*, a GUI front-end to the JMT *simulation engine* that allows for the construction of a model and its performance evaluation through simulation.

The jobs to be elaborated by the modeled system can either be provided by the user or be generated by the system according to a reference distribution.

If generated by the user, they have to be provided in the form of a single file for each job class containing a list of values that each represent the time between the arrival of the next job and the previous one.

The definition of the model is done through a graphical interface where the user can define their own queuing stations and connect them to each other, as well as define the classes of jobs to be handled by the queuing station and how each of them is routed throughout the system.

Each queuing station is defined through the parameters previously shown, but since the routing of the jobs is defined with the knowledge of the whole system in mind the visits do not have to be defined for each queuing station.

To provide the performance indexes shown above, JSimGraph simulates the flow of jobs in the system; while these are being processed, it computes each index requested by the user by considering increasingly narrower confidence intervals until one is obtained that fits the user.

The confidence interval used by default will stop the simulation once it is sure to contain

the real value for the performance index 99% of the times.

In Figure 3.7 is shown a sample of the output of a simulation made with JSimGraph, which shows the response time of the *CPU* station of the system. The blue line in the graph shows the value computed by the tool for the response time as simulation progresses, while the red lines show the confidence intervals for the measurement.

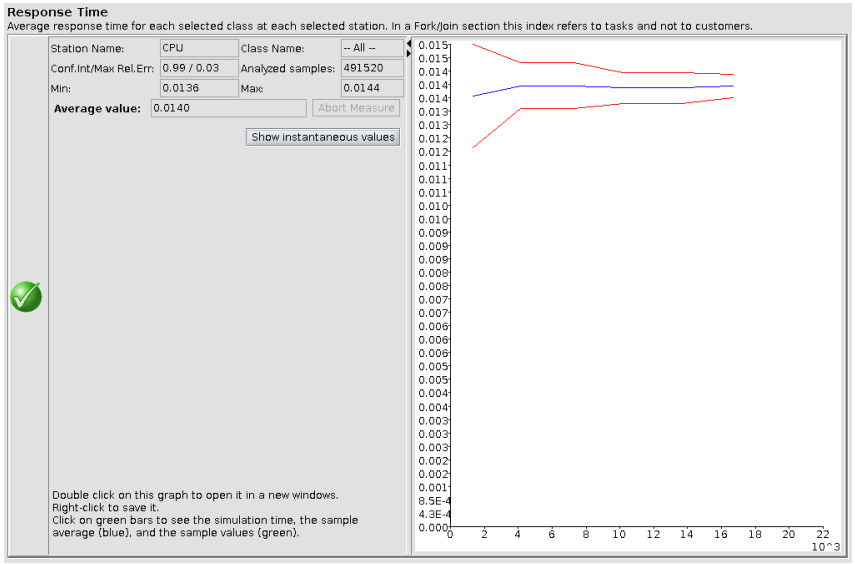


Figure 3.7: Results of an example simulation in JSimGraph.

What-if Analysis

An interesting tool bundled with JSimGraph is the *What-if analysis* tool.

A what-if Analysis consists of a series of simulations in which one or more control parameters are varied over a specified range; this allows the observation of the behavior of the system under a spectrum of conditions, unlike the single simulation run where the system is observed under a fixed set of parameters.

Once the what-if analysis is enabled, it is made possible for the user to specify the range of the variation on each parameter and to specify the number of steps in which such variation shall happen. Then, for each of the step provided, the what-if analysis tool performs a simulation with the newly updated parameter.

In Figure 3.8 is shown the output of a what-if analysis performed for the response time of station *CPU* while changing the service time of all the classes of the system to be 200% of their original service time in three steps.

The graph of the what-if analysis shows in red the change in the performance index during

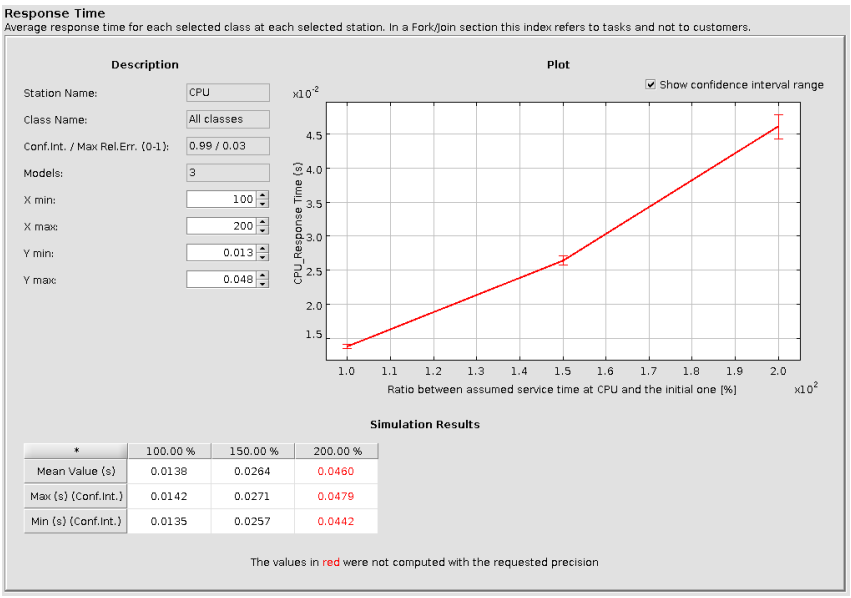


Figure 3.8: Results of an example what-if analysis in JSimGraph.

the change in the reference parameter, showing at each step the confidence interval of the measurement taken.

4 | Analysis

In this chapter we will present the procedures and methodologies applied during the extraction of the temporal distributions of the data contained in MIMIC-III.

To obtain them, we carried out an analysis of these data, focusing on the interaction process between the patients and the hospital that led to their storage in its data system, with the aim of making the generator as close to reality as possible.

After presenting the design choices followed throughout this chapter in Section 4.1 and some preliminary procedures performed on the data in Section 4.2, we will focus on the stages that comprise the interaction process of the patients with the hospital system in Section 4.3.

Once these stages have been identified, we will split the dataset in subsets based on the classification procedure followed in Section 4.4 and, in Section 4.5, we will choose the distributions that correctly fit the duration of the stages previously identified. Then, we will do the same in Section 4.6 and Section 4.7 for each kind of data recorded during the patients' hospital stay, associating it with one or more of the identified interaction stages and choosing a suitable temporal distribution to describe it.

We will finally summarize the outputs obtained from the analysis in Section 4.8, focusing on the information they will provide to the generator developed in Chapter 5.

4.1. Design Decisions for the Analysis

As previously stated, the reference data source chosen for the synthetic data generator is MIMIC-III, a freely-available database comprising data associated with the patients who stayed in critical care units of the Beth Israel Deaconess Medical Center of Boston. A thorough description of the dataset can be found in Section 3.1.

The data that comprises MIMIC-III had to undergo a de-identification procedure before it could be incorporated into the dataset, in accordance with the required privacy standards. In particular, in order to avoid the recognition of the patients based on known habits or

conditions of them, for each individual patient the dates were shifted into the future by a random offset in a consistent manner.

This way, although any time correlation between different patients is lost, all the time intervals between two entries associated with the same patient are kept intact; other than that, only a handful of characteristics of the original date were still valid after the random shift, namely the day of the week, the general season, and the time of the day.

In Figure 4.1 you can find an image that describes the de-identification process with an example.

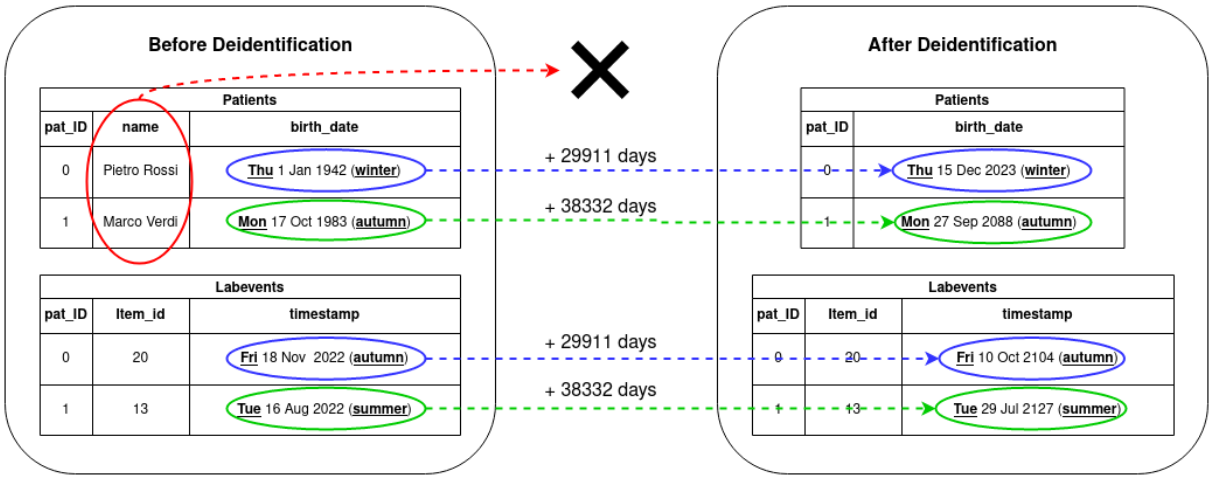


Figure 4.1: De-identification procedure applied on MIMIC-III

Since the focus of the work lies in the simulation of the time distribution of the observations and measurements contained in MIMIC-III, the de-identification procedure applied on the dataset deeply influenced the structure of our work: an analysis of the exchange of data between the patients as a whole and the hospital system was in fact made basically impossible and, for this reason, we decided to focus on analysing of the patients singularly.

4.2. Preliminary Steps

Before introducing the procedures applied to gather insights regarding MIMIC-III, to enable the analyses performed to be reproduced properly, it is useful to list the tools used and the preparatory steps taken.

4.2.1. Tools Used

Once we obtained access to the dataset, we considered the various opportunities for querying the available data; Of the various options, we chose self-hosting, using PostgreSQL as the database system. To avoid conflicts with other system applications we chose to use Docker to contain the database.

We chose to use JupyterLab to perform the analysis step, a web-based development environment, and used Python as the programming language to develop the analysis. To interact with the PostgreSQL database in the JupyterLab development environment we leveraged Psycopg2, a PostgreSQL adapter for the Python language.[1]

4.2.2. Pre-processing the Waveforms

In order to carry out an analysis of the records of the vital signals of the patients (called *waveforms* by the authors of MIMIC-III) with the aim of fitting their temporal distribution, it was necessary to first process them to extract the necessary information concerning timing.

Each waveform is divided into several binary files, also called *segments*, and each segment is associated with its own header file.

The waveforms are each associated to a *master header file* which provides the `subject_id` of the patient, the surrogate time and date of the recording and the segments that comprise such recording. The master header file (of which an example is shown in Figure 4.3) was used to gather most of the necessary information about the waveform recordings.

Moreover, a *layout file* (of which a sample is shown in Figure 4.2) is associated to each waveform and shows which signals (highlighted blue in figure) are registered during each recording.

[1] `https://www.psycopg.org/`

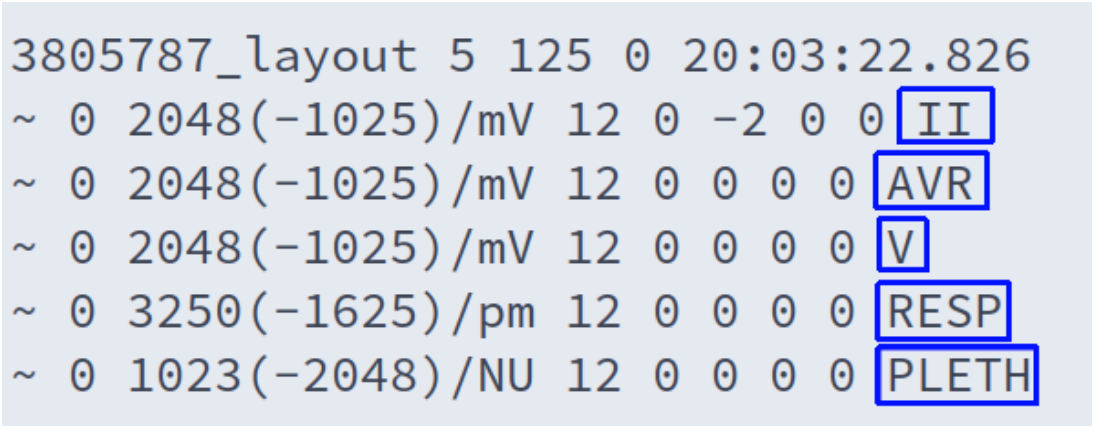


Figure 4.2: Layout file example with the signals information highlighted

The information of interest contained in the master header file is:

- The `subject_id` of the patient on which the recording was performed, which connects the waveform to the other tables contained in MIMIC-III.
- The sampling frequency of the recording (amount of samples transmitted per second), highlighted in green in Figure 4.3.
- The total amount of samples transmitted during the recording, highlighted in red in Figure 4.3
- The timestamp of the recording, highlighted with cyan in Figure 4.3.
- The layout file associated with the recording, highlighted in purple in Figure 4.3.
- The pauses between transmissions, identified by a '∼' at the start of the line, and their duration (in terms of "un-trasmitted samples", or sample frequency units), highlighted in orange in Figure 4.3.
- For each segment, the amount of samples covered by it, highlighted in blue in Figure 4.3.

```
p000107-2121-11-30-20-03/279 5 125 11259250 20:03:22.826 30/11/2121
3805787_layout 0
3805787_0001 125
3805787_0002 6875
3805787_0003 11375
3805787_0004 625
3805787_0005 103375
~ 1250
3805787_0006 125
3805787_0007 125
3805787_0008 185000
~ 750
3805787_0009 125
3805787_0010 125
3805787_0011 148750
```

Figure 4.3: Waveform Master Header example with the information of interest highlighted.

To facilitate the analysis, we gathered the information of interest into a single table, which was then stored in the same database system containing MIMIC-III and matched with the ICU stays in which they were presumably recorded based on their time stamp.

To reduce the amount of tuples in the table and simplify the fitting process seen in Section 4.7, subsequent sample transmissions (not separated by a pause) within the same recording were grouped together; the timestamp of each transmission group was then computed based on the duration of the previous transmissions and pauses while the signals for each transmission group were assigned based on the configuration read from the layout file. An example of the process can be seen in Figure 4.4.

Master Header file

```
p000107-2121-11-30-20-03/279 5 125 11259250 20:03:22.826 30/11/2121
3805787_layout 0
3805787_0001 125
3805787_0002 6875
3805787_0003 11375
3805787_0004 625
3805787_0005 103375
~ 1250
3805787_0006 125
3805787_0007 125
3805787_0008 185000
~ 750
3805787_0009 125
3805787_0010 125
3805787_0011 148750
```

Layout file

```
3805787_layout 5 125 0 20:03:22.826
~ 0 2048(-1025)/mV 12 0 -2 0 0 II
~ 0 2048(-1025)/mV 12 0 0 0 0 AVR
~ 0 2048(-1025)/mV 12 0 0 0 0 V
~ 0 3250(-1625)/pm 12 0 0 0 0 RESP
~ 0 1023(-2048)/NU 12 0 0 0 0 PLETH
```

Waveforms

subject_id	icustay_id	timestamp	samples	frequency	duration [seconds]	signals
000107	XXX	2121/11/30 20:03:22.826	122375	125	979	II, AVR, V, RESP, PLETH
000107	XXX	2121/11/30 20:19:41.826	1250	125	10	PAUSE
000107	XXX	2121/11/30 20:19:51.826	185250	125	1482	II, AVR, V, RESP, PLETH
000107	XXX	2121/11/30 20:54:33.826	750	125	6	PAUSE
000107	XXX	2121/11/30 20:54:39.826	149000	125	1192	II, AVR, V, RESP, PLETH

Figure 4.4: Example of the processing of a waveform record.

4.3. Stages of the Patient's Interaction Process

In addition to information regarding the ICU stays of the patients, like their date and time of start and finish, MIMIC-III also contains information about the entire hospital stay of the patients, covering thus their time outside the intensive care units. In particular, some of the observations registered in MIMIC-III were only made during an ICU stay, while others were made during the entire hospital stay. From now on, these observations will be called *events*, the same term used by the authors of MIMIC-III to denominate them.

Since we want to reproduce the intervals at which the events associated to each patient are registered in the hospital data system according to MIMIC-III, it is useful, as a first step, to try and understand the stages that comprise the interaction process between the patients and the hospital.

The term *interaction process*, in this context, refers to the various time periods during which data has been acquired from the patient and recorded within MIMIC-III. The *stages* into which we are going to partition this process in this section will themselves be time periods, and the transition from one stage to the other will correspond, most of the times,

to the admission and the discharge from an ICU or from the hospital.

Using some MIMIC-III terminology, we will say an event is "associated to an admission" (or a hospital stay) when the table it is registered in has the `hamd_id` attribute, that is to say it was registered during the time when the hospital stay took place, while we will say that the event is "associated to an icustay" when its table possesses the attribute `icustay_id`, that is to say it was registered during the time when the patient was in an ICU.
As one can guess, every event is associated to a `hadm_id`, since all registered events happened during an hospital stay, but not every event is associated to an `icustay_id`. For additional information about the structure of MIMIC-III and its tables, please refer to Section 3.1.

Based on the information carried by the structure of MIMIC-III, we can split the period of interaction between the patient and the hospital into these main stages:

- After the hospital stay has started, there's a time period before the start of the ICU stay of the patient in which they are associated to an hospital stay but not to an ICU stay.
- The patient then gets admitted into an intensive care unit, where their ICU stay starts. During this period inside the ICU, the events associated to the ICU stay are recorded.
- The patient then exits the intensive care unit.
- After a period of time outside the ICU, their admission terminates.

During this entire period of time, the events associated to the hospital stay of the patients are registered.

The stages described above can be formalized in the UML activity diagram[2] presented in Figure 4.5. The boxes on the edges of the pools represent the events that are registered in the hospital data system.

Since it is not the focus of the thesis, the operation of the system has been vastly simplified.

[2]The specification for UML activity diagrams can be found at `https://www.omg.org/spec/UML/2.5.1`

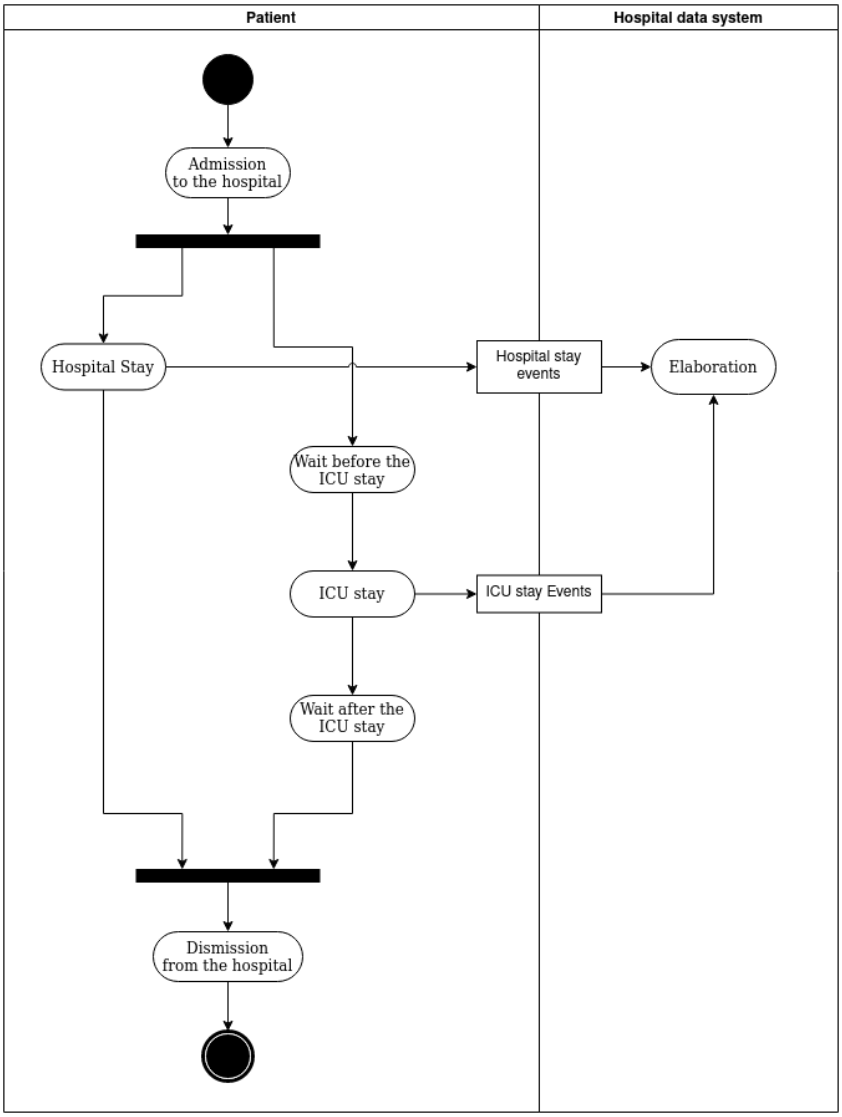


Figure 4.5: Activity diagram representing the stages of the interaction process between the patient and the hospital system

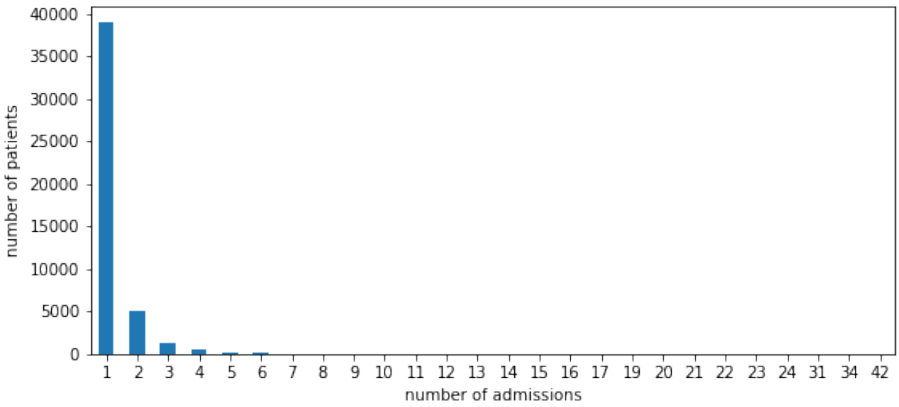
The stages identified in this section, however, do not cover many less frequent cases found during the exploration of the data, which we will highlight in the following sections.

4.3.1. Multiple Hospitalizations and Multiple ICU Stays

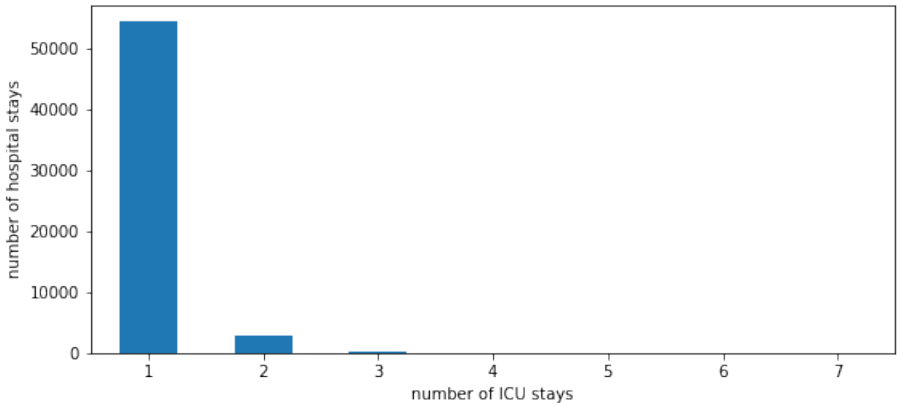
One of the cases not represented in the stages of interactions shown above is the possibility for the patient to have multiple hospital stays and multiple ICU stays within a single hospital stay.

The queries performed on MIMIC-III on this regard supported the hypothesis, as shown

in Figures 4.6.



(a) How many patients have sustained a certain number of hospital stays



(b) How many hospital stays comprised a certain number of ICU stays

Figure 4.6: Results of the queries performed on MIMIC-III regarding multiple hospital and ICU stays.

As we can see, even though the majority of the patients is associated to a single hospital stay and most hospital stays are associated to a single ICU stay, a non-negligible portion of patients has multiple admissions associated to them or multiple ICU stays associated to the same hospital stay.

Given these results, it is useful to update the interaction stages previously shown to also consider the patients mentioned above. The additions associated with this case study made to the previous activity diagram are shown in Figure 4.8 in green, and consider the possibility for an hospital stay to repeat and for another ICU stay other than the first one to happen.

As you can see, between successive ICU stays a time interval has been considered, labeled in the diagram as "wait time between ICU stays". This time period could be considered in conjunction with the period after the ICU stay already introduced in the stages of interaction previously outlined, but we decided to consider them separately in the event that they have a different time distribution.

A similar time period has been considered between successive hospital stays and has been labeled in the diagram as "wait time between hospital stays". During this time period, the patient is not considered to be in the hospital premise and, as such, does not generate any sort of data to be stored in the hospital data system.

4.3.2. Hospitalizations That End Immediately After the Patient Is Discharged From the ICU

During the exploration of MIMIC-III, we realized that an important number of ICU stays (almost 20%) finishes after the hospital stay to which they are associated.

As we can notice in Figure 4.7, which represents the cumulative density function of the duration of these abnormal ICU stays after the end of the hospital stay to which they are related to, the majority of them has at most one day of time distance after the end of the associated hospitalization, with over 75% being below three hours.

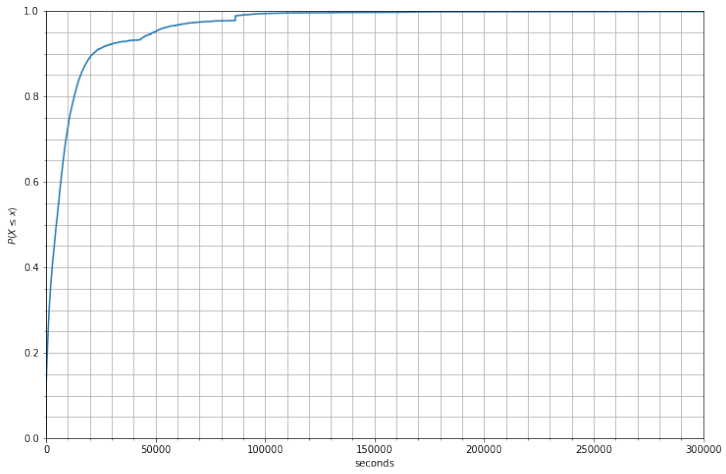


Figure 4.7: Cumulative density function plot of the length of the ICU stays that finish after their hospital stay

Since the documentation of MIMIC-III does not contain any information about them, to prevent any information loss, we consider them as the real finish time of the associated

hospital stays.

In accordance with this consideration, a meaningful number of hospital stays now finish immediately after their last ICU stay. To avoid introducing too many null values for the interval between the end of the last ICU stay and the end of the admissions, we deemed it appropriate to consider this possibility separately by explicitly introducing it in the interaction stages previously outlined.

The update to the activity diagram associated with this case study is shown in Figure 4.8 in blue.

4.3.3. Hospitalizations That Don't Have an ICU Stay

While looking for additional information in MIMIC-III, we found another limit case, less evident (but relevant nonetheless) than the ones previously considered: A small portion of the hospital stays (1190 over 58976, 2% of the total) is not associated to any ICU stay.

Although they do not represent a big portion of the total cases, we deem it useful to consider these hospital stays separately.

The updates to the activity diagram associated with this case study are shown in Figure 4.8 and highlighted in red.

To summarize, while studying the interaction process between the patients and the hospital system, we identified a total of 6 stages, identified by as many time intervals:

1. The time in ICU, in which the events associated with the ICU are created.
2. The time after the hospital admission and before the start of the first ICU stay
3. The time between two consecutive ICU stays.
4. The time between the end of the last ICU stay and the end of the entire hospital stay.
5. The total time in the hospital, which shall be considered as the sum of the times listed above (unless the patient is not admitted in an ICU, in which case the hospital time shall be computed separately).
6. The time between the end of a hospital stay and the beginning of the next one.

In the following sections, the term **interaction** will be used to indicate these time intervals.

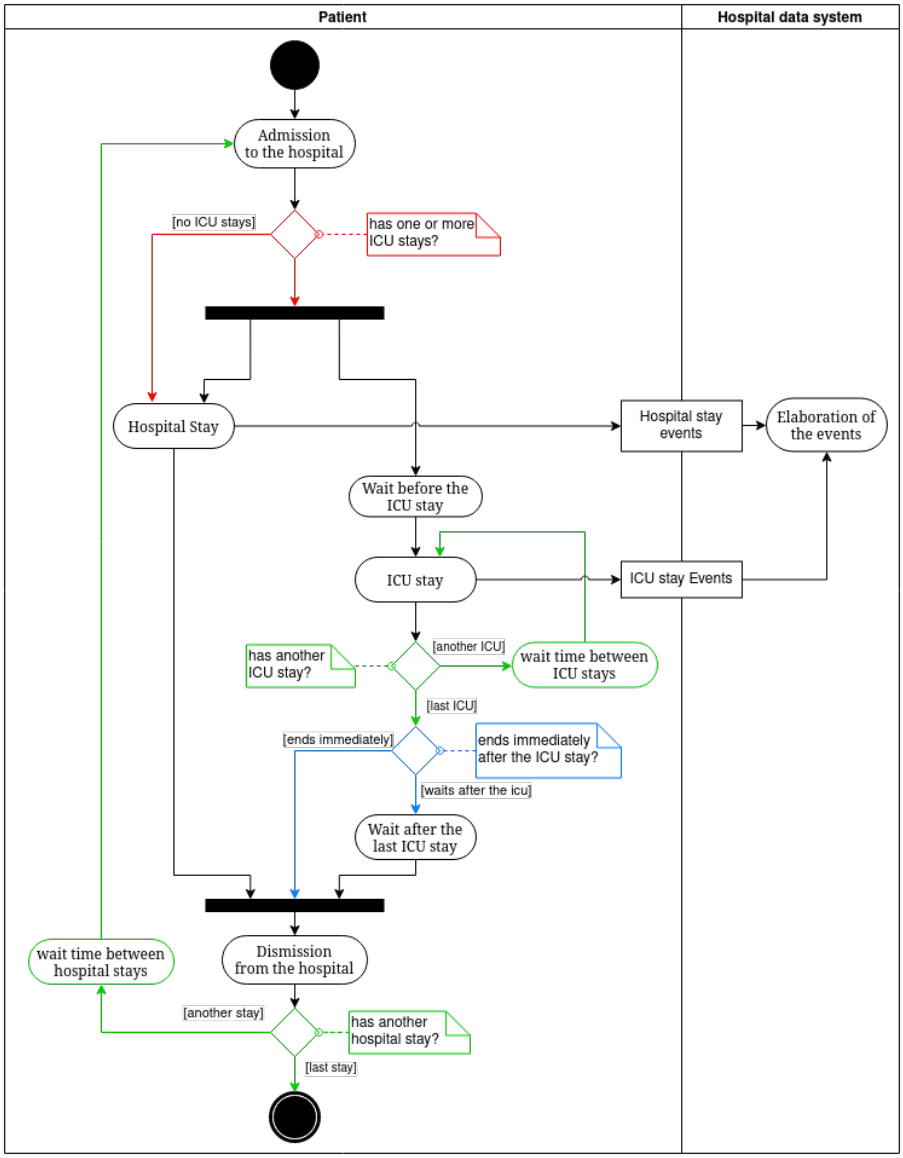


Figure 4.8: Activity diagram representing the stages of the interaction process between the patient and the hospital system

4.4. Classification

Before focusing on the distributions that can be fit to model the interactions identified in the previous section, we decide to split the dataset into multiple subsets based on specific *classifications*. This decision is made to avoid considering a single distribution to model all the events produced by all the patients during their stays, which would risk worsening the fitting results.

By collecting the data into sensible classes, in fact, we can hope to identify those time

intervals that follow similar distributions, thus reducing the complexity and improving the results of the distributions used to fit them. Even if the chosen classifications do not achieve the desired effect, introducing this possibility in our generator benefits its versatility and opens the door to use more sophisticated and better performing classification methods.

The chosen line of work is to base our classes on easily observable features of the data in order to maintain the grouping process simple. With the objective of keeping the groups meaningful, we only consider features that:

- relevant to the medical field, meaning that they are prone to vary the diseases they might cover or the procedures followed by the hospital staff.
- distinctive enough to split the set of data in comparable portions.

4.4.1. Patient-based Classification

The first classification step is centered around the patients of the Beth Israel Institute.

If we aggregate the available patient attributes in the MIMIC-III tables (specifically the `PATIENTS` and `ADMISSIONS` table), the immediately available features for a patient-centered classification of the interactions and the events are:

- Their ethnicity
- Their gender
- Their marital status
- Their language

Of these, we consider ethnicity and gender as the only features relevant to the medical field.

The link between gender and health conditions is well known [18]; for example, the incidence and severity with which diseases occur varies between genders, with some of them being sex-specific.

The link between ethnicity and health conditions, although not as common, has also been documented and studied for a long time [26] and for this reason we can say that its relevance to the medical field has already been established.

However, observing the various categories available for the ethnicity of the patients and their distribution (shown in Figure 4.9), it is clear that splitting the dataset based on the

ethnicity of the patients would be inconclusive; There is in fact a too high number of categories (an aspect that could be overlooked having the possibility to group the categories into 'macro-ethnic' groups) and an excessive imbalance in the distribution of patients among the various categories (the "White" category alone represents approximately 70% of the total population).

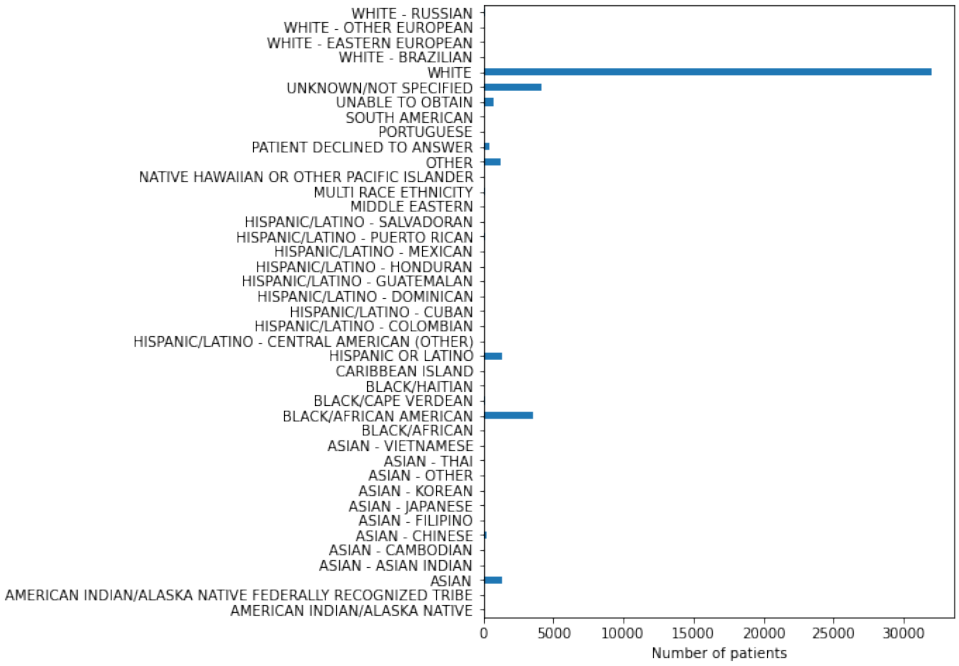


Figure 4.9: Bar plot showing the ethnicity of the patients registered in MIMIC-III.

The categories available for the gender of the patients, on the other hand, are only two (male and female) and the distribution of patients among them is shown in Figure 4.10. As we can see, the patients are well distributed between the two categories, with 56% of patients identified as male and 44% identified as female.

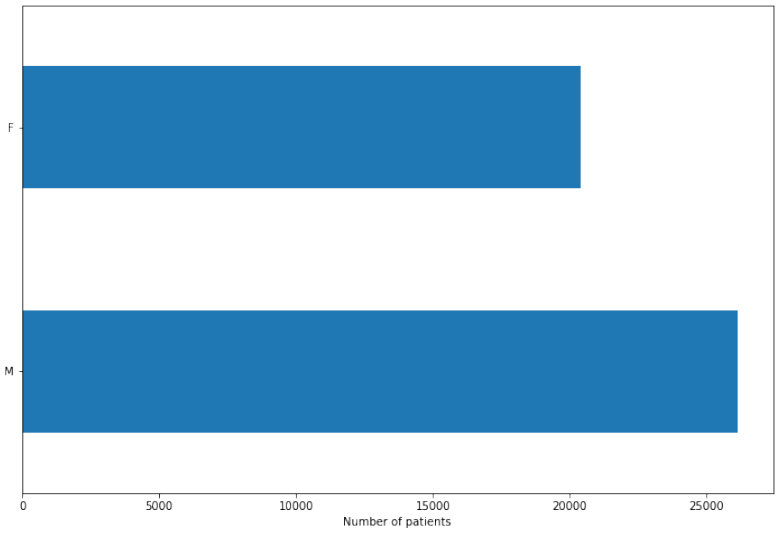


Figure 4.10: Bar plot showing the gender of the patients registered in MIMIC-III.

Having available the patients' date of birth (an attribute of the `PATIENTS` table), another feature with strong medical relevance (although not as readily available) for which we introduce a classification is the patient's age.

The date chosen as the reference for calculating the patient's age (impossible to consider in an absolute manner because of the de-identification process applied on MIMIC-III) is the date of the beginning of their first hospital stay.

Following this consideration, we can see in Figure 4.11 how the patients are distributed for various age groups.

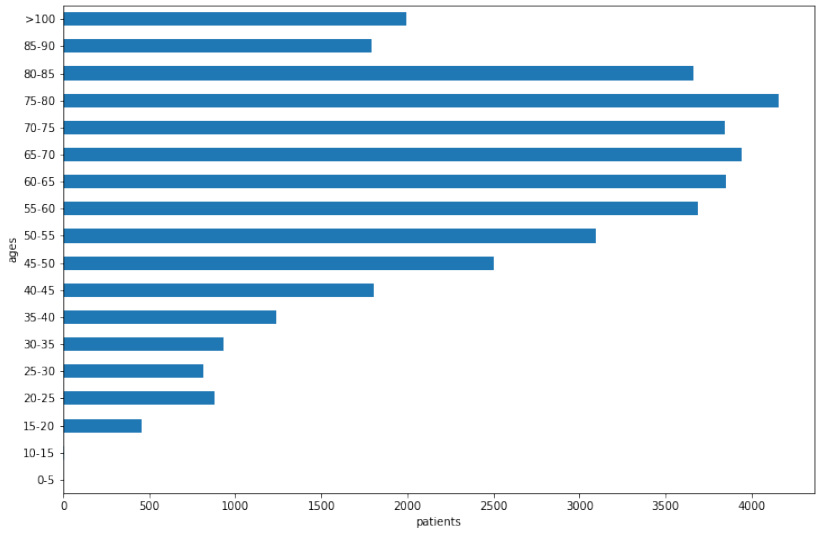


Figure 4.11: Bar plot showing the ages of the patients registered in MIMIC-III at their first admission.

We can see that the age of the patients follows a somewhat normal distribution centered on the age of 70, with a considerable peak in the age range of 75 to 80.

During the de-identification process applied on MIMIC-III, the age information of patients 90 years old or older endured an additional de-identification step, in which they were assigned a birth date that results in a registered age over 300 years. These patients are considered to be in the *over 100* age group, giving a rationale for the sudden spike in the plot.

Before it can be considered as a valid classification feature, however, it is necessary to reduce the number of groups into which the ages are divided (too many classes would in fact lead to sets that are too small to be fit adequately by a distribution).

The chosen alternative subdivision covers 5 age groups:

- Up to age 45
- Between 45 and 65
- Between 65 and 75
- Between 75 and 100
- Over 100

and results in the distribution shown in Figure 4.12.

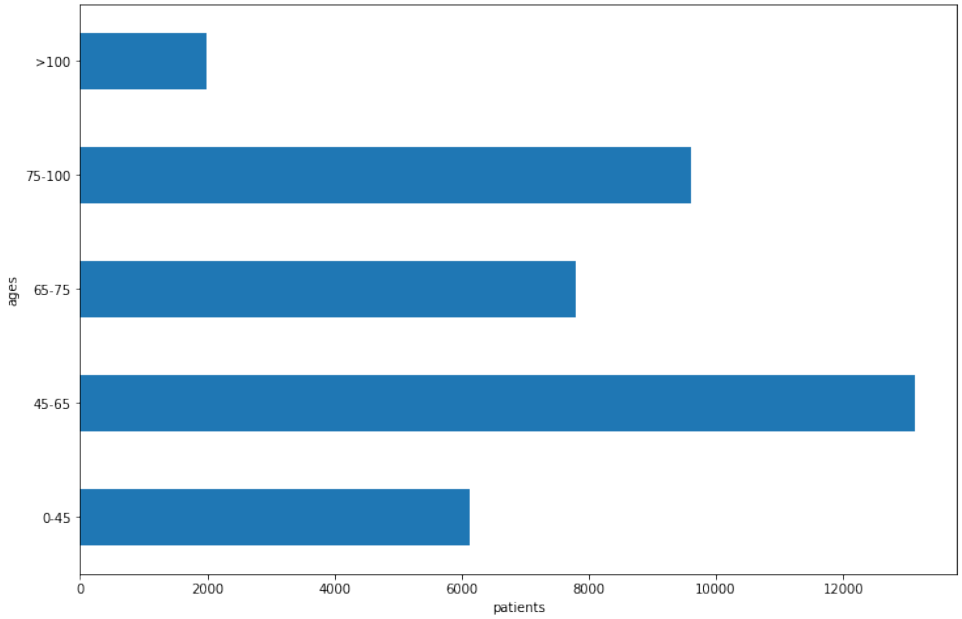


Figure 4.12: Bar plot showing the ages of the patients registered in MIMIC-III at their first admission split into bins.

The groups chosen are not perfectly divided, but they have medical relevance as they represent age groups with different diseases and disorders. The *over 100* group, in particular, was not aggregated with the other groups because it represents a group of patients with a higher risk of hospital death, for which different distributions would probably be applied.

4.4.2. Admission-based Classification

The second and last classification step performed is centered around the hospital stays registered in MIMIC-III.

Having only the information regarding the beginning and end of hospital stays, the only features we can extract are those not touched by the de-identification process, viz:

- The day of the week when the hospital stay begins/ends.
- The time at which the hospital stay begins/ends.
- The approximate season of the hospital stay.

Of these, we chose as a feature for classification the day of the week when the hospital stay began.

Looking at Figure 4.13, which represents the number of hospital stays recorded for each weekday, we can see that hospital stays are well distributed among all days of the week, with a sudden drop during the weekend.

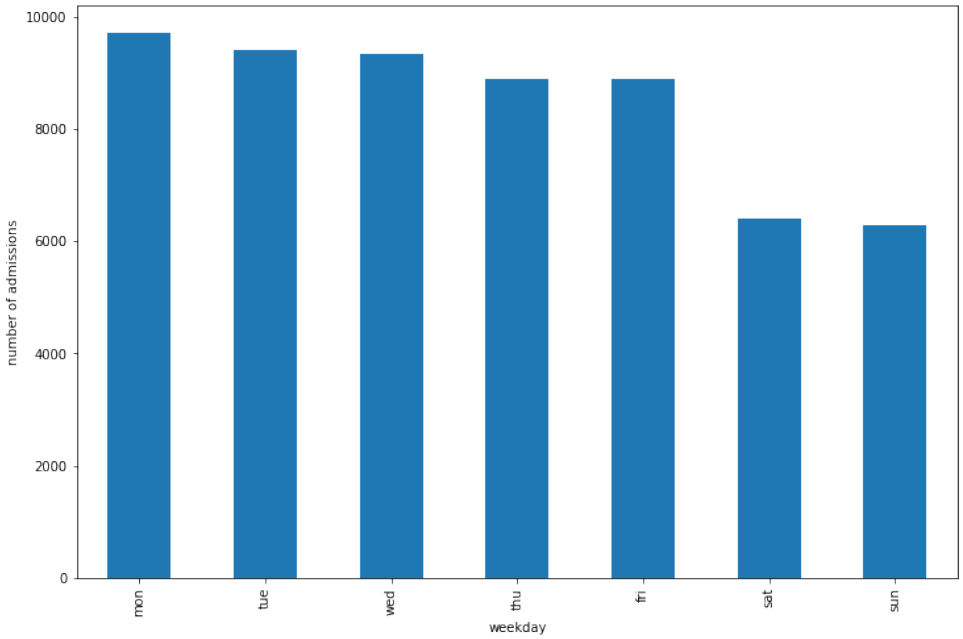


Figure 4.13: Bar plot showing, for each week, the number of admissions.

4.4.3. Summary of the Classifications

To summarize, we have decided to implement our classifications on patients and hospital stays, keeping the focus on considering fairly split and health-related subdivisions.

The features we chose are:

- the gender of the patient, which splits the dataset into 2 groups.
- the age of the patient, which splits the dataset into 5 groups.
- the day of the week when the hospital stay begins, which splits the dataset into 7 groups.

In total, the considered classes amount to $2 * 5 * 7 = 70$ groups in which the dataset has been split.

The results of the chosen classifications are shown in Table 4.1, which presents the number of patients enclosed in the classes identified from the classification made on the patients, and Figure 4.14, which shows the distribution of hospital stays among the various classes.

	M	F
0 - 45	7613	5967
45 - 65	7976	4819
65 - 75	4622	3143
75 - 100	5213	5176
> 100	697	1294

Table 4.1: Table showing the number of patients in each class.

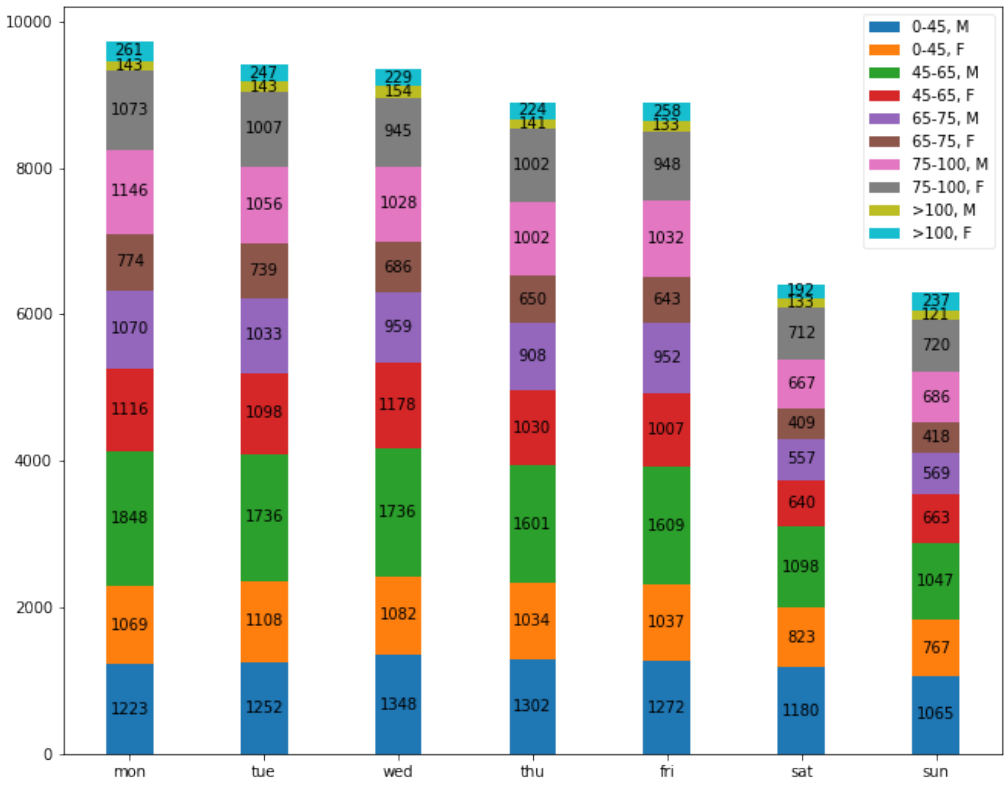


Figure 4.14: Bar plot showing, for each class, the number of admissions.

As we can see, all the groups have a comparable amount of elements, except for the groups related to the *over 100* age group which, as pointed out earlier, contain fewer elements due to the sparsely populated age group.

All classifications, as previously explained, are based on evident and easily usable features of the dataset; the generator, as we will see in Chapter 5, will also be developed to work with classifications based on more complex features.

4.5. Distribution Fitting the Interactions

Once the interactions shown in Section 4.3 have been identified, the next step towards the creation of the generator is to find the distributions that correctly describe their duration.

The classes shown in section 4.4.3 that partition the set of patients and hospital stays are considered in fitting the identified interactions, but not all of them: those interactions identified in the activity diagram shown in Figure 4.8 that indicate borderline cases, due to the scarcity of available samples, cannot be partitioned into all the classes previously listed. Interactions that fall into this category are:

- The time between hospital stays, which although it represents a decreased caseload (most patients endure a single hospital stay, as can be seen in Figure 4.6), since it would only benefit from the patient-based classification (thus reducing the number of classes in which the available samples are split to 10), was deemed appropriate to be split in those groups.
- The hospitalization time in case the patient is not admitted to an ICU.
- The time between subsequent ICU stays, which due to the limited amount of patients with multiple ICU stays was not deemed appropriate to be split based on a classification.

To summarize, Table 4.2 shows which classes are considered for each of the previously identified interactions.

	Patient-based classes	Admission-based classes	Total classes
Time in ICU	10	7	70
Time before the first ICU	10	7	70
Time after the last ICU	10	7	70
Time between hospital stays	10	/	10
Time between subsequent ICU stays	No classes considered		
Duration of the hospital stay without ICU stays	No classes considered		

Table 4.2: Table showing the number of considered classes for each stage of the interaction process of the patient.

To perform the fitting of the identified time intervals, we use Phase Type distributions, specifically Hyper-erlang distributions. These are commonly used for empirical sample fitting as they have the desirable characteristic of being able to approximate any distribution with arbitrary precision.

The tool we use is *HyperStar*, a tool for fitting phase-type distributions to data samples, designed with simplicity and ease of use in mind. The fitting process for phase type distributions is not automated though: in fact, it requires a user interaction step to perform the fitting to a distribution.

Having identified 4 interactions on which to apply the classification previously described and having introduced a considerable amount of classes into which each interaction shall be split, the time required for fitting would be excessive. To overcome this drawback, we choose to group, for each interaction, the classes that can be approximated with the same distribution, so that we can reduce the number of distributions to fit through HyperStar.

This choice is backed up by the fact that the classes found are based on obvious features of patients and hospital stays and thus do not preclude the possibility of some of them following the same distribution between classes.

4.5.1. Class Grouping

To compare the times associated with each class of each interaction and decide whether they are sampled from the same distribution, we use the Kolmogorov-Smirnov 2-samples test. It consists in a non-parametric test that can be used to compare the distributions of two samples and assess whether they were extracted from the same continuous distribution.

The test was performed using the python library *scipy* through the method *ks_2samp*.[3] The method returns the *p-value* of the test, which in our case is a proxy for the probability of the two samples having different distributions, and which shall be compared with a significance level α imposed by us to interpret the result: if the p-value is lower than the threshold α we consider the two samples to have different distribution, otherwise we consider them to come from the same distribution. A more complete explanation of the test can be found in Section 3.3.

Once a lower probability limit α beyond which the test is considered to be successful is set and after it is applied to all possible pairs of classes, we are able to identify which of them have a similar cumulative density function, thus allowing them to be grouped to be fit as a single distribution.

However, it should be kept in mind that the test results are not transitive: if class 'A' exceeds the threshold α when compared with class 'B' (marking them as having the same distribution) and class 'B' exceeds the threshold when compared with class 'C,' it does not imply that the test result applied between class 'A' and class 'C' exceeds the same threshold α.

For this reason, we decided to group into a single distribution only sets of classes in which the Kolmogorov-Smirnov test exceeded the threshold α for each pair of classes within.

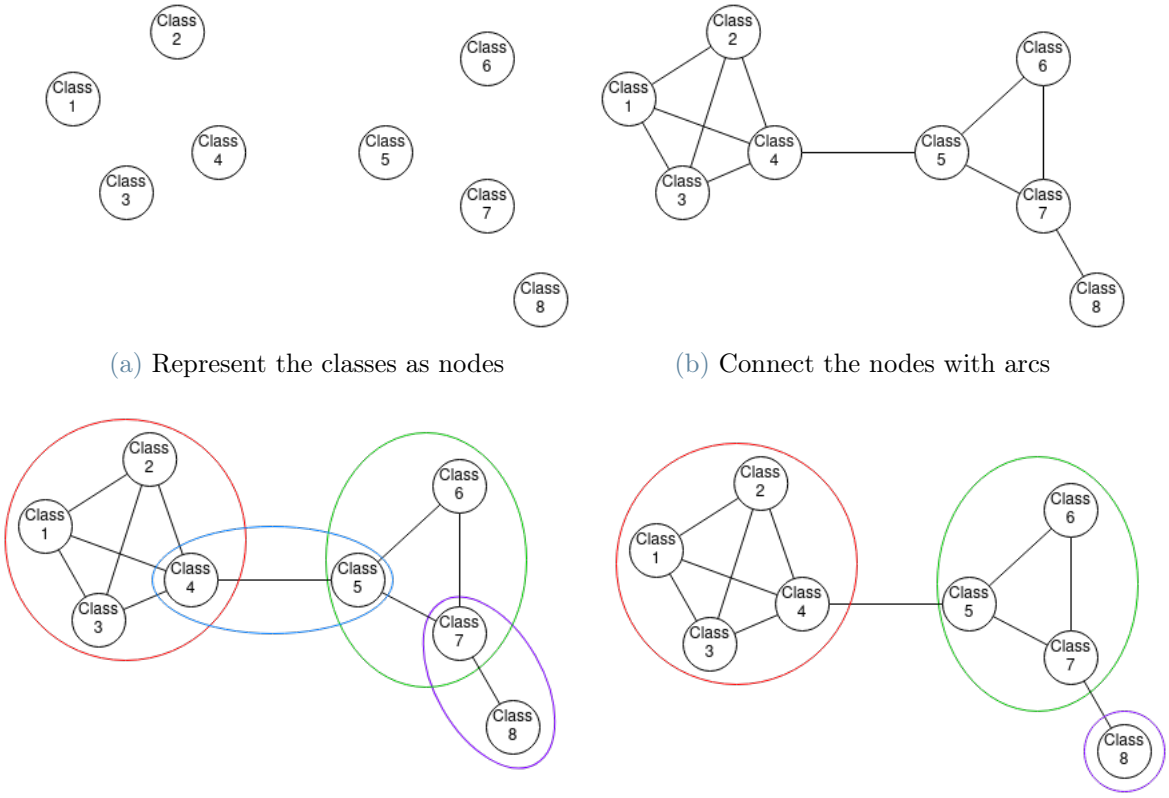
To obtain the aforementioned sets, we used the following method, which exploits the concept of **cliques** in graph theory (also called *maximal complete sub-graphs*) [23]:

1. Represent the available classes as nodes of an un-directed graph (Figure 4.15a).
2. Connect with arcs only those pairs that exceeded the α threshold in the Kolmogorov-Smirnov test (Figure 4.15b).
3. Identify the maximal complete sub-graphs within the graph (Figure 4.15c); these represent the sets of classes in which all each pair of classes passed the Kolmogorov-Smirnov test.

[3] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html`

4. In the case where one or more of the nodes considered belong to several maximal complete sub-graphs at the same time, since they shall not be considered in multiple distinct distributions, assign them only to the maximal complete sub-graph with the largest number of nodes (Figure 4.15d). Due to their completeness, the other sub-graphs will remain complete (or, at most, empty) even after the removal of the offending node.

To represent the available classes as a graph and find its cliques we leveraged the Python module *NetworkX*.[4]



(a) Represent the classes as nodes

(b) Connect the nodes with arcs

(c) Identify the maximal complete sub-graphs

(d) Assign shared nodes to a single sub-graph

Figure 4.15: Visualization of the steps performed to make the groups

Table 4.3 shows the results of the grouping method described above considering an α limit of 0.05 applied on each interaction on which a classification was meant to be applied (according to Table 4.2), showing in particular the reduction in the number of distributions to be fit.

[4] `https://networkx.org/`

	Classes before grouping	Groups formed
Time after the last ICU	70	21
Time before the first ICU	70	27
Time in ICU	70	22
Total	210	70

Table 4.3: Table showing the reduction of the number of distributions to fit.

For each one of the groups identified, we collected samples of the duration of the interaction they refer to for each class belonging to the group and used them to fit a distribution using HyperStar; the mapping between the class and the group it belongs to was then saved in a `.csv` file to be used by the generator later on.

An example of the result of the grouping procedure is shown in Figure 4.16 where we can see how, considering the time spent in the ICU, three classes were grouped together and fitted into a single distribution (shown in figure with a dashed line).

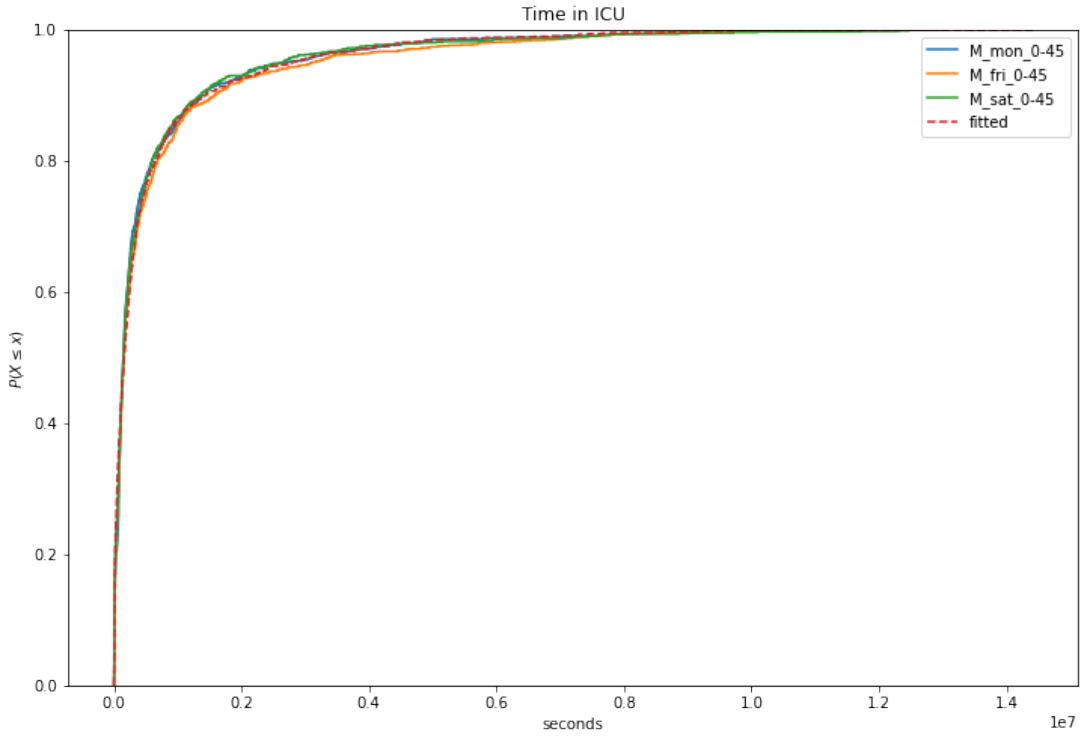


Figure 4.16: ECDF of the classes of one of the groups formed matched against the fitted distribution.

4.6. Distribution Fitting the Events

As mentioned earlier, events belonging to two possible categories are generated during each of the identified interactions:

- events related to a hospital stay
- events related to an ICU stay

As with the case of interactions, these events also require to be fit to distributions in order to be correctly simulated by the synthetic generator.

The classes identified in section 4.4 are considered to split the dataset and perform the fitting of the distribution of identified events. Unlike the procedure seen for the distribution fitting of the patients' interactions, all classes are considered during the partitioning of the samples for the distribution fitting of each event kind.

Due to the de-identification processes applied on MIMIC-III presented in 4.1, we are forced to consider for our fitting purposes the inter-arrival times between two successive events.

The distribution chosen to model them is the *exponential distribution*. It has been chosen because it is commonly used to represent the inter-arrival time in a homogeneous Poisson process, which in turn is commonly used to represent random processes that occur in everyday life [19].

We could have used phase-type distributions as seen in section 4.5 while fitting the duration of the interactions, but the large number of events on which the fitting had to be performed and the number of classes considered, combined with the time required to perform the fitting via HyperStar presented earlier, would have taken an excessive amount of time, even after applying the grouping procedure previously shown.

The fitting of exponential distributions can be carried out with several methods (least squares method, maximum likelihood method, moments method); since in the literature [9] the fittings highlighted as most unbiased are the method of moments and the maximum likelihood method, we decided to apply the method of moments to carry out the fitting since, in case of the exponential distribution, they lead to the same result.

The method of moments consists in the estimation of the parameters of the distribution to be fit by comparing each empirical moment recorded from the samples with the analytical moment of the distribution.

In the case of the inter-arrival time between events of a Poisson process, expressed accord-

ing to the formula $f(t) = \lambda e^{-\lambda t}$, the only parameter to be estimated is λ and, therefore, it is sufficient to use the first empirical moment of the considered samples to derive it.

Since the first moment of an exponential is $E[X] = \frac{1}{\lambda}$, it is sufficient to derive the inverse of the mean (a.k.a. first moment) of the considered sample to estimate the parameter to be used and, consequently, obtain the fit distribution.

This process can clearly be easily automated once the inter-arrival times of the event kind under consideration are obtained.

Since MIMIC-III does not focus on the temporal features of the events, there is no common indication available of the time instant at which the event was observed. Instead, some events have a `charttime` available, which is the time instant at which the event was recorded; others have a `storetime` available, which is the time instant at which the event was introduced into the hospital system; others again have other time indications, and should be considered separately.

The *standard procedure* applied for fitting an exponential to each type of event is thus as follows:

1. Get the events in that category.
2. Split them by the classes previously chosen.
3. Compute their inter arrival time based on the attribute that provides the most accurate time stamp.
4. Find the average of the collected samples.
5. Use the average found to estimate the λ parameter of an exponential distribution via the method of moments described earlier.

The obtained result is, for each kind of event and for each class, a fitted exponential distribution. For most of the events considered, the methodology given above is sufficient, and Table 4.4 shows the attributes used to perform the fitting for these events and the table from which such attributes were extracted. As it can be seen, when available the attribute `storetime` was preferred over other attributes, as it is indicative of the time at which the actual registration of the event in the hospital data system happened.

Event kind	Table	Attribute considered
Lab Events	LABEVENTS	charttime
Datetime Events	DATETIMEEVENTS	storetime
Output Events	OUTPUTEVENTS	storetime
Procedure Events	PROCEDUREEVENTS_MV	storetime
Service Events	SERVICES	transfertime
Chart Events	CHARTEVENTS	storetime

Table 4.4: Table showing the attributes considered during the fitting of events using the standard fitting procedure.

Cases not covered by the procedure just shown are presented in the following sections.

CPT Events and Prescription Events

The `CPTEVENTS` table contains a list of which current procedural terminology codes were billed for which patients.

The only temporal information available about these events is provided by the `chartdate` attribute, which (as the name implies) provides a date but not a precise time stamp.

The same situation applies to the events marked in the `PRESCRIPTION` table which, although they have the attributes `startdate` and `enddate` available to indicate the start and end date of the prescription, do not have more precise indications regarding the time instant at which they were recorded.

Callout Events

The `CALLOUT` table provides information about ICU discharge planning. When a patient is deemed ready to leave the ICU, they are "called out". This process involves:

- a care provider registering that the patient is ready to leave the ICU and detailing any specialized precautions required.
- a coordinator acknowledging the patient requires a bed outside the ward.
- an outcome: either the patient is called out (discharged) or the call out event is canceled.

Each of the steps listed above involves an interaction with the hospital system at different

times, recorded in the `CALLOUTS` table through the following attributes:

- `createtime` provides the time at which the registration of the callout request was performed.
- `acknowledgetime` is the time at which the coordinator acknowledged the callout.
- `updatetime` provides the last time at which the callout event was updated.
- `outcometime` is the time at which the outcome of the callout was registered.

The only attributes that are ensured to be registered for each outcome are the `createtime` and the `outcometime` and, as such, they are the only ones we considered during this analysis.

We opted to model the outcome and the creation of the callout as two distinct events, but related one another.

In fact, while the creation of the callout was modeled with the same standard procedure explained previously, where the time between callout creations was taken as the interval to model, the outcome of the callout was modeled based on the time between the creation of each callout and its outcome. This way in the generator, when the generation of the callout event is performed, we can generate the associated outcome event too.

Figure 4.17 summarises what time intervals are considered for fitting both outcome and creation events.

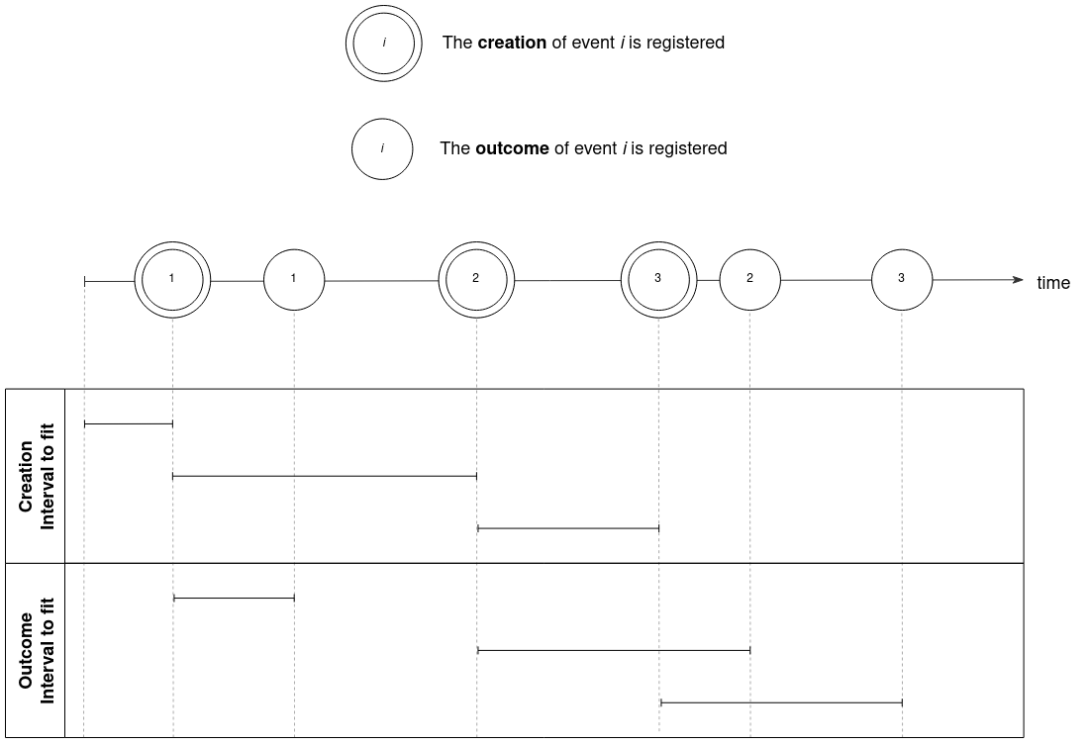


Figure 4.17: Intervals considered for fitting the creation of a callout and the registration of its outcome.

Input Events

Tables `INPUTEVENTS_MV` and `INPUTEVENTS_CV` track the fluids administered to patients during each of their ICU stays. The two tables represent data collected from two separate critical care information systems, namely CareVue and Metavision.

Due to discrepancies in the way the data from the two tables was stored, they could not be integrated. However, since our work is focused on studying the timing of interactions with the hospital system and since both tables possess the `storetime` attribute containing the time stamp of the recording in the hospital system of the event, the two tables were considered together during fitting.

The method then followed for fitting the collected events was the same as seen in the standard procedure presented earlier.

Transfer Events

The `TRANSFERS` table monitors the location of the patients throughout their hospital stay. The `ICUSTAYS` table is derived from this table, but multiple transfer events might happen

during the same stay in ICU.

Each transfer event is associated with an `intime` and an `outtime` that represent the amount of time the patient stayed in the considered ICU ward. Due to the way the table is constructed, there's no time interval between the `outtime` of a transfer and the `intime` of the next transfer during the same ICU stay.

We opted to model the transfer events with a single time reference that represents the time at which the transfer happened during the single ICU stay. The time interval used to model the transfer events is the interval between the `intime` and the `outtime` of the transfer during the same ICU stay. Figure 4.18 visualizes such time interval.

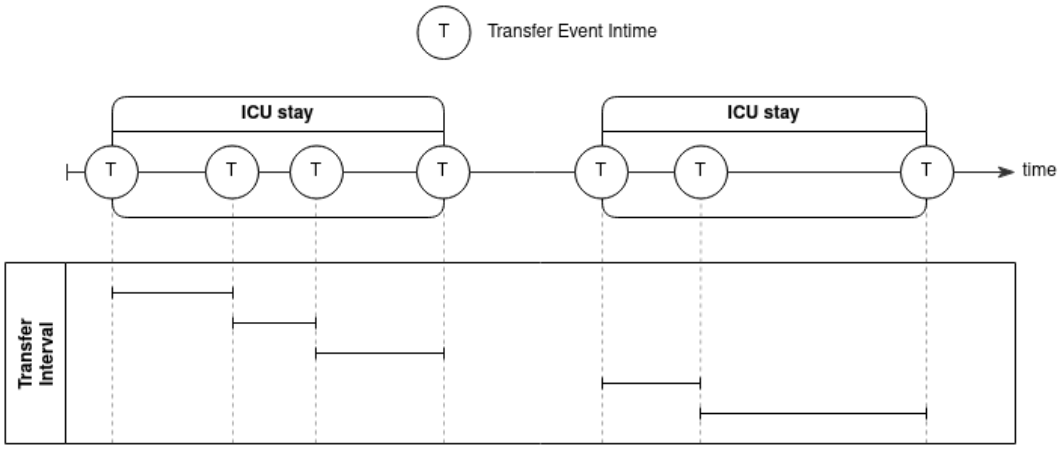


Figure 4.18: Intervals considered for fitting transfer events.

During the generation of the events in the generator, the generated time interval was corrected to avoid transfers for a certain ICU stay happening outside of the time dedicated to the ICU stay.

Note Events

The `NOTEEVENTS` table contains all notes written by the medical staff of the medical institute for their patients. This kind of events, due to their natural language constitution, represents a first important example of heterogeneous data stored in the hospital system of the Beth Israel Medical Institute and recorded by MIMIC.

As many other tables, `NOTEEVENTS` has available both a `charttime` attribute and a `storetime` attribute. Although in these cases we usually prefer to consider the `storetime` attribute as more reliable, MIMIC-III's documentation clearly states that certain kinds of medical notes (namely discharge summaries, ECG notes, radiology notes, and echo notes)

don't have a `storetime` associated but only a `charttime`.[5]

For this reason, we opted to consider the `storetime` attribute when available as a valid time indication and, otherwise, to use the `charttime` attribute. The way Note events are fit does not otherwise change over the standard procedure previously presented.

4.7. Distribution Fitting the Waveforms

Although not events (as the definition in 3.1.2 states) registered in the tables of MIMIC-III, waveforms pose another important source of data. Due to their structure fundamentally different than the rest of the dataset, they had to be pre-processed as shown in Section 4.2.2 to make a correct time model about them.

The results of the pre-processing procedure are in the form of a table that captures the duration of each transmission, the number of samples registered during such transmission, the signals recorded during each transmission and the length of the pauses between each transmission.

The objective distribution we wanted to fit were:

- The duration of the pauses between each transmission.
- The duration of the transmission.

Both of these distributions were split into the classes found in Section 4.4 and fit with an exponential distribution through the method of moments (as the case with all the events presented in section 4.6) described previously. A visualization of the intervals considered is shown in Figure 4.19.

In order to make our generator richer and more featureful, we also leveraged the information gathered during the pre-processing phase to include in each generated transmission the signals it would have registered in the data system.

In this regard, the probability of each transmission to carry a certain signal was computed by checking the number of transmissions carrying that signal in the considered group against the total number of transmission in the group.

4.8. Outputs of the Analysis

To summarize, the analyses performed in this section allowed us to obtain:

[5]The documentation for the Note Events can be found at `https://mimic.mit.edu/docs/iii/tables/noteevents/`

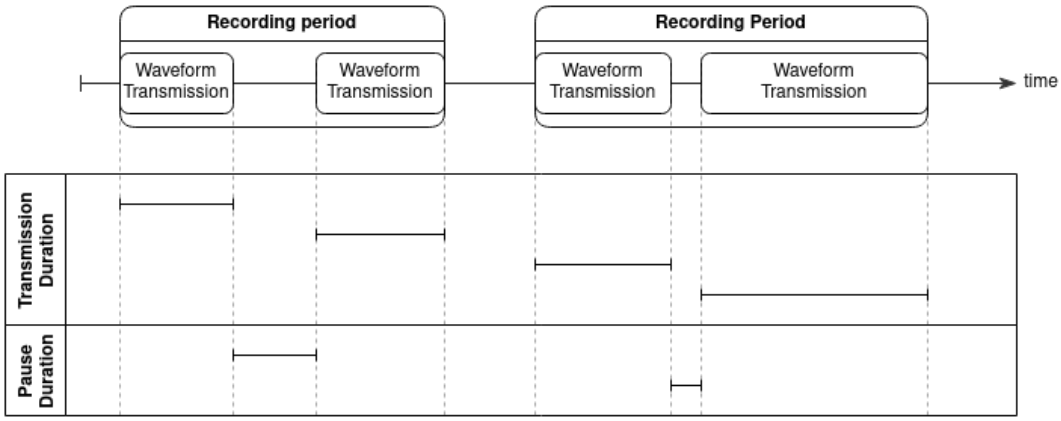


Figure 4.19: Intervals considered for fitting the waveforms.

- The distributions (fit using Phase-Type distributions) of the duration of each interaction previously identified.
- The distributions (fit using exponential distributions) of the inter-time elapsing between successive events of a given kind.

Both categories of distributions were divided into classes. In the case of distributions that fit the interactions, because of the time required to perform the fit, some of them were compared and grouped by similarity so that the number of fittings to be performed could be reduced (see Table 4.3); others, because of the small number of such activities recorded within MIMIC-III, were not divided into classes or underwent a partial division (see Table 4.2).

The outputs of the analyses performed are:

1. For all distributions concerning interactions that have undergone a classification procedure, an HyperStar output file was created for each group (see Section 3.2.1 for the specification of the output of HyperStar) and one mapping file to track the association between the group and the classes.
2. A HyperStar output file for each class for all distributions regarding activities that have undergone partial classification.
3. A single Hyperstar output file for activities that have not undergone classification.
4. For each exponential distribution used to fit the events in MIMIC, a file bearing the estimated lambda parameters for each class.
5. For the exponential distributions used to fit the Waveforms, a file bearing the estimated lambda parameters for each class.

6. For each of the two classifications made (patient based and admission based, seen in section 4.4) a file containing the probability for a patient or admission to belong to each class.
7. For each decision node in the patient activity diagram presented in Figure 4.8, a file containing the probabilities (for each patient class and admission class) of taking one decision branch rather than the other.
8. A single file containing, for each possible signal recorded by the Waveforms, the probability for each class that such signal is transmitted.

These outputs will be the reference parameters read by the generator described in Chapter 5.

The association between the class and the output of the analysis is shown, when in a single file, by writing the class name (written as e.g. "M_tue_0-45" for the class comprising male patients between age 0-45 and who started their first admission on a tuesday) with the considered output. In Figure 4.20 is shown an example of such behavior, where the probability of not having any ICU stays is associated with the class the samples were drawn from by prepending the class name to the computed probability, with a comma as the separator.

Clearly, to make the outputs of the analysis easily readable by the generator, the chosen class names have to be standardized. The proposed standard, as shown in Figure 4.20, is "patientGender_firstStayWeekDay_ageGroup", but any standard (possibly based on the classification features chosen) can be adopted.

```
F_mon_0-45,0.02806361085126286
F_tue_0-45,0.019855595667870037
F_wed_0-45,0.02033271719038817
F_thu_0-45,0.0183752417794971
F_fri_0-45,0.022179363548698167
F_sat_0-45,0.012150668286755772
F_sun_0-45,0.009126466753585397
M_mon_0-45,0.01553556827473426
M_tue_0-45,0.016773162939297124
M_wed_0-45,0.02373887240356083
M_thu_0-45,0.020737327188940093
M_fri_0-45,0.02122641509433962
M_sat_0-45,0.006779661016949152
M_sun_0-45,0.013145539906103286
F_mon_45-65,0.025089605734767026
F_tue_45-65,0.028233151183970857
F_wed_45-65,0.044142614601018676
F_thu_45-65,0.037864077669902914
F_fri_45-65,0.030784508440913606
F_sat_45-65,0.0109375
F_sun_45-65,0.004524886877828055
M_mon_45-65,0.024891774891774892
M_tue_45-65,0.022465437788018433
M_wed_45-65,0.03225806451612903
M_thu_45-65,0.023735165521549032
M_fri_45-65,0.020509633312616533
```

Figure 4.20: Extract of the analysis output showing the probability of not having an ICU stay for each class of samples.

5 | Development of the Generator

The generator for synthetic data (abbreviated to *generator* in the following) of which the development is described in this chapter focuses on the generation of synthetic time stamped events, following the time distribution of the events and the waveforms contained in the MIMIC-III dataset.

The expected output of the generator is a collection of data points related to time, representing the timestamp at which the recording of each medical measurement occurred within the hospital system. The only exception to this rule are the waveforms, in which the signals related to each transmission are also included in the generated output, as presented in Section 4.7.

Since we want the generator to adapt to the results obtained by its users from the analysis previously presented, and for it to adapt to multiple use-cases, the objective features which we focused on during the development are *fine-tunability* and *adaptability*, in line with the requirements of data and workload diversity for the performance evaluation of Big Data systems [11].

Fine-tunability is intended as the ability of the user to fit the model to his or her own results obtained with the procedures shown in Chapter 4 and to have control over the output obtained from the generator.

Adaptability is intended instead with the broader sense of having the ability to reuse the generator with the results of different analysis procedures (such as classifications made with different methodologies) or even different datasets with a structure similar to that of MIMIC-III.

In the following sections we will describe the implementation and architectural features of the created generator, highlighting the reasons behind our choices, from the perspective of the goals just stated.

5.1. Preliminary Implementation Choices

Given the technical focus of the work, we can expect the main users of the generator to be researchers or, at most, physicians or technicians working in the medical field interested in evaluating the performance of a Big Data system.

In order to make the generator more versatile, to limit its complexity and to encourage custom modifications, considering that the users just mentioned are likely to have the knowledge to do so, we chose to build it as a library, to be used as part of a program or a script.

The language chosen for the implementation was Python, because, being an interpreted language and not strongly typed, it makes it easy to extend and reuse existing libraries without having to add excessive boilerplate code and gives the ability to easily and immediately test the changes made to see if and how they work.

Because it is designed as a library, the generator does not include an output presentation part: it is left to the user to choose if and how to show the results obtained.

The generation of samples of the distributions obtained from the MIMIC-III analysis was done using the `ciw`[1] library to obtain samples of Phase-type distributions and `numpy`[2] to obtain samples of exponential distributions.

5.2. Architecture

The generator is divided into 3 modules, relegated to distinct functionalities:

- a **Configuration module**, which contains the necessary components for reading and managing the outputs of the analysis.
- a **Classification module**, containing the components intended to model the classification made during the analysis phase.
- a **Generation module**, the main one, which contains the components needed to generate the synthetic events.

5.2.1. Configuration Module

In order to simplify the user's customization and use of their analysis results, we decide to centralize the management of the information obtained from the analysis within a single

[1] `https://ciw.readthedocs.io/en/latest/`
[2] `https://numpy.org/`

module. For this reason, the Configuration module contains the components necessary for reading the analysis output files presented in Section 4.8 and making them accessible to the rest of the generator.

The model consists of two main components:

- The **Manager** class, responsible for providing the classes in the other modules with the information contained in the output of the analysis.
- The **configuration dictionary**, containing the file paths where to find the outputs of the analysis.

Since the generator is intended as a library, we chose to store the configuration as a multilevel dictionary, in which the file paths from which the various generator parameters are read are stored and catalogued. The files indicated by this configuration dictionary are read by the `Manager` class.

To avoid the use of hardcoded strings in the other components of the generator, **enumerations** were also introduced to enclose and group the keys of the previously mentioned dictionary.

These enumerations are listed in Table 5.1 alongside their elements and a brief description of the information they refer to.

As it can be seen from the list of enumerations created, some event categories have been considered with separate enumerations, while others have been grouped in the `AdmissionEvents` and `ICUEvents` enumerations. This distinction is aligned with the event categories that required a specific distribution fitting procedure and those that used the default fitting procedure shown in Section 4.6.

The enumerations listed also have the role of modeling the events and interactions considered; as we shall see later in the chapter, when possible they have also been used to make known to the other components of the generator some aspects of the structure of the analysis conducted.

The advantage these enumerations provide is that it is possible to alter the analysis without applying excessive changes to the generator: if, for example, one wanted, as part of a study on workload characterization, to consider Note Events as events specifically related to ICU stays (and not to generic hospital stays as it has been done), one would simply move their item from the `AdmissionEvents` enumeration to the `ICUEvents` enumeration.

Enumeration	Elements	Usage
Probabilities	P_NO_ICU	Probabilities associated to each decision to be made for the phases of a patient's stay.
	P_SUCC_ICU	
	P_SUCC_ADMISSION	
	P_IMM_FINISH	
	P_USER_CLASS	Probabilities of belonging to a certain classification group.
	P_ADMISSION_CLASS	
AdmissionEvents	LAB	Parameters used for the generation of the events registered during a patient's hospital stay.
	MICROBIOLOGY	
	SERVICE	
	NOTEEVENTS	
CalloutEvents	CALLOUT_CREATE	Parameters used for the generation of Callout events.
	CALLOUT_OUTPUT	
ICUEvents	DATETIMEEVENTS	Parameter used for the generation of the events registered during a patient's ICU stay.
	INPUTEVENTS	
	OUTPUTEVENTS	
	PROCEDUREEVENTS	
	CHARTEVENTS	
TransferEvents	TRANSFERS	Parameters for the generation of Transfer events.
WaveformEvents	PAUSE	Parameters for the generation of the waveforms.
	TRANSMISSION	
InteractionKind	ICUSTAY	Parameters for the generation of the duration of each interaction stage.
	POSTICUSTAY	
	PREICUSTAY	
	INTERTIMEICU	
	NOICU	
	INTERTIMEADM	

Table 5.1: Enumerations contained in the Configuration module.

5.2.2. Classification Module

As seen in Section 4.4, the classifications made during the analysis to divide the set of patient events and interactions into groups are based on evident features of patients and hospital stays.

The *classification* module collects the enumerations intended to model the groups obtained from those classifications, subdividing them by the type of classification applied: either a patient-based classification identified by the enumeration `PatientClass` or an admission-based classification identified by the enumeration `AdmissionClass`. The enumerations and their elements are shown in Table 5.2.

Also introduced with them is a function (named `get_class_rep`) used to obtain, from

Enumeration	Elements
PatientClass	M_0-45
	M_45-65
	M_65-75
	M_75-100
	M_100
	F_45-65
	F_65-75
	F_75-100
	F_100
AdmissionClass	mon
	tue
	wed
	thu
	fri
	sat
	sun

Table 5.2: Enumerations contained in the Classification module.

the classes just listed, the standardized name used to identify the classification considered during the creation of the analysis outputs (this standardized class name was previously described in Section 4.8).

In line with the objective feature of *adaptability* highlighted at the beginning of the chapter, in the event that it is decided to consider a classification based on different features, it will suffice to modify the components just described to adapt the generator.

5.2.3. Generation Module

The **generation module** contains the main body of the generator. As we will see in the following sections, it reproduces the structure of the interaction process identified in the analysis to generate the synthetic events.

The classes that make up the module can be divided into two main categories:

- The *Generating Classes*, that focus on the creation process of the synthetic events.
- The *Event Classes*, that model the events we intend to generate.

Generating Classes

The classes concerning the generation of events are the main body of the generator, and the ones that make the most use of the results of the analysis carried out earlier.

The structure of MIMIC and the way its analysis was carried out lends itself to the idea of splitting the generation of the events into **multiple layers**; in fact, we can divide the recording of the events in MIMIC-III into the following steps:

1. The *hospital* receives multiple *patients*.
2. Each patient has none, one or more *hospital admissions*.
3. Each hospital admission corresponds to multiple *interactions* between the patient and the hospital system.
4. During each interaction, multiple *events* are registered.

This same layered structure has been adopted to articulate the generation of the events in the generator: each of the layers just listed in fact corresponds to a different software component, and each component is responsible for instancing the components of the following layer up to the components that represent the interactions, which finally generate the synthetic events.

The goal of this layered structure is to give the user the ability to access each of these layers indiscriminately, allowing them to fine tune the output of the generation process (depending on the target characteristics) according to the desired granularity, in line with the objective of *fine-tunability* introduced at the beginning of the chapter.

The classes intended for the generation of the events are thus:

- `Interaction`, an abstract class extended by the classes `StayInteraction` and `ICUStayInteraction`.
- `Admission`, the class representing a patient's hospital stay, which is responsible for the instancing of the classes that extend the `Interaction` class, following the pattern of patient interaction identified in Section 4.3.
- `Patient`, the class that represents the entire interaction process between the patient and the hospital over the course of multiple hospital stays, which is responsible for the generation of the instances of the `Admission` class listed above.
- `Hospital`, the class representing the entire hospital, which creates instances of the `Patient` class.

Each of these classes represents one of the layers listed above.

All the generative components just listed implement the `EventGenerator` interface; this interface defines the `get_events` and `get_waveforms` methods, used respectively to re-

trieve the events and waveforms that result from the generation. A chevron diagram visualizing the layered architecture is shown in Figure 5.1.

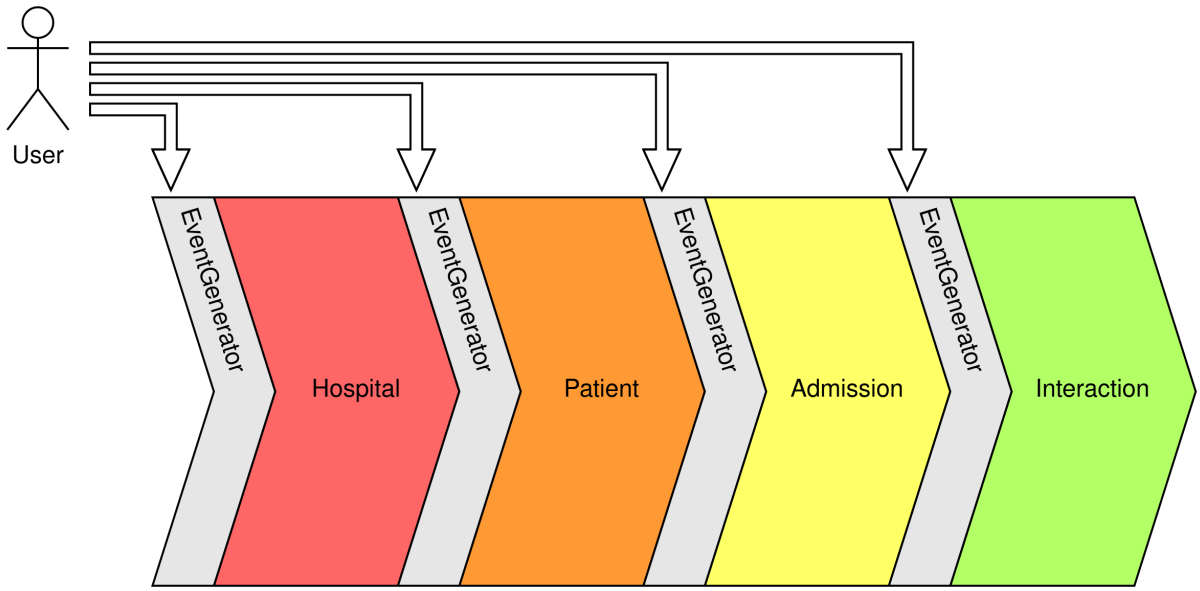


Figure 5.1: Chevron diagram showing the layered architecture used for the Generation module.

When the user instantiates any of these classes, all the classes of the following layers are also created with it. On the other hand, when the user requests the selected layer to generate a collection of synthetic events or waveforms (via the `get_events` and `get_waveforms` methods), the latter requests the components of the following layer to generate the requested output (via the same `get_events` and `get_waveforms` methods). This handover is repeated up to the Interaction layer, which finally generates the synthetic events or waveforms requested.

The classification group of each simulated patient and its identifying id (introduced in similarity to the `subject_id`, the attribute that identifies the patients in MIMIC-III) are chosen by the `Hospital` class, while the classification group of the individual hospital stay (and its identifying id, similar to the `hadm_id` attribute found in MIMIC-III) is chosen by the `Patient` class.

These classification groups are passed to the following layers during the instancing process in the form of the enumerations shown in Table 5.2 and communicated to the `Manager` class. The `Manager`, in turn, uses them to read from the output files of the analysis (shown in Section 4.8) the parameters needed to describe the distributions designed to generate the synthetic events and the duration of each interaction for the specified classification

group.

The entire procedure is summarized in the sequence diagrams shown in Figure 5.3 and Figure 5.2, which represent the process of creating an instance of the `Hospital` class and the process of generating the synthetic events following a request from the user, respectively.

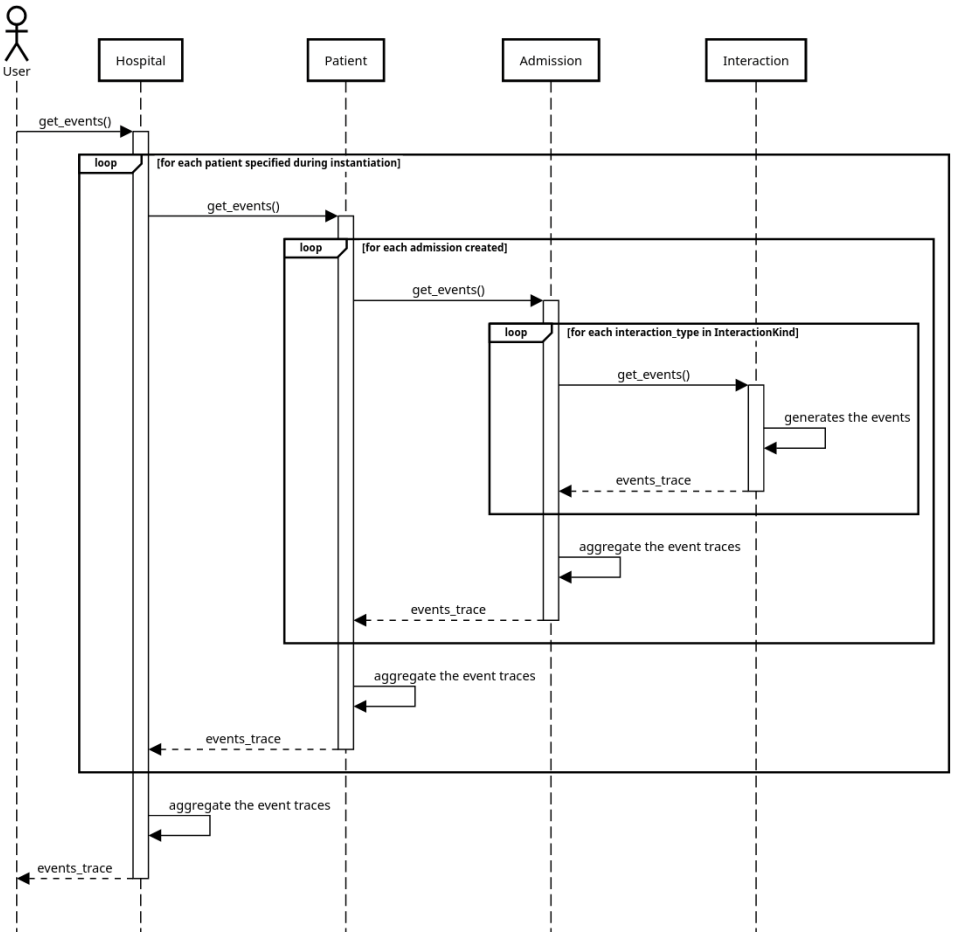


Figure 5.2: Sequence diagram showing the process of generating and obtaining the events.

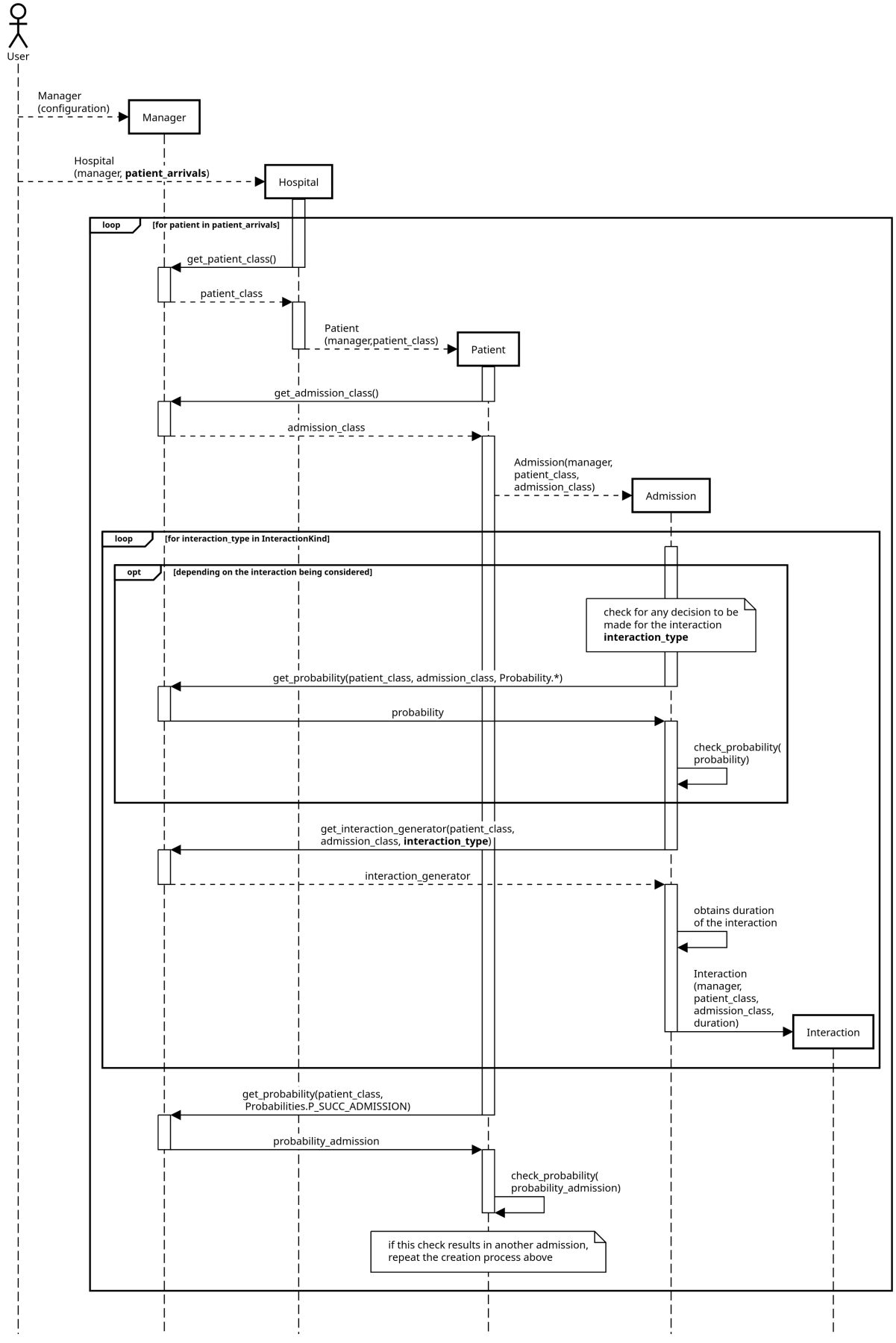


Figure 5.3: Sequence diagram showing the creation of an instance of the Hospital class.

As it can be seen, the `Manager` class is instantiated by the user, provided with the (possibly customized) configuration dictionary described in Section 5.2.1, and is then passed on to the various components of the generation module.

Moreover, the creation of an `Hospital` instance is done by the user by specifying some information regarding the patients to be modeled, in the form of the `patient_arrivals` parameter; this parameter and an instance of the Manager class are the only required parameters to be provided during the instancing of the class.

Much like the Hospital layer, each layer requires several parameters to be provided during construction in order to be able to generate the events. The only common parameter between all the components is the instance of a `Manager` class, as it is necessary to obtain the information resulting from the analysis carried out on MIMIC-III.

The required parameters for each component, accompanied by a brief description, are provided in Table 5.3.

Class	Parameters	Description
Hospital	patient_arrivals	List of floats identifying the time at which each user will start its first hospital stay. The number of users to model in the system is derived from this list.
Patient	patient_class	Class of the patient considered. It must be an element of the enumeration `PatientClass`.
Admission	admission_class	Class of the admission considered. It must be an element of the enumeration `AdmissionClass`.
	patient_class	Derived from the patient that generated this admission or provided by the user.
Interaction	admission_class	Derived from the admission that generated this interaction or provided by the user.
	patient_class	Derived from the patient that generated the admission that generated this interaction or provided by the user.
	duration	Duration of the interaction. Computed by the Admission layer or provided by the user.

Table 5.3: Parameters required by each component of the generation module.

Interaction Classes We pointed out, during the analysis of MIMIC-III, that some types of events occur during a hospital stay, while others specifically during an ICU stay. To reflect this difference, the generation of the interactions was divided into the classes `StayInteraction` and `ICUStayInteraction`, which extend the common abstract class `Interaction`.
Both classes share the `generic_events` method, implemented in the father class; it is responsible for the generation of the event kinds listed in the `ICUEvents` and `AdmissionEvents` enumerations (shown in Table 5.1).
In particular, the `ICUStayInteraction` class uses this method to generate the events listed in the `ICUEvents` enumeration, while the `StayInteraction` class uses it to generate the events listed in the `AdmissionEvents` enumeration.
Each of the two classes also implements the `special_events` method, which is used for the generation of events that represent special cases and require their generation to be treated differently (such as `Callout` and `Transfer` event kinds).
The generation of the waveforms, which in MIMIC-III are associated with the ICU stay and not with the entire hospital stay, is handled by the `ICUEvents` class via the `get_waveforms` method (implemented for the `EventGenerator` interface), which implements the generation of both the waveforms and the related signals following the distributions defined in Section 4.7.

Event Classes

To represent the events within the generator, we chose to collect them within a single class `Event`, and to specify the precise type of event to be considered through the elements of the enumerations highlighted in Section 5.2.1.

When the `get_events` method of any class that implements the `EventsGenerator` interface is called, it returns a collection (in the form of a list) of instances of the `Event` class.

The only relevant method of the `Event` class is the `get_event_dictionary` method, which can be used to obtain the information of interest about the event, such as the synthetic time instant at which it was generated, the classification group it belongs to, the event type, the synthetic patient id (obtained from the `Patient` class) and the id of the synthetic hospital stay (obtained from the `Admission` class) in the form of a key-value dictionary.

The waveforms are represented in the generator by an extension of the `Event` class, called `WaveformsEvent`. The only difference with the `Events` class just described is, in fact, in the method `get_event_dictionary`, for which a method override of the parent class was

applied to include the information about the signals generated during each transmission.

When the `get_waveforms` method of any class that implements the `EventsGenerator` interface is called, it returns a collection (in the form of a list) of instances of the `WaveformEvent` class.

To summarize, a high level UML diagram of the Generation Module is shown in Figure 5.4.

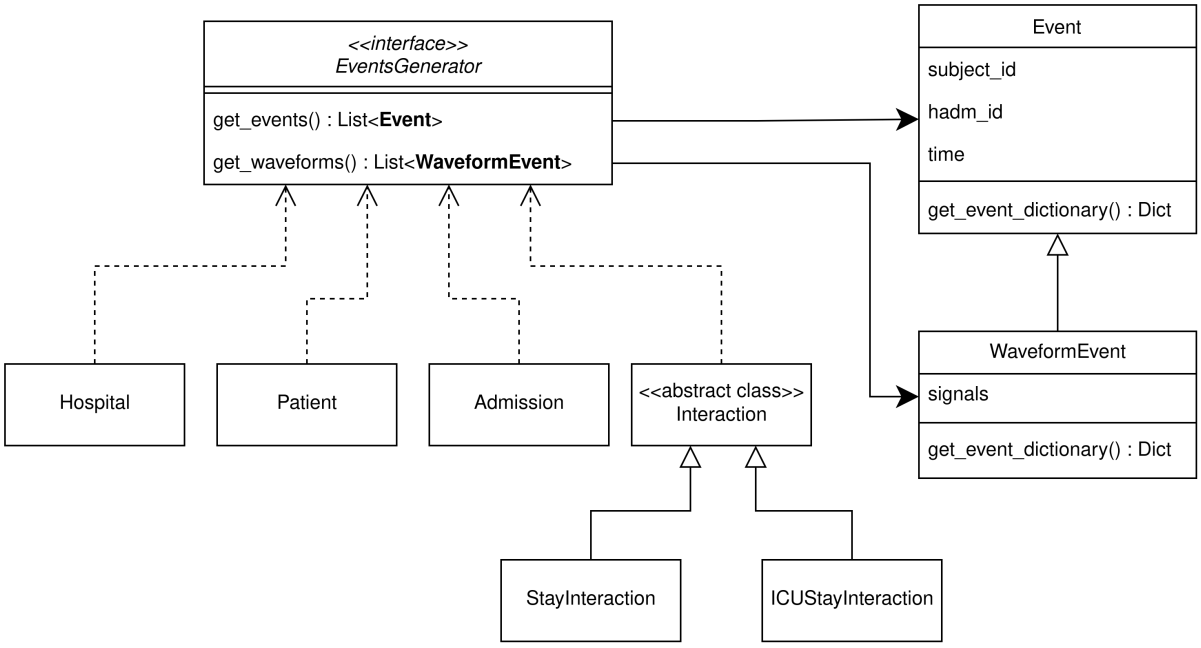


Figure 5.4: UML diagram of the Generation Module.

6 | Experiments and Results

With the aim of showing how to interact with the generator presented in the previous chapter and highlighting its possible usages and features, we present in this chapter two illustrative experiments based on two fictitious scenarios.

Both scenarios involve evaluating the performance of a not yet implemented Big Data system, intended to be used as one of the components of a hospital's information system that records the same data stored in MIMIC-III.

In the **first scenario**, the system is to be used for managing all patient data under consideration. In the **second scenario**, the system is limited to the hospital's ICU data system, and provisionally it is intended to be used to manage only the structured data generated during the ICU stays of the patients.

In both scenarios, our goal is to evaluate the possibilities of downgrading or upgrading the considered system by assessing its performances using the synthetic data generator described in this paper in order to optimize the economic and energy expenditure of the hospital.

The performance analysis and the design of the model is performed in both scenarios using *JMT*, "a suite of tools for performance evaluation of computer systems" [25], described in depth in Section 3.4.

The tool used for this example is *JSimGraph*, a graphical component of *JMT* specialized in systems modeling and performance evaluation using queuing networks and capable of handling multiple job classes.

As we can see in Figure 6.1, the queuing network used to model the reference architecture consists of two queues:

- The *Ingestion* queue, which represents the ingestion activity performed by the Big Data system, in which data gathered from different sources are brought into a single storage.
- The *Processing* queue, which represents the processing activity performed by the

Big Data system, during which data are processed to provide it with a common relational structure.

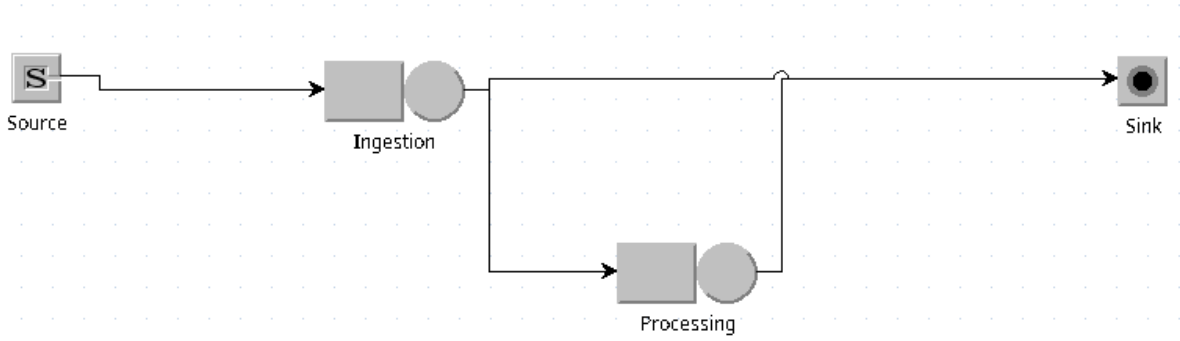


Figure 6.1: Queuing network constructed in JSimGraph to model the system of the first scenario.

Most of the data in MIMIC-III already have a relational structured form, except for two kinds: waveforms, which are recorded as signals, and Note Events, which contain a textual part from which we want to extract features to be saved in a structured format. For this reason, only these two categories of data go through the *Processing* queue of the model, while all data contained in MIMIC-III go through the *Ingestion* queue.

Given the way the events and the waveforms are distributed among the various queues, we use three job classes to model them:

- A class to represent structured data, i.e., all the events collected in MIMIC-III except for Note events.
- A class to represent the waveforms.
- A class to represent the Note Events.

The second scenario, which as mentioned does not include the unstructured data, does not use the *Processing* queue of the second model (a more specific model to represent the system for this scenario is shown in Figure 6.2) and, for the same reason, uses only the job class that represents the structured data.

The service times of each queue remain unchanged between the two scenarios: due to the lack of a physical reference architecture, we used exponential distributions to track the service times of the system for each job class, of which the average values chosen are shown in Table 6.1. Note that each and every queue is comprised of a single service center.

The average service times considered above were obtained from other studies conducted

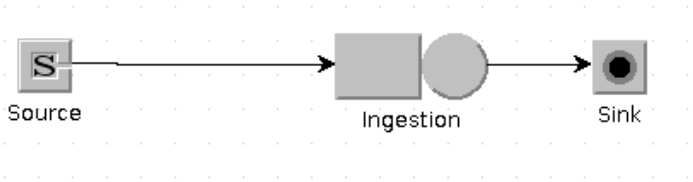


Figure 6.2: Queuing network constructed in JSimGraph to model the system of the second scenario.

Class	Ingestion average service time	Processing average service time
Structured Events	0.448	/
Note Events	0.693	133.453
Waveforms	12.409	69.388

Table 6.1: Service times associated with each job class for each station.

within the Health Big Data project [6].

6.1. Data Generation

The procedure followed to generate the collections of events and waveforms to be supplied to JSimGraph via our synthetic generator varies depending on the scenario under consideration.

For the **first scenario**, since the Big Data system is intended to cover data generated by all patients in the hospital, this procedure follows the sequence diagrams described in Figure 5.3 and Figure 5.2.

We perform event generation using the `Hospital` class, specifically requesting the generation of events for 250 users beginning their hospital stay 10 seconds apart from one another. After that, through the `get_events` and `get_waveforms` methods, we obtain the required data.

Two excerpts of the events and waveforms generated for this scenario are depicted in Figure 6.3 and Figure 6.4, encapsulated within a *dataframe* – a two-dimensional, size-mutable, potentially heterogeneous tabular data container, provided by the Pandas python library[1] – for ease of visualization.

[1] `https://pandas.pydata.org/`

time	class	patient_id	admission_id	event
47.30	F_sat_100	1	1	TRANSFERS
116.78	M_sat_75-100	5	9	LAB
136.46	F_sat_100	1	1	DATETIMEEVENTS
145.53	F_sat_100	1	1	CHARTEVENTS
152.06	F_sat_100	1	1	CHARTEVENTS
...	...	...	...	...
306335178.90	M_fri_65-75	59	75	LAB
306342085.17	M_fri_65-75	59	75	LAB
306348534.32	M_fri_65-75	59	75	NOTEEVENTS
306351029.95	M_fri_65-75	59	75	LAB
306353917.50	M_fri_65-75	59	75	LAB

Figure 6.3: Excerpt of the events provided by the generator for the first scenario.

time	class	patient_id	admission_id	event	duration	transmissions	signals
191.80	M_sat_0-45	7	11	WAVEFORM_TRANSMISSION	10533.89	1316736	[II]
244.89	M_sat_45-65	17	22	WAVEFORM_TRANSMISSION	6808.50	851062	[II]
433.14	M_sat_0-45	0	0	WAVEFORM_TRANSMISSION	12071.09	1508886	[II]
460.94	F_sat_100	35	45	WAVEFORM_TRANSMISSION	20294.14	2536767	[ABP, AVF, AVL, AVR, CVP, I, II, III, MCL, PLE...
469.80	M_sat_45-65	39	50	WAVEFORM_TRANSMISSION	10046.35	1255793	[ABP, AVR, I, II, III, PLETH, RESP, V]
...	...	...	...	...	...	...	...
151953947.22	M_fri_65-75	59	75	WAVEFORM_TRANSMISSION	3806.99	475874	[]
151957832.06	M_fri_65-75	59	75	WAVEFORM_TRANSMISSION	3542.54	442817	[II]
151961719.28	M_fri_65-75	59	75	WAVEFORM_TRANSMISSION	299.60	37449	[II]
151962241.69	M_fri_65-75	59	75	WAVEFORM_TRANSMISSION	10451.43	1306429	[II]
151973000.63	M_fri_65-75	59	75	WAVEFORM_TRANSMISSION	30453.54	3806692	[ABP, AVR, I, II, III, PLETH, RESP, V]

Figure 6.4: Excerpt of the waveforms provided by the generator for the first scenario.

For the **second scenario**, since the Big Data system is intended to cover only the structured data generated during the patients' ICU stays, the class chosen for the generation of the events is `ICUInteraction`, which as pointed out in the previous chapter is used to generate the events pertinent to patients' ICU stays. In this case, the generation procedure was carried out considering 250 users, each with two admissions and each admission corresponding to two ICU stays.

With the aim of showing the adaptability of the generator, in this scenario we choose to simulate a change of the classification features considered during the analysis of MIMIC-III: instead of dividing the ages of patients as seen in Section 4.4.1, we use the age groups

"0-20 years old", "20-40 years old", "40-60 years old", "60-80 years old", and "over 80 years old".

To avoid redoing the distribution fitting from scratch to no avail, we change the output files of the analysis to simulate the new classifications, but kept the same values for the obtained parameters.

The only needed modification to the source code of the generator to make it work correctly with the new classification is to change the enumeration `PatientClass` described in Section 5.2.2 to model the new classes.

Finally, the length of the stays was chosen randomly for each stay between 1 and 10 hours and the classification group of the user and the admissions was chosen randomly between the available classes.

An excerpt of the events generated for this scenario is shown in Figure 6.5. The Note Events, being associated with the admission of the patient, are not created by the class `ICUInteraction` and do not appear in the output of the generator for this scenario.

As we can see from column `class` in the excerpt, the new classifications are correctly incorporated in the output of the generator.

	time	class	patient_id	admission_id	event
0	0.00	M_sun_60-80	0	0	TRANSFERS
1	24.00	M_sat_0-20	1	0	TRANSFERS
2	30.00	M_mon_60-80	0	1	TRANSFERS
3	31.00	M_mon_80	15	1	TRANSFERS
4	72.00	F_fri_40-60	9	0	TRANSFERS
...	...	...	...	...	...
103507	147344.92	F_mon_40-60	239	0	CHARTEVENTS
103508	147454.44	F_mon_40-60	239	0	CHARTEVENTS
103509	147489.30	F_mon_40-60	239	0	CHARTEVENTS
103510	147518.78	F_mon_40-60	239	0	CHARTEVENTS
103511	147584.00	F_mon_40-60	239	0	TRANSFERS

Figure 6.5: Excerpt of the events created by the generator for the second scenario.

Before we can discuss the procedures used for the performance evaluation of the modeled system for the two scenarios, it is necessary to prepare the newly obtained data so that it can be read by JSimGraph.

The tool, in fact, expects an input file for each job class. Each of these files, moreover, must consist of float values separated by commas representing the time elapsed between two consecutive events of the same job class.

For this reason, the events obtained from the generator are processed following the steps listed below for both scenarios:

1. The results obtained are divided among the job classes previously identified in Table 6.1. For the second scenario this step is not necessary, since the synthetic events generated belong only to the structured job class.
2. From these only the information regarding timing is selected (which, as shown in Figure 6.3 and Figure 6.4, is the data provided by the `time` column).
3. From the timings just derived, the inter-time between each event is obtained.

Having done so, the processed events can be written to file and given as an input to JSimGraph.

6.2. Performance evaluation

The performance evaluation of the system was carried out following the same procedure shown below in both scenarios:

1. Perform a single simulation run via JSimGraph to ascertain the model performance index values with the service times described in Table 6.1.
2. Based on the results of that run, decide how to modify the service times to simulate a change in architecture and observe the new performance indices obtained.

To perform the second step, JSimGraph provides a tool called *what-if analysis*, which allows us to change one or more control parameters within a specified range in incremental steps. In our case, the tool would allow us to compute the performance indices for multiple different service times, giving us the ability to easily compare the results obtained.

The performance indices considered for each of the two scenarios are:

- The system's response time.
- In case the model has multiple queues, the response time of each of them.
- The system's throughput.
- The utilization of each of the queues.

6.2.1. First Scenario

Applying the first step of the performance evaluation procedure shown above in the first scenario, the index of greatest interest is utilization. Indeed, the simulation results show that: *a)* the *processing* queue has a very low utilization, around 0.012; *b)* the *ingestion* queue has a rather high utilization, around 0.89.

Based on these results, we follow up with the second step and perform a what-if analysis by increasing the service time of the processing station by 50% in 5 successive steps, thus simulating an architectural downgrade (since it would require more time to perform each job).

The results of this procedure can be seen in the following paragraphs, where for each performance index listed above the values as service time changes are shown.

Utilization In Figure 6.6 and Figure 6.7 are shown, respectively, the utilization of the *Processing* and *Ingestion* queues throughout the change in the service time of the *Processing* queue. As we can see, the utilization of the *Processing* queue increases in line with the increase in queue service time, as it should. The utilization of the *Ingestion* queue, instead, oscillates in the range between 0.89 and 0.91, which indicates that it has not been affected by the modification. This result was predictable, since jobs that receive service from the *Processing* queue then conclude their path within the system and thus do not affect the *Ingestion* queue.

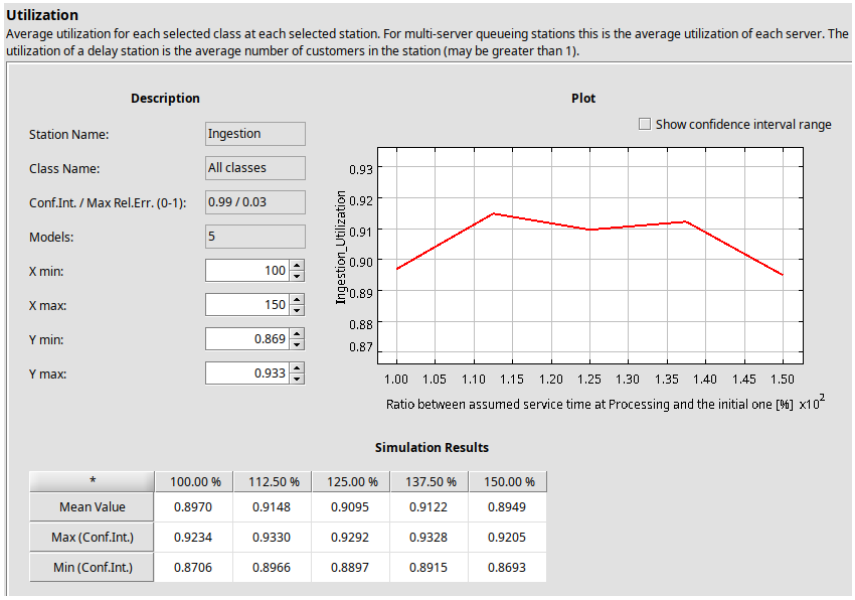


Figure 6.6: Utilization of the *Ingestion* queue (first scenario)

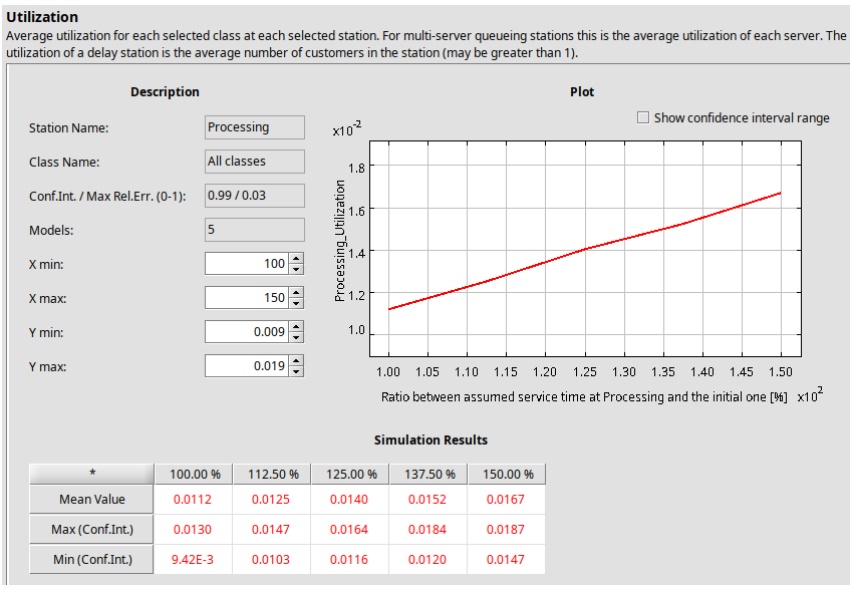


Figure 6.7: Utilization of the *Processing* queue (first scenario)

Throughput In Figure 6.8 the throughput of the system is shown. As we can see, it has remained more or less stable, maintaining values between 1.95 jobs/second and 1.96 jobs/second.

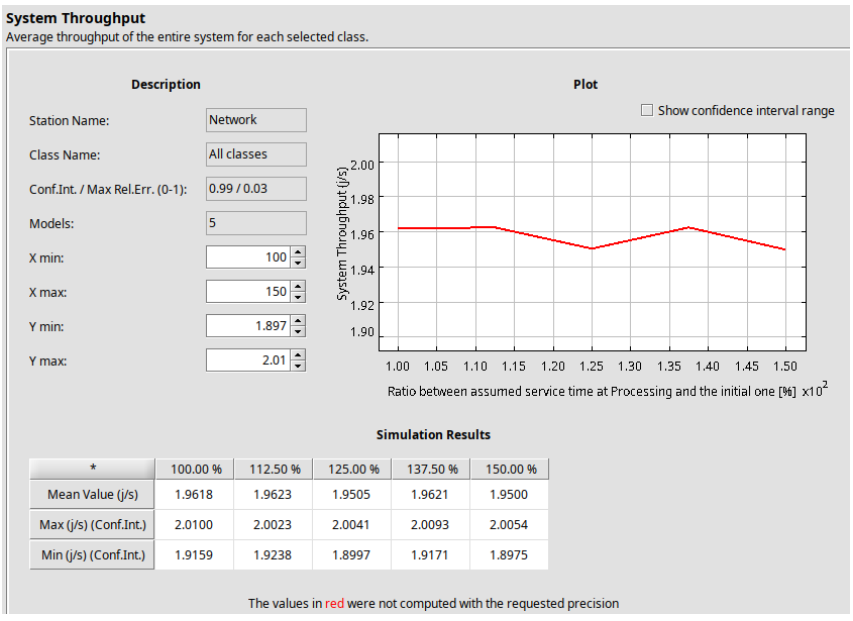


Figure 6.8: Throughput of the system (first scenario)

Response time In Figure 6.9 and Figure 6.10 are shown, respectively, the response times of the *Processing* and *Ingestion* queues throughout the change in the service time of the *Processing* queue. As expected, the response time of the *Processing* queue increases

with the service time of the queue, while the service time of the *Ingestion* queue does not change substantially. In Figure 6.11 we can see the response time of the system, which shows that the total response time increases. This result is expected seen the evolution of the response times of the two queues.

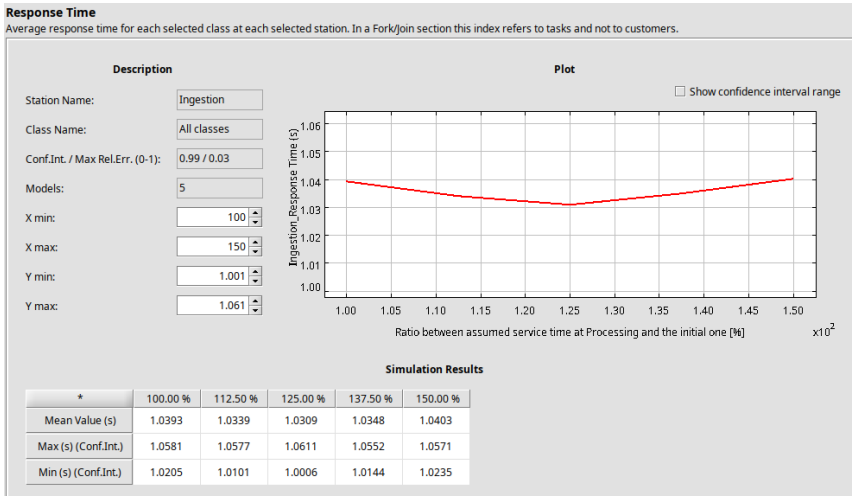


Figure 6.9: Response time of the *Ingestion* queue (first scenario)

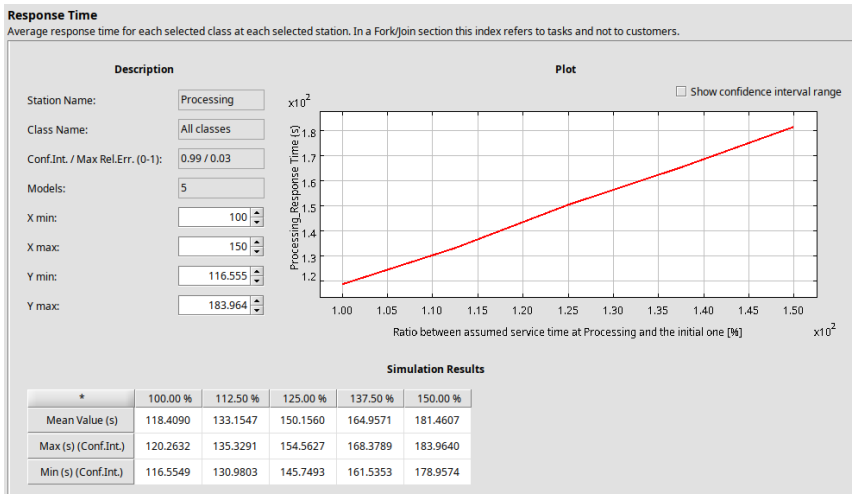


Figure 6.10: Response time of the *Processing* queue (first scenario)

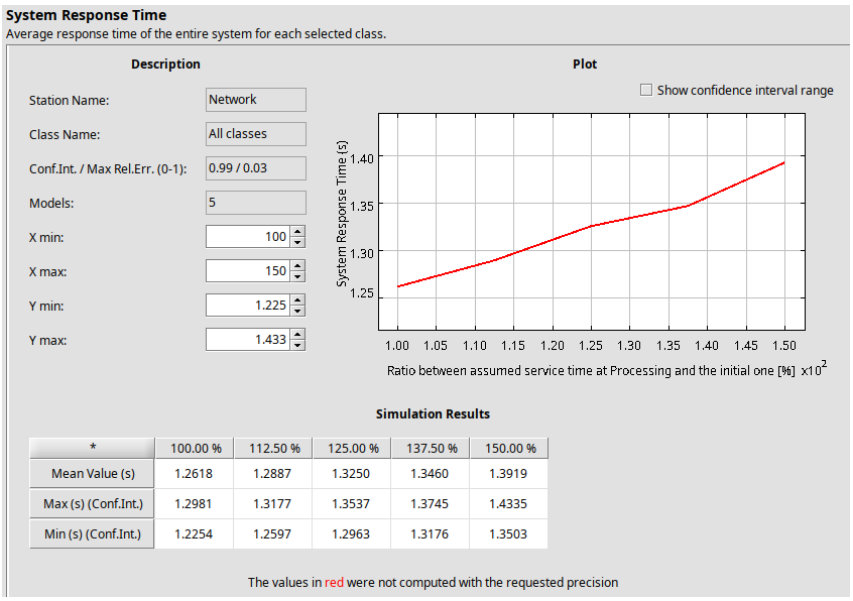


Figure 6.11: System response time (first scenario)

6.2.2. Second Scenario

Applying the first step in the second scenario, again, we observe that the index of greatest interest is the utilization: Although there is only one queue, it is stable with a value around 0.33.

This leads us to think that, as long as the utilization remains low, it is once again possible to downgrade the architecture.

We then perform the what-if analysis by increasing the service time in 5 successive steps from 0.448 seconds to 1 second (an increase of 123%).

The results of this procedure can be seen below, where for each performance index listed above the values as service time changes are shown.

Utilization In Figure 6.12 is shown the utilization of the *Ingestion* queue for each of the service times considered. As expected, the utilization increases with the change in the service time. The rate of this increase ramps up at the 0.72 seconds mark, pushing the utilization to 1 on the 0.862 mark.

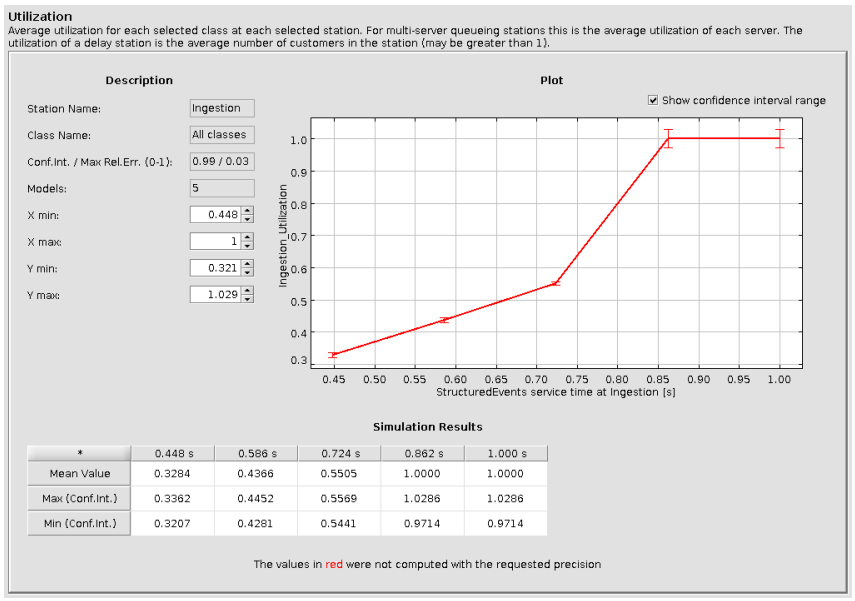


Figure 6.12: Utilization of the ingestion queue (second scenario)

Throughput In Figure 6.13 is shown the change in the system throughput associated with the increase in the service time of the ingestion queue. The throughput remains unaffected until the 0.586 seconds mark, after which it begins its descent.

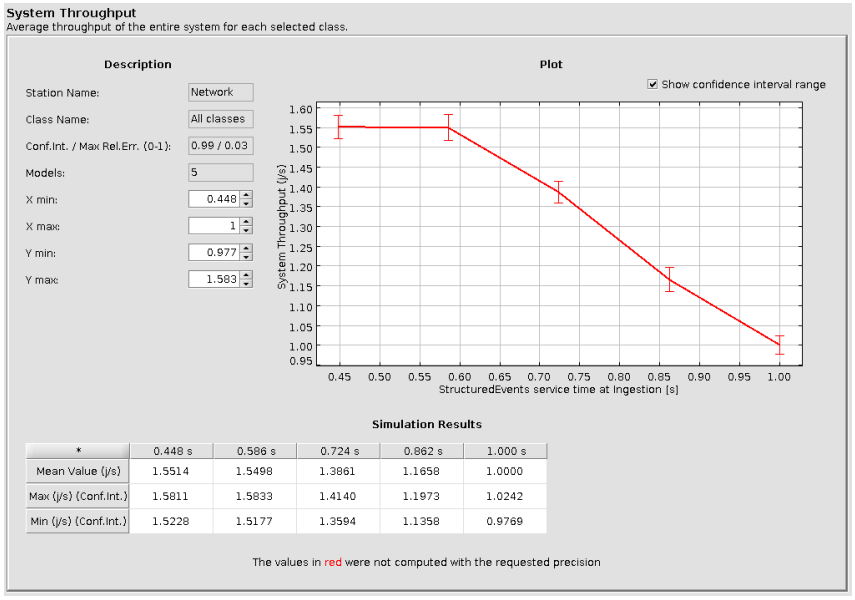
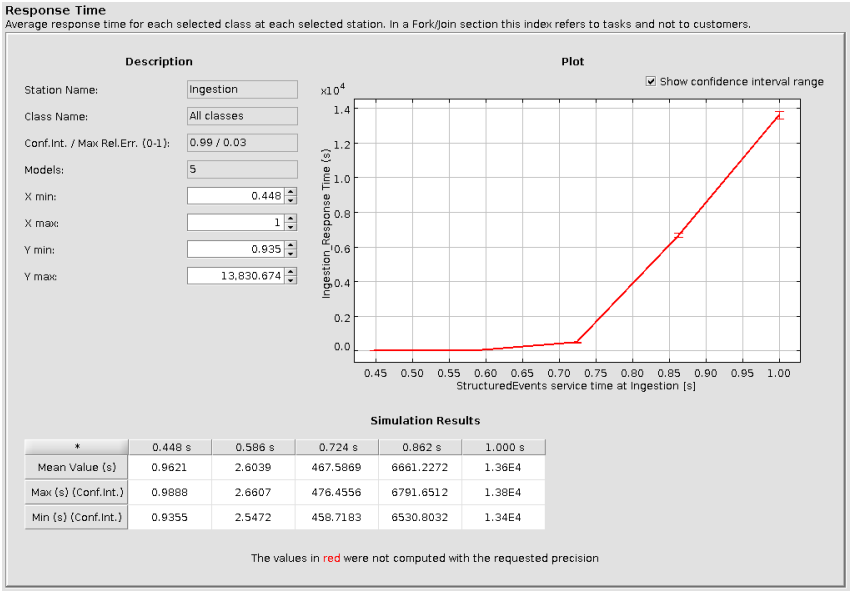


Figure 6.13: Throughput of the system (second scenario)

Response Time In Figure 6.14 is shown the change in the response time of the system associated with the increase in the service time of the *Ingestion* queue. As we can see, the response time increases accordingly to the service time, ramping up substantially after

the 0.724 seconds mark, where the utilization shown in Figure 6.12 starts to increase dramatically.



*	0.448 s	0.586 s	0.724 s	0.862 s	1.000 s
Mean Value (s)	0.9621	2.6039	467.5869	6661.2272	1.36E4
Max (s) (Conf.Int.)	0.9888	2.6607	476.4556	6791.6512	1.38E4
Min (s) (Conf.Int.)	0.9355	2.5472	458.7183	6530.8032	1.34E4

Figure 6.14: System response time (second scenario)

6.3. Results

The data obtained from the what-if analyses performed in the previous section lead us to say that the downgrade can be done, but in a limited fashion.

In the the **first scenario**, we can see that the increase in service time led to an increase in the utilization of the *processing* queue, reflecting in an increase in the system response time, but without leading to an appreciable lowering of the system throughput. In this scenario, choosing to downgrade was a sensible choice and the system could probably be slowed down further.

In the case of the **second scenario**, however, we can see that, by increasing the service time beyond the value of 0.724, the utilization of the *ingestion* queue suddenly rises to 1, making the system *unstable*.

Looking further at the system throughput, we can see that the increase in service time caused an inexorable drop in system performance even before the value that led it to instability. For this reason, we can conclude that in this scenario the downgrade can be carried out, but limited to raising the service time of the *ingestion* station to at most 0.586 seconds for the structured jobs.

Through the experiments conducted on the two scenarios described in this chapter, we were able to show multiple important qualities of the newly created synthetic generator.

Through the first scenario we saw how, thanks to its architecture designed as a library, the generator can be adapted to multiple use cases, allowing the events and waveforms generated to be presented in different ways.

Through the second scenario we instead got to see the usefulness of the layered structure of the generator and, in particular, the control it allows over its output. Additionally, thanks to the changes made to the outputs of the analysis, we were able to verify that the modifications required to the generator would not be excessive were other classifications made on the dataset.

To summarize, the architecture and implementation choices made for the generator yielded the desired results, achieving the characteristic goals of *fine-tuning* of the output and *adaptability* of the generator presented at the beginning of Chapter 5.

7 | Conclusions and Future Works

This final chapter will summarize the contributions and the modalities of the work carried out before moving to the limits of the developed software, with consequential suggestions for future works.

7.1. Summary

A commonly used method for the evaluation of Big Data systems involves the use of modeling techniques; they are usually chosen over other techniques because they do not require a working implementation of the system under consideration. Moreover, they only require that the data, used for the evaluation of the performance, follow the same temporal distribution of the data that is going to be processed by the system, without the need for the actual values.

However, because of the sensitivity of the data being processed, there are few datasets available in the healthcare field capable of meeting that need. While in other domains this problem is usually solved through the use of synthetic data generators, in the domain of healthcare these are few and not specific for performance evaluation.

In this work, we have developed a synthetic data generator to be used for the evaluation through modeling techniques of a Big Data System.

As a reference dataset for the creation of the generator, we relied on MIMIC-III, a publicly available dataset concerning the hospital stays of multiple patients that occurred over several years, containing the measurements and observations (called "events" by the authors of MIMIC-III) made on the patients during that period and recorded by the data system of the hospital in question. Due to the objective usage of the generator, we focused on following the same time distribution between the generated data and the data contained in MIMIC-III.

With the aim of obtaining additional information on the temporal distribution of such events, we first analyzed the interaction process between the patients and the hospital system, focusing on the time periods during which the events described in MIMIC-III

were recorded.

These time periods were then modeled through appropriate distributions, taking advantage of distribution fitting techniques and tools, such as the method of moments and Hyperstar. These were again used to model the inter-time between each kind of event.

The results obtained, such as the parameters of the fitted distributions and the structure identified for the interaction process between patients and the hospital system, were used as the foundation for the generator's architectural choices.

During the development of the latter, we sought two features:

- fine-tunability, defined as the possibility of changing the parameters of the identified distributions from those obtained from the analysis performed, and the ability to control the data generated as output.
- adaptability, defined as the ability to adapt the generator to different analysis procedures and, potentially, different datasets, without requiring excessive and complex modifications.

These characteristics were achieved through the adoption of a layered architecture and keeping in mind, during the development of the generator, the possibility of modifying and customizing its components.

Finally, to test the possibilities just described, we conducted two experiments to evaluate the performance of a Big Data system to be used within a hospital's information system. The experiment were based on two scenarios in which different kinds of events had to be provided to the system. The two experiments resulted in a success, showing for the generator the simplicity with which its output can be tuned for a specific need and its components modified to adapt to the use case.

7.2. Limitations and Future Works

As pointed out in the previous chapters, one of the limitations of our work is represented by the features chosen for the classifications made to split the dataset before performing the distribution fitting; the features considered, taken from patients and hospital stays, had in fact been extracted through an exploration of the contents of MIMIC-III.

However, while describing the development of the generator, we pointed out that the classifications considered during the analysis phase, if changed, could be easily introduced into the generator thanks to the classification module presented during the description of the development of the generator. Should it then be decided to use more complex

classifications to split the set of patients and hospital stays, these could be considered during the generation of the synthetic data without the need to apply excessive changes to the software.

Another limitation, centered on the versatility of the generator, is the inability to use different distributions than those chosen to model the duration of the various stages of interaction and the inter-time between events. Introducing the possibility of using different distributions than phase-type distributions and exponential distributions would be an interesting future addition to our work, which could lead to the creation of synthetic data more similarly distributed to the data contained in MIMIC-III from a temporal point of view.

Another future development of this work is to consider the adaptation of the generator to other datasets structured similarly to MIMIC-III. In this regard MIMIC-IV [14] became recently available; it is an evolution of MIMIC-III that, in addition to many other modifications, covers a more recent time period. It has not been considered for this work because, at the time of writing, the waveform dataset associated with the new version has not yet been published.

Finally, a last future work concerns the use of the synthetic data generator presented here to evaluate the performances of a model of a Big Data system more complete than the simple one presented in this work during the performed experiments. For this purposes, consideration should be given to the work presented in [1], where the authors rendered the model of a hypothetical Data Lake to be used for the storage of heterogeneous data of patients on which an analysis is to be performed to predict the presence of cancer at an early stage.

Bibliography

[1] E. Barbierato, M. Gribaudo, G. Serazzi, and L. Tanca. Performance Evaluation of a Data Lake Architecture via Modeling Techniques. In *Performance Engineering and Stochastic Modeling*, Lecture Notes in Computer Science, pages 115–130, Cham, 2021. Springer International Publishing. ISBN 978-3-030-91825-5.

[2] K. Batko and A. Ślęzak. The use of Big Data Analytics in healthcare. *Journal of Big Data*, 9(1):3, 2022. ISSN 2196-1115.

[3] V. W. Berger and Y. Zhou. Kolmogorov–Smirnov Tests. In *Encyclopedia of Statistics in Behavioral Science*. John Wiley & Sons, Ltd, 2005. ISBN 978-0-470-01319-9.

[4] M. Bertoli, G. Casale, and G. Serazzi. An overview of the jmt queueing network simulator. *Politecnico di Milano-DEI, Tech. Rep. TR*, 2007, 2007.

[5] M. Bladt. A Review on Phase-type Distributions and their Use in Risk Theory. *ASTIN Bulletin: The Journal of the IAA*, 35(1):145–161, May 2005. ISSN 0515-0361, 1783-1350.

[6] C. Cappiello, M. Gribaudo, P. Plebani, M. Salnitri, and L. Tanca. Enabling Real-world Medicine with Data Lake Federation: a research perspective. *The Eighth International Workshop on Data Management and Analytics for Medicine and Healthcare (DMAH)*, 2022. [accepted for publication].

[7] K. Chin-Cheong, T. Sutter, and J. E. Vogt. Generation of heterogeneous synthetic electronic health records using GANs. In *Workshop on Machine Learning for Health (ML4H) at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*. ETH Zurich, Institute for Machine Learning, 2019.

[8] E. Choi, S. Biswal, B. Malin, J. Duke, W. F. Stewart, and J. Sun. Generating Multi-label Discrete Patient Records using Generative Adversarial Networks. In *Proceedings of the 2nd Machine Learning for Healthcare Conference*, pages 286–305. PMLR, Nov. 2017.

[9] R. Fraile and E. García-Ortega. Fitting an Exponential Distribution. *Journal of Applied Meteorology - J APPL METEOROL*, 44:1620–1625, Oct. 2005.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[11] R. Han, X. Lu, and J. Xu. On Big Data Benchmarking. In *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, Lecture Notes in Computer Science, pages 3–18, Cham, 2014. Springer International Publishing. ISBN 978-3-319-13021-7.

[12] M. Harchol-Balter. Real-World Workloads: High Variability and Heavy Tails. In *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*, pages 347–348. Cambridge University Press, Cambridge, 2013. ISBN 978-1-107-02750-3.

[13] M. o. Health. National Minimum Dataset (hospital events), 2012.

[14] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. A. Celi, and R. Mark. MIMIC-IV, 2022.

[15] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[16] T. Kadri and K. Smaili. Convolutions of hyper-erlang and of erlang distributions. *International Journal of Pure and Applied Mathematics*, 98(1):81–98, 2015.

[17] E. D. Lazowska. *Quantitative System Performance, Computer System Analysis Using Queuing Network Models*. Prentice Hall, 1984. ISBN 978-0-13-746975-8.

[18] M. J. Legato and J. P. Bilezikian. *Principles of Gender-specific Medicine*. Gulf Professional Publishing, 2004. ISBN 978-0-12-440907-1.

[19] P. McQuighan. Simulating the poisson process. *Department of Mathematics-University of Chicago*, 23, 2010.

[20] B. Moody, G. Moody, M. Villarroel, G. Clifford, and I. Silva. MIMIC-III Waveform Database Matched Subset, 2017.

[21] S. Rashidian, F. Wang, R. Moffitt, V. Garcia, A. Dutt, W. Chang, V. Pandya, J. Hajagos, M. Saltz, and J. Saltz. SMOOTH-GAN: Towards Sharp and Smooth Synthetic EHR Data Generation. In *Artificial Intelligence in Medicine*, Lecture Notes

in Computer Science, pages 37–48, Cham, 2020. Springer International Publishing. ISBN 978-3-030-59137-3.

[22] P. Reinecke, T. Krauß, and K. Wolter. HyperStar: Phase-Type Fitting Made Easy. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 201–202, Sept. 2012.

[23] F. S. Roberts and J. H. Spencer. A characterization of clique graphs. *Journal of Combinatorial Theory, Series B*, 10(2):102–108, 1971.

[24] S. Sehgal and Y. Singh. Big Data: A Volume or Technology ? *International Journal of Engineering Research & Technology*, 3(10), Apr. 2018. ISSN 2278-0181.

[25] G. Serazzi, G. Casale, and M. Bertoli. Java Modelling Tools: an Open Source Suite for Queueing Network Modelling and Workload Analysis. In *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*, pages 119–120, Sept. 2006.

[26] H. Tang. Confronting ethnicity-specific disease risk. *Nature genetics*, 38(1):13–15, 2006.

[27] A. Thummler, P. Buchholz, and M. Telek. A Novel Approach for Phase-Type Fitting with the EM Algorithm. *IEEE Transactions on Dependable and Secure Computing*, 3(3):245–258, July 2006. ISSN 1941-0018.

[28] J. Walonoski, M. Kramer, J. Nichols, A. Quina, C. Moesel, D. Hall, C. Duffett, K. Dube, T. Gallagher, and S. McLachlan. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *Journal of the American Medical Informatics Association*, 25(3): 230–238, Mar. 2018. ISSN 1527-974X.

[29] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106, 2017.

[30] A. Yale, S. Dash, R. Dutta, I. Guyon, A. Pavao, and K. P. Bennett. Generation and evaluation of privacy preserving synthetic health data. *Neurocomputing*, 416: 244–255, Nov. 2020. ISSN 0925-2312.

List of Figures

List of Tables