

# Systematical conception, design and development of a C++ software for transfer of biomedical data via the hardware API of a clinical patient monitor

## DIPLOMARBEIT

zur Erlangung des akademischen Grades  
Diplom-Ingenieur

im Rahmen des Studiums  
**Biomedical Engineering**

eingereicht von  
**Jakub Matta**

Matrikelnummer 1026447

an der  
Fakultät für Elektrotechnik und Informationstechnik  
der Technischen Universität Wien

Betreuer:  
Projektass. Dipl.-Ing. Florian Thürk  
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eugenijus Kaniusas

# Abstract

Patient monitoring in hospitals belongs to the most important clinical instruments to evaluate the health status of critically ill patients. Especially in the perioperative setting, the probability of patients experiencing adverse health conditions is still very high. Cardiopulmonary complications notably contribute to the patients' health deteriorations, however, the acute organ injuries are the most serious reasons of perioperative mortality. Additional adverse effects such as patient comorbidity and surgical complexity increase the 48-hour and 30-day postoperative mortality about the 12 times and 4 times, respectively. Thus, early and accurate identification of the unfavorable effects is exceedingly important. Nevertheless, the common patient monitoring systems of modern hospitals merely capture the standard vital signs of patients missing novel vitals such as cumulative time of desaturation and hypotension. What's more, the recorded biosignals are rarely persisted with a sufficient sampling resolution for further processing. Those restrictions make the evaluation process of biosignals much more difficult and substantially limit the decision-making process of clinicians in the perioperative setting. Although the most hospitals dispose of a sufficient hardware equipment, the recorded data isn't processed after its storage. This fact substantially limits the effectiveness of the risk prediction systems evoking a timely intervention of the clinical personnel. The present work introduces an easy-to-use, vital sign assessment system supporting the real-time evaluation and high-resolution, multiparametric analysis. The developed tool, *Vital-signs REal-time Analysis for Clinical Translation* (VREACT) provides a simple graphical environment facilitating the real-time, vital sign tracking as well as flexible biosignal recording ensuring a consistent data storage. The tool interacts with the specific patient monitoring device, *Infinity® Delta* ensuring the access to the physiological information. Finally, a short experiment evaluating the accuracy and consistency of the recorded experimental measurements was conducted. Hereby, a test measurement storing the recorded vital parameters across 3 different ICUs was performed. The experiment confirmed the potential of the high-resolution vital sign recorder in the perioperative setting. The tool delivered 20 consistent biosignal records consisted of standard and novel physiological parameters. Conclusively, the software enhanced the automation of the whole signal acquisition process and enabled a real-time remote tracking of the novel physiological parameters. In the future work, additional functions like an advanced database search or a predictive alarm system might extend the capabilities of the tool.

# Kurzfassung

Patientenüberwachung in Krankenhäusern stellt eine wichtige Basis für Evaluierung von Gesundheitsrisiken kritisch erkrankter Patienten dar. Vor allem im perioperativen Umfeld ist das Risiko unvorhersehbarer Gesundheitskomplikationen sehr hoch. Während die kardiopulmonalen Erkrankungen nachweislich zur Gesundheitszustandsverschlechterung führen, tragen akute Organverletzungen am meisten zur perioperativen Mortalität. Diese wird zusätzlich durch weitere negative Einflüsse wie z.B. die Komorbidität oder Komplexität der chirurgischen Eingriffe gestärkt. Gleichzeitig steigt die Mortalität während der postoperativen Phase um einen Faktor 12 innerhalb der ersten 48 postoperativen Stunden und um einen Faktor 4 innerhalb der ersten 30 postoperativen Tage. Es ist daher unabdingbar, derartige Effekte möglichst früh und verlässlich zu erkennen. Die heutigen Patientenüberwachungssysteme nehmen bloß die Standard-Vitalparameter auf und verzichten auf neue, mittlerweile gut bewährte Parameter wie z.B. Kumulative Zeit der Sauerstoffsättigung oder des Blutdruckabfalls. Außerdem werden die aufgenommenen Vitalparameter kaum oder nur mit unzureichender Auflösung gespeichert. Das macht eine genaue Evaluierung von kontinuierlichen Biosignalen fast unmöglich und schränkt die Entscheidungsfähigkeiten des medizinischen Personals erheblich ein. Trotz einer hinreichenden Datenspeicherung seitens der meisten Krankenhäuser werden die physiologischen Daten nur selten weiterverarbeitet. Dieser Fakt vermindert die Effektivität der klinischen Prognosesysteme, die die frühzeitigen Warnmeldungen generieren sollen. Diese Arbeit präsentiert ein neuartiges System zur Evaluierung von Biosignalen mit einer einfachen graphischen Oberfläche. Es soll eine hochauflösende multiparametrische Echtzeitanalyse von Biosignalen gewährleisten. Das entwickelte Tool, *Vital-signs REal-time Analysis for Clinical Translation* (VREACT) bietet einerseits eine vielseitige graphische Oberfläche für Echtzeitüberwachung, andererseits einen Recorder zum Aufnehmen und Speichern von physiologischen Daten. VREACT wurde für einen speziellen Patientenmonitortyp, *Infinity® Delta Monitor* entworfen, welcher als Quelle jeglichen physiologischen Informationen dient. Schließlich wurde die Software im Rahmen eines kurzen Experiments eingesetzt und deren Messungen auf die Genauigkeit und Konsistenz geprüft. Die Daten wurden aus 3 verschiedenen Pflegestationen aufgenommen. Die Software wies im Grunde ein hohes Einsatzpotential auf. Sie hinterließ 20 konsistente biomedizinische Aufnahmen, die nicht nur die üblichen, sondern auch die neuartigen physiologischen Parameter enthielten. Ferner, die Software trug wesentlich zur Automatisierung des gesamten Signalakquisitionsprozesses bei und legte eine gute Basis für die Fernüberwachung neuartiger Vitalparameter fest. Das Tool könnte noch in der Zukunft um zusätzliche Funktionen wie z.B. die Datenbanksuche oder prädiktive Alarmsysteme erweitert werden.

# Acknowledgments

I would like to thank Prof. Eugenijus Kaniusas for the possibility to enter the board of this fascinating project as well as acknowledge his valuable suggestions and constructive feedback. I would also like to express my deepest appreciation to Florian Thürk for his professional mentoring, intensive support and constructive opinions during the entire project work. His valuable advices contributed to a better quality of this scientific work and helped me (personally) develop new valuable technical skills. I would also like to appreciate the collaboration with my colleagues Fatih Kartal and Max Schnetzinger, which essentially contributed to the realization of the project. Thanks to Max, the data for the final test experiment was easily organized. Finally, I wish to thank my girlfriend Andrea as well as the whole family for their optimism and tireless encouragement during the whole research work.

# Table of Contents

Abstract .....	i
Kurzfassung.....	ii
Acknowledgments .....	iii
<b>1. Introduction .....</b>	<b>1</b>
1.1. Motivation.....	1
1.2. Definition and History .....	3
1.3. Real-time assessment systems.....	6
1.3.1. WvAPITest.....	7
1.3.2. WvRecorder .....	9
1.3.3. Other scientific tools.....	10
1.3.4. Dräger Infinity Delta.....	14
1.4. Biomedical signal analysis .....	15
1.4.1. Standard physiological parameters .....	16
1.4.2. Novel physiological parameters.....	20
1.5. Aim of this work.....	24
<b>2. Real-time assessment using VREACT .....</b>	<b>25</b>
2.1. Software design .....	25
2.1.1. Stakeholders .....	26
2.1.2. UML Use Case diagram .....	27
2.1.3. Workflow diagram .....	30
2.1.4. UML Class diagram.....	31
2.1.5. Design patterns .....	35
2.2. System implementation .....	40
2.2.1. Technology .....	41
2.2.2. Infinity® Network .....	42
2.2.3. Dynamic linking.....	44
2.2.4. Application tier structure.....	46
2.2.5. Multithreading .....	48
2.3. User Interface .....	50
2.3.1. Requirements Engineering.....	51
2.3.2. UI Requirements .....	53
2.3.3. Connection Management .....	55
2.3.4. Patient Recorder .....	57
2.3.5. Patient Viewer.....	59
<b>3. Clinical experiment .....</b>	<b>61</b>
3.1. Introduction.....	62
3.2. Methods .....	62
3.3. Results.....	65
3.4. Discussion.....	73
<b>4. Discussion .....</b>	<b>76</b>
<b>Appendix.....</b>	<b>80</b>

# List of Figures

FIGURE 1. THE EINTHOVEN’S STRING GALVANOMETER DEVELOPED IN 1903 [12].	4
FIGURE 2: CONNECTION OF SENSOR EQUIPMENT TO THE COMMERCIAL COMPUTER IBM 1710 [13].	5
FIGURE 3. WvAPITest TOOL USER INTERFACE ENCOMPASSING FUNCTIONS FOR ACCESS TO PHYSIOLOGICAL DATA.	8
FIGURE 4. THE USER INTERFACE OF WvRECORDER FACILITATES THE CONNECTION TO MONITORS AND RECORDING OF SAMPLES.	10
FIGURE 5. REAL-TIME, VITAL SIGN ANALYSIS USING INTENSIVE CARE WINDOW TOOL [23].	12
FIGURE 6. THE MAIN PANE OF THE VITALRECORDER CURRENTLY OPERATED IN THE “TRACK MODE” [24]. AT THE TOP OF THE WINDOW, THE MENU BAR WITH MANY USEFUL FUNCTIONS SUCH AS DATA EXPORT IS PLACED. “TRACK LIST” KEEPS THE LIST OF AVAILABLE (ONLINE) END-POINT DEVICES, WHEREAS THE “TRACK WINDOW” PRESENTS THE PHYSIOLOGICAL DATA IN REAL TIME. THE “TIME SLIDER” PROVIDES AN ACCESS TO HISTORY OF THE RECORDINGS. THE THREE PANES ON THE RIGHT SIDE AID TO SETUP THE END-POINT DEVICES, TO LOG THE EVENTS AND TO START ANALYSIS BY OTHER THIRD-PARTY ALGORITHMS [24].	13
FIGURE 7. INFINITY DELTA MONITORS ACT AS WAVEFORM AND VITAL SIGN SOURCES [25].	15
FIGURE 8. (A) THE TYPICAL ECG CURVE OF HEALTHY INDIVIDUALS, (B) 12-ELECTRODE SYSTEM LOCATED AROUND THE HEART [27].	17
FIGURE 9. THE EMITTED LIGHT OF THE INITIAL INTENSITY $I_0$ PASSES THROUGH THE DIFFERENT STRUCTURES IN LIVING TISSUE ENDING UP WITH THE INTENSITY $I$ . THE LIGHT IS ABSORBED DURING THE PRESENCE AND ABSENCE OF THE PULSATILE COMPONENT DIFFERENTLY [37].	20
FIGURE 10. (A) EXAMPLES OF DIFFERENT SPECTRAL COMPONENTS EXTRACTED FROM ORIGINAL HRV WAVEFORM [8], (B) POINCARÉ PLOT FITTED BY ELLIPSE CONSECUTIVELY PLOTS TWO NEIGHBORING RR INTERVALS AGAINST EACH OTHER [41].	21
FIGURE 11. LIST OF STAKEHOLDERS PARTICIPATING ON THE DEVELOPMENT OF VREACT. THE PROJECT LEADER {ENGINEER} ABSOLVED REGULAR MEETINGS WITH PROJECT LEADER {CLINICIAN} WHERE THE SYSTEM REQUIREMENTS HAVE BEEN FORMULATED. THE REQUIREMENTS HAVE BEEN COLLECTED FROM OTHER CLINICIANS AND STUDENTS PERFORMING CLINICAL RESEARCH. THE PHYSIOLOGICAL DATA SAMPLED FROM MONITORING UNITS WAS ACQUIRED AFTER PATIENTS’ AGREEMENT. THE STUDENT {ENGINEER} PROPOSED AND DEVELOPED SOLUTIONS TO FULFILL PROJECT LEADERS’ REQUIREMENTS.	27
FIGURE 12. THE UML USE CASE DIAGRAM OF VREACT PORTRAYING THE INITIAL DESIGN OF THE SYSTEM. THE SOFTWARE WAS DESIGNED TO MANAGE THE CONNECTION TO ONE OR MORE BEDS (OR PATIENT MONITORS) SIMULTANEOUSLY, TO RECORD AND TO VIEW THE BIOMEDICAL SIGNALS FETCHED FROM THE SELECTED INFINITY DELTA MONITORS. FURTHERMORE, THE USER IS ABLE TO REFRESH THE LIST OF BEDS, VIEW A LICENSE INFO AS WELL AS TO DETERMINE THE STARTING AND END POINT OF SIGNAL RECORDING.	29
FIGURE 13. BPM FLOWCHART ILLUSTRATING THE WORKFLOWS OF PHYSICIANS AND SCIENTISTS USING OUR CLINICAL SOFTWARE VREACT. THE SOFTWARE (SW) RETRIEVES THE AVAILABLE BEDS ON THE SERVER TO WHICH THE USER MIGHT CONNECT. AFTER SUCCESSFUL CONNECTION TO ONE OR MORE BED MONITORS, THE USER CAN RECORD AND/OR VIEW THE BIOMEDICAL SIGNALS FOR THE GIVEN PATIENTS. AFTER THE REAL-TIME ANALYSIS IS FINISHED, THE BEDS ARE DISCONNECTED AND THE SOFTWARE IS CLOSED.	31
FIGURE 14A-B. THE UML CLASS DIAGRAM (PART 1 AND 2) DEPICTING THE BASIC STRUCTURE OF VREACT INCLUDING THE CLASSES (INSTANCES) AND THEIR POSSIBLE COMMUNICATION PATHS (RELATIONSHIPS). EACH CLASS BELONGS TO A PACKAGE (WHICH KEEPS LOGICALLY SIMILAR CLASSES) WRITTEN BEFORE THE CLASS NAME. THE MULTIPLICITIES (SMALL NUMBERS) LIMIT THE NUMBER OF INSTANCES COMMUNICATING WITH EACH OTHER USING A PARTICULAR PATH (LINE). THE BLACK ARROW PLACED NEXT TO THE LINE GIVES THE READING DIRECTION OF THE TEXT SPECIFYING THE RESPECTIVE COMMUNICATION PATH.	34
FIGURE 15. MODEL-VIEW-CONTROLLER (MVC) PATTERN INVOLVES AT LEAST THREE INDIVIDUAL COMPONENTS INTERACTING TO EACH OTHER. THE MODEL CONTROLS THE DATA, THE CONTROLLER HANDLES THE USER INPUTS AND REACTS ON CHANGES IN MODEL. THE VIEW UPDATES THE UI AFTER THE MODEL STATE IS CHANGED [55].	37
FIGURE 16. VREACT CLASSES PARTICIPATING ON THE COMMUNICATION BETWEEN THE MODEL AND THE VIEW ACCORDING TO THE OBSERVER PATTERN. WvAPIConnection KEEPS THE STATUS OF THE SERVER CONNECTION AND THE BED AVAILABILITY. THEREFORE, IT REPRESENTS THE SUBJECT. IN COMPARISON, WvCONTROLLER AND MainWINDOW PERFORMS SPECIFIC UPDATES BASED ON THE MODEL’S STATE. THEY ARE SEEN AS OBSERVERS.	38
FIGURE 17. (A) BedENTITY REPRESENTS THE MOST FREQUENTLY USED DTO IN VREACT. IT TRANSPORTS DATA ABOUT THE CONNECTED BED SUCH AS CONNECTION ID, BED LABEL, PATIENT ID AND THE CURRENTLY AVAILABLE CONNECTION STATUS. (B) WvAPIConnection WAS IMPLEMENTED AS SINGLETON HAVING A PRIVATE CONSTRUCTOR, COPY CONSTRUCTOR AND DESTRUCTOR (IN BOLD). THIS PREVENTS THE OBJECT INSTANTIATION AND DESTRUCTION FROM OUTSIDE OF THE CLASS. THE INSTANCE CAN ONLY BE ACCESS USING A STATIC METHOD “GETINSTANCE()”	39
FIGURE 18. THE RELATIONSHIP BETWEEN THE DRÄGER’S INFINITY® NETWORK AND EXISTING HOSPITAL NETWORK. THE INFINITY® GATEWAY SERVER ENABLES THE EXCHANGE OF ADMINISTRATIVE AND/OR PHYSIOLOGICAL DATA BETWEEN BOTH NETWORKS. VREACT PERFORMS A REAL-TIME ASSESSMENT, HIGH RESOLUTION PARAMETER ANALYSIS BASED ON PATIENT MONITORS	

ACTIVE ON INFINITY® NETWORK. THE GATEWAY SERVER ENABLES THE CONNECTION TO ANY DEVICE OF THE INFINITY® NETWORK. ....	43
FIGURE 19. DYNAMIC LIBRARIES AS SPECIAL FUNCTIONAL MODULES ARE LINKED TO THE APPLICATION AS SOON AS IT IS COMPILED. THEY HAVE TO BE DISTRIBUTED WITH THE APPLICATION, SINCE THEIR CONTENT IS NOT EXPLICITLY EMBEDDED INTO THE APPLICATION EXECUTABLE. THEY ARE LOADED LATER AT APPLICATION RUNTIME [66]. ....	45
FIGURE 20. THE 3-TIER ARCHITECTURE OF THE SOFTWARE AFTER THE IMPLEMENTATION. THE PRESENTATION LAYER FOCUSES ON THE VISUAL PRESENTATION OF BEDS, NOTIFICATIONS AND VISUALIZATION OF BIOSIGNALS. THE BUSINESS LAYER HOUSES THE BUFFER OF BIOSIGNALS, FURTHER PROCESSING ALGORITHMS AS WELL AS THE SAMPLING LOOP FOR BIOSIGNALS. THE INFINITY NETWORK HAS A ROLE OF THE DATA ACCESS LAYER MANAGING THE RESOURCES CRITICAL FOR THE HIGH-RESOLUTION, REAL-TIME ANALYSIS. ....	47
FIGURE 21. POSSIBLE PROGRAM FLOWS IN (A) SINGLE-THREADED AND (B) MULTI-THREADED APPLICATIONS [72]. THE TASKS IN SINGLE-THREADED APPLICATIONS CAN ONLY BE FINISHED IN A SEQUENCE ORDER, WHEREAS THE TASKS IN THE MULTI-THREADED APPLICATIONS CAN BE EXECUTED IN PARALLEL. ....	49
FIGURE 22. THE IMPLEMENTATION OF THE 'LOOPMANAGER' CONSTRUCTOR. THIS CLASS REPRESENTS THE SIGNAL SAMPLING LOOP FOR ONE SINGLE BED ENTITY. THE TIMER DETERMINES THE SAMPLING FREQUENCY, THE BUFFER STORES THE REFERENCES TO THE STORED BIOSIGNALS. SINCE EACH 'LOOPMANAGER' INSTANCE PERFORMS DEMANDING OPERATIONS IN RESPECT TO THE CENTRAL SAMPLING LOOP, THE INSTANCES ARE MOVED TO THEIR OWN THREADS MANAGING THEIR OWN RESOURCES. ....	50
FIGURE 23. THE RESULTS AFTER IMPLEMENTATION OF THE CONNECTION MANAGEMENT MODULE PRESENTING THE AVAILABLE MONITORS. THE USER HAS AN OPTION TO SELECT AND CONNECT TO THE INDIVIDUAL MONITORING DEVICES. (A) THE AVAILABLE PATIENT MONITORS HAVE SUCCESSFULLY BEEN CONNECTED, I.E. THE BIOMEDICAL SIGNAL ANALYSIS MAY BE STARTED, (B) THE AVAILABLE PATIENT MONITOR IS CURRENTLY OFFLINE (BUT REGISTERED ON THE SERVER). ....	56
FIGURE 24. THE APPLICATION INFORMS ABOUT THE PROGRESS OF THE RECORDER MODULE PERSISTING ALL THE BIOSIGNALS AVAILABLE ON THE CORRESPONDING MONITORS. (A) AFTER THE BUTTON "RECORD" IS CLICKED, ITS NAME IS CHANGED TO THE "STOP" BUTTON AND THE CSV EXPORT STARTS. (B) AFTER PUSHING THE BUTTON "STOP", THE SAMPLES AREN'T STORED ANYMORE AND THE BUTTON CHANGES TO THE "RECORD" BUTTON AGAIN. ....	58
FIGURE 25. PATIENT RECORDER CREATES A SPECIFIC FOLDER AND FILE STRUCTURE FOR EACH SINGLE MONITORING DEVICE. AFTER STARTING THE RECORDER, A FOLDER DEFINING THE CURRENT RECORDING RUN WITH A SPECIAL NAME IS CREATED. WITHIN THIS FOLDER, A FOLDER PER CARE UNIT AND A FOLDER PER BED POSITION ARE CREATED. FINALLY, SEVERAL CSV FILES (IN DEPENDENCE OF THE SIZE AND THE NUMBER OF THE RECORDED BIOSIGNALS) MIGHT BE CREATED AND STORED. ....	59
FIGURE 26. PATIENT VIEWER PRESENTS THE BIOSIGNALS OF A SINGLE PATIENT. ON THE LEFT SIDE OF THE SCREEN, THE OVERVIEW ABOUT THE AVAILABLE BIOMEDICAL SIGNALS IS GIVEN. UNDERNEATH THIS WINDOW, AVAILABLE CHART TYPES FOR THE SELECTED BIOSIGNAL ARE PRESENTED. THE SIGNALS CAN BE THEN EASILY DRAG & DROPPED TO THE TWELVE INDEPENDENT SLOTS PREPARED FOR VISUAL ANALYSIS (RIGHT SIDE). ....	60
FIGURE 27. THE FINAL HOSPITAL INFRASTRUCTURE USED FOR THE BIOSIGNAL RECORDING. CA. 2-HOUR MEASUREMENT ON 20 PATIENTS WAS ACHIEVED BY RUNNING VREACT ON THE TOSHIBA LAPTOP (ORANGE). THE VITAL SIGNS WERE PRIMARILY RECORDED ON (DRÄGER INFINITY® DELTA) PATIENT MONITORS (GREEN) AVAILABLE WITHIN THE SURGICAL AND RECOVERY ROOMS. VREACT ALSO SUPPORTS POSTPROCESSING OF THE PRIMARILY RECORDED DATA AND CALCULATES NOVEL PHYSIOLOGICAL PARAMETERS (E.G. HRV). THIS FIGURE WAS DRAFTED AND PUBLISHED IN THE SUBMITTED CONFERENCE PAPER [3]. ....	63
FIGURE 28. A BIOSIGNAL SET COMPARISON ACROSS ALL ANALYZED CARE UNITS. THE PARAMETERS P1 M, P1 D, P1 S AND GP1 ARE RECORDED BY ONLY ONE PATIENT, WHEREAS THE OTHER TOP 10 PARAMETERS SUCH AS CUMULATIVE TIME OF DESATURATION (HYPOXEMIA), HEART RATE (HR), OXYGEN SATURATION (SpO2), ETC. ARE OBSERVED BY ALL PATIENTS IN ANALYSIS. ....	67
FIGURE 29. EXAMPLES OF THE BIOSIGNAL RECORDED BY VREACT. (A) THE HR PARAMETER EXTRACTED FROM THE UNDERLYING ECG RECORDING, (B) THE MEAN ARTERIAL PRESSURE (ART M) EXPORTED FROM MONITOR, DIRECTLY AND (C) THE CUMULATIVE TIME OF HYPOXEMIA EXTRACTED FROM THE UNDERLYING SpO2 (OXYGEN SATURATION MEASURE) SIGNAL.....	69
FIGURE 30. STATISTICAL DISTRIBUTION OF 8 DIFFERENT BIOSIGNALS RECORDED OVER ALL PATIENTS IN THE 3 ICUs, 13B1, 13C2, 9D. THE STANDARD PARAMETERS EXPORTED FROM THE INFINITY® DELTA MONITORS (A) ART M, (B) SpO2, (C) HEART RATE AS WELL AS NOVEL PARAMETERS (D) VLF POWER, (E) LF POWER, (F) HF POWER, (G) SDNN, (H) RMSSD ARE CHARACTERIZED UPON ITS DISTRIBUTION OF THE DIFFERENT ICUs. IN GENERAL, ALL PLOTS CONTAIN RATHER VARYING DISTRIBUTIONS, THOUGH THE PARAMETERS (D) LF POWER AND (G) SDNN SHOW (ALMOST) UNIFORM MEDIANS ACROSS ALL ICUs. THE MOST EQUALLY DISTRIBUTED SERIES ARE OBSERVED IN (B) SpO2 AND (C) HEART RATE ON THE GROUP "ALL". 7 BOXPLOTS EXHIBIT AN OUTLINER APART FROM THE WHISKERS. ALL THE DISTRIBUTIONS ARE COMPOSED OF BIOSIGNAL MEDIANS, I.E. EACH REPRESENTATIVE VALUE PER BIOSIGNAL WAS OBTAINED AS MEDIAN OF THE WHOLE BIOSIGNAL DATA SERIES. THIS VALUE WAS SUBSEQUENTLY USED AS THE REPRESENTATIVE OF THAT PARTICULAR PATIENT. ....	71
FIGURE 31. THE FINAL UML CLASS DIAGRAM OF VREACT. THE DIAGRAM GIVES A GLIMPSE ABOUT THE STRUCTURE OF THE APPLICATION BACKBONE (EXCEPT OF THE PATIENT VIEWER CLASSES). TO THE MOST IMPORTANT COMPONENTS OF THE SOFTWARE BELONG THE CLASSES OF THE MVC PATTERN (MAINWINDOW, WvCONTROLLER, WvCONNECTION). THESE	

CLASSES ARE RESPONSIBLE FOR THE CENTRAL CONNECTION MANAGEMENT OF PATIENT MONITORS (SEE SECTION 2.1.5). FURTHER, THE LOOPMANAGER ALONG WITH THE BEDENTITY ARE THE BASIC COMPONENTS OF THE SAMPLING LOOP USED FOR THE SAMPLE ACQUISITION. OTHER ENTITIES SUCH AS WAVEENTITY, VSignENTITY AND BIOSIGNALS ARE THE CENTRAL CARRIERS OF THE BIOMEDICAL SAMPLES FROM THE MONITOR TO THE APPLICATION BUFFER. ....	80
FIGURE 32. OVERVIEW ABOUT THE RECORDED BIOSIGNALS ON THE ICU LABELED AS 13B1. 26 OUT OF 40 BIOSIGNALS ARE EXPORTED FOR EVERY MONITORING UNIT IN 13B1 AND 13C2, WHICH IS THE HIGHEST FREQUENCY AMONG THE THREE CARE UNITS. HOWEVER, THERE ARE 6 BIOSIGNALS (RA.1, RA, P1 M, P1 D, P1 S, GP1) WHICH ARE OBSERVED BY NONE OF THE PATIENTS IN THE ICU. ....	85
FIGURE 33. OVERVIEW ABOUT THE RECORDED BIOSIGNALS ON THE ICU LABELED AS 13C2. 26 OUT OF 40 BIOSIGNALS ARE EXPORTED FOR EVERY MONITORING UNIT IN 13C2 AND 13B1, WHICH IS THE HIGHEST FREQUENCY AMONG THE ALL THREE CARE UNITS. HOWEVER, 13C2 EXHIBITS THE HIGHEST NUMBER OF UNTRACKED BIOSIGNALS FROM ALL ICUs BEING ANALYZED. ....	86
FIGURE 34. OVERVIEW ABOUT THE RECORDED BIOSIGNALS ON THE ICU LABELED AS 9D. ONLY 11 OUT OF OVERALL 40 BIOSIGNALS ARE EXPORTED FOR EVERY PATIENT, WHICH IS THE LOWEST FREQUENCY AMONG THE ALL THREE CARE UNITS. HOWEVER, 9D KEEPS NO BIOSIGNALS WITH THE ZERO OCCURRENCES, I.E. THE PARTICULAR CARE UNIT MEASURES MUCH MORE BIOSIGNAL TYPES THAN THE OTHER ICUs 13B1 AND 13C2. ....	87
FIGURE 35. THE SUMMARY OF THE EXPORTED BIOSIGNALS FOR A PATIENT VISUALIZED IN PYTHON. CA. 2-HOUR MEASUREMENT WAS DONE VIA THE VREACT BIOSIGNAL RECORDER. THE EXPORTED DATA HAVE BEEN PROCESSED AND VISUALIZED BY UTILIZING PYTHON ALONG WITH JUPYTER NOTEBOOK [85]. THE X-AXIS REPRESENTS THE TIME (IN MINUTES) IN THE RECORDING AND THE Y-AXIS KEEPS THE RANGE OF VALUES FOR THE GIVEN BIOSIGNALS BEING EXPORTED BY VREACT. THE Y-AXIS WAS LABEL ACCORDING TO THE EXPORTED BIOSIGNAL. THE MEASUREMENT INCLUDES BOTH THE STANDARD (E.G. ECG, ART, RESP) AND NOVEL (CUMULATIVE TIME OF HYPOXEMIA, ECG-RMSSD, ECG-SDNN) PARAMETERS. SINCE THE NOVEL PARAMETERS WERE EXTRACTED FROM THEIR BASIC COUNTERPARTS, ITS LABEL COMPRISES THE NAME OF THE BASE PHYSIOLOGICAL PARAMETER (FROM WHICH IT'S ACTUALLY DERIVED). ....	89



# List of Tables

TABLE 1. THE SYSTEM ENGINEERING REQUIREMENTS ON VREACT FORMULATED ACCORDING TO THE ISO/IEC AND IEEE INTERNATIONAL STANDARDS [76] AND IEEE RECOMMENDATIONS [78] IN SOFTWARE ENGINEERING. THESE REQUIREMENTS WERE PRIMARILY FORMULATED FOR THE BED MANAGEMENT AND RECORDER PART OF THE SOFTWARE, WHICH ARE ALSO THE CENTRAL TOPIC OF THIS WORK. ....	53
TABLE 2. OVERVIEW ABOUT THE CARE UNITS AND THEIR RELATED PATIENT MONITORING DEVICES ACCESSED DURING THE EXPERIMENT. INSTEAD OF PATIENT NAMES, THE MONITOR LABELS (E.G. BETT-01) WERE RETRIEVED FOR THE BIOSIGNAL ANALYSIS. THE TABLE PRESENTS ALL 20 MONITORING POSITIONS ACROSS THE 3 DIFFERENT CARE UNITS. THE “SUM” COLUMNS REVEAL THE NUMBER OF BIOSIGNALS BEING RECORDED AT THE RESPECTIVE MONITORING POSITION. ....	64
TABLE 3. UTILIZED DISK SPACE ON THE TOSHIBA LAPTOP AFTER FINISHING THE BIOMEDICAL RECORDING. THE DATA SIZE OF THE CA. 2-HOUR RECORDING WAS MAPPED ONTO THE AVERAGED DISK SPACE OCCUPIED BY A CARE UNIT, A PATIENT (OR BED POSITION) AND A BIOSIGNAL. THE REQUIRED AVERAGED DISK SPACE WAS SPECIFIED FOR DIFFERENT TIME PERIODS, PER HOUR, DAY, MONTH AND YEAR. ....	65

# List of Abbreviations

Abbreviation	Definition
AKH	Vienna General Hospital
AKI	Acute Kidney Injury
API	Application Programming Interface
BP	Blood Pressure
BPM	Business Process Modeling
CO	Cardiac Output
CSV	Comma Separated Value
DLL	Dynamic Link Libraries
DTO	Data Transfer Object
ECG	Electrocardiogram
EMR	Electronic Medical Record
HR	Heart Rate
HRV	Heart Rate Variability
ICU	Intensive Care Unit
IQR	Interquartile Range
LOS	Length of Stay
MAP	Mean Arterial Pressure
MVC	Model View Controller
OR	Operating Room
UML	Unified Modeling Language
VR	Vascular Resistance
VREACT	Vital-signs Real-time Analysis for Clinical Translation

# List of Symbols

Symbol	Definition
$aVF$	augmented vector foot
$aVL$	augmented vector left
$aVR$	augmented vector right
$dBp$	diastolic pressure
$RR_i$	heart's beat-to-beat interval
$RR_{i-1}$	previous heart's beat-to-beat interval
$S_aO_2$	oxygen saturation
$S_pO_2$	pulse oximetry
$sBP$	systolic pressure
$\tau_{R-R}$	peak-to-peak time
$V1-6$	precordial leads one to six

# 1. Introduction

Clinical patient monitoring as the most commonly used method for continuous collection of biomedical data in hospitals has gained a great popularity in the recent years. The main objective of a monitoring system is the continuous recording of vital parameters on patients at risk [1], [2]. Many of those variables are subsequently used to assess an immediate patient's health condition, e.g. heart rate, respiration, blood pressure and oxygen saturation etc. [1]. Furthermore, long-term detection of various vital parameters provides ideal input data for other analytical algorithms, which may reliably predict further patient care complications and deteriorations. Thus, the supported monitoring systems must include or at least offer an interface for connection to a robust warning system indicating any patient deterioration [2]. However, the common monitoring devices in today's hospitals often deliver only the golden standard parameters and miss the novel parameters e.g. *cumulative time of desaturation or hypotension*, *heart rate variability* (HRV) etc. [3]. Moreover, the mass-manufactured biomedical appliances deliver robust detection procedures, but they usually suffer from insufficient sampling resolutions of data, what in turn introduces further limitations for physicians and complicates their patient-care oriented decisions [3].

## 1.1. Motivation

Though the patient care monitoring systems are still evolving and the vital parameter stack is gradually extended, the general mortality rate of patients with cardiopulmonary complications reaches considerable values ( $4 - 17\%$  *p.a.*) [2], [4]. Correspondingly, an adequate, on-time prediction of high risk patients can assist to detect the unexpected patients' state deteriorations. For example, early detection of sepsis, which typically results in heavy organ malfunction or even organ failure, is crucial and can drastically decrease the mortality rates [4]. Especially, the perioperative patients tend to experience unexpected health conditions mostly induced by acute organ injuries (e.g. myocardial infarction) representing the major cause of the patients' death [3], [5]. After the surgery, the patients are usually transported to the intensive care units (ICUs) and connected to the bedside monitors ensuring full access to the patients' vital signs and waveforms. In the postoperative phase, it is crucial to perform an accurate, real-time measurement of vital parameters, since the mortality during the first 48-hour period after surgery is considered to be the highest [6].

The postoperative mortality is caused by numerous risk factors such as presence of co-occurring diseases or disorders, application of anesthesiologic substances and complexity of the surgical interventions. Those and many other perioperative adverse events are clearly associated with higher mortality rates after patient's surgery [7]. Detrimental cardiovascular and pulmonary events emerging during the surgical procedure increased the 48-hour and 30-day postoperative mortality about the *12 times* and *4 times*, respectively. Furthermore, an application of an invasive monitoring environment during surgery such as *arterial line, cardiovascular catheter or pulmonary artery catheter* intensifies the negative effect on the 48-hour and 30-day postsurgical mortality [7]. Conclusively, it is crucial to identify the adverse effects on the therapeutic care immediately during or after surgery and introduce an easy-to-use, non-invasive and accurate detection environment. In addition, more and more biomedical information is being stored in the electronic medical records (EMR), which provide valuable sources for visual, real-time analysis and can definitely support the decision-making process in hospitals [2].

Paradoxically, the most of the hospital departments still lack an appropriate sensing and monitoring technology essential for recording of vital parameters [2]. E.g. in general wards, the vitals are measured manually by nurses at irregular intervals and an additional, early-warning system is hardly applicable. Although the most ICUs are equipped by a modern sensing technology, a continuous data collection linked with other data-mining systems is typically excluded. Therefore, it is extremely difficult to predict and handle risky situations without the underlying real-time recordings of vitals [2], [4]. Although the larger hospitals usually dispose of a storage capacity for the huge amount of physiological data recorded perioperatively, the data is sampled at a too low resolution and is only accessible through a couple of stationary appliances. This limits the capability to add novel vital sign parameters to the real-time signal analysis and prevents the on-time evaluation of parameters (e.g. HRV [8]) being crucial for advanced patient care [3].

According to the recent study [9] “*the health care in the 21<sup>st</sup> century will require the intensive use of information technology and clinical informatics to acquire and manage data, transform the data to actionable information, and then disseminate this information so that it can be effectively used to improve patient care*”. That means, the data acquisition will play a crucial role in the next years and influence further information processing steps included in real-time analysis techniques as well as the application of machine learning algorithms. Hand by hand

with the better information exchange and continuous clinical education among physicians, it should be possible to access the required clinical information timely and evaluate the pathophysiological factors more precisely [9].

## 1.2. Definition and History

Patient monitoring systems have significantly evolved over the last few decades and have become an integral part in the “*care of critically ill patients*” [1]. According to Hudson in [10], patient monitoring represent “*repeated or continuous observations or measurements of the patient, his or her physiological function, and the function of life support equipment, for the purpose of guiding management decisions, including when to make therapeutic interventions, and assessment of those interventions*”. Conclusively, a monitoring system must deliver repeatedly sampled physiological data portraying the health conditions of currently monitored patients. It is considered as a tangible apparatus, “*something that watches for—and warns against—serious or life-threatening events in patients, critically ill or otherwise*” [1]. In conjunction with ICUs, such a device “*follows the patient’s cardiac, hemodynamic and respiratory statuses*” [11]. Therefore, valuable tracking systems are extremely important within the perioperative state of patients and stationary patients admitted to ICUs.

To evaluate a monitoring system in respect to its operational characteristics, a short insight to the history has to be given. In the early 17<sup>th</sup> century, Santorio from Venice has been investigating methods for measuring the body temperature by using a *spirit thermometer* [1]. In the nearly same time, he also collaborated with the famous scientist Galileo developing a simple *pendulum* system, which oscillated at uniform periodicity at his own pulse frequency. The work of both scientists, however, has longer been overlooked, until 1707, when the memorable work “*Pulse-Watch*” from Sir John Floyer emerged. In 1852, Ludwig Taube was the first man, who published a longer fever curve of a patient. Some years after, the “*temperature, pulse and respiratory rate became the standard vital signs*” [1].

At the end of the 19<sup>th</sup> century, an Italian physician Scipione Riva-Rocci invented the “*sphygmomanometer*” or “*blood pressure cuff*”, which allowed for the first time to have a look at the arterial blood pressure (BP) [1]. Subsequently, the Russian physician Nikolai Korotkoff developed a method of auscultation for listening to body sounds generated by internal anatomic structures such as heart, lung, blood vessels etc. However, the actual breakthrough in the vital

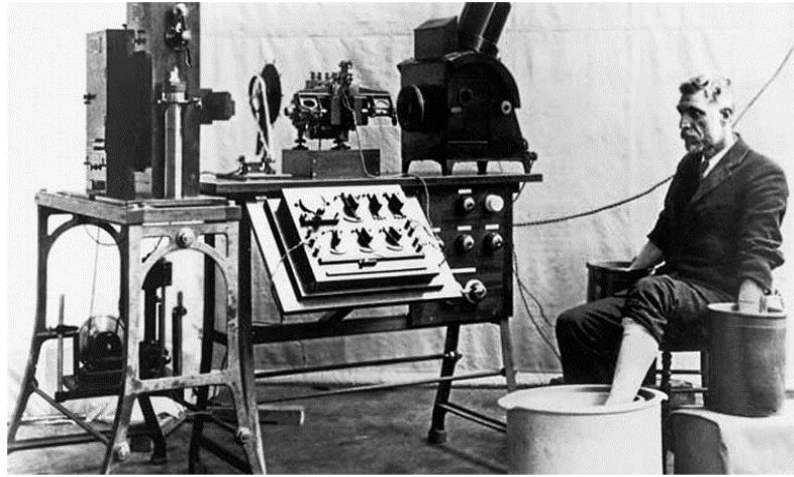


Figure 1. The Einthoven's string galvanometer developed in 1903 [12].

sign monitoring came true in 1903, when Willem Einthoven developed his “string galvanometer” applicable for measuring the electrocardiogram (ECG), a continuous wave chart of electrical potential curves derived from the heart activity [1], [12]. His refined ECG apparatus delivered much more accurate results and reduced the electrode number from five to three (Figure 1) [12]. After his inventory, the ECG has become soon an essential part of the daily clinical routine and is still acting as one of the most frequently used tools in the today's hospital diagnostics [12].

In the 20<sup>th</sup> century, a new epoch of monitoring has started with the development of the all-purpose computer “*Electronic Numerical Integrator and Computer*” and the first computer for the public sector introduced by IBM [9]. Increased popularity of those commercial systems, which have been primarily developed for the business sector, encouraged hospitals to develop their own information management systems similar to the today's EMR. However, the EMR systems technically differed from systems in other hospitals and had no connection to real-time assessment modules, what have led to further development and introduction of the bedside monitors in 1966 [9], [13].

Around that time, the usage of the *analog-to-digital converter* for connection between a commercial computer and bedside monitor enabled the physicians to collect biomedical data during and after surgery in ICU. Hereafter, the first analytical software solutions (e.g. for calculation of trend data) have been developed and integrated to the existing data acquisition tools [9], [13]. Figure 2 depicts the recording system proposed from Shubin and Weil in 1966. The analog physiological data of a patient is converted into the digital bit stream processed by the *central processing unit*. This central unit receives the signal after it was selected by the

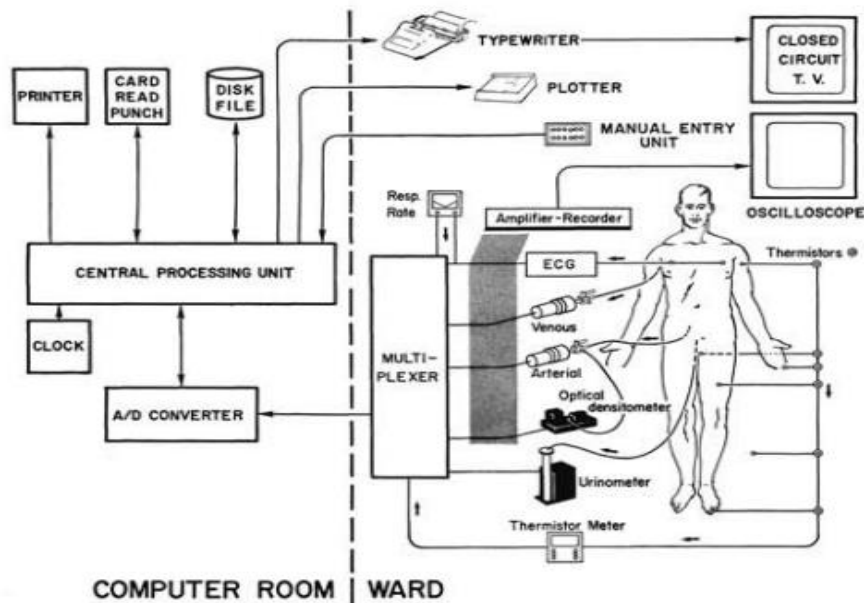


Figure 2: Connection of sensor equipment to the commercial computer IBM 1710 [13].

multiplexer and digitized by the analog-to-digital converter. The physiological signal is then forwarded to the processing unit by using a multiplexer. Finally, the information is stored to both the punch card serving for the subsequent statistical analysis and the disk card used for temporary data storage [13].

In 1980s, Hewlett-Packard developed its own healthcare management system known as “*Patient Data Management System*”, integration of what was associated with high costs and low adaptability [9]. In surgical and ICU division, the software couldn’t keep pace to constant changes of such a dynamic environment. Subsequent improvements in the computer graphics, digital system architecture, computer networks and graphical user interfaces (UIs) led to the emergence of electronic flow sheets, server/client architectures, local area networks and more user-friendly graphical interfaces [14], [15]. At that time in anesthesiology, the anesthesiologists must spend heaps of time to conscientiously prepare their patients for intubation or surgery. Because of that, Gravenstein (1986) proposed an extensive “*anesthesia recording system*” to track the patient’s health state during the operational tasks such as preparation and placing of endotracheal tubes, regulation of gas flow and preparation of the surgical drapes and kits for the subsequent use [9], [16]. The aim of the work was to facilitate and automate the process of recording vital signs during these time-consuming tasks. The software should improve the sampling accuracy, storage and visualization of perioperative data and support not only the anesthesiologists, but also other physicians contributing to the individual patient care [9], [16]. Despite all these improvements in the clinical monitoring area, the monitoring systems missed



intercommunicating components to freely exchange and accumulate the physiological data at one place. This issue has successfully been resolved by introduction of internet technologies, i.e. web-based browsers for creating modern user interfaces in 1990s [9].

### 1.3. Real-time assessment systems

Real-time data acquisition and processing tools became popular in time, the microcomputer units appeared on the market [1]. Those tools represent the essential parts of the modern bedside monitors using the microchips to provide much more powerful analysis than their predecessors from the past. According to Meyer et al [17], a *real-time integrated system* is considered as a tool for analysis of “*preoperative and operative data aimed to ensure that all members of the operating room (OR) team hold the same basic information, might improve OR communication, team performance, OR efficiency, and patient safety by increasing situational awareness*”. Moreover, they discuss the advantages of some non-expensive, portable systems with high-level synchronization and data extraction during different perioperative phases [17].

Georgia et al [9] developed a promising real-time assessment tool, an integrated system for data storage and visualization on the local storage media. It is able to connect to a variety of medical appliances, access and parse the data for calculation of the novel physiological parameters. Moreover, it “*provides real-time data acquisition, integration, time-synchronization, and data annotation of multimodal physiological waveform data (both analog and digital)*” [9]. In addition, the authors proposed an intelligent open middleware architecture for a complex multi-systemic data analysis. The data recorded from bedside monitors and other portable accessories was stored to a local database ready for a full access. By this, the clinicians got “*an integrated overview of the patient state (past, present, and predicted futures)*” every time about every patient [9].

Mothukuri and Kumar [18] utilized a low-cost technology to develop a remote monitoring system with an integrated microcontroller and mobile phone interface. Basically, they see a great potential in use of a wireless technology and describe a real-time assessment as “*continuous monitoring of the patient*”. Improvements in signal detection accuracy and warning system infallibility are essentials being seriously handled in their work. The defined such a system as “*patient management system*” according to [19], which “*deals with the*

*constant monitoring of health parameters using a palm-top like device and informing the service providers when ambulating conditions arise. It acts like a point-to-point system” [18].*

The following sections present some examples of the existing real-time vital sign assessment tools used in the clinical research and discuss their capabilities and limitations in respect to the biomedical data acquisition and real-time patient monitoring. At the very end of this chapter, the standard monitoring device, *Dräger Infinity® Delta* [20], used for tracking of the biomedical information of critically ill patients, is thoroughly discussed.

### **1.3.1. WvAPITest**

*WvAPITest* software was developed for demonstration purposes rather than for a high-resolution data acquisition and real-time analysis. The specialty of the tool is, that it was developed for programmers of the third-party applications using the recorded physiological data of patient monitors sold by *Drägerwerk AG & Co. KGaA* company. This meanwhile international company founded in Lübeck produces a broad palette of medical equipment, especially for anesthesiology, emergency, intensive care and pediatrics [21]. Since the tool needs at least one monitoring unit registered on the *Infinity® Network*, representing the Dräger’s proprietary secure network, it has to be installed on devices being in the same network. A detailed description about how the *Infinity® Network* devices are interconnected and how the data is retrieved from that network is presented in section 1.4.

Figure 3 presents the user interface (UI) of the *WvAPITest* tool. At first glance, it seems to be very complicated to identify, what all the functions actually mean and how they work. Moreover, it is not fully clear, how the monitoring device is accessed and what steps are relevant for the physiological data retrieval. In regards to the UI, the software can be divided into two separate components; the *feature set* component covers the operational units performing special data management tasks such as connection to the desired monitoring unit, whereas the *feedback* component notifies the user about the status of currently executing actions. In addition to the initial confusion about a huge variety of buttons, the software does not provide any information about the action’s order required to fetch and record new physiological samples.

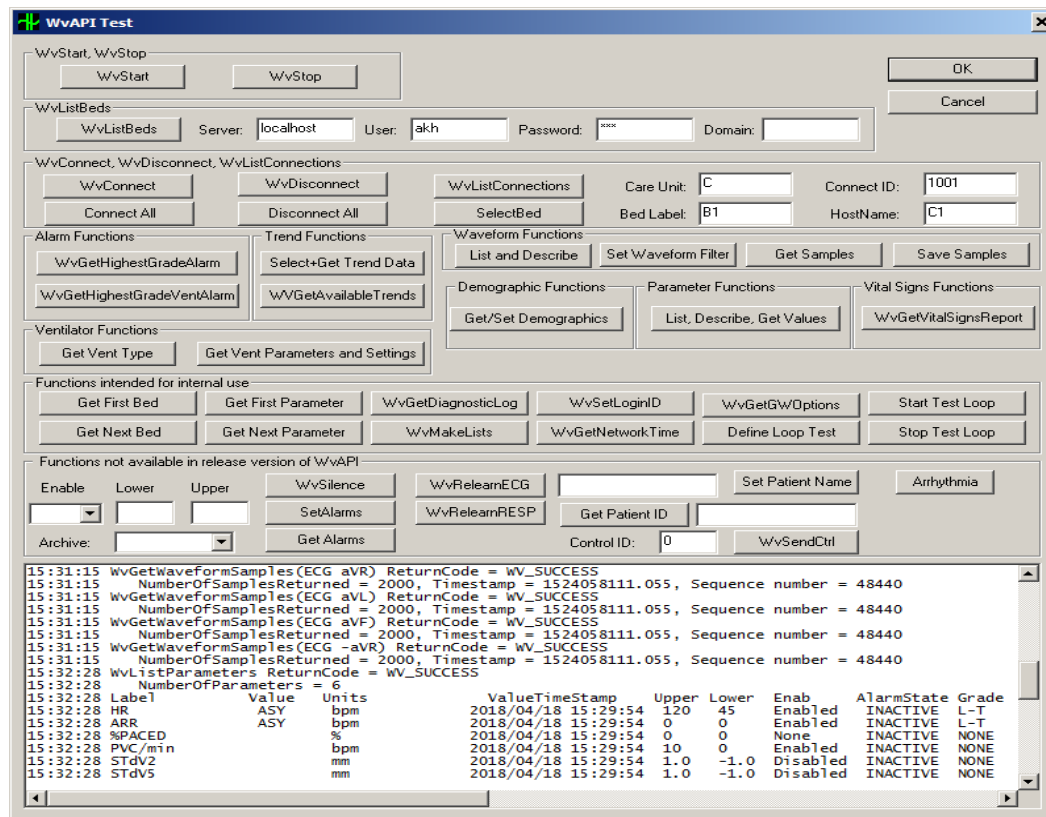


Figure 3. WvAPITest tool user interface encompassing functions for access to physiological data.

WvAPITest interestingly offers its users a possibility to retrieve the momentary information from a single, currently available monitor operating on the *Infinity® Network*. However, it is not possible to retrieve the data from more than one bedside monitor at once. Therefore, the software has to be run ten times, if ten different monitors are being accessed. This results to additional complexity for physicians and scientist monitoring several patients and makes their biomedical analysis unnecessarily difficult. In addition, the software does not provide constant monitoring of vital signs, it retrieves and stores only few samples in time, i.e. it is nearly impossible to perform any real-time analysis of vitals on patients at risk.

Finally, the tool does not provide any additional modules for data annotation and visual analysis of waveforms and vitals. The whole information fetched from the connected bedside monitor is presented by a series of symbols and timestamps (Figure 3). Moreover, the data fetched from the monitor is only exportable for a couple of seconds, what makes the successive data transfer for deeper analysis impossible. That's a huge drawback limiting the software application in real-time assessment area.

### 1.3.2. WvRecorder

*WvRecorder* has been developed by our *Institute of Electrodynamics, Microwave and Circuit Engineering, Vienna University of Technology* to support the data mining and data evaluation procedures conducted by the physicians and researchers in Vienna General Hospital. The new software implementation should address some of the problems related to the *WvAPITest* tool such as feedback transparency, short-term recording times as well as data storage and export management. The most significant improvement against the old tool was definitely achieved by simplification of the whole recording process, i.e. by listing of available monitors, managing their connections and finally implementation of a one-click export function for storage of physiological samples fetched from multiple monitors.

Figure 4 presents the results after *WvRecorder* development. The huge number of buttons on the previous tool was drastically reduced and a new table listing all currently available monitors was added. Thus, the user is able to view all monitors displayed in the alphabetical order according to the corresponding care unit. Furthermore, the identification of the patient became more transparent, since a more detailed information about the monitoring device was provided to the user, e.g. patient's name, its managing care unit, status about the connection and recording procedure (if clicking on the buttons *connect* and *start*, respectively). The whole process of data acquisition was stabilized and automatized, hence the users must only select their monitors of interest and start recording over a long period of time. All available biomedical signals were exported to one or more *comma separated value* (CSV) files and processed within more sophisticated analysis tools such as MATLAB from Mathworks [22].

Nevertheless, there are still some limitations worthy to discuss in regard to the *WvRecorder's* feature set. The monitoring systems doing a real-time analysis in e.g. anesthesiology must address the issue about the too low sampling resolution of the standard monitoring units [3], [9]. Although *WvRecorder* simplified the data acquisition procedure, it merely exports the data from the connected monitors, ignoring the calculation of novel vital signs and waveforms at higher resolution rates. This is a serious problem limiting the capabilities of physicians to explore and interpret the crucial adverse events [9]. The next issue associated with the *WvRecorder's* UI refers to presentation of the sluggish log information, which often raised confusion and time investment for studying of unnecessary messages having no relation to the vital sign and waveform data analysis. Finally, a sophisticated, real-time analysis tool has to

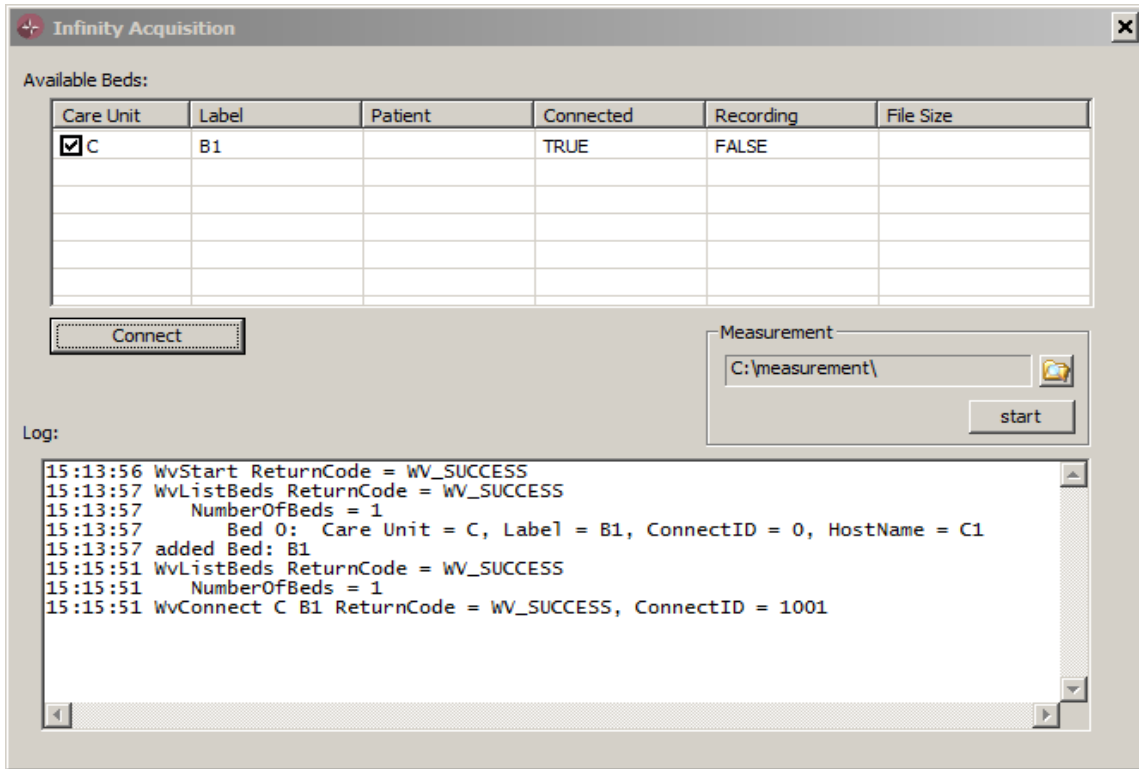


Figure 4. The user interface of WvRecorder facilitates the connection to monitors and recording of samples.

provide a convenient and highly-adaptable UI supporting proper data visualization [3], [9]. However, *WvRecorder* does not provide any graphical interface suitable for continual signal visualization. Thus, the user can only verify its exported data by either looking at the related bedside monitor at the same time or using additional scientific tools (e.g. MATLAB) to check the data in retrospect.

### 1.3.3. Other scientific tools

Though some non-commercial data acquisition tools successfully demonstrated their effectivity and efficiency in physiological data acquisition, storage and identification of the novel physiological trends, these tools are not always available for a wider spectrum of biomedical community and are mostly applied within the specific clinical research [9]. This limits a broader clinical acceptance as well as the integration of the tools into the hospital's daily routine. This chapter takes a focus on further developed tools having great impacts on health care and patient's safety. It also gives some examples of such tools and explains limitations in respect to their availability and adoption to patient's perioperative monitoring.

Meyer et al [17] utilized the common computer technology used in the modern ORs and developed a system for multimodal data acquisition from monitoring devices used in surgery and anesthesiology. The software run on the available technology, used the OR's secure

network and employed both the standard and the proprietary communication protocols provided by device's manufactures. Furthermore, it should ensure the availability of the recorded data after surgical interventions and avoid possible data losses to increase the data consistency and real-time data evaluation, in general. Next, a special attention was paid for appropriate data selection and display's justification presenting only the most important clinical information relevant for participated surgeons and anesthetists. In addition, the software should sophisticatedly select and present only sets of vital parameters critical for at least two participants on the surgery. This should minimize additional user interaction with the system and increase the personnel's attention to patients in surgery.

However, there should be mentioned, that the patient's real-time analysis performed by [17] is primarily focused on standard physiological parameters exported from ORs devices improper for high-resolution vital sign analysis as reported in [3], [9]. Furthermore, the underlying relational database running on a standard desktop computer archives the whole physiological data accumulated at disparate OR devices. Therefore, masses of data being potentially unused are stored to the database increasing the storage capacity demands. While the authors attach great importance to introduction of reliable alarm systems, there is no discussion about the integration of those systems to their existing software.

Stylianides et al [23] developed a promising vital sign assessment tool called "*Intensive Care Window*" acquiring real-time samples exported from diverse sensory systems. The tool was developed as an open-source software and supports consolidated data acquisition and evaluation on ICUs. It should also address several issues occurring during the real-time vital sign analysis such as lack of interoperability between disparate sensing and monitoring systems, use of heterogeneous communication protocols, discontinuity of signal acquisition as well as missing post-processing units evaluating the data in retrospect. The tool is basically composed of the *ICW bedside controller* and *ICW application*, ensuring a stable communication to the data sources as well as long-term data acquisition, respectively. Moreover, the developers successfully enhanced the central real-time assessment unit and integrated some additional algorithms for post-processing of the recorded data.

As shown in [17], a well-designed, easily-structured user interfaces for data visualization along with automatic display configuration can significantly reduce the times of physicians needed for interaction with the software. However, the *Intensive Care Window* tool partially requires

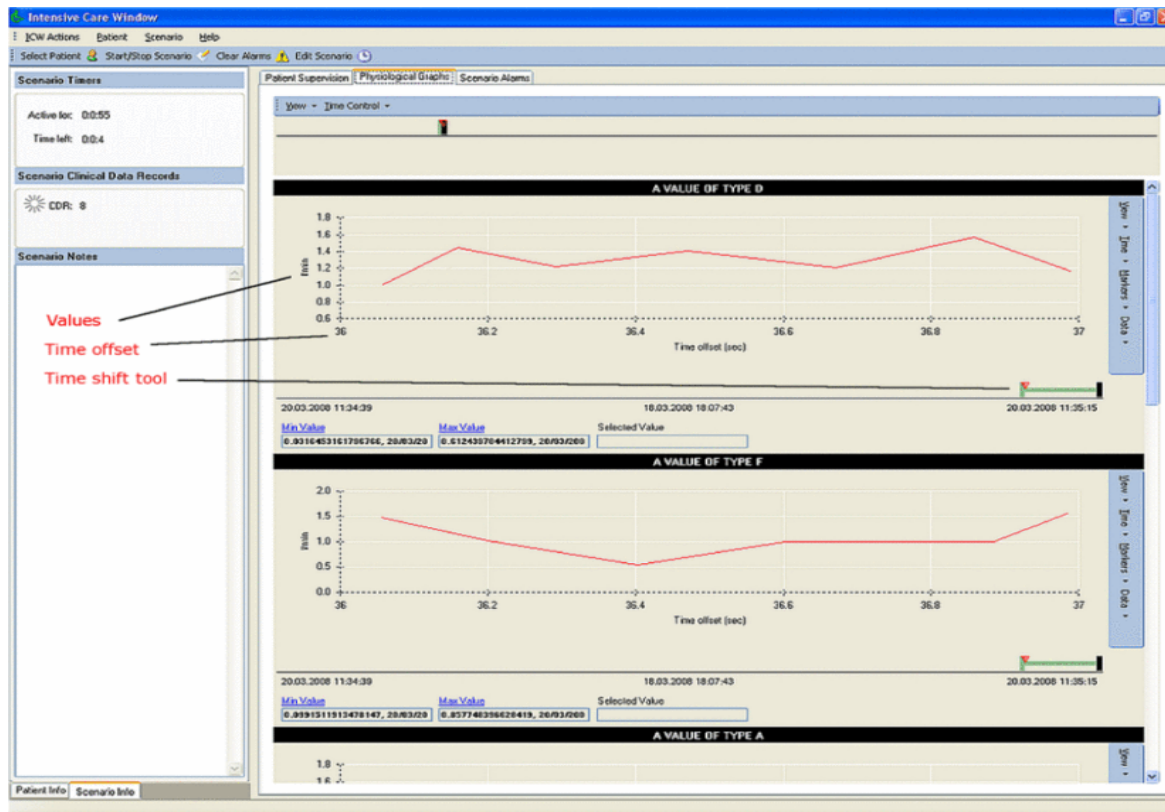


Figure 5. Real-time, vital sign analysis using Intensive Care Window tool [23].

a lot of user interaction for e.g. a deeper signal and vital sign search, alignment of particular signals within a single pane, marking adverse events during the analysis [23]. Allowing further configuration steps, the medical professional may spend several minutes for tool operation during the surgery. Many powerful features like playback of data history or visualization of numerous standard parameters might be useful, but they also contribute to the increased complexity in software operability and could negatively influence the recording accuracy (which wasn't verified during the project, yet), if retrieving data from multiple devices. The users of the tool have also missed an export tool for recorded biomedical information displayed only in "textual and graphical views" [23] as shown in Figure 5.

Recently, Lee & Young [24] proposed a very promising tool, *VitalRecorder*, facilitating the "high-resolution", "time-synchronized" signal analysis. It also supports automated recording of the physiological data on a variety of medical devices, especially "patient monitors", "infusion pumps", "anesthesia machines" as well as "cardiac output monitors" [24]. Furthermore, the recordings provide a high-resolution data in dependence on the analog-to-digital converter. E.g. 500Hz sampling frequency is typically applied during the ECG recording. *VitalRecorder* is runnable on almost on each Windows platform occupying the minimum amount of disk space (10MB). During recording, the data is exported to a local





Figure 6. The main pane of the VitalRecorder currently operated in the “track mode” [24]. At the top of the window, the menu bar with many useful functions such as data export is placed. “Track list” keeps the list of available (online) end-point devices, whereas the “Track window” presents the physiological data in real time. The “Time slider” provides an access to history of the recordings. The three panes on the right side aid to setup the end-point devices, to log the events and to start analysis by other third-party algorithms [24].

Windows machine as well as backed up on a network-attached storage. In conclusion, the *VitalRecorder* offers an effective multiparametric analysis performed on multiple devices simultaneously. It utilizes a minimum of disk space and is executable on all common Windows platforms leading to a better software usability and portability.

In comparison to other tools, *VitalRecorder* offers several unique functions on area of biosignal acquisition such as “automatic recording” or “remote monitoring”. However, the devices of various brands require additional hardware for operation (e.g. analog-to-digital or serial-to-USB converters). Moreover, the configuration of several communication protocols on the end-point devices is necessary before the real-time analysis [24]. This can mightily rise the costs of the initial software configuration in the ORs and cause synchronization problems during the subsequent recording procedures. Figure 6 presents the graphical UI of software. Although the device setup procedure (using the “Device pane”) seems to be straightforward, the “Track list” along with the “Track window” presents all vital parameter tracked in real time. Hence, the UI is rapidly flooded by masses of curves, numbers, acronyms and setting options being worthless for the actual real-time analysis. Finally, the tool supports a variety of the anesthesia and monitoring devices, however our target monitoring device, *Dräger Infinity® Delta* [20], which



is a standard of patient monitoring in Vienna General Hospital (AKH) is unfortunately not supported. Therefore, the biosignals sampled by the attached sensors and accessories of the monitor cannot be stored nor analyzed in real-time.

#### 1.3.4. Dräger Infinity Delta

*Dräger Infinity® Delta* [20] represents a multiparametric monitoring device utilized for monitoring of the vital signs of critically ill patients [25] (see Figure 7). It is one of the commercial medical systems applied within the current ICUs (especially on those in AKH Vienna). The device makes use of the manufacturer's patented technology "*Pick and Go®*" supporting continuous patient monitoring, especially during periods of patient's absence and transport. Thus, it ensures a sustained data collection, reduces the patient's unsupervised times and minimizes the usage of additional technical equipment during transport and visits of other house-intern specialists [20], [25].

As a bedside monitor, it is usually put on, so called "docking stations", which provide a capability to set, edit and store the properties about the presented waveforms and vital signs (e.g. location on screen, colors or numeric alarm limits). Furthermore, the *Infinity® Delta* monitors automatically establish a wireless connection if they are picked up from their docking stations. Thus, the recorded vital parameters are sent over the network to the central station, where they are remotely accessed [20]. Due to this feature, *Infinity® Delta* monitors ensure a continuous data acquisition throughout the whole perioperative period and represent therefore an ideal source of the biomedical information.

Each *Infinity® Delta* monitor facilitates a visual presentation of a variety of vital parameters. Examples for those represent the electrocardiography (ECG), respiration, pulse oximetry ( $S_pO_2$ ), temperature, the invasive and noninvasive BP, ST segment detection etc. [25]. Those parameters are regularly sampled and visualized (as continuous waves or discrete values) on the screens of those monitors. Nevertheless, the devices are capable of displaying standard physiological parameters only, missing the acquisition and visualization of the *novel*, clinically important parameters such as HRV, cumulative times of desaturation or hypotension [3]. The next chapter explains the advantages of the biosignal analysis based on the *novel* physiological parameters.

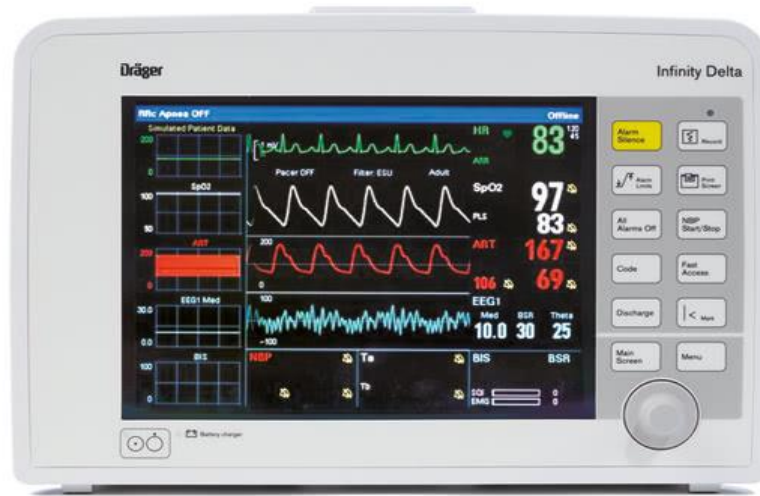


Figure 7. Infinity Delta Monitors act as waveform and vital sign sources [25]. .

## 1.4. Biomedical signal analysis

The previous sections present various data acquisition and real-time assessment tools and discuss their important features and limitations in regard to the biomedical signal analysis. Moreover, the standard patient monitor, *Infinity® Delta*, fetching the valuable biomedical information of critically ill patients, is introduced for the first time. Because of its unique features [20], it represents an ideal source of biomedical information crucial for real-time analysis and subsequent data processing. This work introduces a novel real-time assessment tool “*Vital-signs REal-time Analysis for Clinical Translation*” (VREACT) addressing the problems of the modern data acquisition and processing tools in the perioperative setting [3]. The tool was systematically designed and developed to access the biomedical information of the monitors and to ensure a consistent data evaluation and visualization. The target information sources represent the commercially available multiparametric monitors, *Infinity® Delta*, described in the previous section. Unfortunately, those monitoring devices doesn’t natively support the visualization of the *novel* physiological parameter. This chapter presents the important clinical parameters being exported and visualized by VREACT. It especially focuses on the *novel* physiological parameters extending the capabilities of the current patient monitoring systems [3].

### 1.4.1. Standard physiological parameters

This chapter presents the most important physiological parameters measured and visualized from a wide spectrum of commercial monitoring systems found in the modern hospitals. Especially, the parameters being fetched and stored by VREACT from the *Infinity® Delta* monitors are examined in a more detail. The *novel* parameters showing a great clinical importance, however, are not the topic of this section and are thoroughly discussed in next section 1.4.2.

### Electrocardiogram

An electrocardiogram (ECG) records the electrical activity of the heart and represents a valuable characteristic of the cardiac fitness. It is examined by placing electrodes on the specific body positions and by measuring the electrical potential differences between them [26]. The currents produced by the sinus node of the heart are conducted across the heart muscle, triggering the contractions of atria and ventricles filled with oxygenated and deoxygenated blood. The electrical pulses are introduced by changes of ionic forces as described in [27]. Dynamic changes in electrical conductivity of the heart can be summed up, creating an electric dipole, whereas its direction and size adapts according to the repetitive heart rhythm. Hence, the ECG represents a projection of the electric dipole onto the respective electrode axes in the course of some specific time period [26]. As a result, a continuous voltage-over-time curve is coined and measured by e.g. 12-lead system approved by *Society for Cardiological Science & Technology* [28].

Figure 8a [27] describes the most important part of the ECG curve mapping the process of cardiac activity. The P-wave starts the repetitive sequence of curve deflections towards the positive (depolarization) and negative (repolarization) voltage areas. It reflects the depolarization of atria. The consecutive QRS complex lasts up to 100 ms and represents the ventricular muscle contraction. After repolarization of ventricles (known as a T-wave), the U-wave reflects either the residual repolarization currents or activity of Purkinje fibers. The PR-time represents the time between depolarization of atria and ventricles, whereas the QT interval reflects the period of ventricular depolarization terminated by their complete repolarization.

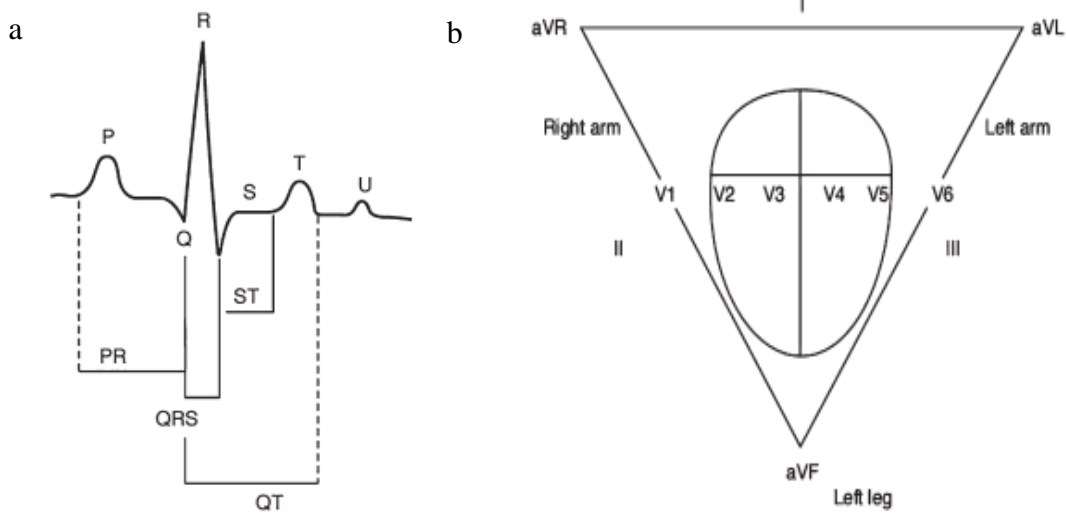


Figure 8. (a) The typical ECG curve of healthy individuals, (b) 12-electrode system located around the heart [27].

The standard 12-lead ECG combines the electrode configurations proposed by Einthoven, Goldberger and Wilson. It consists of 3 limb leads (I, II, III), 3 augmented leads ( $aVF$ ,  $aVR$ ,  $aVL$ ) and 6 precordial leads (V1-6) [29]. Thus, the combination of horizontal and vertical axes supports a more advanced analysis of the electric heart vector. Figure 8b visualizes the position of the heart and surrounding electrodes. The heart lies in the center and the leads surround the heart creating an abstract triangle [27]. Whereas the limb electrodes does not necessarily have to be placed on wrist and ankles, the placement of precordial leads is critical [29]. E.g. the electrode V1 maps the horizontal plane of the vector must be placed at “fourth intercostal space at the right sternal border”; V2 must be stuck at “fourth intercostal space at the left sternal border”, etc. Since the resulting ECG curves, especially their amplitudes and corresponding axes, are prone to the position of the body and recording electrodes as well as surface (skin) preparation before measurement [29], the clinical practices should comply with the guidelines and recommendations of international committees as described in [28].

## Heart rate

Heart rate (HR) represents the number of heart beats per minute and is derived from ECG's peak-to-peak (R-R) times  $\tau_{R-R}$  as stated in [26] :

$$HR = \frac{60}{\tau_{R-R}} [\text{min}^{-1}] \quad (1)$$

According the American Heart Association [30], HR is very effective and easy-to-do measure indicating the physiological fitness of each individual. It can easily be measured by counting the beats on the wrists, elbows, on the neck or even “top of the foot” [30]. The other handy and

noninvasive approach for HR measurement is so called *photo-plethysmography* method [26]. Hereby, the HR of an individual is determined upon its blood oxygenation level by measuring the overall absorbance of light passing through body's anatomic structures, especially arteries. A less common method for measurement of HR is *impedance plethysmography* [26]. In this case, small currents of 20 – 100 kHz frequency are injected into the body measuring the impedance changes caused by pulsating blood vessels. However, there are reasonable limitations about IPG in respect to high energy consumption and artefacts created by movements and small cardiac and muscular currents.

## Arterial blood pressure

Arterial BP is a measure of stress exposed to arteries during the blood ejection caused by contractile heart ventricles [31]. It can be decomposed into two separate measures, *systolic* and *diastolic* BP. The systolic BP defines the maximal force exerted on arteries after the heart's blood ejection, whereas the diastolic BP defines the minimal pressure in arteries, immediately before the ventricular contraction occurs. It is measured in *mmHg* meaning “*millimeters of mercury*” [32], since mercury was frequently used for precise BP measurements in the past. The overall mean BP in arteries (MAP) is determined by multiplication of cardiac output (CO) by vascular resistance (VR) [31], [33]:

$$MAP = CO * VR [mmHg] \quad (2)$$

whereby the CO specifies how much blood is being transported by cardiovascular system per unit of time and the VR reflects to the “*resistance of blood vessels to blood flow*” [33]. Hereby, MAP is used to determine the strength of the arterial blood flow crucial for organ nutrient delivery. A next suitable approach for the MAP evaluation is its representation as the double of the diastolic BP (*dBp*) multiplied by the systolic BP (*sBP*), divided by number of BPs in formula [31]:

$$MAP = \frac{2 * dBp + sBP}{3} [mmHg] \quad (3)$$

The clinical importance of regularly examined BP values can be critical in respect to patients with hyper- or hypotension [31], [32]. Whereas the hypertension is clearly associated with development of acute cardiovascular diseases (e.g. stroke, heart attack, etc.) and kidney malfunction, the hypotension is mostly a result of the insufficient CO [31].

## Oxygen saturation

In anesthesiology, a regular and systematical verification of the organ oxygenation is crucial for early detection of hypoxia and hypoxemia [34], [35]. Furthermore both, *oxygen saturation*  $S_aO_2$  and *hemoglobin concentration*  $[Hb]$ , exhibit a great clinical importance in respect to amount of oxygen in arteries [34]. Cardiopulmonary disturbances can already appear during the earlier metabolic phases such as pressure-driven oxygen uptake in capillaries or relate to the later oxygen transport phases, such as cellular respiration. Due to the complexity of the whole aerobic metabolism in our bodies, it is extremely difficult to develop one monitoring procedure to ensure a reliable run-time evaluation of the “oxygenation” profiles [34].

Spectrophotometry, a concept based on the Labert-Beer’s law, has already been used for determination of the  $[Hb]$  in early 1930s [35]. Several invasive and non-invasive techniques utilize this approach to measure “fractional” or “functional” saturation based on the hemoglobin type [34]. Especially, oxygenated and deoxygenated hemoglobin differ in their optical and structural properties, i.e. the oxygenated hemoglobin absorbs more light at the infrared electromagnetic spectrum and less light in the red spectrum than the deoxygenated hemoglobin. Pulse Oximetry ( $S_pO_2$ ) exploits the concept of light absorbance at different wavelengths (typically 940 nm and 660nm) and represents the current standard for oxygen saturation monitoring in both ambulatory and critical care [34], [35]. Pulse oximetric devices are typically composed of two Light Emitter Diodes (LEDs) alternately emitting in the red and infrared spectrum [36]. The emitted light passes through various body structures at different times being absorbed and finally caught by the opposite photodetector [36], [37]. The arrived signal carries two separate components, AC representing the light absorbance of the pulsatile arterial blood and DC referring to the absorbance of light by other surrounding structures as portrayed in Figure 9. The  $S_pO_2$  parameter depends on both AC and DC measures examined at two different wavelengths as mentioned above. Thus, the final  $S_pO_2$  value is determined as ratio of the pulsatile and non-pulsatile component obtained for two different wavelengths:

$$S_pO_2 = \frac{AC_{660}/DC_{660}}{AC_{940}/DC_{940}} \quad (4)$$

where 660 and 940 correspond to the wavelengths in  $[nm]$  [34], [36].

Similar technique used for more accurate oxygen saturation detection is referred to as reflectance spectrophotometry, which continuously monitors the status of oxygenation inside

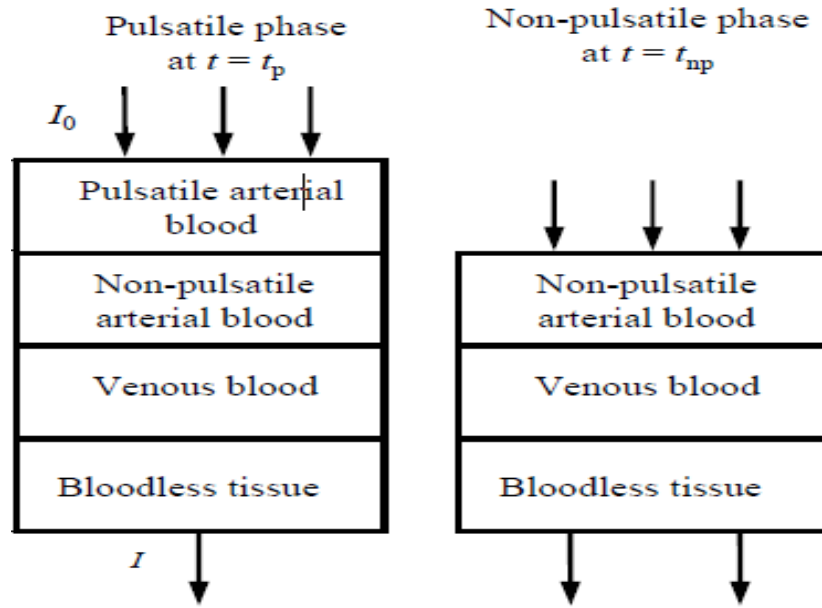


Figure 9. The emitted light of the initial intensity  $I_0$  passes through the different structures in living tissue ending up with the intensity  $I$ . The light is absorbed during the presence and absence of the pulsatile component differently [37].

a blood vessel. However, this technique is highly invasive and merely applicable in conjunction with an intravascular catheter [34].

### 1.4.2. Novel physiological parameters

This chapter takes a focus on the *novel* physiological parameters, which have not fully been integrated into the most hospital monitoring devices yet, but gained popularity in specialized clinical studies of surgical patients. These parameters (such as HRV, cumulative time of hypoxemia or hypotension) may reveal severe adverse effects and organ failures in the perioperative time periods such as atrial fibrillation [38], respiratory arrests [39] or acute kidney injury (AKI) [40]. Therefore, their studying and integration to the modern real-time assessment tools is exceedingly important. This chapter introduces the novel clinical parameters, reviews their essential metrics and discusses their relevance for the real-time analysis within the perioperative care.

#### Heart rate variability

HRV is a measure, closely related to ECG and HR measures, representing the extent of the heart beat fluctuations in time. Thus, the parameter describes the variations in beat-to-beat (RR) interval lengths [38]. HRV is a result of highly-adaptive, physiological reactions of organisms, especially of the cardiovascular system, to abrupt environmental changes influencing the homeostasis of each individual. A detailed investigation of the HRV parameter within the

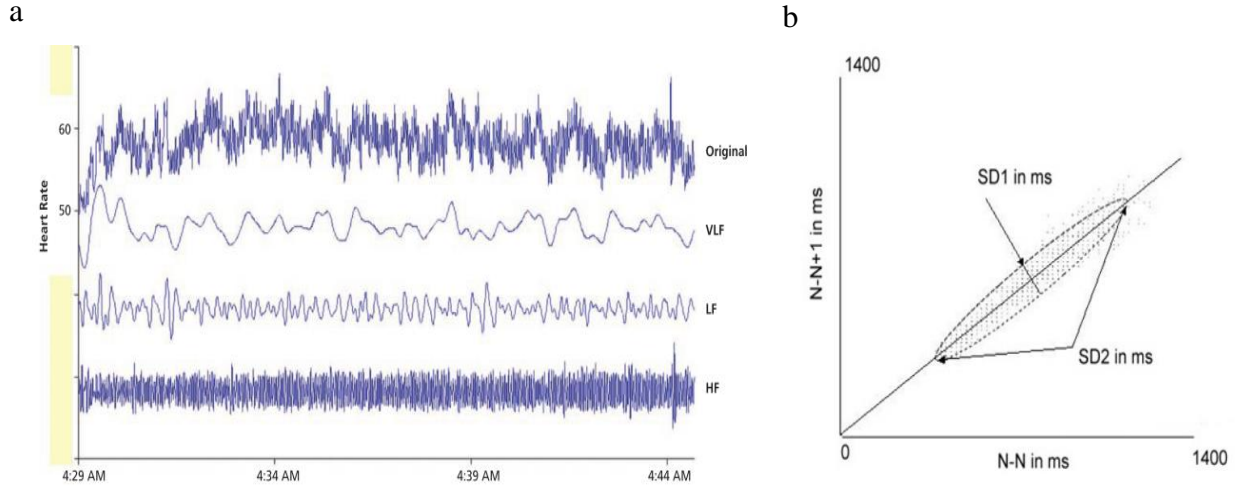


Figure 10. (a) Examples of different spectral components extracted from original HRV waveform [8], (b) Poincaré plot fitted by ellipse consecutively plots two neighboring RR intervals against each other [41].

different physiological setting revealed its non-linear association to other dynamic regulatory systems, especially autonomous nervous system (ANS) [38], [8]. Thus, HRV is the essential physiological measure evaluating the “*self-regulatory capacity, adaptability, or resilience*” of organisms and helps to predict severe cardiovascular complications such as *post-myocardial infarction* or *coronary atherosclerosis* [8].

In general, there are several measures and notations expressing the HRV in terms of detection time. There exist long-term (ca. 24h), short-term (ca. 5min) and ultra-short-term (< 5min) HRV measures belonging to the one of the three analytical approaches [38]:

- *Time domain analysis* is the simplest approach among those three, providing the statistical measures derived from variations in RR interval times. The most common parameters calculated upon a sequence of numerical values are standard deviation of normal-to-normal (SDNN), the mean of all standard deviations index (SDNNI) and the root mean square of successive differences (RMSSD) [38], [8]. The word “normal” in this context relates to the RR interval origin, i.e. those intervals generated by sinoatrial node are considered as “normal”, whereas the ectopic beats triggered outside the node are considered as “abnormal” [38]. In addition, HRV triangular index (HTI) belongs to geometric HRV measures calculating “*the integral of the density of the RR interval histogram divided by its height*” [38]. HTI along with RMSSD are strong indicators of arrhythmia.



- *Frequency domain analysis* offers broader analytical insights to the heart beat variations monitored over a long period of time (long-term HRV). Usually, those variations are detected as oscillations within a particular frequency range, hence, they overall frequency spectrum is divided into the four specific frequency bands [8]:
  - *High frequency (HF)* ranging from 0,15 Hz to 0,4 Hz is known as respiratory arrhythmia band, since it captures variations associated with the normal respiratory cycle,
  - *Low frequency (LF)* band ranging from 0,04 Hz to 0,15 Hz comprises the variations triggered by baroreceptors inside the heart and blood vessels,
  - *Very-low frequency (VLF)* range from 0,0033 Hz to 0,04 Hz is related to higher “all-cause” mortality rates rather than to a specific physiological phenomenon,
  - *Ultra-low frequency (ULF)* band comprises the variations with frequency less than 0,0033 Hz. “*Circadian rhythms, core body temperature, metabolism, hormones and intrinsic rhythms generated by the heart*” are the most influential contributors within this frequency spectrum [8].

Figure 10a presents an example of an HRV waveform being decomposed into three individual frequency components. Usually, a power spectral analysis is applied to visualize the different frequencies against its respective power (or amplitude) [8].

- *Non-linear analytics* assumes that RR interval variations exhibit a certain amount of dynamics and have no “fixed period” in time [38]. Thus, the variations neither happen randomly nor follow an exact repetitive pattern. Those methods account on unpredictable events resulting from the complex interplay of various regulatory systems. Thus, they occasionally deliver more accurate predictions and show correlation to certain time- and frequency domain measures [38]. E.g. A greater non-linear HRV index is generally associated with cardiovascular mortality and represents a better risk determinant in respect to patients with cardiovascular disease [41]. A *Poincaré plot* is very common graph utilized to capture those non-linear relations. It plots the RR interval times  $RR_i$  against their predecessors  $RR_{i-1}$  out of the whole interval time series and reveals new (normally masked) HRV patterns as stated in [41]. Fitting an ellipse to the transcribed Poincaré plot results in formation of new indices SD1, SD2 and their ratio SD12 as depicted in Figure 10b.

## Cumulative time of hypotension

Intraoperative hypotension, as a physiological reaction on homeostatic disturbances, has been closely associated with acute organ injuries caused by reperfusion of tissues after surgical intervention [39], [42]. Next, ischemia largely increases the prevalence of those injuries because it disturbs oxygen supply balance in organic tissues and may actually result in postoperative myocardial infarction. However, the postoperative AKI is more common [42]. Despite several definitions and indicators about the intraoperative hypotension, there is still a doubt about how to characterize an optimal measure for evaluation of this phenomenon. One possibility could be the detection of the lowest MAP (see section 1.4.1, Mean Arterial Pressure) or measurement of the time periods during hypotension by definition of *absolute* and *relative* MAP thresholds as reported in [42]. Therefore, cumulative times of intraoperative hypotension may be linked to other postoperative adverse effects, however no clear correlation to their exact duration times has been evidenced. Cumulative time of MAP can be understood as the duration in [min], at which the patient's MAP is less than a specific hypotonic threshold, e.g. for 65mmHg would be the time  $\tau_{MAP < 65}$  [40], [42].

## Cumulative time of desaturation

Similar to intraoperative hypotension, the postoperative desaturation is attributed to severe clinical complications such as “brain dysfunction, dysrhythmias, and myocardial ischemia” [39]. Those life-threatening complications may occur independent on type of surgery due to the low level of oxygen saturation followed by the “respiratory insufficiency” or even “respiratory arrests”. Moreover, the oxygen saturation is usually recorded manually in irregular time intervals and at patients waking hours. Therefore, the hypoxemia of patients at rest or during the sleep is often not recognized. Hence, a continuous, uninterrupted and regular measurement can reliably detect the times during hypoxemia and advise the nursing personnel for immediate intervention [39]. Analogous to intraoperative hypotension, cumulative times of desaturation can be interpreted as periods of time in [min], which are spent by patients below a specific oxygen saturation value, e.g.  $S_pO_2 < 85\%$  corresponds pulse-oximetric saturation level lower than 85% [39].

## 1.5. Aim of this work

The previous sections discuss the common problems associated with standard patient monitoring and data acquisition tools used in the modern patient care. A couple of data acquisition and real-time analysis tools utilized in the modern clinical research are presented. In spite of numerous attempts to develop a versatile data recording system, there is still no golden standard tool on the market satisfying the needs of both, the medical and scientific community. Although many patient monitors applied within the perioperative setting (e.g. *Dräger Infinity® Delta* [20] from section 1.3.4) produce a high-resolution data, the data is rarely used for the consecutive expertise, i.e. no retrospective analysis or scientific data processing are done. Moreover, calculations of some vital parameters (such as HR or HRV) lack accuracy or are completely missing. The above-mentioned aspects have a great impact on the real-time biosignal analysis and the results of studies based on the recorded physiological data.

This work tries to address the discussed limitations and propose a novel, real-time analysis, data acquisition tool, VREACT. Our tool facilitates a continuous, high-resolution vital sign monitoring and recording during the whole perioperative care, i.e. application of anesthetics, surgery, recovery room and intensive care unit treatment. It supports the anesthetists and surgeons by multiparametric real-time tracking of biosignals and offers more flexibility in the visual biosignal presentation. Moreover, it supports the clinical research by providing the automatic data acquisition on several monitoring devices simultaneously. Finally, the software is easily *extendable*, i.e. new software modules can be implemented and easily integrated to the existing software. Hence, the currently available set of parameters is easily extendable about the novel parameters, every time.

Beside the scientific analysis, we wish the tool will improve the decision-making processes made during the perioperative phase and contribute to a better patient-oriented care. What is more, we seriously handle the *functional* as well as the *non-functional* requirements of the software regarding its *usability* and *portability*. The tool should provide a more accurate and multimodal sample acquisition for multiple patients, simultaneously. By this, the clinicians should get much more flexibility to analyze the patients of interest by tracking the important biosignals in real-time and/or by exporting the biosignals for the ensuing clinical research.

## 2. Real-time assessment using VREACT

The previous chapter has introduced several analytical tools suitable for real-time measurements in the clinical area, especially in anesthesiology and on the ICUs. Since a continuous and effective biomedical signal analysis can assist the physicians and researchers with evaluation of the patient's state of health, a real-time assessment tool must minimize the user interaction times, support transparency of the software interface and provide more comprehensive data analysis than the standard monitors actually do. This work proposes an easily portable recording software called VREACT allowing a deeper analysis of vital signs in real-time.

The tool was developed at Vienna University of Technology (TU Wien) and successfully introduced to the clinical routine of anesthesiologists and intensivists in AKH. It consists of three different modules, namely the *Connection management*, *Patient recorder* and *Patient viewer* (as thoroughly discussed in Chapter 2.3). The design and evolution of the first two components (*connection management* and *recorder*) are the topics treated by this thesis. The *viewer* module, on the other hand, was developed by Fatih Kartal, a college from the equal field of study, Biomedical Engineering. The whole project was organized by our common project assistant Florian Thürk, which took a role of the project leader managing the most important resources of the project.

This chapter describes the design and development process of the VREACT software (with special regard to the *Connection management* and *Patient recorder* module) in more detail and reveals the technology and heuristics used during the software evolution. Furthermore, it examines the problems and difficulties coming out during the project and suggests clever design solutions and implementation methods. Finally, this chapter summarizes the requirements on the software and presents the results after the software development phase.

### 2.1. Software design

Before the implementation of the software started, a user domain and system analysis was performed to familiarize with the requirements of clinicians and with the current software development techniques guiding programmers to achieve a better software quality in a short period of time. Furthermore, an application of the structural patterns in the programming phase

could minimize the written amount of code and support the readability and understandability by other developers or programmers touching the software after a long time. Adding new components as well as applying extra patches on existing modules shouldn't cause grave issues, if a system of rules is followed during the whole development process. Several books have been issued helping developers to solve many problems of early software lifecycle including the process of software design [43], [44], [45]. Besides the evaluation of architectural and design patterns suitable for development purposes, the application domain and participating actors (stakeholders) have to be defined followed by definition and formation of the structure and workflow diagrams emphasizing the relationships between fundamental application components.

### **2.1.1. Stakeholders**

*Stakeholders* play an important role in software projects and include all representatives coming from different industrial sectors and business areas [46]. Hence, there are several roles and institutions participating on software projects delivering a solution for a specific user domain. Each project role put a variety of requirements on the system influencing the construction process of the software product. E.g. users of the system usually require functions facilitating their daily routine, the owners and managers of the company prepare a budget needed for the software development and finally the system administrators call for easy customizability of the system running in the product environment [46].

In addition, the requests from stakeholders shape the overall design of the system and have a great impact on its functional and non-functional properties (e.g. “performance and reliability, memory utilization, extensibility, usability and interoperability with other systems”) [46]. Therefore, requirements coined by the project participants was thoroughly analyzed and evaluated. Thereafter, a system was proposed and divided into the smaller functional pieces, such as administrative component (presenting the available care units and patient monitoring units), sampling loop component (gathering samples of physiological parameters), recording unit (exporting the samples to persisted files) and patient viewer (visualizing the available patients' vital parameters). Those design decisions have been met upon a deeper analysis of requirements worked out (primarily) from the clinicians, which have also become the main users of the system.

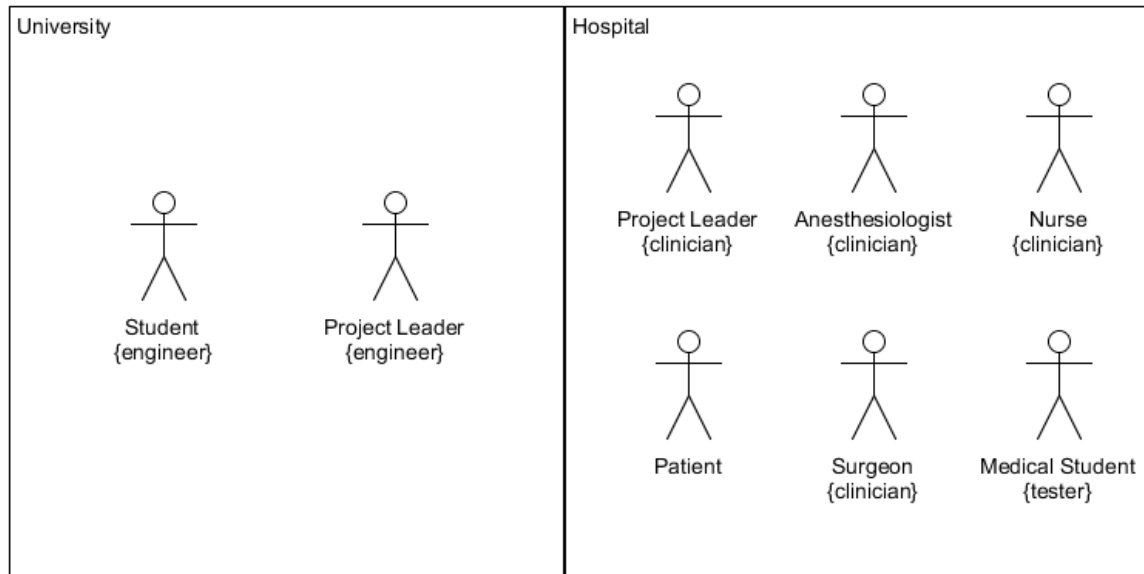


Figure 11. List of stakeholders participating on the development of VREACT. The project leader {engineer} absolved regular meetings with project leader {clinician} where the system requirements have been formulated. The requirements have been collected from other clinicians and students performing clinical research. The physiological data sampled from monitoring units was acquired after patients' agreement. The Student {engineer} proposed and developed solutions to fulfill project leaders' requirements.

Figure 11 lists the stakeholders participating on the VREACT project. The project was organized by Vienna University of Technology along with AKH. At the hospital side, a senior doctor from department of anesthesiology took over a role of the project lead gathering the requirements from other medical specialists, especially other surgeons and anesthesiologists. The nurses and students were also potential users of the developed software. On the university side, the project lead engineer helped programmers to derive and clarify tasks resulting from the requirements. What is more, both project leaders kept regular meetings to discuss the project key parameters and features being implemented by programmers in the next time. Since the physiological data primarily originated from perioperative patients hospitalized for a surgical intervention, particular declarations of consent had to be obtained before the recording procedure. The software was developed by two students coming from biomedical engineering discipline split to the two independent sections organized as two master thesis projects. The third (medical) student conducted a research study using the data visualized and recorded by our software implementation of VREACT. Thus, he operated as a software tester and verification engineer as well.

### 2.1.2. UML Use Case diagram

Use case diagrams usually offer an overview of the application domain and demonstrate possible interactions between the users and the systems [47], [48]. The diagrams highlight the

most important situations which may happen during the user interaction with the system. Even the best IT professionals recommend this modelling method as an addition to the textual definition of the requirements [47]. Use case diagrams merely portray the behavior of a system from the functional point of view, hiding specific details about complex structures and technologies integrated in the system. Thereby, the “goals”, “scenarios” as well as the primary “scope” of the system can be constantly followed by different roles and organizations during the project to avoid confusions and to encourage discussions about the evolved system [47], [48].

Unified Modeling Language (UML) defines the descriptive notation of modeling components crucial for formation of the use case diagrams. The most important model elements in UML use case diagrams are the system, actors, use cases and associations as described in [47], [48]:

- A *system* represents a collection of related use cases or actions being implemented under a specific scope. Usually, it is modelled as a rectangle with solid borders including one or more use cases related to the intended functionality.
- An *actor* “is a person, organization, local process (e.g., system clock), or external system that plays a role in one or more interactions with your system” [48]. Usually, an actor is visualized as a sticky figure outside the system.
- A *use case* represents a self-contained list of activities being performed to achieve a measurable and visible output for the actor. It is represented by a concise text surrounded by a solid-line ellipse usually placed within a system box.
- An *association* represents a defined relationship between two actors, two use cases or an actor and a use case. It mostly specifies allowed interactions, i.e. which actor is permitted to perform which action or denotes the generalization concept between actors and use cases.

Figure 12 presents the *use case* diagram of VREACT system. On the left side of the diagram, two main human actors of the system are described, a *clinician* and a *scientist*. Normally, a clinician wants to have a look at the currently available beds and the biomedical signals available for the particular bed. Therefore, a simple list over all available monitors (and patients) within a particular care unit must be presented first. In addition, the software must inform the user about the status of connection after each connection attempt. Since another actor, *Gateway Server* is requested to establish connections to the correct beds, a couple of use cases (e.g. connect to beds, list beds, refresh bed list, etc.) need both, a user interaction and

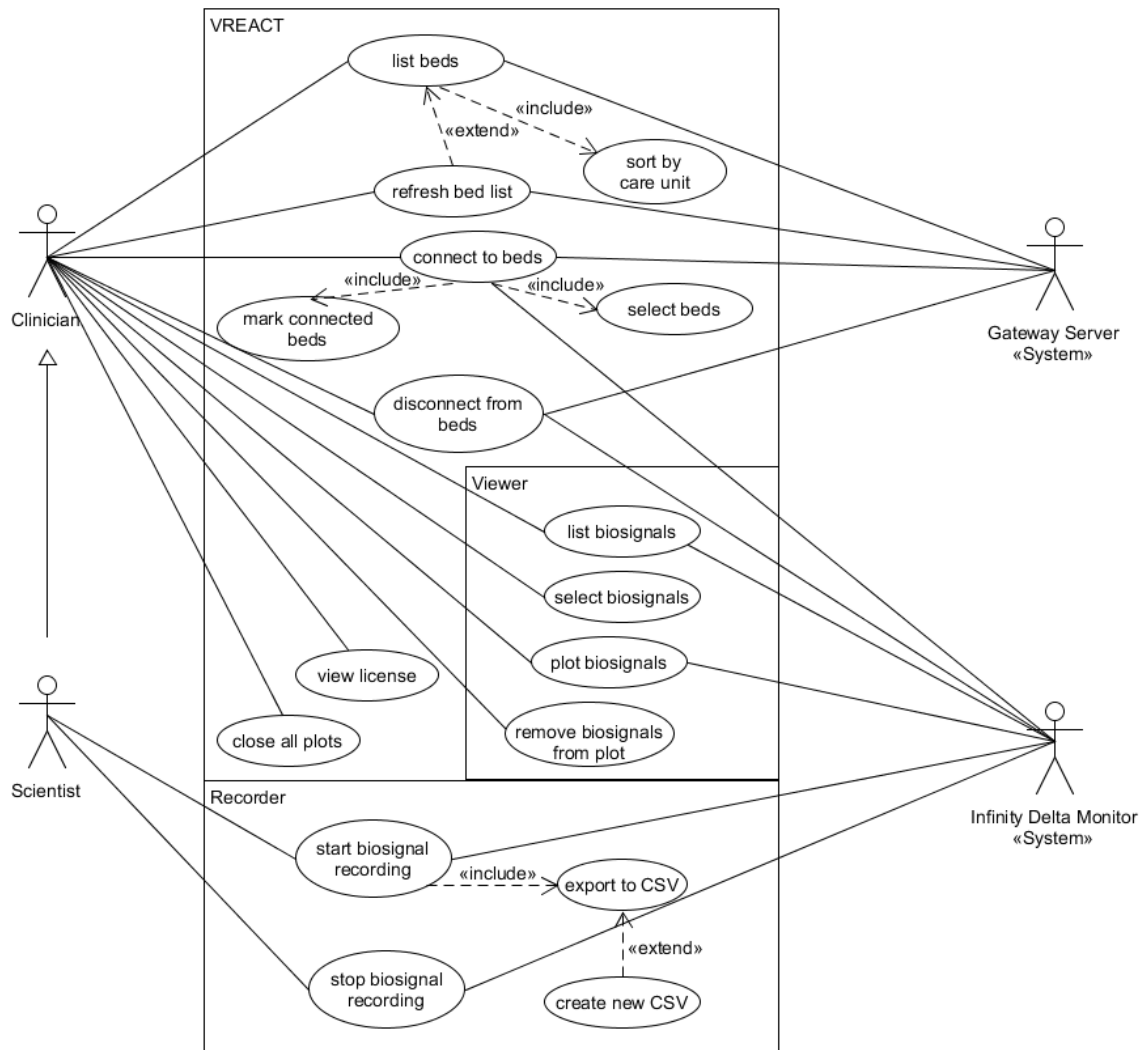


Figure 12. The UML Use Case diagram of VREACT portraying the initial design of the system. The software was designed to manage the connection to one or more beds (or patient monitors) simultaneously, to record and to view the biomedical signals fetched from the selected Infinity Delta monitors. Furthermore, the user is able to refresh the list of beds, view a license info as well as to determine the starting and end point of signal recording.

system respond as described in the Figure 12. Moreover, the system must list all the beds registered on the server and let the user select only the beds, he is interested in.

At the bottom of the diagram, two central components supporting the real-time analysis of the biomedical data are presented, the *viewer* and the *recorder*. The *viewer* must include a clear and concise list of all available biomedical signals for a given bed. The user must get the possibility to open a plot window, select one or more biomedical parameters and track them as continuous graphs or constantly changing numbers in real time. In contrast, the *recorder* must fetch new physiological samples from the *Infinity® Delta Monitors* in a continuous manner and persist the information without any gaps on a local machine. These two components must be run independently, i.e. the user can either record samples, view signals or perform both



actions simultaneously. Conclusively, the system should be very flexible including all features needed for real-time analysis in the biomedical field. Similarly, it shouldn't constrain the user to accomplish his analytical tasks and orders without others' help or additional support.

### 2.1.3. Workflow diagram

Previous sections introduced the most essential roles of the project and several feature requirements put on our real-time assessment tool, VREACT. The imperative requirements have been defined and summarized, several use case scenarios were tested bringing us to the diagram as specified in Figure 12. Unfortunately, a *use case* diagram doesn't map the chronological aspect of user interaction focusing merely on functional views of the system. What is more, the stakeholders are usually coming from different sectors and business areas. Therefore, we need a clear modeling method which provides a good understanding of all participants about the project's scope and of course, the technology fulfilling the requirements and specifics of the particular business sector.

*Business Process Modeling* (BPM) is a broad discipline often applied during the early software development phases [49], [50]. It is primarily used to design workflows of different organizations supporting the analysts to identify the most important steps within a business process. Second, BPM should encourage the optimization and implementation of the whole business process. A result of the modeling is normally a rigorous flowchart consisted of several model elements (e.g. blocks and transitions) resembling the procedural (step-by-step) programming approach [49]. Data flow approach “*outlines the flow of the activities in the model*” [50] and is the most commonly used technique utilizing the text and visual graphics at once.

Figure 13 presents the flowchart diagram portraying the workflows of clinicians performing a real-time analysis using VREACT. After starting the software, a connection to a server is automatically established and the list of currently available beds is retrieved. Subsequently, the user selects one or more beds to connect getting an immediate feedback about the success of the connection for each single bed. If no bed is present, the user may register a new bed on the server and then refresh the list of available beds. All the registered monitors being online should then appear in the list. If the connection to the monitors succeeds, the software starts sampling, i.e. retrieving the biomedical samples from the connected beds. The samples are stored (at

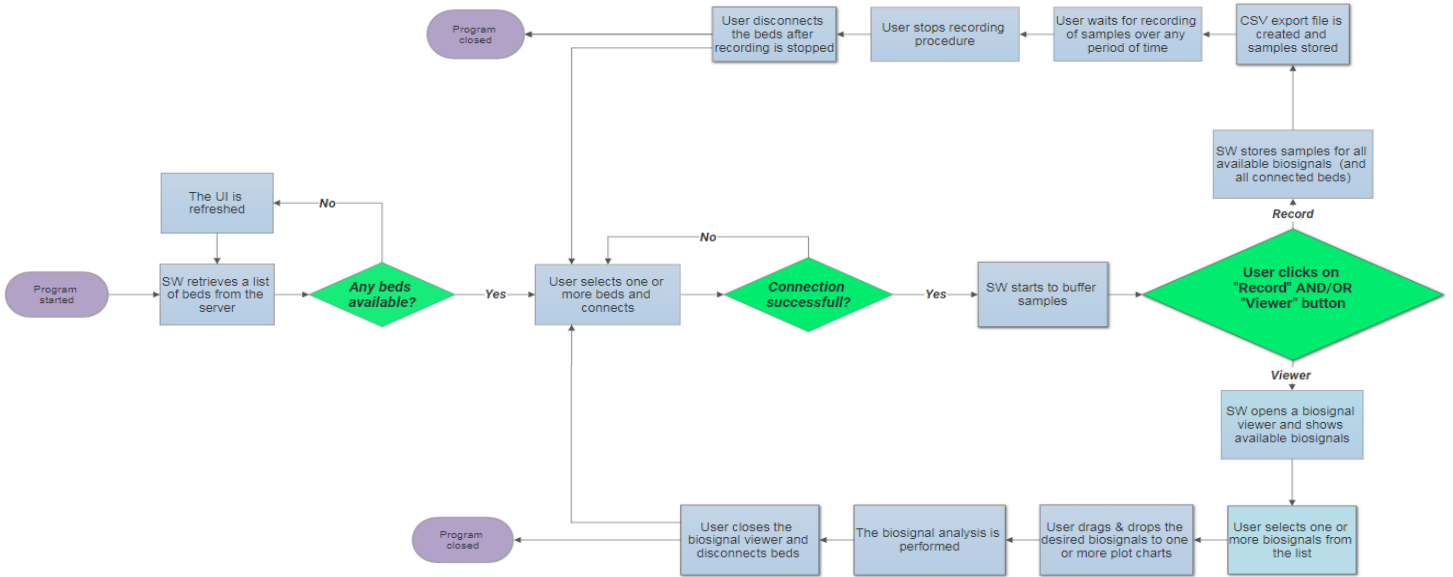


Figure 13. BPM flowchart illustrating the workflows of physicians and scientists using our clinical software VREACT. The software (SW) retrieves the available beds on the server to which the user might connect. After successful connection to one or more bed monitors, the user can record and/or view the biomedical signals for the given patients. After the real-time analysis is finished, the beds are disconnected and the software is closed.

least) in a short-term buffer, which is the main resource for the constant visual presentation and the biomedical recording of physiological values. After the user finishes the above-mentioned operations, he simply disconnects from the recorded beds and the program can be closed.

Since the core functionality of the software is associated with the recording and visualization of biomedical parameters, the special attention to these processes must be paid. Generally, the user has an option to open a plot window for any bed, getting an overview about the signals which are actually sampled and presented on the given patient monitor. He merely selects the signals of interests and starts the signal visualization for some time. Simultaneously, the user can record samples for any bed of choice. Thus, the software exports the samples to a specific file which might be imported and processed by another software environment (e.g. MATLAB). In both cases, the user takes a control over the processes and must terminate the procedures after finishing his analysis.

## 2.1.4. UML Class diagram

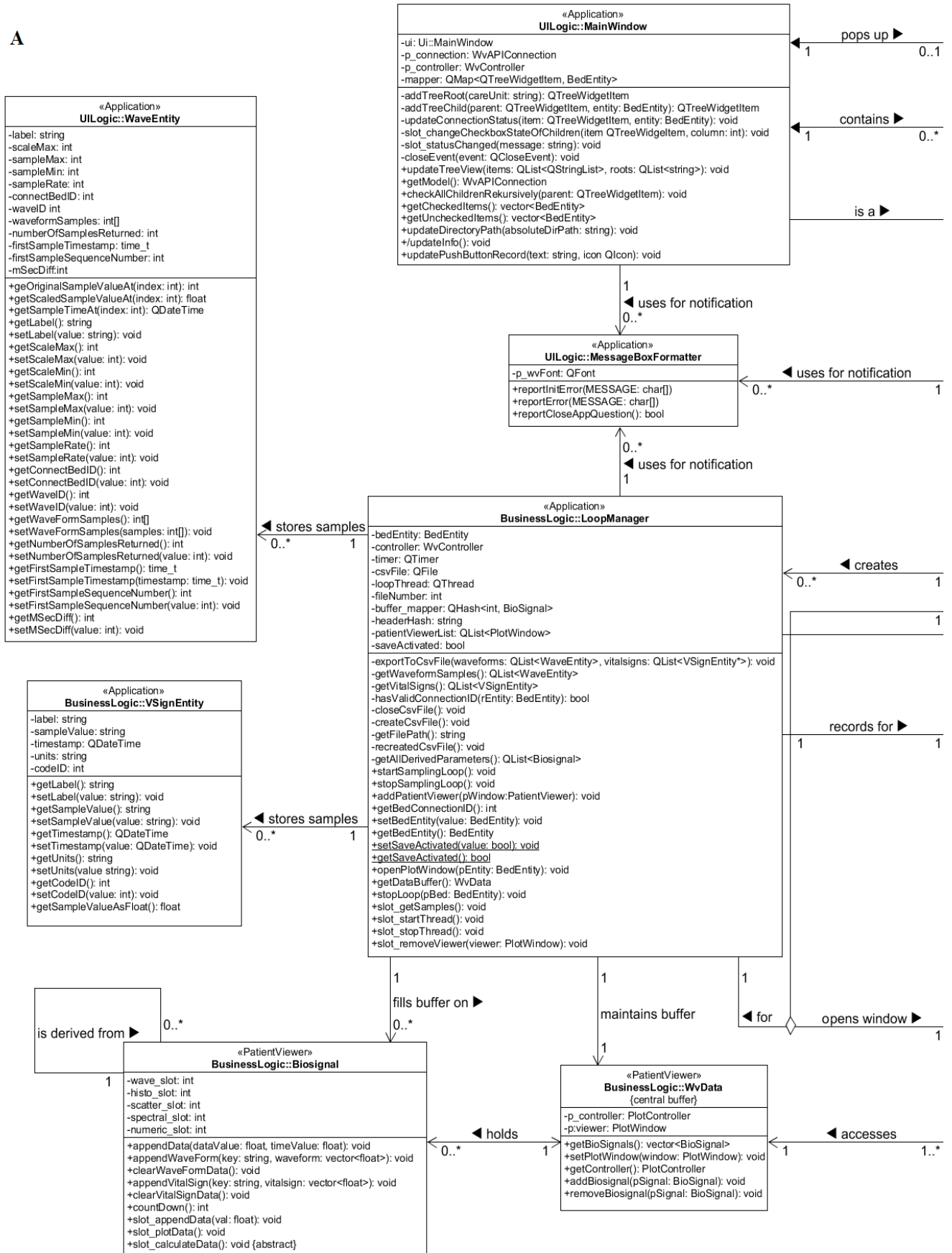
Class diagrams enable a structural view of the system and belong to the best formal modeling methods in area of conceptual analysis [51], [52]. They are often used in the object-oriented programming world to clarify relationships, structure and behavior of instances living within a certain domain. Each instance follows a recipe or a template being used for its construction.

Those templates are known as *classes* [52]. Moreover, the classes specify the *attributes*, i.e. the static properties of instances as well as their *operations*, i.e. functional units stashing a certain logic behind a precise single-row definition. Since a great majority of the classes represent objects of real world, the class diagrams describe relationships between them in an intuitive way. Thus, all usual UML concepts such as “associations, aggregation, composition, dependencies, inheritance, and realizations” [52] are applicable for modelling a system structure with classes.

Similar to other software modeling techniques, the class diagrams utilize several model elements worthy to mention. The *classes* are normally modelled as solid-line rectangles divided into the (at least) three separate parts. The first section describes the class name, whereas the second encompasses all *class* and *object* specific attributes. The last section lists the operations modifying the object states. The “*interactions*” between the classes define their mutual relationship and are visualized by solid lines traveling from one class boundary towards the other one [53]. Adding arrows to the specific line positions usually defines the direction of communication, i.e. the relationship can be limited to “one class uses the other” (*unidirectional relationship*) or “both classes correlate with each other” (*bidirectional relationship*). Finally, the multiplicities (small numbers near the line arrows) determine the number of instances (of class A) being in relation to other instances (of class B). Usually, they are specified as numeric intervals (e.g. “0..2”, “1..\*”) to restrict the actual number of instances existing in the same time. Those numbers are positioned near the second class in the reading direction, i.e. the class to which the arrows points [52], [53].

Figure 14a-b illustrates the UML class diagram of VREACT’s *connection management* and *recorder* part. Hereby, the most important classes represent *WvAPIConnection* and *LoopManager*, since they both maintain connections to the patient monitors by using specific *WvAPI* function calls. If the connection succeeds, the user interacts with the main panel (*MainWindow* class), which in turn delegates his queries to the *WvController* keeping the event handling logic. The controller modifies the state of *WvAPIConnection* model and waits for notifications from the model. If some notification arrives, the controller executes the corresponding logic and informs the view component (*MainWindow* class), if the model haven’t done so, yet. This programming approach is very powerful and commonly used as the *Model-View-Controller* (MVC) pattern [54]. For more reading, see the section in 2.1.5, MVC pattern.

A





Beyond the connection procedure, which is primarily managed by classes designed according to the MVC pattern, there must be a way for getting new samples from the monitors in a periodic manner. A *LoopManager* instance may exist only for a single *BedEntity* instance. The class covers the central functionality for signal detection and sample storage. Its instances facilitate the detection of biomedical signals, retrieve the physiological samples and persist the corresponding samples on the underlying local machine. The lifecycle of the *LoopManager* instances is managed by the *WvController*, which controls the whole sampling process. In addition, the *LoopManager* instances constantly add new physiological samples to the pertinent biosignal buffers (*BioSignal* class) and transport data throughout application tiers by using *data transfer objects* (DTOs) [55]. The DTOs are objects storing data, which is passed either through different processes or application tiers. Our class diagram presents three different DTO object classes:

- *BedEntity* instances carry the information about the connected bed such as patient and care unit name, patient ID, bed connection ID, list of available vital parameters, etc.
- *WaveEntity* instances store the information about the vital parameters available as sample sequences, i.e. waves. They also keep a detailed information about the sampling rate and timestamps specifying the number of samples and times of their recording.
- *VSignEntity* instances carry a single vital sign value sampled at the particular timestamp. They also store the information about the vital sign name and measuring units needed for plots (or files), to which the data is sent and visualized (or stored).

### 2.1.5. Design patterns

Over many years, tones of successful software applications enjoyed a broad acceptance throughout the heterogeneity of industrial sectors. Long-time experiences of many project leaders and engineers have demonstrated useful approaches contributing to a better software quality and a solubility of dozens technical problems [43]. These effective methodologies have gradually evolved to powerful tools inevitable for modern programming praxis and are known as "patterns"[43]. *Design patterns* represent a majority of them and solve repetitive problems emerging during the conception of the software design [54]. They provide practical templates guiding engineers to identify possible implementation defects and propose concrete solutions across several application domains. The following sections focus on the most important design patterns applied during the software design and development phase of VREACT. A short summary about the advantages and disadvantages of those patterns is also given.

## MVC pattern

The *MVC pattern* gives an elegant solution for separation of the UI from the underlying logic [54], [55]. Its use goes back into the 1970s, when a first MVC framework was developed.

Using this pattern, the application's backbone consists of (at least) three smaller object groups:

- a *model*, which represents an object operating on the data,
- a *view* with the aim to graphically present the data after processing,
- a *controller* managing the communication between the model and the view.

Figure 15 depicts the roles of the individual MVC components and describes the relations to each other. The *model* contains the domain data providing various operations in dependence on the character of business. Those operations are triggered by the *controller* (or the view), which typically receives and verifies the user input. After that, the input data is passed to the model for processing. The *view* waits for the output data and forces updates on different UI widgets (e.g. tables, lists, labels etc.) [54], [55]. Conclusively, there are two important separations achieved using the MVC approach: (1) the separation of the model from the view and (2) the separation of the view from the controller [55].

In case of VREACT, the core application components have been designed according to the MVC pattern as follows:

- *WvController* class enjoys the properties of a typical *controller* depicted in Figure 15. Its implementation includes two important pointers referencing the *model* and the *view*. Since the *controller* object is already associated with the *view* during its instantiation, the information about the bed availability is presented to the UI in a timely manner. Thereafter, the *controller* is prepared to take any inputs from the view or send updates to it, if necessary. Figure 14b demonstrates the structure of the *WvController*. The class provides an access to the most important components of the application: the recording and visual presentation of the biomedical signals. In addition, it implements a simple error messaging system informing users about restrictions and failures occurring during the real-time analysis.
- *MainWindow* class (Figure 14a) is the starting window of the application. According to the MVC methodology (Figure 15), the class is considered as a *view*, since it specifies the UI widgets and their layout on the screen. It is derived from the *QMainWindow* class which provides an additional framework-specific code to it. *MainWindow* instance contains important UI widgets like buttons, lists and bars utilized for a user

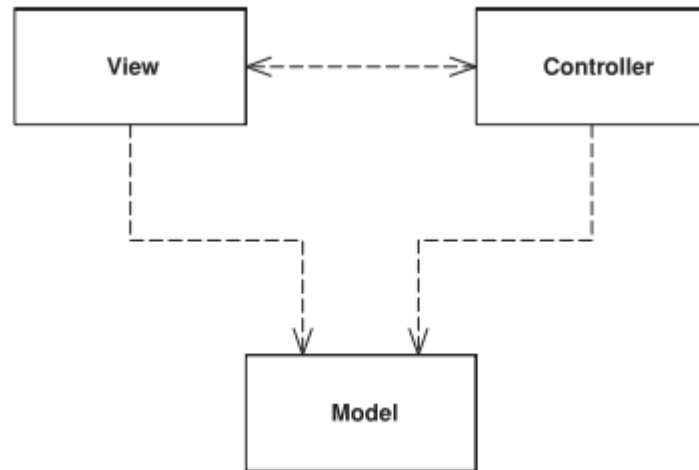


Figure 15. Model-View-Controller (MVC) pattern involves at least three individual components interacting to each other. The model controls the data, the controller handles the user inputs and reacts on changes in model. The view updates the UI after the model state is changed [55].

interaction. Since the cross-platform application framework from QT has been used for implementation of the UI, various features enhanced the visual presentation of the UI components and improved the communication between various application modules (such as *Signal & Slot mechanism* [56]).

- *WvAPIConnection* class (Figure 14b) defines the model of our MVC. It provides the connection to the server, which in turn maintains the connections to the individual bed monitors. Furthermore, it creates and updates the list of available bed monitors being accessed from the user. The state of the *model* is constantly changing, since the information about the connections and monitors is highly dynamic. *WvAPIConnection* can occasionally throw *WvAPIException* objects in case of errors. After an error message is created, it is sent to the controller, formatted and presented in the UI.

## Observer pattern

The *Observer pattern* belongs to other popular patterns solving the “one-to-many” object communication problems [44]. It focuses on objects (*subjects*), changes of which affect other objects (*observers*, *listeners*) reacting on these modifications. Usually, a *subject* has several *observers* being automatically notified if the state of the subject changes. Therefore, the subjects don’t necessarily need to know about its observers, i.e. they merely send information about the particular changes to all subscribers waiting for any kind of observer’s notification [44].

This pattern is commonly applied along with other design patterns, such as MVC pattern [44].



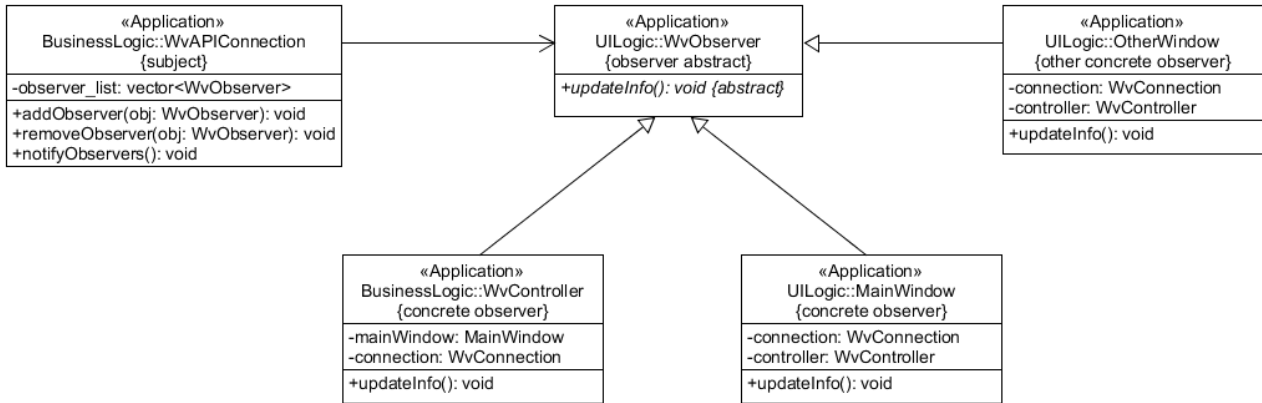


Figure 16. VREACT classes participating on the communication between the model and the view according to the Observer pattern. *WvAPIConnection* keeps the status of the server connection and the bed availability. Therefore, it represents the subject. In comparison, *WvController* and *MainWindow* performs specific updates based on the model's state. They are seen as observers.

It efficiently solves the problems with fast updates of different UI components and drastically reduces the coupling between the UI and business logic. Moreover, several implementations of the graphical UI can be run without a need to reimplement the underlying model [44], [55]. Figure 16 presents some classes of VREACT designed according to the *Observer pattern*. Besides the list of available monitoring devices, *WvAPIConnection* maintains a vector of observers being registered by the *addObserver()* method. Whenever the status of the object changes, the function *notifyObservers()* triggers updates on all “concrete” observers including the *WvController* and the *MainWindow*. Thus, the *WvController* changes its internal state and notifies further external modules belonging to the *recorder* or the *viewer* part (see Figure 12). Analogously, the UI object *MainWindow* retrieves new data from the model and triggers updates on other UI widgets ensuring the consistency of the presented content.

## DTO pattern

The *Data Transfer Object (DTO) pattern* solves the problem of passing objects throughout different structure layers and processes in applications [55], [57]. It allows a consistent transfer of data being sent throughout various application modules using only one single call. A DTO represents a container, which stores the domain specific data missing any additional logic, which could otherwise modify the state of the DTO. The Observer pattern helps to bundle the associated data and to reduce coupling between the different application layers [57]. Finally, it optimizes the number of calls and the size of formal parameter lists [55], which might negatively influence the code complexity and its readability.

a

«Application» <b>BusinessLogic::BedEntity</b>
-connectionID: int -hostName: string -careUnit: string -bedLabel: string -patientID: string -connectionStatus: ConnectionStatus {Enumeration}
+getConnectionID(): int +getHostName(): string +getCareUnit(): string +getBedLabel(): string +getPatientID(): string +getPatientName(): string +setConnectionStatus(CONNECTION_STATUS: ConnectionStatus) +setConnectionID: int

b

«Application» <b>BusinessLogic::WvAPIConnection</b> {Singleton}
- p_hostname: char[] - p_username: char[] - p_password: char[] - online_beds: vector<BedEntity> - observer_list: vector<WvObserver> - countForConnectedBeds: int
<b>- WvAPIConnection() {private constructor}</b> <b>- WvAPIConnection(conn: WvAPIConnection) {private copy constructor}</b> <b>~ WvAPIConnection() {private destructor}</b> <b>+getInstance(): WvAPIConnection</b> - notifyObservers(): void + connectToServer(): void + disconnectFromServer(): void + connectToBeds(beds: vector<BedEntity>): vector<BedEntity>

Figure 17. (a) *BedEntity* represents the most frequently used DTO in VREACT. It transports data about the connected bed such as connection ID, bed label, patient ID and the currently available connection status. (b) *WvAPIConnection* was implemented as Singleton having a private constructor, copy constructor and destructor (in bold). This prevents the object instantiation and destruction from outside of the class. The instance can only be access using a static method “*getInstance()*”.

Figure 14 presents three DTO classes transferring a bed specific (*BedEntity*) and patient related (*WaveEntity*, *VSignEntity*) information throughout different layers of application. The classic 3-tier architecture was applied [54]. Each individual patient monitor in hospital provides information about the network connection, care unit and physiological data of patient being monitored. Since this information is extremely important for the user, the data must be transferred to the UI in a consistent and efficient way. For example, the *BedEntity* instances focus on the most critical data required for identification of patient monitors across different care units (Figure 17a). The remaining DTOs, *WaveEntity* and *VSignEntity*, transport the physiological information of hospitalized patients. Moreover, they serve as temporary buffers storing the recorded samples and transport samples to the global biosignal buffer (*WvData*). Since the pattern is very powerful if using a rather smaller amount of domain entities [57], its positive effects on the application structure were immediately evident.

## Singleton pattern

The *Singleton* pattern allows the existence of only one instance of the same class at the same time [44]. In comparison to others, the instance can be accessed globally, i.e. from any program location. However, programmers have to be careful if they use Singletons in multi-threaded applications [55]. Since VREACT is built as a multithreaded application, i.e. various code sequences can basically be executed in parallel [58], the usage of singletons is restricted to minimum implementing only the *WvAPIConnection* class maintaining the access to the server. Figure 17b presents the exact implementation of the *WvAPIConnection* having the typical Singleton class features. The private constructors and destructors prevent any repetitive object

construction or destruction. The static method *getInstance()* provides a global access to the singleton instance passing its reference to the caller. The instance is created during the program start and has a role of the *model*. Since it is merely accessed from the main thread of the program, no synchronization code was required.

## Summary

In general, design patterns definitely contributed to a better code quality and offered effective instruments to solve serious design problems during the early software development phase. Furthermore, they simplified sharing of the global application resources and introduced useful functions for UI synchronization. Next, they facilitated the extensibility of the existing modules and enabled the exchangeability of application code, i.e. the UI layer code can be easily exchanged any time without touching the underlying logic. Finally, the application of patterns improved the speed and the consistency of the data transfer throughout the application tiers and raised the cohesion of the transported data. Conclusively, the effort put into the studying and realization of the design patterns was profitable.

## 2.2. System implementation

The previous chapter presents the crucial design methods applied before and during the development of the software. In addition, it takes a focus on several design and modelling techniques reflecting the software structure and its functionality. Finally, it briefly reports about the project environment, supported clinical workflows as well as code design techniques facilitating the product implementation and its future application. By this, the chapter summarizes all methods and techniques contributing to a better software quality and software extensibility for other features implemented in the future.

In contrast, this chapter presents the technology used for the development of the software. It primarily focuses on the tools and technical environments necessary to accomplish the most challenging programming tasks. As a next, the operation of the proprietary hospital's environment acting as underlying platform for the software application, is thoroughly discussed. *Infinity® Gateway Developer's Tools* from Dräger [59] offer important instruments and standards for communication between the proprietary hospital networks and other third-party applications. Along with the common 3-tier application structure, they produce other

important topics for discussion in this chapter. Finally, the chapter introduces the concept of multithreading being crucial for the biomedical signal acquisition and recording procedure.

### 2.2.1. Technology

At the very beginning of the project, there was a deeper discussion about the technologies and tools being relevant for a realization of the real-time analysis tool, VREACT. Generally, two programming languages came into question: (1) MATLAB and (2) C++ language [60]. Both languages are high-level programming languages, however there are some essential differences worthy to mention. MATLAB scripts are parsed directly on an interpreter, whereas C++ code must be translated into the machine code before it's executed. This seems to be a plus for MATLAB missing the time-consuming compiling process. However, larger projects require storing code in MATLAB proprietary .m file format limiting the *speed of the interpreter*. This is a common issue if using several loops in systems performing demanding real-time operations [60].

The next problem of MATLAB programs is their *portability*. Since MATLAB code must be translated into the C code before the deployment on target machines can start, C/C++ code is highly portable on different platforms bypassing additional installation of interpreters and another runtime environment. C also gained its popularity due to the availability of built-in compilers in processors of various brands [60]. Although the proprietary programming environment of MATLAB is highly optimized for *rapid prototyping* in its pertinent programming language, many external C++ frameworks were improved (especially in the last decades) and offer convenient programming in C++. One of the most popular C++ frameworks is definitely QT application framework, which is a cross-platform and (partially) free environment offering many profitable features extending the standard libraries of C++ [54].

VREACT was primarily developed for the patient monitoring devices used in the AKH. Since all those biomedical appliances have been distributed from *Dräger* company [21], which in turn introduced its own proprietary network and tools, the license for *Infinity® Gateway Developer's Tools* must be obtained first. These tools actually grant an access to a variety of medical information. By using their application programming interfaces (APIs), other external C and C++ applications access information hosted on the proprietary network. The above-mentioned criteria favor the selection of C++ programming language, i.e. the whole project was developed in this programming language.

The technology stack used during the development of VREACT was extended by:

- *QT framework*. QT framework represents a mature framework proved by the industry over 20 years. It facilitates the development of cross-platform desktop applications and is often a choice for engineers developing in the “*in-vehicle, medical and industrial automation*” context [61].
- *VirtualBox*. Virtual box is the “*cross-platform virtualization platform*” [62]. It is usually installed as a classical desktop application supporting various AMD or Intel-based computers irrespective of their running operating system. What is more, it is able to run multiple operating systems different to the host operating system, i.e. the system which physically runs on the computer. Since the Infinity® Gateway Server as a part of the *Infinity® Network* operates on the *Windows Server 2008 R2* platform [59], VirtualBox is a perfect choice to easily install and switch to the platform during the development and deployment of the software [62].
- *GitLab*. GitLab is a universal project management platform including a powerful version control system *Git*. It facilitates the process of code sharing by managing the existing Git repositories, supporting testers and project leaders to track issues and automates the code integration and delivery. For programmers, it provides a simple access to the codebase any time at any place. In our project, this feature was used to keep our code secure and ready for immediate download, review or supervision [63].

### 2.2.2. Infinity® Network

*Infinity® Gateway Suite* provides a simple access to biomedical information in hospitals and facilitates the exchange of physiological data being recorded during the patient monitoring periods. It includes the above mentioned licensed *Infinity® Gateway Developer's Tools* delivering libraries necessary for access to the proprietary *Infinity® networks* [59]. *Infinity® network* contains several patient monitoring and signal recording devices being grouped to a particular care unit or another organizational entity [64]. In contrast, the standard hospital network grants an access to administrative data of patients, diagnostic reports, electronic medical records, disparate billing systems etc. The particular data is usually retrieved from one of many software solutions being integrated into the single clinical system [65].

Figure 18 illustrates how those existing networks interoperate. The hospital network provides an access to the administrative data as well as the medical record of patients. Once any

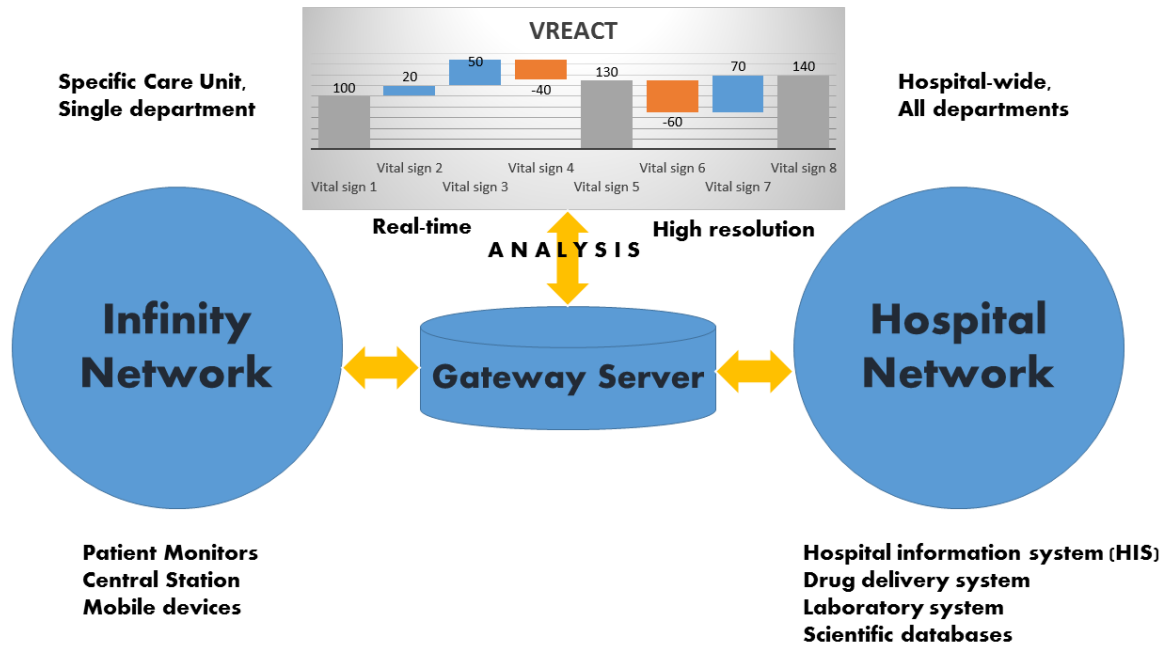


Figure 18. The relationship between the Dräger's Infinity® Network and existing hospital network. The Infinity® Gateway server enables the exchange of administrative and/or physiological data between both networks. VREACT performs a real-time assessment, high resolution parameter analysis based on patient monitors active on Infinity® Network. The Gateway server enables the connection to any device of the Infinity® Network.

information about the patient must be sent to an *Infinity® Network* device, the host sending the data is authenticated by *Infinity® Gateway Server*, which acts as an intermediary component routing the data to the target device. The same occurs in the opposite way, if any biomedical data recorded by an *Infinity®* device need to be transferred to an external device or application.

In case of VREACT, the ideal location for the software deployment is any runtime environment inside the *Infinity® Network*. This simple strategy resulted in the configuration of the *Windows Server 2008 R2* platform, where the Gateway Server was installed. Next, the application was moved and executed on the same platform as the Gateway Server, i.e. *Windows Server 2008 R2*. The whole computer environment was configured for the existing *Infinity® Network* in anesthesiology department, where the patient monitors were located.

After the above-mentioned configuration of the server and application environment, the application was finally able to communicate with the *Infinity® Gateway Server* and was able to obtain the physiological data from monitors in network. The whole communication procedure between the server and the application was actually realized by specifying special API queries by using the central C/C++ library from Dräger, *WvAPI*. This library found in the *Infinity® Gateway Developer's Tools* is the next topic of our discussion.

### 2.2.3. Dynamic linking

*Dynamic libraries* represent files, which contain special functions utilized by other custom applications. Normally, those files are loaded later, during the application runtime [66]. They provide necessary implementations distributed to other third-party developers, which may incorporate the files into their evolved software application. The names used for the libraries slightly differ from platform to platform. E.g. on Windows, which is also the underlying platform of VREACT, the notion *Dynamic Link Library* (DLL) with the same file extension *.dll* is common. Microsoft defines a DLL as “a module that contains functions and data that can be used by another module (application or DLL)” [67].

Figure 19 depicts the process of the DLL file distribution. Although a DLL is linked to another module at the compile time of the application, it is not loaded until any of its functions is required. That means, the DLL being linked to the compiled application must be available during the whole application run. If the DLL depends on other libraries, those libraries must be loaded at runtime as well [66]. After successful linking, the library cannot be exchanged by another version of library or be moved somewhere else, since the DLL search path became invalid. DLL sharing is easy and efficient. It enables integration of the same functionality into more than one application without foregoing compilation. Finally, DLLs bring advantages against static libraries in the memory management, caching and file replaceability [66].

*WinView Application Programming Interface* (WvAPI) is the most important DLL used for a standardized access of biomedical data hosted on the *Infinity® Network*. It is a C library distributed within the *Infinity® Gateway Developer's Tools* [59]. It provides access to the physiological patient data such as “demographics, parameters, alarms, and waveforms”. Furthermore, it offers other useful functions like *WvStart*, *WvStop*, *WvConnect* and *WvDisconnect* needed for proper connection to the Gateway server and other monitoring devices. The DLL functions are implemented as prototypes in the *WvAPI C++ header* file, which must merely be incorporated into the existing QT application template.

After studying the technology and the *Infinity® Network* architecture, the embedding of DLLs to applications built within the QT framework was considered. In general, three separate files having the same name (WvAPI) and different file extensions (.h, .lib, .dll) were provided. Hence, two different ways of linking were possible [68]:

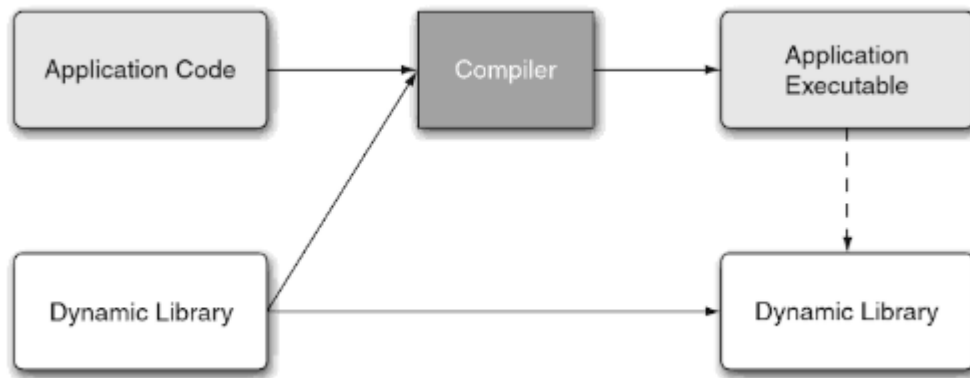


Figure 19. Dynamic libraries as special functional modules are linked to the application as soon as it is compiled. They have to be distributed with the application, since their content is not explicitly embedded into the application executable. They are loaded later at application runtime [66].

- *Load-time dynamic linking* requires the import library (.lib) containing information about how to link a DLL to its target application,
- *Run-time dynamic linking* omits the import library by using other helper functions.

Those helpers fetch the addresses of the DLL functions, which are loaded at runtime.

*Infinity Gateway Developer's Tools* deliver both a WvAPI.dll and a WvAPI.lib file. Therefore, the *load-time* dynamic linking approach was followed.

Next, the Dräger libraries had to be linked to the QT Widget Application template. The whole procedure was achieved by following the QT documentation about the shared libraries [69].

Thus, the following steps were necessary:

1. The header file of the dynamic-link library (WvAPI.h) was moved into the source code directory. Then, it was included to other header files (of the *WvAPIConnection* and *LoopManager* class).
2. The library file of the dynamic-link library (WvAPI.lib) was moved into the source code directory *Libraries*.
3. The dynamically linked library (WvAPI.dll) was moved into the build directory of the project, since the application's executable file (which is created there) must find and load the corresponding DLL at runtime.
4. The other dynamically linked library files (WvSvc.dll, IGAcMsg.dll, Wvresources\_\*.dll) acting as dependencies of the WvAPI.dll were moved into the same directory, where the executable and WvAPI.dll files are located.
5. The directories containing the header (.h) and the library (.lib) files were registered in the project file. The *QT project file* (.pro, .pri) keeps the build-specific



information of the project including the resources and other platform-specific instructions [70]. The project file of the application part (not the viewer part) can be viewed in appendix below.

6. After inclusion of the files' directory, the header (.h) and library (.lib) files were registered in sections "HEADERS" and "LIBS" of the project file, respectively.
7. The remaining library files (WvAPI.lib, WvScv.lib, IGAcMsg.lib) were then registered in the project file using the compiler specific variable "LIBS" as well. Special attention was paid to the prefix "-l" before names of linked files.
8. In the last step, the application was rebuilt and run within the linked libraries.

At the very end of this procedure, the fundamental application structure containing all necessary tools was given. Now, the application utilized the framework specific code as well as the functions specified in the integrated library. Before any of the application specific features was developed, the application as a whole was split into three (logically different) tiers. The next section describes this strategical decision and its advantages.

#### **2.2.4. Application tier structure**

The section 2.1.5 introduced the design patterns as powerful means, which produce effective solutions for smaller architectural problems and affect certain classes in a local context [54]. However, the implementation of a larger software requires a definition of the global policies, which are followed during the whole development process. Presence of the large software applications and complex "domain logic" largely contributed to the formation of application layers (or tiers) [43], [71]. Since VREACT will provide transparent graphical interface for the biomedical signal visualization and simultaneously perform demanding operations on a large physiological data, the *three-tier architecture* would imply the best architectural solution for the tier decomposition [71].

*Three-tier architecture* represents the most common layer decomposition approach splitting the system into the *presentation*, *domain (business)*, *data access* constituents [54], [71]. Each constituent (called as *tier*) contains classes accomplishing their specific jobs such as visual presentation of data, data storage or data processing procedure. Our application takes advantage of this architectural model in order to increase transparency of intermodular communication, enable code exchangeability and finally reduce internal coupling [54], [71].

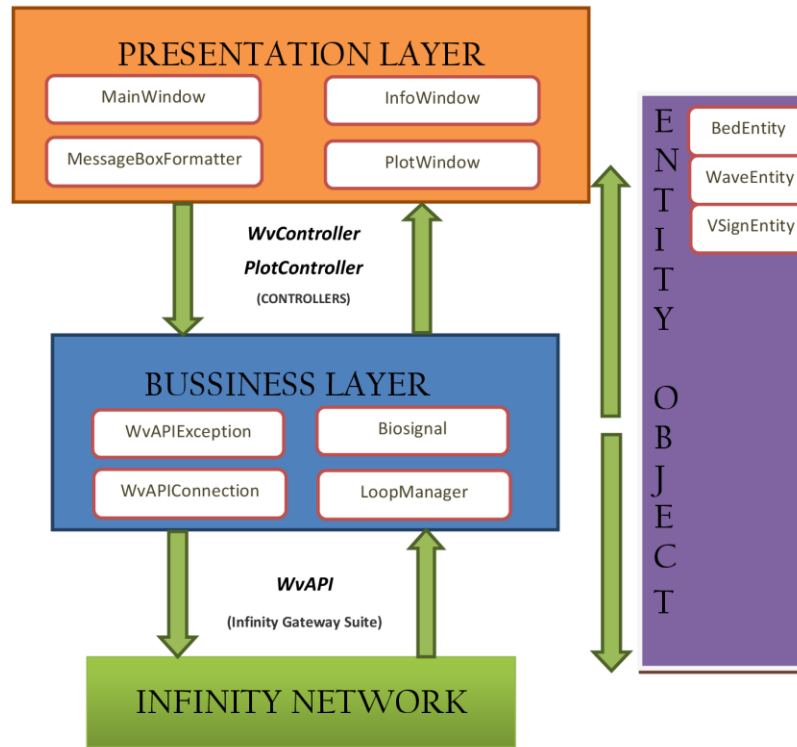


Figure 20. The 3-tier architecture of the software after the implementation. The presentation layer focuses on the visual presentation of beds, notifications and visualization of biosignals. The business layer houses the buffer of biosignals, further processing algorithms as well as the sampling loop for biosignals. The Infinity network has a role of the data access layer managing the resources critical for the high-resolution, real-time analysis.

Figure 20 depicts the result of the decomposition getting three separate tiers. Controllers were implemented according to the *MVC pattern* as described in the section 2.1.5. They define essential interfaces specifying the contract between the presentation and business layer. Analogously, the *WvAPI* library implements a bridge to the data access layer (Infinity® Network layer) containing the resources of interest (e.g. Gateway Server, patient monitors etc.). The *entity objects* (BedEntity, WaveEntity, VSignEntity) transferred across the layers doesn't necessarily belong to any of the layers, i.e. they primarily encapsulate the monitor specific information and transfer biological samples and other metainformation from the Infinity® Network up to the presentation layer. The remaining classes were assigned to one of the tiers as defined in [71].

The *presentation layer* concerns with the visualization of data on the screen [54]. Therefore, it is made up of classes, which represent the graphical part of the application such as *MainWindow*, *InfoWindow*, *PlotWindow* and other message boxes. Those classes specify other widgets and their properties related to the structure, size and layout within the containing window. E.g. The plotting rectangle can be set for presentation of biomedical information on

the screen. Furthermore, the QT Framework offers several graphical widgets like *QTreeWidgetItem*, *QStatusBar*, *QMenuBar*, *QPushButton*, being useful in UI programming. Finally, there are further classes being necessary for communication between the widgets or widgets and other controller classes. In this case, QT proprietary concepts such as *Signals and Slots mechanism* were employed [56].

The *domain (business) layer* houses the processing algorithms operating on the domain specific data [71]. The layer also maintains the communication between the outermost (*presentation*) and the innermost (*data access*) using strong-typed interfaces. The classes here store information about the connection status of various monitoring devices (*WvAPIConnection*), fetch, process and store the biomedical samples returned from monitors (*LoopManager*) or temporarily fill buffers with new biological samples (*Biosignal*). Moreover, there are further classes derived from *Biosignal* calculating physiological parameters such as HR, HRV or evaluating the experimental data using Fourier analysis. Finally, the standard exception concept (*WvAPIException* along with *MessageBoxFormatter*) of error handling was employed in order to present the failure specific notifications coming from the *Infinity® Gateway Server*.

The *data access layer* covers the topics of data encapsulation and persistence [54]. It contains classes providing *external* (API-specific) and *internal* operations used for the access to the *Infinity® Network* resources. The *external* part of the data access layer is represented by the libraries from *Infinity® Gateway Developer's Tools*, especially by the *WvAPI* library, which can be seen as the entrance door to the *Infinity® Gateway Server*. In contrast, the *internal* part includes other dependencies and functions being managed by the server itself. Next, the server maintains connections to the monitoring devices and cares for secure transfers of biomedical data from the monitors to the third-party applications. The necessity of a proper server configuration is therefore crucial, otherwise no monitors and biosignals could be obtained.

### 2.2.5. Multithreading

*Multi-threaded* applications parallelize the execution of instructions by using several threads [72]. Usually, an application requires at least one single thread, i.e. “a single stream of instructions” [72] to be started. A *single-threaded* application executes their instructions (tasks) by using a strict sequence order. That means, the task 2 can only be executed, if the task 1 finished. This way of program flow is illustrated in Figure 21a.

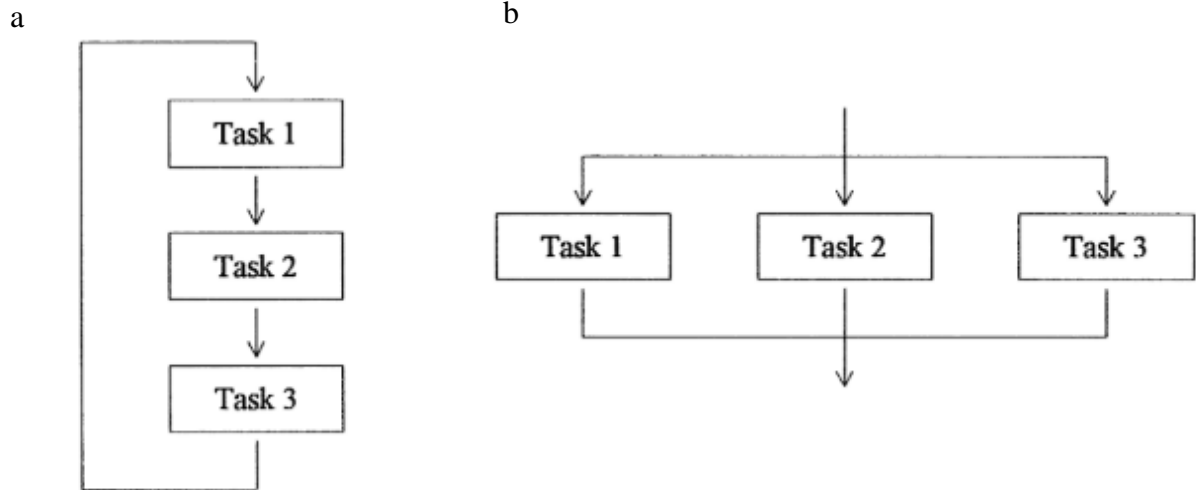


Figure 21. Possible program flows in (a) single-threaded and (b) multi-threaded applications [72]. The tasks in single-threaded applications can only be finished in a sequence order, whereas the tasks in the multi-threaded applications can be executed in parallel.

*Multi-threading* mechanism introduces more threads running simultaneously. Basically, there is a “primary (main) thread”, which is created by the underlying operating system during the application start. Then, the program starts additional (“secondary”) threads at certain places in code defined by programmers. Depending on the application runtime environment, the tasks are executed in parallel on systems with multiple integrated processors [73]. Figure 21b demonstrates the completion of three tasks running in different threads. The task1, task2 and task3 are normally assigned to different processor cores running the instructions at the same time [72]. If fewer processor cores were available, their built-in schedulers would regulate the reserved processor times of threads, i.e. each single thread would get a while to operate on its particular processor core. Therefore, a certain parallelism degree is still given [73].

QT framework offers its own multi-threading concept and allows outsourcing the CPU-intensive operations to separate threads [74]. VREACT should keep its UI components responsive to the user. Thus, the demanding operations such as sampling and real-time plotting of biosignals may negatively influence the UI responsiveness and let the application freeze. Therefore, those operations need to be run in their own threads.

*QThread* class represents the central component of multithreading in QT [74]. Usually, a custom class extends the *QThread* class, i.e. the programmers may reimplement the logic in the thread’s specific function *run()*. Conclusively, the framework executes the code implemented in the function by using a new thread. Another approach for running new threads

```

// Create the timer and the central buffer for the sampling loop
p_timer = new QTimer(this);
p_buffer = new WvData();

// Create a new thread instance and move the 'worker' instance to it
p_loopThread = new QThread();
this->moveToThread(p_loopThread);

// Connect timer to the sampling loop function
connect(p_timer, SIGNAL(timeout()), this, SLOT(slot_getSamples()));

// Notify the 'controller' to perform the necessary clean up.
connect(this, SIGNAL(stopLoop(std::shared_ptr<BedEntity>)),
p_controller, SLOT(slot_stopBedLoop(std::shared_ptr<BedEntity>)));

// Initialize the sampling loop, if the thread is started
connect(p_loopThread, SIGNAL(started()), this, SLOT(slot_startThread()));

// Terminate the sampling loop, if the thread is finished
connect(p_loopThread, SIGNAL(finished()), this, SLOT(slot_stopThread()));

```

Figure 22. The implementation of the *LoopManager* constructor. This class represents the signal sampling loop for one single bed entity. The timer determines the sampling frequency, the buffer stores the references to the stored biosignals. Since each *LoopManager* instance performs demanding operations in respect to the central sampling loop, the instances are moved to their own threads managing their own resources.

is known as “*worker model*” [74]. A worker represents a special object with the `QObject` macro keeping the code for the new thread. Figure 22 presents an example of a worker class, the *LoopManager*. The code snippet stems from the constructor of this class. *LoopManager* implements the logic of sample retrieval and buffering, i.e. the samples are pulled from the already connected patient monitors and temporarily stored in the versatile memory (RAM). It is also responsible for sample export to a .csv file as thoroughly described in section 2.3.2. Although the instances of *LoopManager* are created in the primary thread, they are immediately moved to their own secondary threads. The pointer to the *QThread* instances are defined as their instance variables. After starting the secondary thread, the function *startThread()* is executed, initializing the central sampling component for the particular patient monitor. Analogously, the sampling loop is stopped after stopping the corresponding thread of the instance. If any problem with the connection occurs, the *WvController* (a controller instance according to the MVC pattern as discussed in section 2.1.5) is requested to terminate the sampling loop by using *slot\_stopBedLoop()* function. Conclusively, *WvController* keeps the control over all sampling loops running their separate threads.

## 2.3. User Interface

The previous section examined the most important technical concepts employed by the application programming. It introduced the entire technology stack and discussed special

criteria needed for the selection of appropriate tools. Furthermore, it thoroughly described the application runtime environment with a special regard to the *hospital* and the proprietary *Infinity® network*. Other topics like dynamic linking, software tier architecture and multithreading in QT have been discussed due to its importance during the software development phase.

However, this section presents the results from the design (section 2.1) and development (section 2.2) phase and goes through the most important functional aspects of the application in regard of the real-time analysis procedure. It also gives a glimpse of the dynamic process of UI implementation and link the postulated functional requirements of clinicians with the individual UI functions of the software. Finally, the graphical UI components being critical for the real-time assessment system will be explained in the more detail.

### **2.3.1. Requirements Engineering**

*Requirements Engineering* is a special field of the software engineering with the goal to explore, derive, document and verify requirements of a specific domain shaping the system of interest [75], [76]. It provides useful concepts about the requirements' specification and management, what has a great impact on the subsequent design and development phases in software projects. There exist several definitions of requirements [75], however Sommerville and Sawyer describe requirements as “*a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system*” [77]. I.e. the requirements basically describe the product's features and specify the behavior of the developed system.

Table 1 presents the most important requirements on VREACT, which were intensively discussed among the whole stakeholder community, especially during the early development phases. The most of the requirements have been prioritized, i.e. the content of the highest priority was treated as the first. Simultaneously, all the requirements were formulated according to the recommendations and standards specified by ISO, IEC and IEEE [76], [78]. The underlying IEEE policy was issued under its unique reference number *ISO/IEC/IEEE 29148:2011(E)*© [76]. Table 1 focuses on the functional requirements explicitly discussed in conjunction with the application as a whole. Next sections address the problem of requirements in regard of the UI and present the results of their implementation.

<b>Requirement ID</b>	<b>Description</b>	<b>Priority</b>
# 1.	After starting the system, the system shall list all available monitors.	HIGH
# 1.1	The system should list all available monitors within five seconds.	LOW
# 1.1.1	Each monitor should be characterized by its label name and the corresponding care unit, to which it belongs.	LOW
# 1.2	If any monitors are available, the system should sort the available monitors according to their care unit.	MEDIUM
# 1.3	If any monitors are available, the system should support sorting of the monitors within a single care unit.	LOW
# 1.4	The system shall provide a simple selection system for the available monitors.	HIGH
# 1.4.1	If any monitors are selected, the system shall connect to the monitors using one single function / user click.	HIGH
# 1.4.1.1	The system shall provide a color-based feedback according to success of connection within five seconds. The successfully connected monitors should be labelled by a “green” color. The other (connected) monitors should be marked with a “red” color.	HIGH
# 1.4.1.2	For all successfully connected monitors, the system shall enable a function for opening a biosignal analysis slot.	MEDIUM
# 1.4.1.3	For each successfully connected monitor, the biosignal analysis slot function should appear within the list (within the same list item).	LOW
# 1.4.1.4	For each successfully connected monitor, the system shall implement a sampling loop (i.e. a loop, which pull all the biomedical samples for a given monitor repetitive in a specified time interval).	HIGH
# 1.4.1.5	For each successfully connected monitor, the system shall temporarily store the fetched samples for at least five minutes.	HIGH
# 1.4.1.6	For all successfully connected monitors, a function for storing the biomedical samples (to a CSV file) shall be enabled.	HIGH
# 1.4.2	If any monitors are deselected, the system shall disconnect from the monitors via one single function / click.	MEDIUM
# 1.5	The system shall provide function to refresh the list of available monitors every time.	MEDIUM

# 1.6	The system shall export the biomedical samples for all connected monitors and biosignals being accessible from that monitors.	HIGH
# 1.6.1	The system should inform the user about the start of the CSV export.	LOW
# 1.6.2	The system should inform the user about the end of the CSV export.	LOW
# 1.7	The system shall facilitate the selection of paths for the CSV file export.	MEDIUM
# 1.8	The system should present the license information about the licensed development tools.	LOW
# 1.9.1	When the application main window is closed, the system should close all the other opened windows as well.	LOW
# 1.9.2	When the application main window is closed, the system should automatically disconnect from the already connected monitors.	LOW
# 1.10	If system start or the connection to the Gateway Server fails, the system prints an error message and closes after user's confirmation.	LOW

Table 1. The system engineering requirements on VREACT formulated according to the ISO/IEC and IEEE international standards [76] and IEEE recommendations [78] in software engineering. These requirements were primarily formulated for the bed management and recorder part of the software, which are also the central topic of this work.

### 2.3.2. UI Requirements

*User Interface* (UI) involves the interactive part of the software or environment, which communicates with the user by triggering reactions on the operated system. It is a part of the software, that “*people can see, hear, touch, talk to, or otherwise understand or direct*” [79]. It is composed of widgets receiving “inputs” from the user and presenting “outputs” after successful processing. The best UIs follow the needs of users as closely as possible and perfectly adhere to the rules of the corresponding business domain. Therefore, “*the best interface is one that it not noticed*” [79].

Before presenting the VREACT's UI structure, some advantages of the graphical UIs will be discussed. There is a lot of research published in respect to the design of appropriate graphical user interfaces [79]. E.g. It is well-known, that the graphical symbols are preceived more precisely than a text of any length. Therefore, any message being shown to the user be presented with an graphical symbol such as “i” meaning information, “?” emphasising question



or exclamation depicting a warning about the particular action [79]. Those messages have therefore been integrated to the VREACT software and presented by means of the dialog boxes.

Quick and easy learning process of working with the software was a special requirement of clinicians. According to [79], a proper “visual or spatial representation of information” may encourage the learning and memorizing effects of users with regard to the operational aspects. I.e. the clinicians looking at a well-structured graphical component (e.g. list of available beds) can easily learn and remember the operations performed in association with it. This implies a more accurate and faster completing of tasks realized by the underlying software.

Graphical presentation of biosignals was the next important clinical requirement being intensively discussed and translated into the graphical UI. Since the biosignals are normally represented by a long series of numbers (as a function of time), a spacial representation of biosignals could support the “visual thinking” of each individual [79]. Therefore, various curves, lines and histograms have been implemented. The visual presentation may also intensify the context of the information. By this, the relationship between the graphics and the presented information can be easier to notice.

As introduced in section 1.3, the real-time assessment systems such as VREACT should not only support the clinicians in their daily clinical routine, but should also improve the patients` safety and reduce the mortality numbers. This is probably the most important requirement on the system. A higher degree of concretization results in fewer errors and lower numbers of error situations in general [79]. Therefore, transparent description of widgets and the possibility to repeat (or to revert) the executed functions was also given.

Fast feedback represents the next valuable requirement on the UI. The UI should be self-explained and should inform users about the important milestones and progresses [79]. Immediate feedback delivery represents the one of the most important UI characteristics. VREACT implements a powerful messaging system including various message boxes, message colors and status bars in accordance with this requirement.

Despite various recipes for a proper UI programming, there are some disadvantages of the graphical UIs worthy to mention [79]. Typically, the design of graphical UIs is much more

complex than the design of the text-based screens. The enormous number of different controls and their properties does not necessarily simplify the process of UI evolution. What is more, the process of familiarizing with the software lasts typically longer, since the diverse controls, icons and labels are often misunderstood. Finally, various companies deploy similar software solutions using its own terms and notations in the UI. This considerably complicates the learning process of users and leads to further confusions in the operability of the system [79].

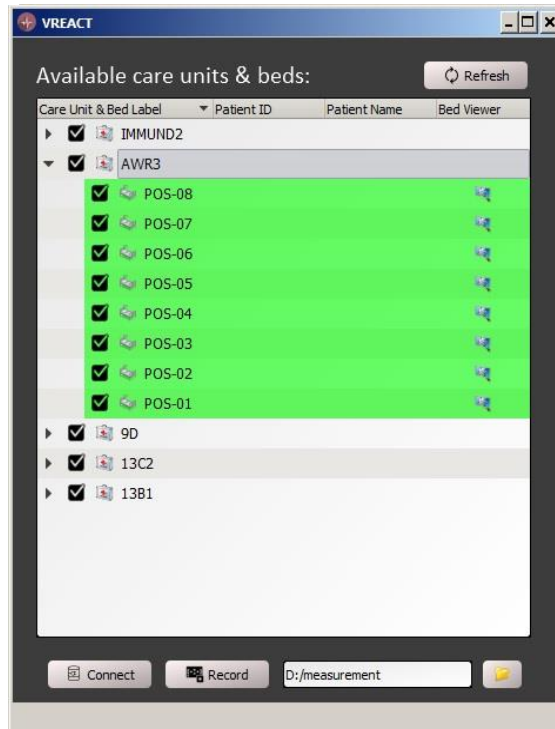
### 2.3.3. Connection Management

Before starting the real-time analysis and recording of biomedical signals, a desired patient monitoring device must be accessed and connected. Figure 23a-b presents the simple main application window after starting and connection to the corresponding devices. First, the application asks the server for a list of currently available monitors and presents it to the user. Subsequently, the checkboxes are enabled for each single item of the list, which stands for one single monitor. Finally, the monitors are sorted and grouped according to the corresponding care units, i.e. after the application start, merely the care units are presented to the user. The user is able to expand the care unit item and select individual monitors. The second option is to check the care unit item. In this case, all monitors of the care units are automatically checked.

The list of available monitors can be easily refreshed, whenever a new monitoring device is added, removed or simply turned off. In addition, the new monitors can be plugged in and configured for the current *Infinity® network*. The implementation of the *refresh* functionality ensures the consistency of the already running recordings, i.e. the (already started) analysis doesn't necessarily have to be interrupted, if new devices appear. Only the list of available monitors and their corresponding care units will be updated.

After device selection, the monitors can be *connected* and *disconnected* at once by making a single click on the *connect* button. This feature significantly simplified the whole process of resource management and assisted users by their selection of the appropriate devices for analysis. Furthermore, the user gets immediate visual feedback about the status of the connected devices. If the devices are online, the list items are marked with the *green* background color (Figure 23a), however if they are offline or unplugged from the operating network, the device's background color will be *red* (Figure 23b). By this, the user immediately sees the devices ready for analysis and the other ones being problematic.

a



b

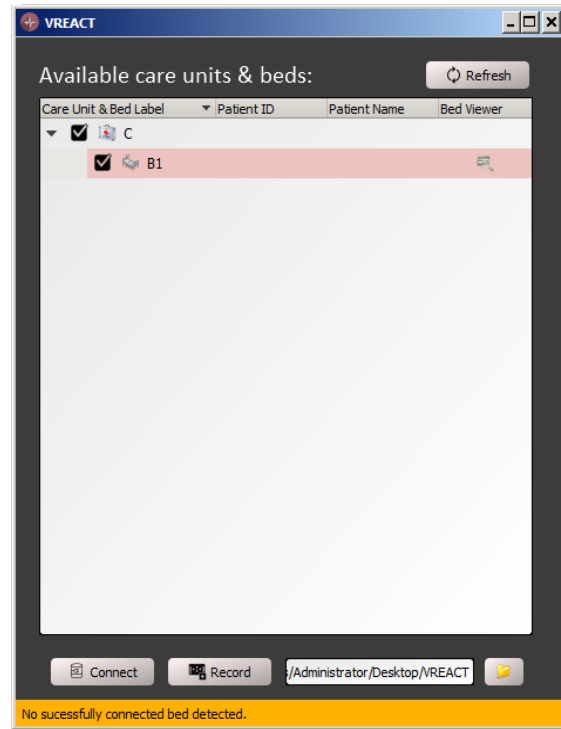


Figure 23. The results after implementation of the connection management module presenting the available monitors. The user has an option to select and connect to the individual monitoring devices. (a) The available patient monitors have successfully been connected, i.e. the biomedical signal analysis may be started, (b) the available patient monitor is currently offline (but registered on the server).

A next important feature of the software is the data persistence. The user must have an option to *export* the recorded data into some other flexible format. The CSV files are commonly applied for data storage, export or exchange with other “spreadsheet” systems [80]. Thus, the physiological data stored within the CSV files can be processed by other powerful tools like Excel or MATLAB. Generally, the data is only exported from the successfully connected monitors providing some kind of physiological information, i.e. the monitors and its connected sensors must be online and record any physiological information. Thus, the user simply connects to the desired beds and press the *record* button. After some time of data recording, he can stop the running export algorithm and open the CSV file of interest.

In conjunction with the data persistence feature, the user is allowed to select any preferred folder for the data export (at the bottom of the Figure 23a, b). Furthermore, this feature facilitates the usage of any external medium connected to the running machine. The selected folder path is retained after each program’s reboot. Therefore, the user needs to choose his desired folder once only, i.e. at the very beginning of the program usage.

Each single item of the list, which represents a monitoring device (not the care unit), contains an icon representing a monitor with the magnifier glass. This icon is clicked to open the biosignal analysis window for the given monitor after the its successful connection (Figure 23a). If the connection to the monitor wasn't established, the icon is changed and disabled (Figure 23b). The more information about the visual biosignal analysis is given in section 2.3.5.

Table 1 contains the functional requirements shaping the resulting UI in the Figure 23. However, the requirements #1.4.1.2 - 6 and #1.6, 1.6.1-2 specify the functionality of the *Patient recorder* and *Patient viewer* components discussed in the following two sections. Other requirements reflect functions of the main application window and are associated with the functionality of the *Connection management* module.

#### **2.3.4. Patient Recorder**

*Patient Recorder* implements the transfer and recording of the biomedical samples from monitors accessed by the *Connection management* module. The samples are transferred from the monitors via the Infinity® Gateway Server to the *data access layer* of the software (see section 2.2.4). From there, the samples are stored to the DTO objects and transported either to the central biosignal buffer or to the UI (for visual presentation) as discussed in one of the subsections 2.1.5, DTO pattern. Figure 20 illustrates the process of the biomedical sample transfer throughout different application tiers using special entity objects having a role of the DTOs.

Aside from adding the transported samples to the central biosignal buffer, the data can also be exported to the CSV files as mentioned in section 2.3.3. Before the samples are actually transported, a short interaction with the user is required. First, the user must connect to the desired monitors and press the “*Record*” button placed in the main application window (Figure 23a). The button's name and icon change to the “*Stop*” state and the recorder is started (Figure 24a). If the user decides to stop the recorder, he must press the same button again. By this, the data export function immediately stops and the button's name and icon change to the original “*Record*” state as presented in the Figure 24b.

In background, the application retrieves current information about the DTOs created by the sampling loop and generates the required folder and file structure (see Figure 25) on the local

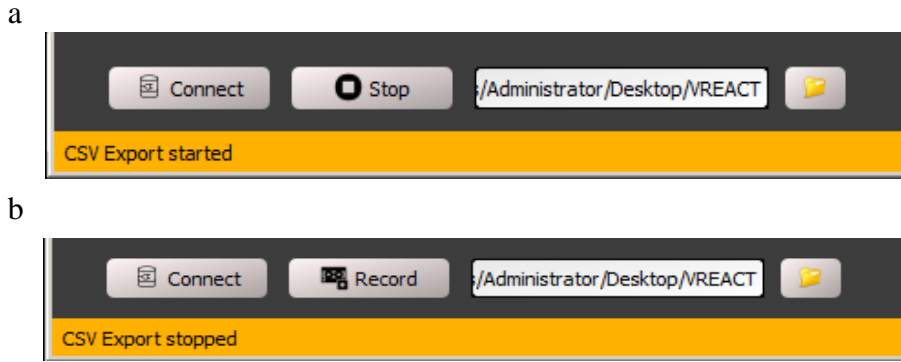


Figure 24. The application informs about the progress of the recorder module persisting all the biosignals available on the corresponding monitors. (a) After the button “Record” is clicked, its name is changed to the “Stop” button and the CSV export starts. (b) After pushing the button “Stop”, the samples aren’t stored anymore and the button changes to the “Record” button again.

machine. The DTOs are regularly filled up with new data in dependence of the sampling frequency. The content of the objects is then written to CSV files for each single biosignal (defined as a CSV file column) and timestamp (represented as one single CSV file row). The samples are stored in a row-wise order until the user stops the recorder or the file becomes larger than 100MB of size. In this case, the CSV file is immediately closed and a new file is created. The recorder proceeds the export procedure by storing the samples to the second file.

Beyond intensive discussions of the entire sample storage procedure providing one or more CSV files, there have been concerns about the management of the stored information. In general, there are dozens of monitoring devices registered within a care unit representing a potential source for the recorder. Furthermore, several physicians (or researchers) export the clinical data several times per day. This may result in tens of recordings a day and thousands of samples being exported to the corresponding files. To find those files after each data analysis, a simple data location system was proposed.

Figure 25 depicts the specific autogenerated folder structure after starting the data export. Normally, a user specifies the path to the folder, to which the samples are stored (Figure 24a-b). After pressing the “Record” button, the application autogenerated several folders and files for each single monitoring device being recorded in the predefined folder. The structure is composed as follows:

- a folder specifying the *current recording*,
- one or more folders specifying the *care units* of the corresponding monitors,
- one or more folders specifying the *bed positions* or patient monitors,
- one or more CSV files storing the *actual names and samples* of biosignals.

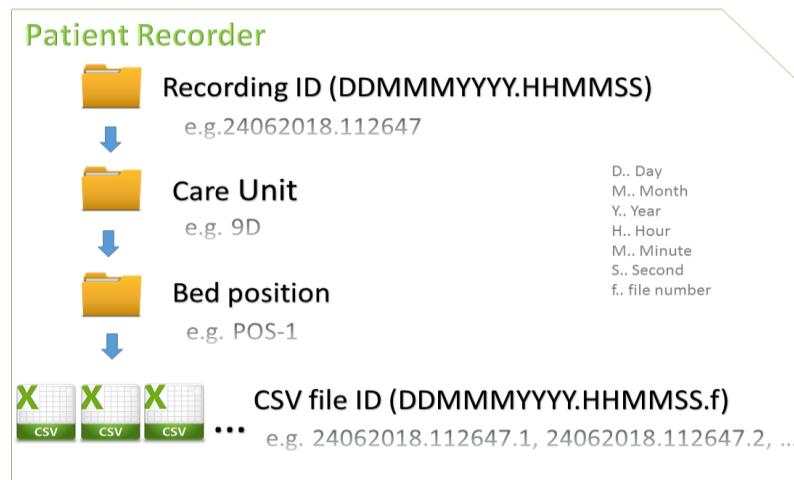


Figure 25. Patient Recorder creates a specific folder and file structure for each single monitoring device. After starting the recorder, a folder defining the current recording run with a special name is created. Within this folder, a folder per care unit and a folder per bed position are created. Finally, several CSV files (in dependence of the size and the number of the recorded biosignals) might be created and stored.

The aforementioned folders and files are generated in the order presented in Figure 25. First, the recording folder is generated and named by using a specific date-time format. Since any recording might access monitors of different hospital care units, folders with particular care unit names are created as the next. In addition, different bed positions (corresponding to the different patient monitors) obtain different folders carrying the name of the corresponding position. At the very end of this structure, one or more CSV files are generated to store biosignal samples from the different monitoring units.

### 2.3.5. Patient Viewer

*Patient Viewer* enables visual presentation of the biomedical parameters on data sampled from the connected patient monitors. Hence, it allows the user to track not only the standard, but also the novel physiological parameters calculated by the software in real time. On the one hand, it promotes the selection of biosignals and the appropriate plot-visualization style. On the other hand, it enables a remote long-term tracking of biosignals and other physiological parameters (e.g. HR, HRV, etc.), which are determined by using more accurate algorithmic solutions. It provides several visualization techniques such as bar charts, dot and scatter plots as well as live curves and numeric presentation.

Figure 26 presents the main window of the *viewer* being opened for a single patient. The patient's name, bed position and care unit are presented on the top left side of the window. All available physiological parameters of the patient are listed in the tree-like structure. Since a couple of standard biosignals (such as *ECG*, *SpO2*) provide data suitable for calculation of



Figure 26. Patient Viewer presents the biosignals of a single patient. On the left side of the screen, the overview about the available biomedical signals is given. Underneath this window, available chart types for the selected biosignal are presented. The signals can be then easily drag & dropped to the twelve independent slots prepared for visual analysis (right side).

additional (derived) biosignals (e.g. HR, SDNN, RMSSD, etc.), the tree-like structure for data selection was suitable to implement. After any of the biosignals is selected, the possible charts for the plotting appear below the list. The user selects the symbol representing the chart of interest, drags and drops it to one of the twelve slots prepared for the real-time analysis. The respective slot starts the visual presentation of samples in real time. Subsequently, the user may visualize other biosignals using different charts transferred to their own slots at the same time. Some visualizations require more space to ensure a consistent analysis. Therefore, some charts use two slots simultaneously.

Constant plotting requires a constant data delivery. Therefore, the exported physiological samples are temporarily stored in the *central biosignal buffer* for at least five minutes. It is an important implementation step to ensure continuance of the plotted curves and provide a short history of the recordings. In addition, the algorithms extracting the derived biosignal parameters require data accumulated over a longer period of time (e.g. for calculation of HRV parameters). Thus, the novel parameters (see section 1.4.2) can be determined more accurately. Since the buffer collects samples of the derived biomedical parameters too, those parameters are easily exported to the CSV files as they were the standard parameters of the monitor. The implementation of the central biosignal buffer reflects the req. # 1.4.1.5 from Table 1.

### 3. Clinical experiment

The previous chapters discuss the current difficulties in the perioperative monitoring and propose the easy-to-use real-time assessment tool, VREACT. Our software allows tracking of the novel vital parameters to enhance the real-time biosignal analysis and to record the physiological data at higher resolution rates. Other state-of-the-art systems similar to VREACT were introduced and thoroughly discussed. However, many of them are close-sourced and don't meet the most critical requirements on the continuous patient monitoring, especially within the anesthesia and surgical units. Next, the most essential standard and novel physiological parameters affecting the patient's state evaluation by clinicians are presented. Finally, the requirements on the developed real-time analysis system were obtained thanks to the clinical specialists from the Department of Anesthesia, Critical Care and Pain Medicine in AKH. The tool was subsequently developed, tested and applied within the above-mentioned department to support the real-time biosignal analysis and the running clinical research.

This chapter introduces a short experiment conducted at the Department of Anesthesia, Critical Care and Pain Medicine in AKH. The aim of the experiment was to assess the tool and its capability to record biological samples for the successive analysis. The provided measurement consists of recordings carried out on 20 anonymized ICU patients. The experiment demonstrates the utility of the analytical software assisting by a clinical research. Beside this, it examines the association of the recorded biomedical signals by inspecting different patients across 3 different care units. Finally, the biosignals recorded by VREACT are assessed and graphically visualized.

In the course of this master thesis project, the scientific paper "*Real-time assessment of high resolution vital signs recording for calculation of perioperative clinical parameters*" [3] was submitted. The paper substantiated the necessity of the real-time assessment and high-resolution recording tools in the perioperative setting. Generally, the paper discussed the current issues of the biosignal monitoring in the perioperative periods and introduced the tool VREACT along with its integrated modules (e.g. HR, HRV, BP, SpO<sub>2</sub>). It also gave a short glimpse on the architecture and integration of the software into the existing hospital environment. Finally, it presents a few examples of studies working with the physiological data recorded by VREACT [3]. The next sections show a short experiment utilizing VREACT.



### 3.1. Introduction

A short clinical experiment was organized during the evaluation of VREACT. An approximately 2-hour measurement over the currently hospitalized patients from 3 different ICUs was taken. The analyzed ICUs belonged to the clinical departments of Anesthesia and Surgery in AKH, especially the Department of Anesthesia, Critical Care and Pain Medicine and the Department of Surgery. In total, 20 patient recordings have been exported by VREACT during the real-time biomedical analysis. The patients were fully anonymized, i.e. no personally identifiable information was neither stored nor used for the subsequent analysis. The tool was deployed and run on the *Windows Server 2008 R2* platform being preinstalled on the notebook *Toshiba Satellite Pro* with Intel Core i3-6006U processor, 4GB RAM and 1000GB HDD. A detailed information about the hardware requirements and installation procedure of the Windows Server is provided in [81].

### 3.2. Methods

Before starting the analysis of the biomedical recording, the most important features of the VREACT's recorder were analyzed. The feature of the concurrent export of biosignals for several patients was verified at first. Concurrent export of biosignals includes measurements of the biomedical data over multiple patients and ICUs and is normally triggered by a one single button click. The consistency and validity of the multiparametric biosignal recording was the next important point of analysis. A biomedical recording should contain all exportable biosignals and store a continuous data vector for every recorded parameter during the whole measurement time. The recorded data must be easily attributed to the corresponding biosignal label to avoid confusions. Although the modern digital storage media are constantly enhanced and furnished with more and more memory power, the size of the recorded data wasn't known before the experiment. Hence, the tool's memory utilization was examined in a more detail and the several records of multiple patients were done. Since VREACT is able to extract novel physiological parameters such as HRV measures or cumulative time of desaturation, those parameters were (similar to the standard physiological parameters) written to the CSV export files and were subsequently used within the short experiment. The analytical and graphical analysis was realized in Python, where the exported physiological parameters were processed and finally visualized by means of the powerful Python tools presented in the next sections.

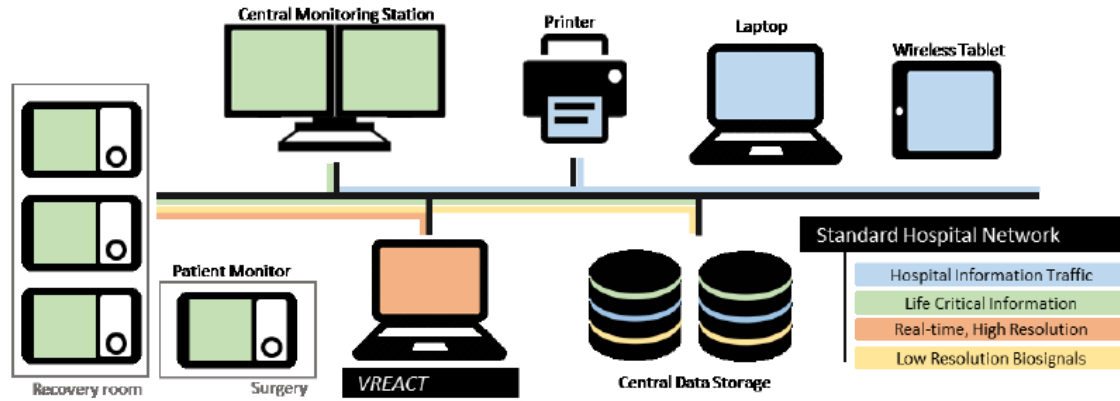


Figure 27. The final hospital infrastructure used for the biosignal recording. Ca. 2-hour measurement on 20 patients was achieved by running VREACT on the Toshiba laptop (orange). The vital signs were primarily recorded on (Dräger Infinity® Delta) patient monitors (green) available within the surgical and recovery rooms. VREACT also supports postprocessing of the primarily recorded data and calculates novel physiological parameters (e.g. HRV). This figure was drafted and published in the submitted conference paper [3].

To assess the above-mentioned features of the VREACT's recorder, the tool was deployed to the *Toshiba Satellite Pro* laptop, where the Infinity® Gateway Server has already been launched (see Figure 27 adapted from [82]). First, the patient monitors (out of the Dräger Infinity® Delta series) out of the 3 different surgical and anesthesia ICUs were registered on the Infinity® Gateway Server to access the recorded biomedical information. Then, a connection to the *Dräger Infinity® Network* was established by using a standard network switch as described in the Figure 27. The recording tool was started and the registered monitoring units were successfully accessed. Finally, a recording of the current patient population was started. The measurement took approximately 2 hours and 10 minutes. During the measurement, the software recording was regularly checked for the continuity of the exported samples. After that, the recorder was stopped and the exported data underwent the automated biosignal analysis in *Python*.

The programming language *Python* belongs to the scripting languages, since its source code needs to be merely parsed and run by the powerful Python interpreter without any additional steps as compilation and linking [83]. It is a very flexible programming language suitable for both, fast undemanding prototyping of a variety of systems as well as evolving of complex solutions for the most challenging tasks. Furthermore, it provides useful libraries in the *scientific data analysis* area [84]. Because of these unique features, the whole scientific data analysis was primarily realized in Python and its helpful scientific libraries *NumPy* and *Pandas* [84]. All the necessary data evaluation algorithms were written in an easy-to-use, open-source web editor, Jupyter Notebook [85].

	<i>Care Unit</i>	<i>Patient Monitor</i>	<i>Sum</i>		<i>Care Unit</i>	<i>Patient Monitor</i>	<i>Sum</i>
<b>1</b>	13B1	BETT-01	31	<b>11</b>	13C2	BETT-06	26
<b>2</b>	13B1	BETT-02	26	<b>12</b>	13C2	BETT-07	26
<b>3</b>	13B1	BETT-04	28	<b>13</b>	13C2	BETT-08	26
<b>4</b>	13B1	BETT-05	31	<b>14</b>	9D	POS-01	22
<b>5</b>	13B1	BETT-08	29	<b>15</b>	9D	POS-03	35
<b>6</b>	13C2	BETT-01	26	<b>16</b>	9D	POS-04	31
<b>7</b>	13C2	BETT-02	26	<b>17</b>	9D	POS-06	31
<b>8</b>	13C2	BETT-03	29	<b>18</b>	9D	POS-07	18
<b>9</b>	13C2	BETT-04	26	<b>19</b>	9D	POS-09	14
<b>10</b>	13C2	BETT-05	26	<b>20</b>	9D	POS-10	22

Table 2. Overview about the care units and their related patient monitoring devices accessed during the experiment. Instead of patient names, the monitor labels (e.g. Bett-01) were retrieved for the biosignal analysis. The table presents all 20 monitoring positions across the 3 different care units. The “sum” columns reveal the number of biosignals being recorded at the respective monitoring position.

In the course of the biomedical recording (as described in Figure 27), 3 folders with the ICU names containing 20 patient folders and 174 CSV files were created according to Figure 25. In total, 529 biosignal slots were active during the measurement of the 20 patient monitoring units. Some biosignals were merely exported from the patient monitors, whereas the other (novel) biosignals were continually calculated from standard parameters by the post-processing HRV and SpO2 modules. However, it has to be mentioned, that not every biosignal slot was active during the whole measurement time (i.e. 2 hour and 10 minutes), since VREACT detects and adds new connected slots in real time. In such a case, a completely new CSV file with an updated header is created without a recording interrupt. The same situation happens, if some biosignal slots are unexpectedly removed.

Table 2 presents the particular monitoring units and the corresponding biosignal numbers recorded on each of the positions. The equal number of biosignals on diverse positions doesn't necessarily mean, that the same types of biosignals were measured. While the care unit *13C2* shows rather uniform distribution of the recorded biosignals, the distributions of the both other care units *13B1* and *9D* are more heterogenic. The positions *POS-03* and *POS-09* from *9D* exhibit the highest and the lowest number of recorded biosignals, respectively. This summary table gives an overview about the contents of the measurement, which underwent a more detailed scientific analysis in Python. The next section presents the results of the analysis.

	Per Hour	Per Day	Per Month	Per Year
Care Unit	258.55 MB	6.06 GB	42.42 GB	2.16 TB
Patient	38.78 MB	930.77 MB	6.36 GB	331.77 GB
Biosignal	1.42 MB	34.13 MB	238.93 MB	12.17 GB

Table 3. Utilized disk space on the Toshiba laptop after finishing the biomedical recording. The data size of the ca. 2-hour recording was mapped onto the averaged disk space occupied by a care unit, a patient (or bed position) and a biosignal. The required averaged disk space was specified for different time periods, per hour, day, month and year.

### 3.3. Results

This section describes the results of the short clinical experiment based on the VREACT measurement analysis in Python. The main goal of the short experiment was to assess the features of the VREACT *recorder* (see section 3.1). A proper data visualization is crucial for its subsequent interpretation [86], [87]. To achieve a consistent analysis of the physiological data (recordings) and other categorical data (based on the different care units and monitoring positions), several visualization graphs and charts such as *box plots*, *2D line plots*, *histograms* and *bar charts* were taken into account. Concerning the type of the visualized data (nominal, numerical, textual, etc. [87]), the particular chart technique was carefully selected.

#### Memory utilization

Starting with the Table 3, the *disk space utilization* of the VREACT recorder is presented. First, the disk space of the recording for each bed position as well as each care unit was retrieved. First, the applied *Python algorithm* checked the individual CSV file sizes and summed them up to a final value per patient and care unit. Then, the final sum (per patient) was divided by the number of biosignals of the particular position to get the average disk space taken by recording of a one single parameter. Finally, the values were scaled for different time periods as shown in Table 3.

According to the Table 3, the longer the recording and the more biosignal slots present, the more is the size of the whole recording on disk. Less than 1GB constantly recorded data per patient a day (including the *novel* as well as the *standard* parameters) is basically a very promising outcome. In the US and European countries lies the mean length of stay (LOS) on the ICUs by ca. 3-4 days [88], [89], which is in comparison to the median of LOS on the

hospital in general (i.e. 11-14 days [89]), relatively short. Therefore, an average patient would require the storage capacity of less than 3GB high-resolution physiological data per admission.

## Concurrent biosignal analysis

Parallel recording of multiple biosignals is often achieved by utilization of multiparametric sensors [90]. In general, the (target) *Infinity® Delta* monitors allow an integration of external sensors like *MultiMed*, *HemoMed pod*, etc. [25]. The physiological data is often transferred via several ports, simultaneously. The arrived data is stored in the monitor's cache for short, i.e. VREACT continuously reads and assigns the heterogeneous samples to the correct biosignal slot. Due to this fact, VREACT matches the requirements on a multiparametric recorder, since it allows to store samples of different biosignals and patients at the same time.

Figure 28 gives an overview about the biosignals and their frequencies in the patient recordings, i.e. the occurrences of the biosignals over all care units and patients in analysis. The special *Infinity® Delta* parameters with labels P1 M (Pressure1-Mean), P1 D (Pressure1-Diastole), P1 S (Pressure1-Systole) and GP1 (General Pressure 1) were present by only one single patient. In contrast, the top 10 biosignals (on the bar chart) such as Cumulative time of hypoxemia, Heart rate (HR), Oxygen saturation (SpO2), etc. are available in all patient records. Except of the Cumulative time of hypoxemia, there are other novel physiological parameters calculated by VREACT post-processing modules such as SDNN, SD1, SD2, RMSSD etc. Those parameters are derived from the ECG II data slot, so, the name of the exported parameter contains the name of the slot as well. Since those derived parameters are automatically determined if any ECG recording is running, the frequency of the underlying ECG II slot and its derived parameters must perfectly match. This is also the case, since the frequency of the discussed biosignals is exactly 19 (see Figure 28). Measurements of the arterial BP (ART M, ART S, ART D, ART) on patients in the three different ICUs is also quite common, since 18 out of the 20 available patients had the arterial BP records. Finally, there are a couple of exported biosignals with acronyms, which aren't clear at the first glance. Therefore, a further research on this area is necessary. All available acronyms and its (available) meanings have been taken from the official guide, *Infinity® Gateway Suite* [59].

Basically, there is a certain heterogeneity in biosignal types and their occurrences in the recording across the different ICUs (see appendix, figures Figure 32, Figure 33, Figure 34).

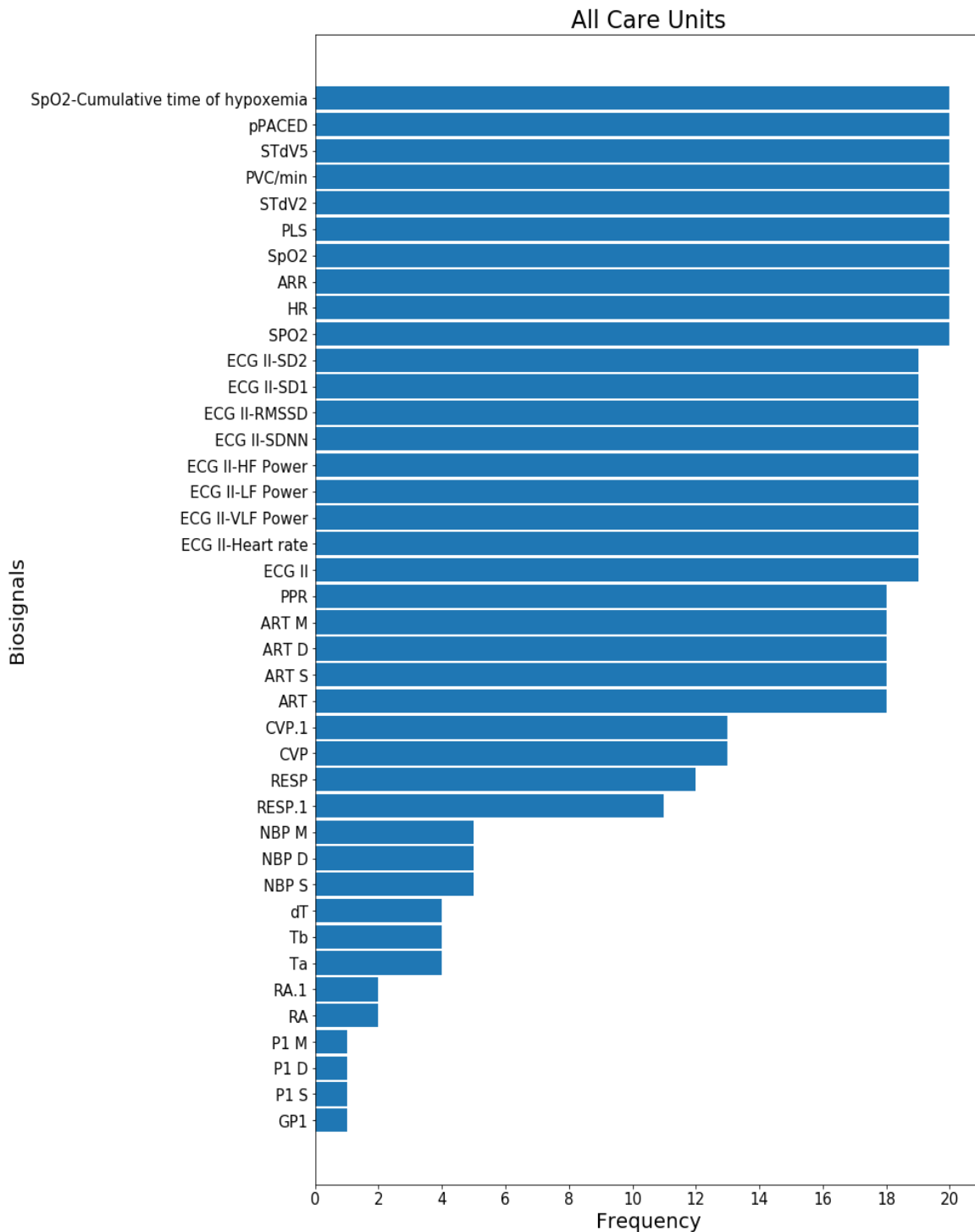


Figure 28. A biosignal set comparison across all analyzed care units. The parameters P1 M, P1 D, P1 S and GP1 are recorded by only one patient, whereas the other top 10 parameters such as cumulative time of desaturation (hypoxemia), Heart Rate (HR), oxygen saturation (SpO2), etc. are observed by all patients in analysis.

Whereas the most biosignals measured on all patients are present in 13B1 and 13C2, the care unit 9D carries the largest biosignal set among the all three ICUs. 26 out of overall 40 biosignals were exported from all available inpatients of the 13B1 and 13C2 (Figures Figure 32, Figure

33). Only 11 out of the 40 biosignals were obtained or calculated within the care unit 9D (Figure 34). However, the top 10 measured biosignals were:

- %PACED, STdV5, PCV/min, STdV2, PLS, SpO2, ARR, HR, SPO2 belonging to the standard vital signs,
- Cumulative time of hypoxemia *derived* from the standard vital signs.

On 13C2, there are only three different occurrences detected among patients (Figure 33). The most biosignals (26) are coming from the whole patient population. By one patient, three additional body temperature curves (Ta, Tb, dT) were detected. The rest of biosignals (11) wasn't exported, i.e. no measurements of those signals were done on the ICU.

## Consistency of biomedical records

Figure 26 presents a snapshot from a multiparametric patient recording monitored by VREACT, consisted of both the *standard* and *novel* vital parameters. The *consistency* and *validity* of the measurement was done on both parameter types. The *standard* vital parameters visualized on the VREACT's interface can be easily compared to the vitals displayed on the *Infinity® Delta* monitors. Since the biosignals are visualized and exported from the monitor to the application at the (nearly) same time, both screens can be verified simultaneously. Second, the *novel* vital parameters, which haven't been integrated into the *Infinity® Delta* series yet, must be compared with similar measurements achieved by other tools (e.g. *WvRecorder* presented in section 1.3.2). However, this procedure is still complex due to the synchronization of measurements, because both measurements must be started and stopped at the same time.

To make a brief validation of exported biosignals, the contents of the CSV export files were processed in Python, i.e. the whole analytical process was automatized. The results of the analysis were charted as continuous 2D line plots, each biosignal was plotted in a separate slot. Finally, the patient's data was replaced and the data of another patient was processed in the same way. Figure 29 presents three plots visualized in Python. The complete results of processing for the particular patient are demonstrated in the appendix (Figure 35).

Figure 29 presents an example of three different biosignals stored by VREACT's recorder. Each of the presented biosignals has its unique properties in regard to the origin and type. E.g. *ECG II-Heart Rate* measures HR of the patient based on its ECG data recorded on the *second* ECG slot. The biosignal is calculated by VREACT and its post-processing HRV module on

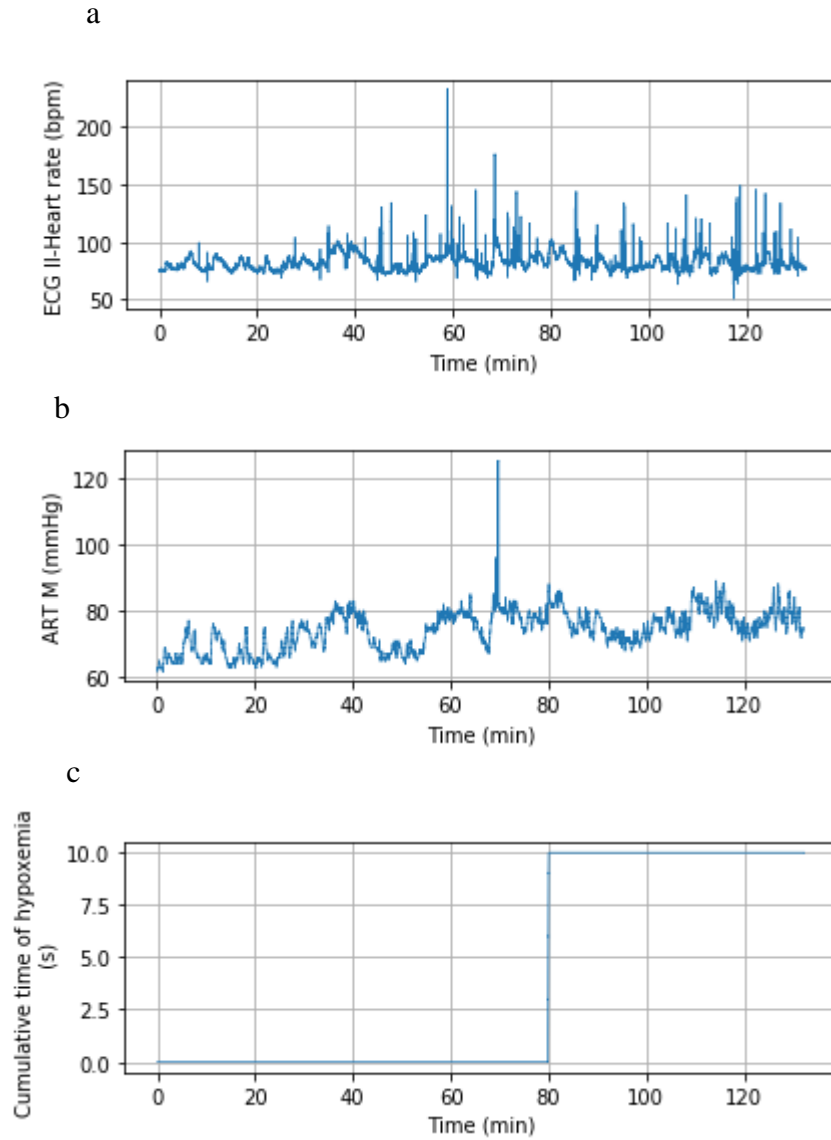


Figure 29. Examples of the biosignal recorded by VREACT. (a) The HR parameter extracted from the underlying ECG recording, (b) the mean arterial pressure (ART M) exported from monitor, directly and (c) the cumulative time of hypoxemia extracted from the underlying SpO2 (oxygen saturation measure) signal.

the raw ECG data retrieved from the *Infinity® Delta* monitor. HR is similar to the *pulse* and can be defined as the “*number of heartbeats per minute*” [38]. Figure 29a presents the diagram charting the HR against the time of the measurement. Except of the most visible artefact occurring shortly before the 60<sup>th</sup> minute, the HR seems to have less variations in the first half of the measurement. In the second half, there are more obvious variations,  $\pm 50$  of the HR. Without knowing the patient medical record and his exact diagnosis, it can’t be definitely decided, what the deviations mean and why they occur. Thus, a detailed knowledge about the subjects of analysis would be very helpful in the future work.



The biosignal labeled as *ART M* (see Figure 29b), which actually represents the MAP according to Dräger [59], belongs to the group of *standard* parameters. Its samples are exported one after the another from the *Infinity® Delta* monitor, i.e. there are no modifications of the biosignal by the recording software. MAP is a powerful indicator about the blood pressure in arteries [31]. The particular MAP in Figure 29b is slightly varying between 75 and 80. Around the 70<sup>th</sup> minute of the recording, there is a clear peak of the measurement by ~125 representing (probably) an artefact in the measurement. This interpretation is the most likely, since the peak lasts only a couple of seconds. In the last half of the measurement, the MAP reaches higher values, up to 90. This could imply an event changing the patient condition, e.g. by application of drugs, some post-surgical intervention or similar.

*Cumulative time of hypoxemia* is a completely *novel* parameter being intensively studied as e.g. in [39]. It represents the overall duration of desaturation, i.e. “*the raw minute-by-minute values below various hypoxemic thresholds*” [39]. Its calculation is treated by the VREACT’s post-processing *SpO2* module, i.e. the continuous raw *SpO2* signal is received and regularly checked for drops below the defined threshold value of 90%. Figure 29c summarizes the hypoxemia times of a patient during the measurement. Though the patient has been reaching the *SpO2* values over the hypoxemia threshold for almost 80 minutes, the value suddenly dropped below the threshold for exactly 10 seconds. After this time, the patient didn’t experienced any desaturation event up to the end of the measurement time.

## Example for a biosignal analysis

Suppose, there would be a question about the distribution of the parameters among the patients hospitalized on the specific care units or across all care units present in the measurement. In order to determine the characteristics of the statistical distributions, box plots offer a powerful tool of visualization for “*quantities associated with a set of items*” [91]. The box typically starts with the first quartile (Q1) and ends with the third quartile (Q3) of the distribution, i.e. it covers its “*interquartile range (IQR)*” [91]. The line traversing the box represents the median of the mathematical series. The whiskers are defined as lines, which protrude from the boxes, starting at the quartiles (Q1, Q3) and ending (typically) by the values within “ $Q1 - 1.5 \times IQR$ ” and “ $Q3 + 1.5 \times IQR$ ”, respectively. The values being outside of the box are represented by other marks, such as points or crosses [91]. Figure 30 presents the box plots of the different ICUs and all ICUs for eight representative biosignals.

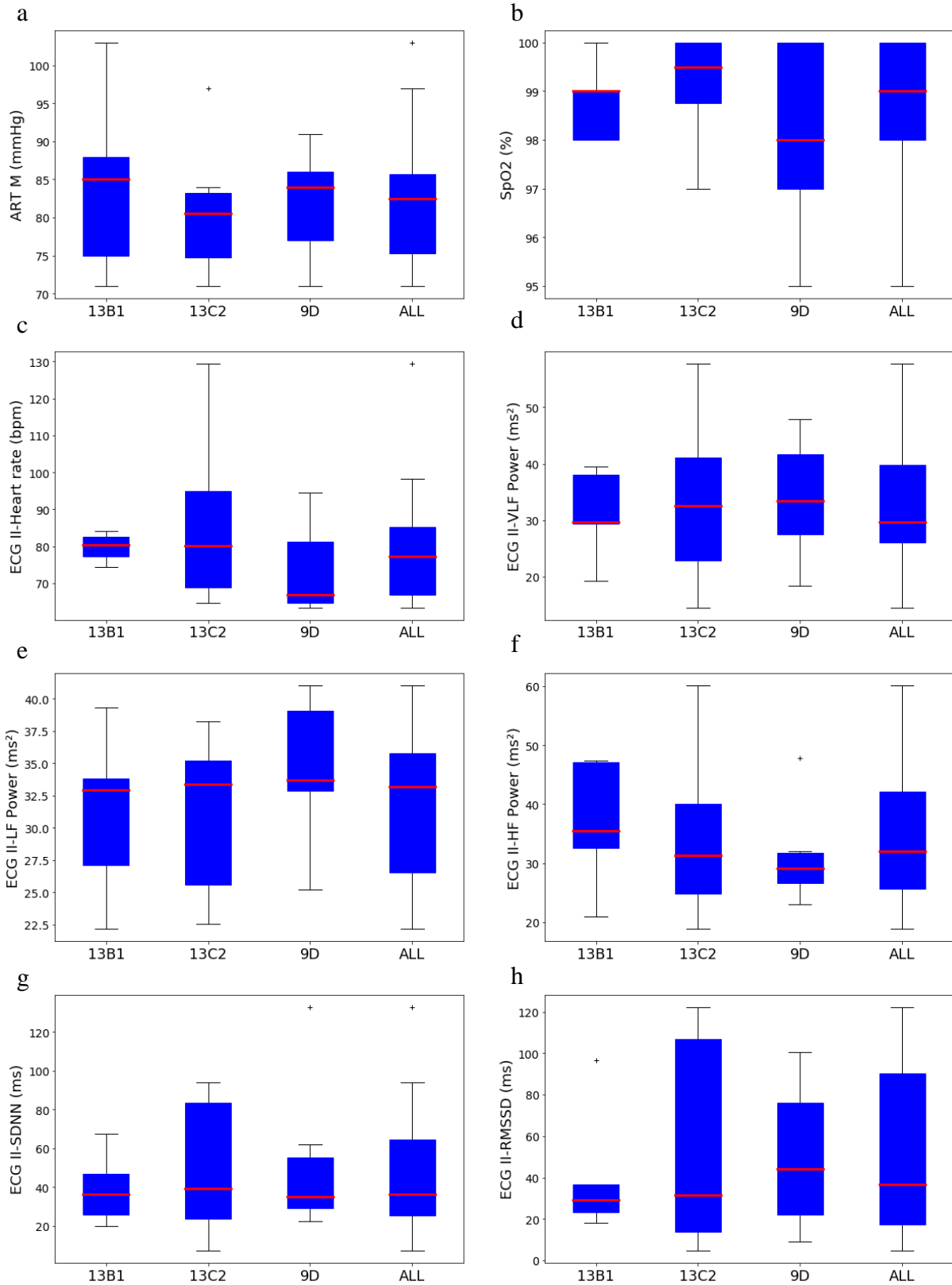


Figure 30. Statistical distribution of 8 different biosignals recorded over all patients in the 3 ICUs, 13B1, 13C2, 9D. The standard parameters exported from the Infinity® Delta monitors (a) ART M, (b) SpO2, (c) Heart Rate as well as novel parameters (d) VLF Power, (e) LF Power, (f) HF Power, (g) SDNN, (h) RMSSD are characterized upon its distribution of the different ICUs. In general, all plots contain rather varying distributions, though the parameters (d) LF Power and (g) SDNN show (almost) uniform medians across all ICUs. The most equally distributed series are observed in (b) SpO2 and (c) Heart Rate on the group “ALL”. 7 boxplots exhibit an outlier apart from the whiskers. All the distributions are composed of biosignal medians, i.e. each representative value per biosignal was obtained as median of the whole biosignal data series. This value was subsequently used as the representative of that particular patient.

Figure 30 presents the results of the statistical biosignal analysis considering the individual ICUs as well as all ICUs at once. The analysis was automatized in Python. First of all, the contents of the CSV export files have been transferred to the Python code and the long biosignal sample vectors for each patient were read. Then, a representative value for each biosignal vector was calculated. The values representing biosignals of patients have been sorted according to the care unit, to which the patient belonged. Finally, four different box plots on four different distributions (13B1, 13C2, 9D, ALL) were plotted for each biosignal of interest.

Three *standard* biosignals were chosen during the analysis as described in Figure 30a-c. All of them depend on a continuous stochastic variable  $X$  keeping mathematical distributions of around thousands of values pro signal. Per patient and biosignal, only one single representative value is accepted, since the overall distribution across the ICUs is needed. Therefore, the median value as the representative of  $X$  was obtained from each biosignal data series.

Considering the box plots of the *ART M*, which corresponds to the above-mentioned MAP (Figure 30a), the box variations are slightly fluctuating on the individual ICUs. The IQRs of the boxes are markedly of similar lengths and the highest MAP value is on the unit 13B1, whereas the lowest value is observed on the 9D. Similar to the MAP, the SpO<sub>2</sub> value exhibits only slight variations across different ICUs (Figure 30b). Looking at the overall distribution (“ALL”) of the chart, the median achieves 99% (with  $\pm 1$  distance to quartiles Q1 and Q3). There is also at least one patient with the averaged SpO<sub>2</sub> 95%. Finally, the analysis of the HR parameter (Figure 30c) brought interesting results. Considering the median values on 13B1 and 13C2; the medians are identical, the both distributions significantly differ, though. The IQR of the 13C2 is almost four times higher than the IQR of the 13B1. The highest value among all distributions is reached by 130bpm on the unit 13C2. Although the distributions of HR across the individual ICUs vary, the overall distribution (see “ALL”) is of a nearly uniform structure.

Five *novel* HRV parameters were chosen for the distribution analysis to demonstrate the utilization of the VREACT’s *recorder* (see Figure 30d-h). The hereby mentioned HRV parameters are thoroughly discussed in the section 1.4.2 and its subsection “Heart rate variability”. Let’s consider the most interesting distributions out the five visualized biosignals, SDNN and RMSSD. In case of SDNN, the medians across different ICU distributions are almost identical. That means, that the overall mean value over all patients (the box plot of “ALL”) must be around those three medians, what is actually correct if looking at the box plot

of the variable “ALL” (Figure 30g). The IQR of the SDNN box plots is varying from ICU to ICU. The largest variation is observed on the 13C2, whereas the narrowest SDNN variation is visible on 13B1. In contrast, 9D contains one outlier reaching the value around 130ms. RMSSD is the next HRV parameter having an unusual statistical distribution. Similar to SDNN, the RMSSD variation of 13B1 is extremely small probably due to the smallest number of patients (5) as shown in the appendix (Figure 32-Figure 34). The highest median value was found in 9D unit, its IQR is double as small as in case of the 13C2, though. The IQR of the 13C2 reaches approximately a value of 100ms, which is rather high in comparison to 70ms reached on the distribution across all ICUs. Out of the 5 patients hospitalized on 13B1, there is one outlier lying slightly below the 100ms. This value is much higher than its remaining four representatives lying between 20 – 35ms.

### 3.4. Discussion

The objective of this experiment was to evaluate the properties of the VREACT’s *recorder* as well as to demonstrate the utilization of the tool in the perioperative setting. VREACT is a powerful tool allowing real-time assessment and high-resolution recording of the biomedical data. It consists of three different components, the *Connection management*, *Patient viewer* and *Patient recorder*, which all perform different operations such as connection to monitors, visualization and recording of the biomedical information. The short clinical experiment primarily focused on the evaluation of the *Patient recorder*, since it belongs to the main objectives of this thesis. Despite the technical point of analysis, VREACT was utilized for recording of the ca. 2-hour biosignal measurement at the Department of Anesthesia, Critical Care and Pain Medicine in AKH. The resulting biomedical data was obtained from 20 ICU patients involved in the analysis. All data was anonymized and deleted after the experiment.

The tool was compiled and deployed to the notebook *Toshiba Satellite Pro* with Intel Core i3-6006U processor, 4GB RAM and 1000GB HDD. These hardware requirements were sufficient for installation on the *Windows Server 2008 R2* [81]. Although the laptop was satisfactory for this particular measurement, it should be noted, that a future database server environment might enhance the existing hospital setup in order to facilitate longer multiparametric measurements by multiple patients at once. Furthermore, the application of the server within the setup might increase its whole performance as well as automate the backup procedure of the recorded

biomedical data [24]. During the experiment, however, the recorded data was only persisted on the laptop's local storage and transferred to another machine for the successive data analysis.

The biomedical measurement consisted of 20 patient folders and 174 CSV files, which corresponds to the (approximately) 19 files per patient recording. Despite several rules implementing the new CSV file generation, the number of files (per patient) containing the exported physiological vectors is still too high in comparison to the current patient number. The recorded data should be immediately ready for analysis without a necessity of further processing or advanced file manipulation. On the other hand, the 529 parallel recorded biosignals stored as continuous alphanumeric vectors represent a promising result in respect to the multiparametric biosignal analysis. Several of the recorded biosignal slots consists of the novel physiological parameters, which extend the patient's state analysis and contribute to a better decision-making process on the ICU rooms. Since the examined biomedical measurement took the 2 hours and 10 minutes, the presented results are limited to this (relatively) short period of time. Thus, a more accurate evaluation of the (mostly) long-term recordings is planned for the future.

Regarding the *memory utilization* of VREACT's recorder, the 34,13 MB of data in average recorded during the 24 hours seems to be very promising outcome (Table 3). By this, a multiparametric patient recording would occupy approximately 930,77 MB of disk space per patient in average, what in turn results in the 18,41 GB disk space for 20 parallel records (a day). Usually, the researchers must perform more than 20 measurements at once and expect rather long-term biomedical records, i.e. the tool should seamlessly record samples for several days. Integration of the data compression algorithms, such as GZip [24], during the storage on local disk could be an acceptable solution for this problem. Integration of some additional logic for a proper data selection before analysis would represent a second solution. On the other hand, it should be noted, that implementation of (at least) one of the above-mentioned solutions would increase the complexity regarding the software manipulation and prolong the analysis times due to the necessity for the initial configuration.

One of the main ambitions towards the recorder implementation was the *consistency* of multiparametric records as well as the possibility to *record biosignals of multiple patients* at the same time. This feature was successfully realized and verified at AKH by using the setup described in Figure 27. At the end of the measurement, the software realized 20 continuous

recordings without any visible dropouts and distortions in records. The records were thoroughly analyzed in Python and a whole multiparametric record is presented in appendix (section *VREACT recorder example*). In terms of the software reliability, the results of the 2-hour duration test were principally positive, i.e. the software was running until stopping the measurement, manually. It neither crashed nor froze and exported both the standard and novel parameters to their full extend. Despite this great success, it must be noted, that some additional long-term measurements across more than 3 ICUs must be done yet, to verify the longstanding operation of the recording software.

The utilization of the *recorder* unit has more or less no limitation in the perioperative setting. The researcher may select and choose several patients from different care units starting and stop his recording any time. In the previous chapter, an example for the biosignal analysis based on a 20-patient recording was done. The standard and novel vital parameters and their distributions on the corresponding ICUs (and over all ICUs) were examined. According to the Figure 30, the most variations were noticed on the 13C2 unit, while the largest distribution (the broadest box plot) was observed by the derived RMSSD. Similar to the presented analysis (and its results), a more comprehensive study working on a much larger data set is also realizable (assumed that the study is allowed by EUREC [92] and the patient explicitly gives his consent about the usage of his personally identifiable information).

In conclusion, the utilization of the VREACT's recorder during the short clinical experiment was advantageous, since the *novel* physiological parameters, especially the HRV measures and the cumulative time of desaturation, were calculated and exported along with the *standard* vitals. 20 parallel records were actually present in the measurement, i.e. the recorder reliably exported the biosignal set for each of the patients in analysis. 529 biosignal vectors consisted of the continuously exported samples. The subsequent analysis in Python confirmed the consistency of the exported data. An example for the biosignal analysis demonstrated the application of the tool in the perioperative setting. Despite these powerful features of VREACT, the number of the exported files is still too high and must be decreased in the future. In addition, the user is currently forced to export all the signals during the patient's recording. Hence, a simple signal selection process (implemented on the recorder) would suppress the export of a superfluous data. Finally, additional duration tests of the recorder are necessary to validate the data in analysis. Few of them have already been passed successfully (especially the 4-day recording). However, this is an objective of another thesis authored by Fatih Kartal.

## 4. Discussion

The primary objective of this work was to design and develop a system for acquisition and visualization of both, the *standard* and *novel* physiological parameters. The work outlines the importance of the patient monitoring systems in current hospitals and deals with their limitations in the modern intensive care. In spite of the enormous progresses within the intensive care achieved during the last years, the basic culture in the biomedical data acquisition and visualization hasn't significantly changed. According to the current clinical research [3], [9], the resolution of the recorded perioperative data is fairly limited on the most commercial monitoring systems. Due to the large heterogeneity of the recorded data and massive incompatibilities between the sensing, monitoring and storage systems, the recorded perioperative data is hardly integrated to the existing EMRs. Lastly, the *novel* physiological parameters such as HRV [8] or cumulative time of hypoxemia [39] bear a huge potential to be integrated for the patient monitoring technology. This work deals with the current technological limitations within the perioperative setting and proposes a novel, real-time assessment and high-resolution recording software, VREACT.

The system was developed in cooperation with the anesthetists and surgeons from the Department of Anesthesia, Critical Care and Pain Medicine in AKH Vienna. Later, it was thoroughly tested and finally introduced into the perioperative setting of the hospital. The whole project focused on the design and evolution of the three elementary software components, namely the *Connection management*, *Patient recorder* and *Patient viewer*. The *first* component solved the problem of the parallel monitoring, i.e. the user may now select more patient monitoring devices at once without any configuration in advance. The *second* component enabled the storage of the *novel* perioperative data to a local storage medium. The *third* component was implemented as an extensible module system responsible for the biosignal visualization facilitating the analysis of the *novel* physiological parameters. Implementation of the *first* and the *second* component was the main topic of this thesis, whereas the *third* component was developed by Fatih Kartal, a colleague from the equal field of study.

The ICUs in AKH Vienna dispose of various appliances tracking the patients' state of health. Nevertheless, the monitoring devices of the *Infinity® Delta* series [24] belong to the most common tracking systems utilized within that perioperative setting. Those monitors are

primarily used for a real-time analysis of the vital parameters ensuring a continuous data acquisition on patients being bedside and on transport [25]. Due to the high acceptance of the system and availability of the *Infinity® Gateway Developer's Tools* [59] (and its special hardware API, *WvAPI*), VREACT was evolved for this particular device type. Nevertheless, the utilization of the famous architectural and design patterns (e.g. MVC Pattern) ensured a better software extendibility and adaptability. This was mainly achieved by a clear logical separation of the UI and the underlying logic [54]. Hence, the current implementation can be (theoretically) adapted to any monitoring device omitting a large rework of the graphical UI.

The *Infinity® Delta* monitors [24] continuously track and visualize the biomedical information of the ICU patients. However, the stored physiological information isn't utilized by default. To ensure a smooth data analysis, the data must be exported either to a machine hanging on the *Infinity® Network* or by using the (physical) serial ports of the devices. The first solution would mean a certain overhead and would load the corresponding network if transferring the additional data. The second solution would mean that the exporting storage medium must constantly be around the monitor (due to the serial connection cables). Moreover, the physical storage medium keeping the samples shall be easily portable to ensure the communication with other computer systems, e.g. those, primarily applied for the research purposes. Because of this, the application was deployed on the *Windows Server 2008 R2* installed on the "virtualization platform" called *Virtual Box* [62]. In spite of the initial effort, put into the installation and configuration of both platforms, the system became much more portable to systems configured at other hospital departments. By this, the exporting device didn't necessarily need to be located near the ORs, it must only access the corresponding *Infinity® Network*. This substantially simplified the process of the *biomedical data acquisition* and enabled the *remote-tracking* of the (even *novel*) physiological parameters.

In addition, the standard patient monitors such as *Infinity® Delta* don't naturally support recording of the *novel* physiological parameters such as HRV [8] or cumulative time of hypoxemia [39]. Though VREACT calculates and routinely evaluates the both respective vitals, their relevance in the clinical setting and during the decision-making process has to be proven, yet. Similarly, the recorded parameters must additionally be evaluated for the appropriate recording resolution in order to produce consistent and precise measurements. Therefore, a deeper evaluation is planned in the future.



Basically, VREACT's recorder exports samples of both, *standard* and *novel* physiological parameters to the one specific location, i.e. the clinician doesn't necessarily need to process multiple export files during analysis. The implementation of this feature was challenging because of the issues with time synchronization of the exported samples. This feature was finally enabled and works surprisingly well. Nevertheless, there is still some potential to improve the storage management of the VREACT's *recorder*. For example, the current implementation allows a new (CSV) file generation in some special cases (such as exceeding the 100MB threshold size or adding/removing of biosignals during the patient's record). A next unresolved problem was the exportability of some *novel* biosignals (such as *HRV interpolation*, *HRV Fourier Transform*) delivering samples with delay i.e. the samples attributed to the timestamps from the past. Those issues must definitely be addressed in a future work.

To verify the implemented features of the software, a short test measurement was performed. The measurement took ca. 2 hours and consisted of 20 individual patient records. Finally, the recorder could successfully generate 174 export files containing 529 biosignals. After that, the data underwent the automated biosignal analysis in Python. The processed data included records out of the three different ICUs of Anesthesia and Surgery in AKH. The results of the analysis confirmed the functionality of the *recorder* against its specification in Table 1.

The described test measurement was primarily supposed to evaluate the *consistency* of the recorded data. This was partially achieved by *visualization* of the physiological data in *Python 3.7* [83] and *Jupyter Notebook* [85]. The results present the biosignals of an unknown patient (Figure 29 and 35). 6 out of 18 recorded biosignals represent the novel physiological parameters calculated by the VREACT's *post-processing* modules. E.g. the *hypoxemia times* (Figure 29c) for this particular patient are rather minimal during the 2-hour perioperative period. In the ca. 80<sup>th</sup> minute of the measurement, the value suddenly drops below the threshold for exactly 10 seconds. After this time, the patient didn't experienced any desaturation event up to the end of the measurement time. Considering an example for a *standard* physiological measure, HR was also analyzed during the test measurement (Figure 29a). Except the numerous artifacts in the diagram, the HR of the patient slightly varies between 70 – 90 bpm during the whole measurement time. The highest variation is observed between the 40<sup>th</sup> and 80<sup>th</sup> minute of the measurement. The above-mentioned examples let me to a conclusion, that VREACT delivered *consistent* biosignal records, however, the exact evaluation of the perioperative data and its significance must be proven, yet.

In conclusion, this work addressed several issues associated with the current perioperative patient monitoring. The developed tool, VREACT, enables a *high-resolution recording* and a *real-time tracking* of multiple biosignals and automates the *biosignal acquisition* and the *remote-tracking* of the *standard* and especially the *novel* physiological parameters. VREACT's *connection manager* and *recorder*, representing the most important software components of this work, facilitate handling of multiple monitoring devices on several ICUs and ensure the high-resolution multiparametric export of biosignals, respectively. In general, the tool fulfilled the most important functional requirements of clinicians and passed the initial system tests in the clinical environment. Subsequently, the tool's *recorder* was utilized during the 2-hour test measurement, demonstrating its functionality and capability to provide a consistent *biosignal analysis* on the recorded data.

On the other hand, there are still some requirements on the software, which haven't been realized yet. First, the most clinicians desire to run the tool several days by making tens of records in parallel. This requirement has partially been addressed in the Fatih Kartal's work including the *duration* and *load testing* of the software in the perioperative setting. Second, the recorder should export the physiological samples to the (ideally) one export file. This feature would eliminate the need to handle multiple files by other processing tools like MATLAB or Python. Third, there are currently two post-processing modules extending the tool's parameter set, namely the *HRV* and *SpO2* module. In the future, further meaningful modules such as *cumulative time of hypotension* or *blood pressure variation* are planned to be integrated. In regard to the existing modules, some of the *derived* biosignals (such as *HRV interpolation* or *HRV Fourier Transform*) aren't exportable at present. This is because of the difficulty to modify entries of the already stored CSV rows. For that reason, the current way of the file manipulation must be enhanced in the future. Fourth, VREACT doesn't include any warning system, which would currently alarm the medical personnel in case of severe patients' state deteriorations. However, this feature plays an important role within the perioperative setting, because the most patients are considered as the critically ill patients. Altogether, this work points out the current problems of the perioperative monitoring and presents an easy-to-use, real-time analysis system as a solution for some of them. Finally, it should motivate other scientists and engineers to enhance the monitoring technology on the current ICUs and encourage for further clinical research on this critical area.

## UML Class Diagram in summary

The section presents the application *class* diagram of VREACT. Especially a look at the whole *connection management* and *recorder* classes is given. The diagram gives a structural view on the system. For more detail, refer to section 2.1.4

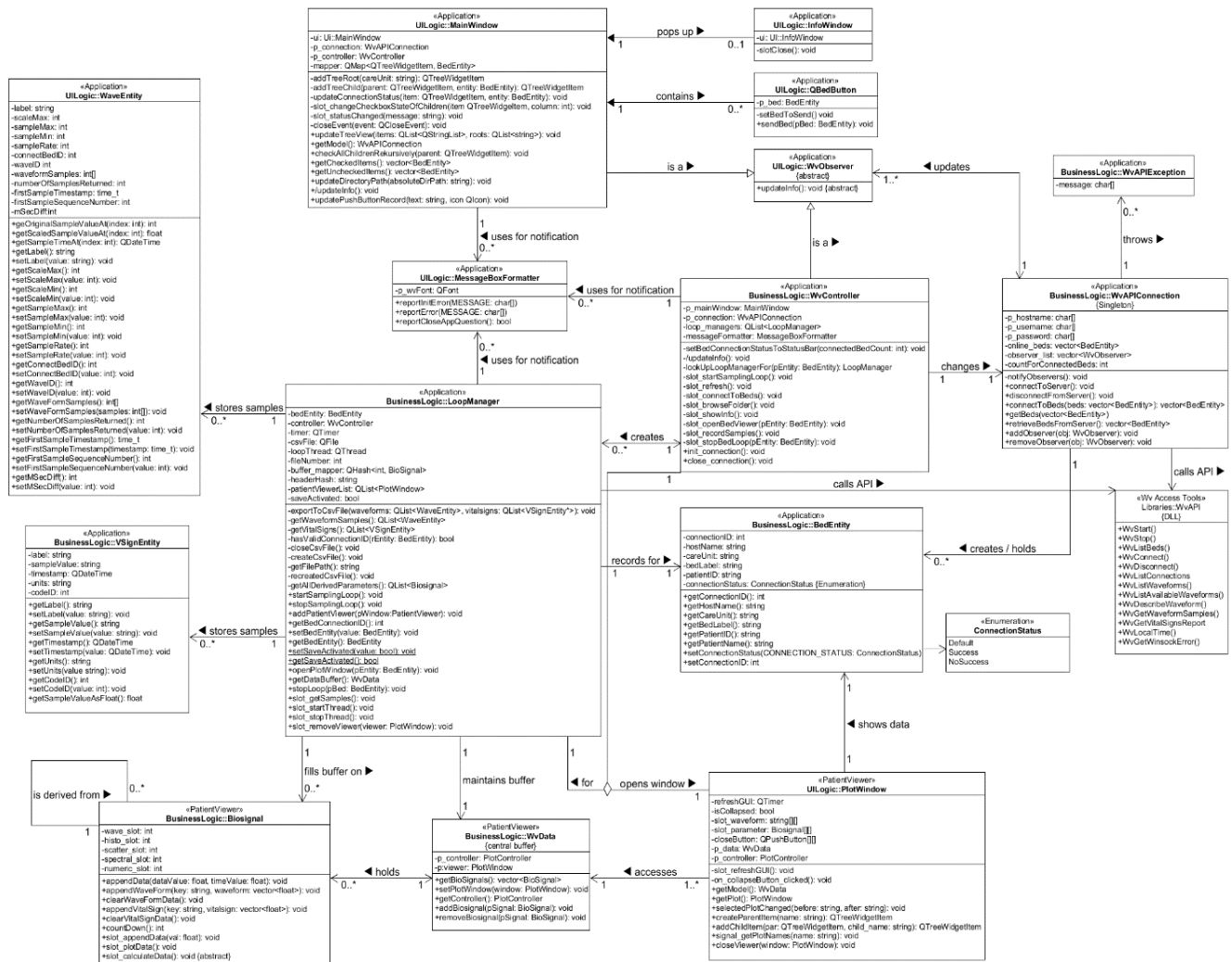


Figure 31. The final UML class diagram of VREACT. The diagram gives a glimpse about the structure of the application backbone (except of the Patient Viewer classes). To the most important components of the software belong the classes of the MVC Pattern (MainWindow, WvController, WvConnection). These classes are responsible for the central connection management of patient monitors (see section 2.1.5). Further, the LoopManager along with the BedEntity are the basic components of the sampling loop used for the sample acquisition. Other entities such as WaveEntity, VSignEntity and Biosignals are the central carriers of the biomedical samples from the monitor to the application buffer.

## QT project file

This section presents the contents of the project file template of the *Connection management* and *Patient recorder* part. The file contains the necessary source and header file paths used for linking during the application build. The file also contains the information about the dynamic linking of the delivered DLLs (section 2.2.3) and specifies the directory and rules for the application build (and deployment).

```
#-----  
#  
# Project created by QtCreator 2017-08-23T12:29:00  
#  
#-----  
  
QT      += core gui  
  
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets  
  
TARGET = WvApp  
TEMPLATE = app  
  
# The following define makes your compiler emit warnings if you use any feature of  
# Qt which has been marked as deprecated (the exact warnings depend on your compiler)  
# Please consult the documentation of the deprecated API in order to know how to port  
# your code away from it.  
  
DEFINES += QT_DEPRECATED_WARNINGS  
  
# QT flag to prevent Windows to create additional folders "debug" and "release"  
# which are not available for another OS.  
CONFIG -= debug_and_release  
  
# Include the PatientViewer.pri file, which will be added to this file if qmake is  
# invoked. The WvApi.pri file defines classes from the PatientViewer subproject  
# which are supposed to be used in this subproject (DEFINE AND USE ONLY API  
# CLASSES!).  
# include(C:/Users/Administrator/Desktop/QT Projects/Repository/WvRecorder/Premerge/  
# WvRecorder/src/PatientViewer/PatientViewer.pri)  
INCLUDEPATH += $$PWD/UILogic \  
              $$PWD/BusinessLogic \  
              $$PWD/FormLogic \  
              $$PWD/Libraries \  
  
# At this point, HEADERS (WvAPI.h) and LIBRARIES (e.g WvAPI.lib, ...) are found or  
# pointed to. Don't forget to put the corresponding .dll file in the application  
# directory or in the global PATH. # 'WvAPI', 'WvSvc', 'IGAcMsg' are mandatory  
# Dräger-related libraries, 'qtcsv' is an open-handling the .csv format files.  
LIBS      += -L $$PWD/Libraries/ -lWvAPI -lWvSvc -lIGAcMsg -lqtcsv  
  
SOURCES   += $$PWD/main.cpp \  
              $$PWD/BusinessLogic/WvController.cpp \  
              $$PWD/UILogic/MainWindow.cpp \  
              $$PWD/BusinessLogic/WvAPIConnection.cpp \  
              $$PWD/BusinessLogic/BedEntity.cpp \
```

```

    $$PWD/BusinessLogic/WvAPIException.cpp \
    $$PWD/UILogic/MessageBoxFormatter.cpp \
    $$PWD/Libraries/easylogging++.cc \
    $$PWD/UILogic/WvObserver.cpp \
    $$PWD/UILogic/QBedButton.cpp \
    $$PWD/BusinessLogic/WaveEntity.cpp \
    $$PWD/BusinessLogic/LoopManager.cpp \
    $$PWD/BusinessLogic/VSignEntity.cpp \
    $$PWD/UILogic/InfoWindow.cpp

HEADERS    +=    $$PWD/BusinessLogic/WvController.h \
    $$PWD/UILogic/MainWindow.h \
    $$PWD/BusinessLogic/WvAPIConnection.h \
    $$PWD/Libraries/WvAPI.h \
    $$PWD/Libraries/easylogging++.h \
    $$PWD/BusinessLogic/BedEntity.h \
    $$PWD/BusinessLogic/WvAPIException.h \
    $$PWD/UILogic/MessageBoxFormatter.h \
    $$PWD/UILogic/WvObserver.h \
    $$PWD/UILogic/QBedButton.h \
    $$PWD/BusinessLogic/WaveEntity.h \
    $$PWD/Libraries/stringdata.h \
    $$PWD/Libraries/writer.h \
    $$PWD/Libraries/reader.h \
    $$PWD/Libraries/qtcsv_global.h \
    $$PWD/Libraries/abstractdata.h \
    $$PWD/Libraries/variantdata.h \
    $$PWD/BusinessLogic/LoopManager.h \
    $$PWD/BusinessLogic/VSignEntity.h \
    $$PWD/UILogic/InfoWindow.h

FORMS      +=    $$PWD/FormLogic/mainwindow.ui \
    $$PWD/FormLogic/infowindow.ui

RESOURCES  +=    $$PWD/resources.qrc \

RC_FILE    =    $$PWD/ConfigFiles/WindowsResources.rc

```

## Biosignal sets on diverse ICUs

This section presents the bar chart diagrams depicting the biosignal occurrences on the special ICUs (13B1, 13C2, 9D). Although, the biosignal sets measured on the different care units slightly differ from each other, the occurrences of a biosignal on the ICUs are similar. E.g. 24 out of the 40 measured biosignals enjoy full or almost full representation on a given care unit, i.e. the group of biosignals found in all patient records except of one or two patients, by which the biosignals are missing (Figure 28). The most biosignals enjoying a maximal representation were observed on the care units 13B1 (Figure 32) and 13C2 (Figure 33). However, the largest set of exported biosignals was obtained in 9D (Figure 34).

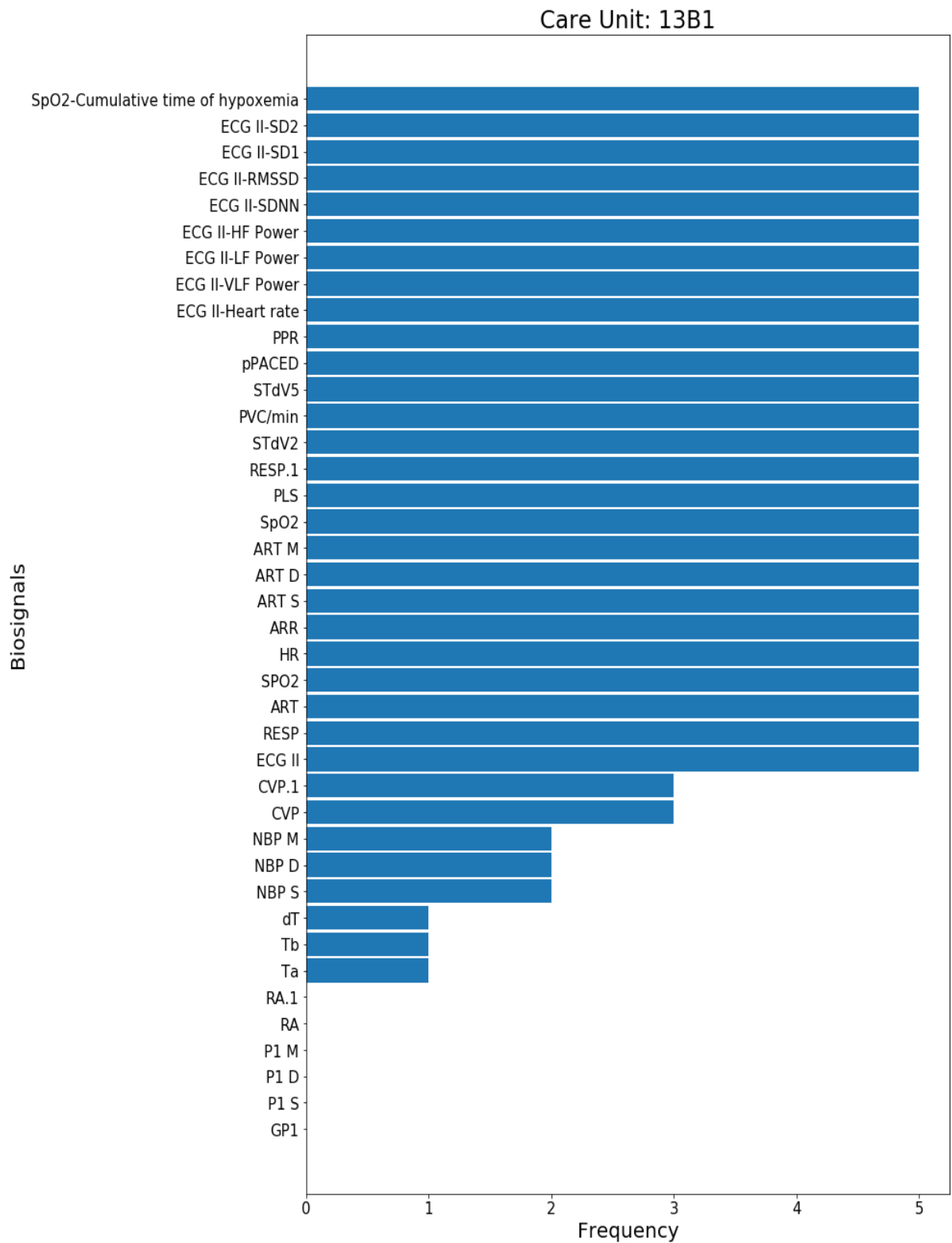


Figure 32. Overview about the recorded biosignals on the ICU labeled as 13B1. 26 out of 40 biosignals are exported for every monitoring unit in 13B1 and 13C2, which is the highest frequency among the three care units. However, there are 6 biosignals (RA.1, RA, P1 M, P1 D, P1 S, GP1) which are observed by none of the patients in the ICU.

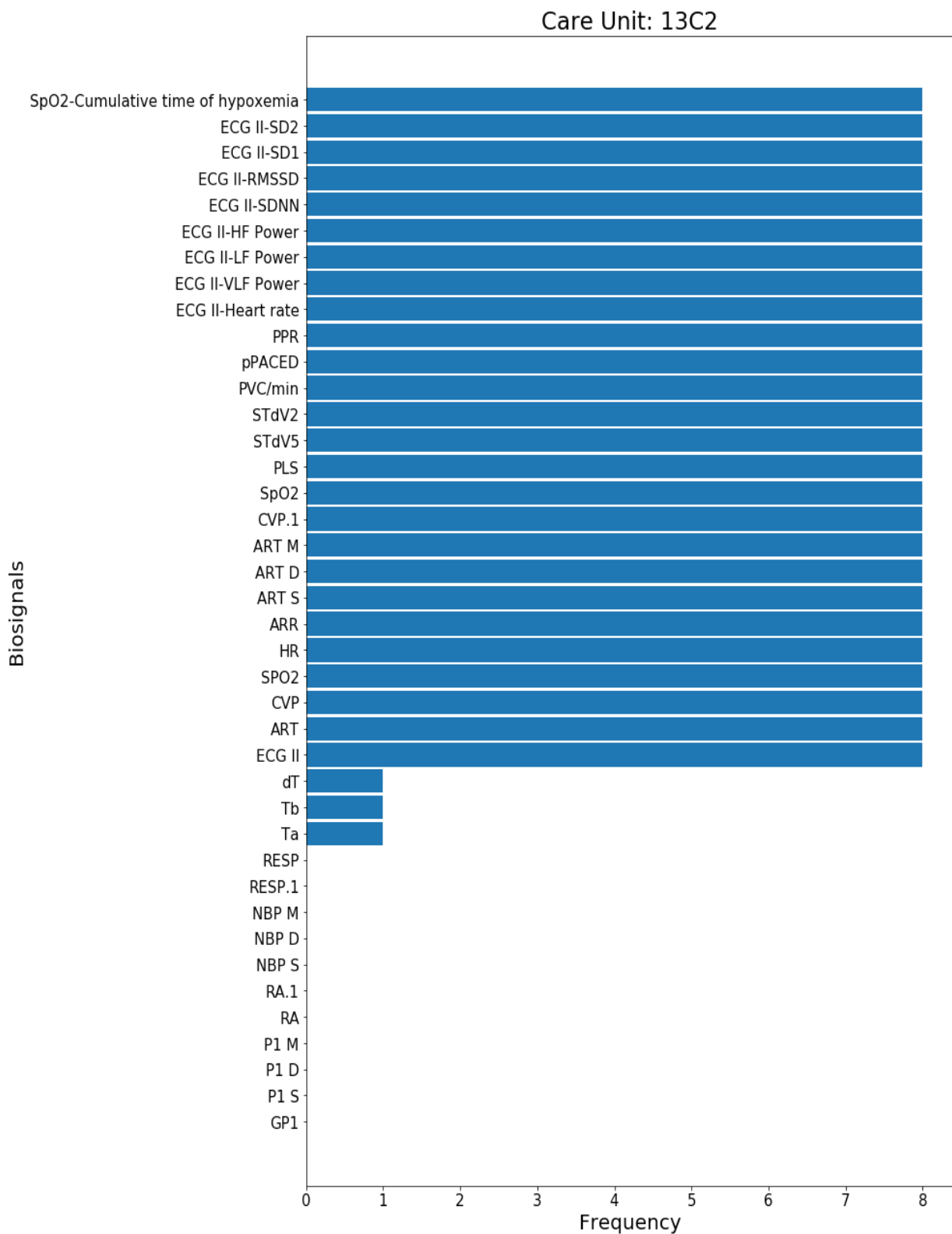


Figure 33. Overview about the recorded biosignals on the ICU labeled as 13C2. 26 out of 40 biosignals are exported for every monitoring unit in 13C2 and 13B1, which is the highest frequency among the all three care units. However, 13C2 exhibits the highest number of untracked biosignals from all ICUs being analyzed.



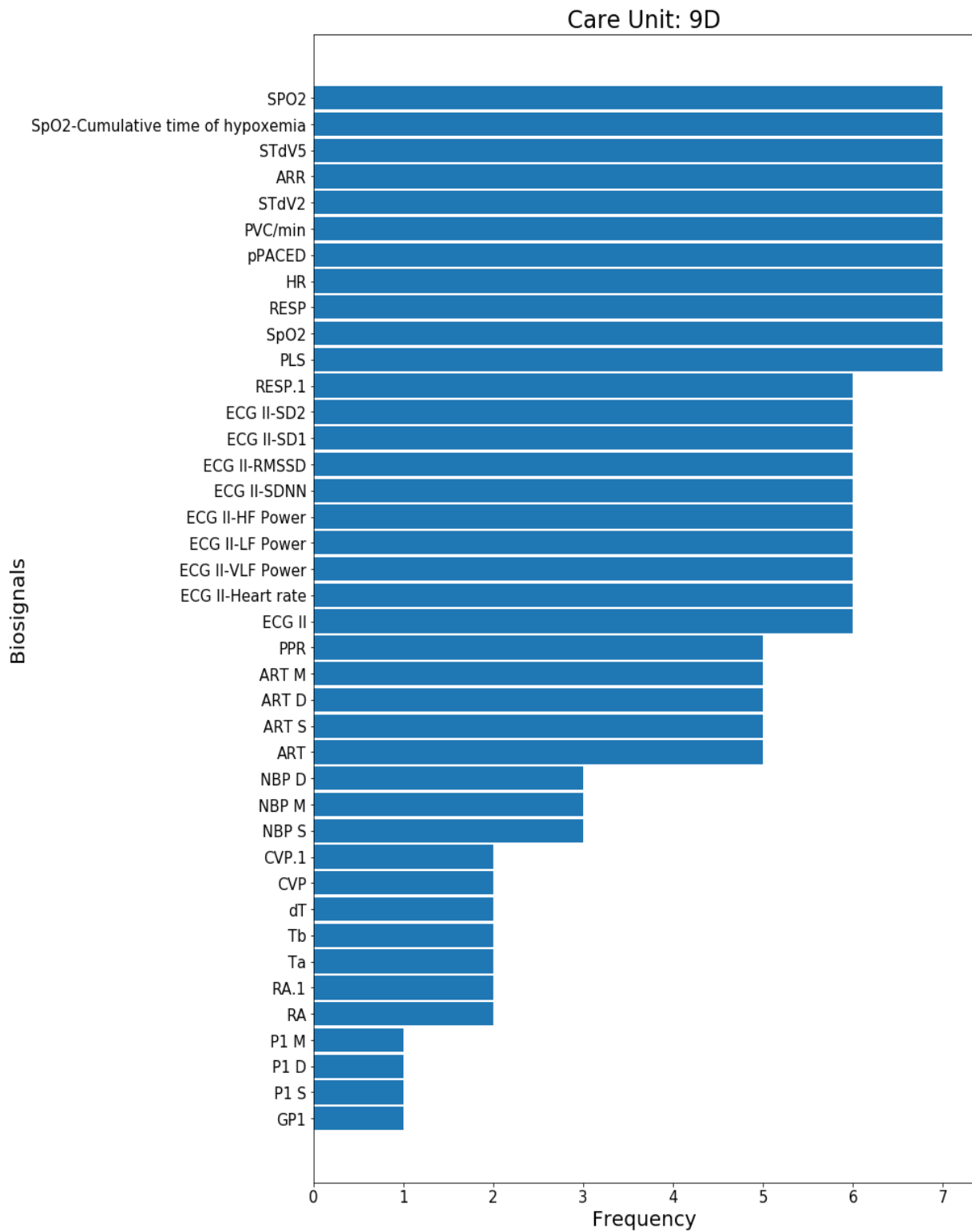
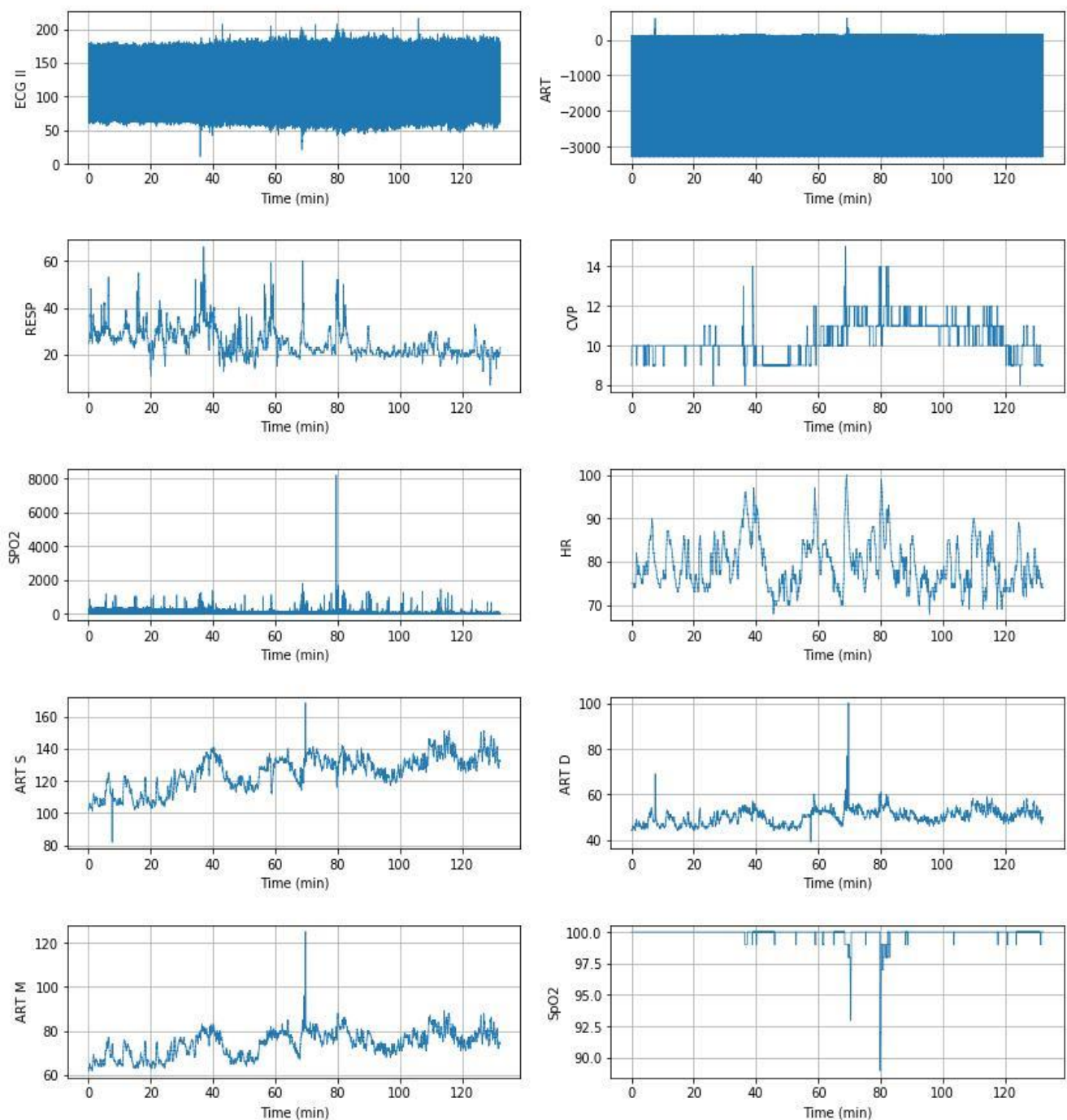


Figure 34. Overview about the recorded biosignals on the ICU labeled as 9D. Only 11 out of overall 40 biosignals are exported for every patient, which is the lowest frequency among the all three care units. However, 9D keeps no biosignals with the zero occurrences, i.e. the particular care unit measures much more biosignal types than the other ICUs 13B1 and 13C2.

## VREACT recorder example

This chapter presents the different vital parameters of the patient on position *Bett-04* being a part of the ICU *13B1* in AKH. The exported biosignals were obtained during the ca. 2-hour recording of vitals of stationary patients (see Chapter 3). All of the parameters were recorded by the newest version of VREACT.

### Exemplary export of biosignals



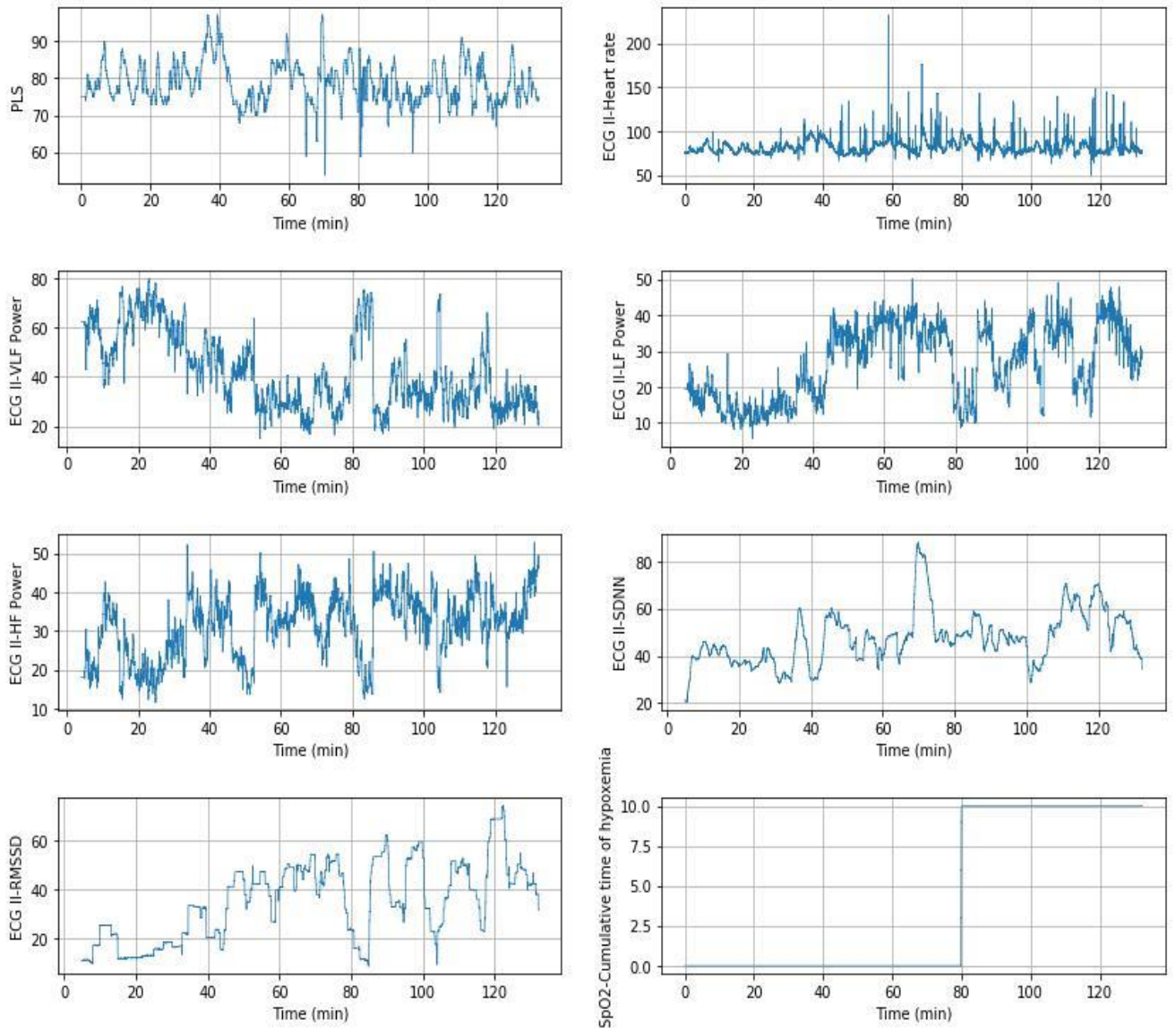


Figure 35. The summary of the exported biosignals from an anonymized patient visualized in Python. Ca. 2-hour measurement was done via the VREACT biosignal recorder. The exported data was processed and visualized by utilizing Python along with Jupyter Notebook [85]. The x-axis represents the time (in minutes) in the recording and the y-axis keeps the range of values for the given biosignals being exported by VREACT. The y-axis was labeled according to the exported biosignal. The measurement includes both the standard (e.g. ECG, ART, RESP) and novel (Cumulative time of hypoxemia, ECG-RMSSD, ECG-SDNN) parameters. Since the novel parameters were derived from their basic counterparts, their labels keep the name of the base physiological parameter (from which it was actually derived).

## References

- [1] Reed M. Gardner and M. Michael Shabot, "Patient-Monitoring Systems," in *Biomedical Informatics*, New York: Springer, New York, NY, 1995, pp. 585–625.
- [2] Y. Mao *et al.*, "An Integrated Data Mining Approach to Real-time Clinical Monitoring and Deterioration Warning," 2012.
- [3] F. Thürk *et al.*, "Real-time assessment of high resolution vital signs recording for calculation of perioperative clinical parameters," *Med. Meas. Appl. (MeMeA)*, 2018 *IEEE Int. Symp.*, 2018.
- [4] O. Chipara *et al.*, "Reliable Real-time Clinical Monitoring Using Sensor Network Technology," 2009.
- [5] G. W. Smetana, S. L. Cohn, and V. A. Lawrence, "Update Update in Perioperative Medicine," *Ann. Intern. Med.*, pp. 452–461, 2004.
- [6] F. Cavaliere *et al.*, "Intensive care after elective surgery: a survey on 30-day postoperative mortality and morbidity," *MINERVA Anesthesiol.* 459 *MINERVA ANESTESIOLOGIA*, vol. 7474, no. 9, pp. 459–68, 2008.
- [7] K. Fecho, A. T. Lunney, P. G. Boysen, P. Rock, and E. A. Norfleet, "Postoperative mortality after inpatient surgery: Incidence and risk factors," *Ther. Clin. Risk Manag.*, vol. 4, no. 4, pp. 681–688, 2008.
- [8] R. McCraty and F. Shaffer, "Heart rate variability: New perspectives on physiological mechanisms, assessment of self-regulatory capacity, and health risk," *Glob. Adv. Heal. Med.*, vol. 4, no. 1, pp. 46–61, 2015.
- [9] M. A. De Georgia, F. Kaffashi, F. J. Jacono, and K. A. Loparo, "Information Technology in Critical Care: Review of Monitoring and Data Acquisition Systems for Patient Care and Research," *Sci. World J.*, vol. 2015, pp. 1–9, Feb. 2015.
- [10] L. D. Hudson, "Design of the intensive care unit from a monitoring point of view," in *Respir Care*, 7th ed., 1985, pp. 549–559.
- [11] S. Miodownik, "Intensive Care," in *Clinical Engineering Handbook*, Joseph F. Dyro, Ed. Academic Press, 2004, pp. 373–376.
- [12] M. Alghatrif and J. Lindsay, "A brief review: history to understand fundamentals of electrocardiography," 2012.
- [13] H. Shubin and M. H. Weil, "Efficient monitoring with a digital computer of cardiovascular function in seriously ill patients.,," *Ann. Intern. Med.*, vol. 65, no. 3, pp. 453–460, 1966.
- [14] F. Booth, "Patient monitoring and data processing in the ICU," *Crit. Care Med.*, vol. 11, no. 1, pp. 57–58, 1983.
- [15] A. S. Sado, "Electronic medical record in the intensive care unit," *Crit. Care Clin.*,

- vol. 15, no. 3, pp. 499–522, 1999.
- [16] J. S. Gravenstein, “The automated anesthesia record Department of Anesthesiology , University of Florida College of Medicine ,” *Int. J. Clin. Monit. Comput.*, vol. 3, no. 2, pp. 131–132, 1986.
  - [17] M. A. Meyer *et al.*, “A computerized perioperative data integration and display system,” *Int. J. Comput. Assist. Radiol. Surg.*, vol. 2, no. 3–4, pp. 191–202, 2007.
  - [18] C. Mothukuri and K. C. P. Kumar, “Patient Monitoring System,” *Int. J. Sci. Res.*, vol. 2, no. 2, pp. 418–422.
  - [19] A. V. Halteren *et al.*, “Mobile Patient Monitoring: The MobiHealth System.,” in *The Journal on Information Technology in Healthcare*, 2004, vol. 2, pp. 365–373.
  - [20] Drägerwerk AG & Co. KGaA, “Infinity® Delta Monitor,” 2018. [Online]. Available: [https://www.draeger.com/en-us\\_us/Hospital/Products/Patient-Monitoring/Patient-Monitors/Infinity-Delta-Series](https://www.draeger.com/en-us_us/Hospital/Products/Patient-Monitoring/Patient-Monitors/Infinity-Delta-Series). [Accessed: 25-Apr-2018].
  - [21] “Dräger, Technik für das Leben,” 2018. [Online]. Available: [https://www.draeger.com/de\\_de/Home](https://www.draeger.com/de_de/Home). [Accessed: 20-Apr-2018].
  - [22] © 1994-2018 The MathWorks. Inc., “What is MATLAB?,” 2018. [Online]. Available: <https://www.mathworks.com/discovery/what-is-matlab.html>. [Accessed: 22-Apr-2018].
  - [23] N. Stylianides, M. D. Dikaiakos, H. Gjermundrød, G. Panayi, and T. Kyprianou, “Intensive care window: real-time monitoring and analysis in the intensive care environment,” *IEEE Trans. Inf. Technol. Biomed.*, vol. 15, no. 1, pp. 26–32, 2011.
  - [24] H. C. Lee and C. W. Jung, “Vital Recorder- A free research tool for automatic recording of high-resolution time-synchronised physiological data from multiple anaesthesia devices,” *Sci. Rep.*, vol. 8, no. 1, pp. 1–8, 2018.
  - [25] Draeger Medical Systems (Inc), “Infinity® Delta and Delta XL Patient Monitors,” 2018. [Online]. Available: <https://www.draeger.com/products/content/delta-xl-ds-9051660-en.pdf>. [Accessed: 25-Apr-2018].
  - [26] A. Dosinas, M. Vaitkunas, and J. Daunoras, “Measurement of human physiological parameters in the systems of active clothing and wearable technologies,” *Elektron. IR Elektrotechnika*, vol. 7, no. 7, pp. 77–82, 2006.
  - [27] R. Vecht and N. S. Peters, *ECG Diagnosis in Clinical Practice Second Edition ECG Diagnosis in Clinical Prattice Second Edition*. London: Springer-Verlag, 2009.
  - [28] J. Eldridge *et al.*, “Clinical guidelines by consensus recording a standard 12-lead electrocardiogram an approved methodology by the society forr cardiological science & technology,” *Soc. Cardiol. Sci. Technol.*, pp. 1–26, 2017.
  - [29] P. Kligfield *et al.*, “Recommendations for the standardization and interpretation of the electrocardiogram: Part I: The electrocardiogram and its technology: A scientific statement from the American Heart Association Electrocardiography and Arrhythmias

- Committee, Council on Cli,” *Circulation*, vol. 115, no. 10, pp. 1306–1324, 2007.
- [30] American Heart Association, “All About Heart Rate (Pulse),” 2015. [Online]. Available: [http://www.heart.org/HEARTORG/Conditions/HighBloodPressure/GettheFactsAboutHighBloodPressure/All-About-Heart-Rate-Pulse\\_UCM\\_438850\\_Article.jsp#.WuHm1S5ubIU](http://www.heart.org/HEARTORG/Conditions/HighBloodPressure/GettheFactsAboutHighBloodPressure/All-About-Heart-Rate-Pulse_UCM_438850_Article.jsp#.WuHm1S5ubIU). [Accessed: 26-Apr-2018].
  - [31] W. Brzenzinski, “Blood Pressure; Clinical Methods - The History, Physical and Laboratory Examinations,” in *Clinical Methods, 3rd edition*, Emory Univ., M. Editors: H Kenneth Walker, MD, W Dallas Hall, MD, and J Willis Hurst, Ed. Boston: Butterworths, 1990, pp. 95–97.
  - [32] American Heart Association, “Understanding Blood Pressure Readings,” 2017. [Online]. Available: [http://www.heart.org/HEARTORG/Conditions/HighBloodPressure/KnowYourNumbers/Understanding-Blood-Pressure-Readings\\_UCM\\_301764\\_Article.jsp#.WuH6qi5ubIU](http://www.heart.org/HEARTORG/Conditions/HighBloodPressure/KnowYourNumbers/Understanding-Blood-Pressure-Readings_UCM_301764_Article.jsp#.WuH6qi5ubIU). [Accessed: 26-Apr-2018].
  - [33] J. M. Maier, “What Is the Formula for Calculating Blood Pressure?,” *Leaf Group Lifestyle*, 2017. [Online]. Available: <https://healthfully.com/formula-calculating-blood-pressure-6328819.html>. [Accessed: 29-Apr-2018].
  - [34] P. Schober and L. A. Schwarte, “From system to organ to cell: Oxygenation and perfusion measurement in anesthesia and critical care,” *J. Clin. Monit. Comput.*, vol. 26, no. 4, pp. 255–265, 2012.
  - [35] K. K. Tremper, T. W. Rutter, and J. A. Wahr, “Monitoring Oxygenation,” *Curr. Anaesth. Crit. Care*, vol. 4, pp. 213–222, 1993.
  - [36] S. Lopez, “Pulse Oximeter Fundamentals and Design,” *Free. Semicond. Inc.*, pp. 1–39, 2012.
  - [37] E. Kaniusas, *Biomedical Signals and Sensors II*. 2015.
  - [38] F. Shaffer and J. P. Ginsberg, “An Overview of Heart Rate Variability Metrics and Norms,” *Front. Public Heal.*, vol. 5, no. September, pp. 1–17, 2017.
  - [39] Z. Sun *et al.*, “Postoperative Hypoxemia Is Common and Persistent: A Prospective Blinded Observational Study,” *Anesth. Analg.*, vol. 121, no. 3, pp. 709–715, 2015.
  - [40] M. Walsh *et al.*, “Relationship between Intraoperative Mean Arterial Pressure and Clinical Outcomes after Noncardiac Surgery,” *Anesthesiology*, vol. 119, no. 3, pp. 507–515, 2013.
  - [41] P. K. Stein and A. Reddy, “Non-linear heart rate variability and risk stratification in cardiovascular disease,” *Indian Pacing Electrophysiol. J.*, vol. 5, no. 3, pp. 210–220, 2005.
  - [42] V. Salmasi *et al.*, “Relationship between Intraoperative Hypotension, Defined by Either Reduction from Baseline or Absolute Thresholds, and Acute Kidney and Myocardial Injury after Noncardiac Surgery,” *Anesthesiology*, vol. 126, no. 1, pp. 47–

- 65, 2017.
- [43] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, *Patterns of enterprise application architecture*. Addison-Wesley Professional, 2002.
  - [44] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns – Elements of Reusable Object-Oriented Software,” Pearson Education, 1994, pp. 144–152, 249, 326–337.
  - [45] S. McConnell, *Code Complete: A practical handbook of software construction*, 2nd ed. Redmond, Washington: Microsoft Press, A Division of Microsoft Corporation, 2004.
  - [46] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley Professional, 2003.
  - [47] Lucid Software Inc., “UML Use Case Diagram Tutorial,” 2018. [Online]. Available: <https://www.lucidchart.com/pages/uml-use-case-diagram>. [Accessed: 30-May-2018].
  - [48] S. W. Ambler, “The Elements of UML™ 2.0 Style,” Cambridge University Press, 2005, pp. 33–46.
  - [49] M. Havey, *Essential Business Process Modeling*. O’Reilly Media, Inc., 2005.
  - [50] R. C. Papademetriou and D. A. Karras, “Business Modeling and Software Design: 6th International Symposium, BMSD 2016, Rhodes, Greece, June 20-22, 2016, Revised Selected Papers,” B. Shishkov, Ed. Rhodes: Springer, 2017, pp. 161–183.
  - [51] A. Cali, G. Gottlob, G. Orsi, and A. Pieris, “Querying UML class diagrams,” in *Foundations of Software Science and Computational Structures*, Springer, Berlin, Heidelberg, 2012, pp. 1–25.
  - [52] S. W. Ambler, “The Elements of UML™ 2.0 Style,” in *The Elements of UML™ 2.0 Style*, Cambridge: Cambridge University Press, 2005, pp. 47–72.
  - [53] “UML Class Diagram Tutorial,” *Lucid Software Inc.*, 2018. [Online]. Available: <https://www.lucidchart.com/pages/uml-class-diagram>. [Accessed: 05-Jun-2018].
  - [54] A. B. Singer, “Practical C ++ Design,” in *Library of Congress Control Number*, 2017, p. xviii, 9-17.
  - [55] M. Fowler, “Patterns of Enterprise Application Architecture,” in *Source*, vol. 48, no. 2, Pearson Education, Inc., 2003, pp. 330–333, 401–415.
  - [56] “Signals & Slots,” *Qt Company Ltd. Documentation contributions*, 2018. [Online]. Available: <http://doc.qt.io/qt-5/signalsandslots.html>. [Accessed: 07-Jun-2018].
  - [57] D. Esposito, “Cutting Edge - Pros and Cons of Data Transfer Objects,” *MSDN Magazine*, 2009. [Online]. Available: <https://msdn.microsoft.com/en-us/magazine/ee236638.aspx>. [Accessed: 08-Jun-2018].
  - [58] “Threading Basics,” *Qt Company Ltd. Documentation*, 2018. [Online]. Available: <http://doc.qt.io/qt-5/thread-basics.html>. [Accessed: 08-Jun-2018].

- [59] Draeger Medical Systems Inc., “Instructions for Use Infinity Gateway Suite,” Telford, USA, 2014.
- [60] N. Kehtarnavas and M. Gamadia, “Software Methods for Real-Time Image and Video Processing,” in *Real-Time Image and Video Processing*, Morgan & Claypool Publishers, 2006, pp. 55–78.
- [61] “QT Official Homepage,” © 2018 *The Qt Company*, 2018. [Online]. Available: <https://www1.qt.io/company/>. [Accessed: 12-Jun-2018].
- [62] “Welcome to Oracle VM VirtualBox!,” *Oracle*, 2018. [Online]. Available: <https://www.virtualbox.org/manual/ch01.html>. [Accessed: 12-Jun-2018].
- [63] “GitLab is the first single application for all stages of the DevOps lifecycle.,” *GitLab B.V.*, 2018. [Online]. Available: <https://about.gitlab.com/product/>. [Accessed: 12-Jun-2018].
- [64] “Infinity® Gateway Suite,” *Draeger Medical Systems (Inc)*, 2018. [Online]. Available: [https://www.draeger.com/en-us\\_us/Hospital/Products/Patient-Monitoring/Connectivity-and-Remote-Access/Infinity-Gateway-Suite](https://www.draeger.com/en-us_us/Hospital/Products/Patient-Monitoring/Connectivity-and-Remote-Access/Infinity-Gateway-Suite). [Accessed: 13-Jun-2018].
- [65] M. D. Ciampa and M. Revels, “Document Imaging and Problem Solving,” in *Introduction to Healthcare Information Technology*, Course Technology, 2012, pp. 191–193.
- [66] M. Reddy, “Libraries,” in *API Design for C++*, Morgan Kaufmann, 2011, pp. 391–399.
- [67] “Dynamic-Link Libraries,” © *Microsoft 2018*, 2018. [Online]. Available: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms682589\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx). [Accessed: 14-Jun-2018].
- [68] “About Dynamic-Link Libraries,” © *Microsoft 2018*, 2018. [Online]. Available: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms681914\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms681914(v=vs.85).aspx). [Accessed: 14-Jun-2018].
- [69] “How to link to a dll,” © 2018 *The Qt Company*. [Online]. Available: [https://wiki.qt.io/How\\_to\\_link\\_to\\_a\\_dll](https://wiki.qt.io/How_to_link_to_a_dll). [Accessed: 14-Jun-2018].
- [70] “qmake Project Files,” © 2018 *The Qt Company*. [Online]. Available: <http://doc.qt.io/archives/qt-4.8/qmake-project-files.html>. [Accessed: 14-Jun-2018].
- [71] K. Brown *et al.*, “Introduction,” in *Enterprise Java Programming with IBM WebSphere*, 2nd ed., Addison-Wesley Professional, 2003, pp. 4–13.
- [72] M. Walmsley, “Introduction,” in *Multi-Threaded Programming in C++*, London: Springer London, 2000, pp. 1–8.
- [73] M. Walmsley, “Threads,” in *Multi-Threaded Programming in C++*, London: Springer London, 2000, pp. 9–24.



- [74] G. Lazar and R. Penea, “Keeping Your Sanity with Multithreading,” in *Mastering Qt 5*, Packt Publishing Ltd, 2016, 2016, pp. 307–340.
- [75] K. E. Wiegers, “On Essential Requirements Concepts,” in *More About Software Requirements : Thorny Issues and Practical Advice.*, Microsoft Press, 2006, pp. 3–11.
- [76] © ISO/IEC 2011 and © IEEE 2011, “International Standard - ISO/IEC/IEEE: Systems and software engineering — Life cycle processes — Requirements engineering,” 2011.
- [77] I. Sommerville and P. Sawyer, “What are Requirements?,” in *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, 1997, p. 4.
- [78] J. Doe, “Recommended Practice for Software Requirements Specifications,” 2011.
- [79] W. O. Galitz, “Characteristics of Graphical and Web User Interfaces,” in *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*, Robert Elliott and Emilie Herman, Eds. John Wiley & Sons, Inc, 2002, pp. 15–27.
- [80] Y. Shafranovich, “Common Format and MIME Type for Comma-Separated Values (CSV) Files,” *The Internet Society*, 2005. [Online]. Available: <https://www.ietf.org/rfc/rfc4180.txt>. [Accessed: 21-Jun-2018].
- [81] T. Smith, “Install Windows Server 2008 and Windows Server 2008 R2,” 2009. [Online]. Available: <https://docs.microsoft.com/en-us/iis/install/installing-iis-7/install-windows-server-2008-and-windows-server-2008-r2>. [Accessed: 20-Jul-2018].
- [82] Drägerwerk AG & Co., “Infinity® OneNet,” 2018. [Online]. Available: [https://www.draeger.com/de\\_de/Hospital/Products/Patient-Monitoring/Network-Solutions/Infinity-OneNet](https://www.draeger.com/de_de/Hospital/Products/Patient-Monitoring/Network-Solutions/Infinity-OneNet). [Accessed: 20-Jul-2018].
- [83] Python Software Foundation, “The Python Tutorial,” *1. Whetting Your Appetite*, 2018. [Online]. Available: <https://docs.python.org/3/tutorial/appetite.html>. [Accessed: 22-Jul-2018].
- [84] M. Fuxjaeger, “Numeric and Scientific,” 2018. [Online]. Available: <https://wiki.python.org/moin/NumericAndScientific>. [Accessed: 22-Jul-2018].
- [85] Jupyter Project, “Jupyter,” 2018. [Online]. Available: <http://jupyter.org/>. [Accessed: 22-Jul-2018].
- [86] C. Chen, W. Härdle, and A. Unwin, “Scientific Design Choices in Data Visualization,” in *Handbook of Data Visualization*, 2008, pp. 63–70.
- [87] A. a T. Bui and W. Hsu, “Medical data visualization: Toward integrated clinical workstations,” *Med. Imaging Informatics*, no. Mvc, pp. 139–193, 2010.
- [88] C. M. Lilly, I. H. Zuckerman, O. Badawi, and R. R. Riker, “Benchmark data from more than 240,000 adults that reflect the current practice of critical care in the United States,” *Chest*, vol. 140, no. 5, pp. 1232–1242, 2011.

- [89] M. Capuzzo *et al.*, “Hospital mortality of adults admitted to Intensive Care Units in hospitals with and without Intermediate Care Units: a multicentre European cohort study,” *Crit Care*, vol. 18, no. 5, p. 551, 2014.
- [90] E. Kaniusas, “Fundamentals of biosignals,” in *Biomedical Signals and Sensors I: Linking Physiological Phenomena and Biosensors*, 2012, pp. 1–27.
- [91] M. Streit and N. Gehlenborg, “Points of View: Bar charts and box plots,” *Nature Methods*, vol. 11, no. 2. p. 117, 2014.
- [92] D. Lanzerath, “European Network of Research Ethics Committees - EUREC,” 2018. [Online]. Available: European Network of Research Ethics Committees - EUREC. [Accessed: 18-Aug-2018].

## Eidesstattliche Erklärung

*Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.*

Wien, 16.09.2018

.....  
Jakub Matta