MASTER

Using an advice system and a simulator to improve patient safety in anesthesia
a practical proof-of-concept

van de Vin, R.W.M.

Award date:
2008

Link to publication
Using an advice system and a simulator to improve patient safety in anesthesia

– a practical proof-of-concept –

by Rico van de Vin

Master of Science thesis

Project period: August 2006 – June 2008
Report Number: 8-09

Commissioned by:

Supervisors:
prof. dr. Erik Korsten (TU/e)
dr. ir. Paul de Clercq (Medecs)

Additional Commission member:
prof. dr. ir. Jan Bergmans (TU/e)
# Table of contents

TABLE OF CONTENTS .......................................................................................................................... III

SUMMARY ............................................................................................................................................ V

ACKNOWLEDGEMENTS ..................................................................................................................... VII

LIST OF FIGURES ................................................................................................................................ IX

LIST OF ABBREVIATIONS .................................................................................................................. XI

1 INTRODUCTION ............................................................................................................................... 1

1.1 PROJECT GOALS ......................................................................................................................... 1

1.2 CHAPTER OVERVIEW .................................................................................................................. 1

2 BACKGROUND .................................................................................................................................. 3

2.1 FACTS AND FIGURES ABOUT MEDICAL ERRORS .................................................................... 3

2.2 PATIENT SAFETY IN ANESTHESIA .......................................................................................... 4

2.3 FORMER SPS RESEARCH ON IMPROVING PATIENT SAFETY ................................................. 5

2.4 MEDICAT: CLINICAL DATA MINING AND DECISION SUPPORT SYSTEMS ............................. 7

3 TECHNICAL FRAMEWORK ............................................................................................................. 9

3.1 DATA ACQUISITION: VIERKLEUREN PEN AND CHIPSOFT EZIS ............................................. 9

3.1.1 Vierkleurenpen ....................................................................................................................... 9

3.1.2 CS-EZIS ................................................................................................................................. 9

3.2 DECISION SUPPORT: GASTON ................................................................................................. 10

3.2.1 Decision support systems: a subclass of expert systems ....................................................... 10

3.2.2 Gaston applications ............................................................................................................. 10

3.3 SIMULATION: WINDOWS APPLICATION DEVELOPED USING BORLAND DELPHI ............ 11

3.3.1 Windows Messages ................................................................................................................ 12

3.3.2 ADO ..................................................................................................................................... 12

3.3.3 XML ..................................................................................................................................... 12

3.4 SPEECH RECOGNITION: DRAGON NATURALLY SPEAKING (DNS) ....................................... 12

4 MAJOR IMPLEMENTATION ISSUES ............................................................................................. 13

4.1 VIERKLEUREN PEN .................................................................................................................. 13

4.2 CS-EZIS DATABASE ................................................................................................................... 14

4.3 DECISION SUPPORT SYSTEM ................................................................................................. 14

4.3.1 Coupling with databases ...................................................................................................... 14

4.3.2 Work flow ............................................................................................................................ 15

4.4 PATIENT SIMULATOR ................................................................................................................ 16

4.4.1 Work flow ............................................................................................................................ 17

4.4.2 Simulation routines ............................................................................................................. 20

4.4.3 Case description file debugging and fine tuning ................................................................. 24

5 RESULTS ............................................................................................................................................ 25

5.1 ADVICE SYSTEM PROOF-OF-CONCEPT ................................................................................ 25

5.2 PRACTICAL EXAMPLES ............................................................................................................. 26

5.2.1 Malignant Hyperthermia (MH) protocol guidance ................................................................. 27

5.2.2 Adapting sevoflurane concentration to patient’s age and desired MAC ............................ 30

5.2.3 Hypovolemia ....................................................................................................................... 31

5.3 SPS PH.D. RESEARCH ............................................................................................................... 32

6 CONCLUSIONS AND RECOMMENDATIONS ............................................................................ 33

7 TEMPORAL ABSTRACTION AND TREND DETECTION ............................................................. 35

7.1 TEMPORAL ABSTRACTION ..................................................................................................... 35

7.1.1 Knowledge-based Temporal Abstraction Method .............................................................. 35

7.1.2 Bellazzi’s TA method ........................................................................................................... 37

7.1.3 Multivariable Fuzzy Temporal Profile (MFTP) model ......................................................... 39
7.1.4 Temporal abstraction in the VIE-VENT system ................................................................. 39
7.1.5 Basic Temporal Abstraction and Temporal Relation Extraction .................................. 40
7.2 TREND DETECTION ............................................................................................................. 41
7.3 DISCUSSION ....................................................................................................................... 41

8 LITERATURE .......................................................................................................................... 43

APPENDIX A. SOURCE CODE PATIENT SIMULATOR .......................................................... 47

APPENDIX B. READING AND WRITING VIERKLEURENPEN ........................................ 71

B.1. ADMINISTRATION OF MEDICATION .............................................................................. 71
   B.1.1. Structure of th_events .............................................................................................. 71
   B.1.2. Registering the administration of medication .......................................................... 71
   B.1.3. Predefined medication ........................................................................................... 72
   B.1.4. Example .................................................................................................................... 72
B.2. START OR CHANGE OF A PUMP ..................................................................................... 73
   B.2.1. Structure of th_given_pumps ................................................................................... 73
   B.2.2. Registering (the change of) a pump .......................................................................... 73
   B.2.3. Predefined pump fluids ......................................................................................... 73
   B.2.4. Example .................................................................................................................... 74
B.3. ATTACHMENT OF A DRIP ............................................................................................... 74
   B.3.1. Structure of th_given_infuus .................................................................................... 74
   B.3.2. Registering the attachment of a drip ........................................................................ 75
   B.3.3. Predefined drip fluids .............................................................................................. 75
   B.3.4. Example .................................................................................................................... 76
B.4. ACTIONS ............................................................................................................................. 76
   B.4.1. Predefined actions .................................................................................................... 76
   B.4.2. Example .................................................................................................................... 77
B.5. COMPLICATIONS .............................................................................................................. 78
   B.5.1. Registering a complication ....................................................................................... 78
   B.5.2. Complication templates .......................................................................................... 79
B.6. UPDATING THE VIERKLEURENPEN CLIENT WINDOW ........................................... 79

APPENDIX C. WRITING CASES FOR PATIENT SIMULATOR ............................................. 81

APPENDIX D. DEFINING MEDICATION AND EVENTS FOR PATIENT SIMULATOR .......... 87

D.1. READING MEDICATION AND EVENTS FROM VIERKLEURENPEN ............................. 87
D.2. SPECIFYING MEDICATION IN A XML FILE ................................................................. 87
D.3. SPECIFYING EVENTS IN A XML FILE .......................................................................... 89

APPENDIX E. HOW TO USE THE PATIENT SIMULATOR ...................................................... 91
Summary

In 1999 the Institute of Medicine in the United States published a report on avoidable errors in hospitals. The report stated that on a yearly base between 44,000 and 98,000 people die in US hospitals due to preventable (human) mistakes. The Dutch Inspectie voor de Gezondheidsdienst states in its annual report over 2002 that no exact figures are known for the situation in the Netherlands but that there is no reason to believe that these differ from those in other countries. Internationally anesthesia is regarded as a leader in improving patient safety because of former results in the reduction of deaths due to anesthesia. However there still are lives to save.

During this project an advice system is developed and implemented that supports anesthetists in their work. The system uses two knowledge sources. The anesthesia recording database of the Vierkleurenpen application provides data about the running case and previous cases of the same patient. The hospital information system database of the Chipsoft EZIS application contains general information about the patient and his physical condition. Based on this information the advice system interacts with these data and the data generated by the anesthesia machine and the system comes up with advices or warnings. The anesthetist can overrule the suggestions and make his own decisions.

Events are registered by the advice system into the Vierkleurenpen database. The advice system can monitor this data and physiological parameters of the patient (such as heart rate, blood pressure, temperature, ventilation and anesthetics). Protocols and guidelines can be used to interpret the data and generate warnings if things are about to go wrong.

To simplify the task of anesthesia recording the system has been developed in such a way that touch screen and speech can be used for input tasks.

In order to practice the use of the system and to practice protocols for seldom occurring complications a simulator has been developed and implemented. Simulations scripts in xml can be written and run on the simulator. The simulator simulates a patient connected to an anesthesia monitor. The simulator can be used in combination with Vierkleurenpen and the advice system for training purposes.

The result of this study is a proof-of-concept of an advice and support system for anesthesia. The system is setup in a stand-alone network using some computers and a router.

The system can help to improve patient safety in anesthesia. A quantitative study of the impact of the use of this system takes years and therefore was no part of this project. It is however recommended to perform such a study in the future.

Some improvements should be made to the system, maybe even before taking the system into use. The medical contents should be increased by medical personnel. It should be investigated how the system can be safely integrated in the hospital network. A solid speech recognition solution should be implemented. The fact that medical staff wear masks and the rumor in an operating room complicates the use of speech recognition. However, it should be possible to solve this problem, due to
recent advances in speech recognition and hardware-improvements (selective microphones). Several speech recognition solutions should be compared qualitatively.

In the future it is conceivable to be able to predict certain events. Data- and process mining will be useful for that. Therefore the system should be able to represent temporal abstract concepts like ‘increasing’ and ‘decreasing’ as well. A small literature study has been performed on temporal abstractions and trend detection. This study can be taken as a start for a new project aiming on the integration of data- and process mining with the presented advice system.

This work is a direct result of the Medicast project – in which the TU/e, Medecs and the Catharina Hospital participated. That project was aimed to combine predictive modeling with decision support. The work presented in this thesis focuses on anesthesia. The resulting system enables to bring the results of theoretical research to clinical practice.
Acknowledgements

It is not common to write acknowledgements in a graduation report. However my graduation has been a certain struggle for me and some people motivated me to keep on going. I want to thank them for their support.

In the first place I want to thank my supervisors Erik and Paul for all their patience. There have been months with nearly no progress but they gave me the chance to renew my energy and complete the project. At the moment that I could not see the end coming any nearer anymore, Herman stepped in the role of mental coach en kept me going until the project was finished. Thanks for that!

During the total length of my study my contact with my mentor Paul Goemans has been inspiring for me. He always convinced me that studying slow is not that bad as long as one develops him selves in other fields, as I did with building up my company.

With concern to the contents of this project I’m thankful to Leo, who has always been prepared to adapt ‘his’ Vierkleurenpen to my wishes. Maarten and Niek have had a great contribution to the final implementation of the advice system as a part of their graduation project. Their entrance in the project renewed my energy and enabled me to complete the project.

For the past few years Harald has been my roommate and he was always prepared to help me solving problems and to give advice.

Thanks to all!

Rico.
List of figures

Figure 1, time line SPS research on patient safety improvements (Ph.D. projects) ......7
Figure 2, GASTON structure (figure taken from [17])..............................................11
Figure 3, Main flow chart for the DSS (figure taken from [19]) .....................16
Figure 4, part of the state diagram of a simulation script ...............................17
Figure 5, simulation flow chart ........................................................................24
Figure 6, test setup advice system, an HP laptop serves as a server and three touch
screen pc’s serve as clients ..................................................................................25
Figure 7, some screenshots from the advice system (clockwise starting upper left:
preoperative information, latest laboratory results, induction medication and tube
setting) ..............................................................................................................26
Figure 8, the advice system recognizes an increase in heart rate (red line) and blood
pressure (purple dotted line) and shows a message that there is possible occurrence of
MH ..............................................................................................................28
Figure 9, the advice system can support the user during treatment of MH ........28
Figure 10, in the simulator one can activate events or administer medication ....29
Figure 11, after registration of an event in Vierkleurenpen the advice system shows
the next step in the treatment protocol ..............................................................29
Figure 12, nomogram relating age, total MAC and end-expired concentrations of
volatile agent and nitrous oxide (figure taken from [21]).................................30
Figure 13, a screenshot of how the nomogram of Lerou can be used in clinical
practice using the advice system developed in this project .........................31
Figure 14, Knowledge-based temporal abstraction method (figure taken from [27]) 36
Figure 15, decomposition of the temporal abstraction task (figure taken from [29])..38
Figure 16, temporal relations used in Temporal Relations Abstraction method (figure
taken from [32]) ..........................................................................................41
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4kp</td>
<td>Vierkleurenpen (anesthesia recording application)</td>
</tr>
<tr>
<td>ADO</td>
<td>ActiveX Data Objects</td>
</tr>
<tr>
<td>AEP</td>
<td>Auditive Evoked Potential</td>
</tr>
<tr>
<td>ASP</td>
<td>Active Server Pages (Microsoft web scripting language)</td>
</tr>
<tr>
<td>CAF</td>
<td>Cognitive Activity Flow</td>
</tr>
<tr>
<td>CS-EZIS</td>
<td>Chipsoft Elektronisch Ziekenhuisinformatiesysteem (hospital information system)</td>
</tr>
<tr>
<td>CZE</td>
<td>Catharina Ziekenhuis Eindhoven (hospital)</td>
</tr>
<tr>
<td>DNS</td>
<td>Dragon Naturally Speaking (speech recognition software)</td>
</tr>
<tr>
<td>DSS</td>
<td>Decision Support System</td>
</tr>
<tr>
<td>EEG</td>
<td>Electroencephalography</td>
</tr>
<tr>
<td>FTP</td>
<td>Fuzzy Temporal Profile</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HL7</td>
<td>Health Level 7 (medical data exchange protocol)</td>
</tr>
<tr>
<td>HP</td>
<td>Hewlet-Packard (computer manufacturer)</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language (standard webpage markup language)</td>
</tr>
<tr>
<td>IIS</td>
<td>Internet Information Services (Windows’ build-in web server)</td>
</tr>
<tr>
<td>IOM</td>
<td>Institute of Medicine (US)</td>
</tr>
<tr>
<td>KBTB</td>
<td>Knowledge-based Temporal Abstraction</td>
</tr>
<tr>
<td>MAC</td>
<td>Minimal Alveolar Concentration</td>
</tr>
<tr>
<td>MFTP</td>
<td>Multivariable Fuzzy Temporal Profile</td>
</tr>
<tr>
<td>MH</td>
<td>Malignant Hyperthermia (medical complication)</td>
</tr>
<tr>
<td>MHAUS</td>
<td>Malignant Hyperthermia Association of the United States</td>
</tr>
<tr>
<td>MS</td>
<td>Microsoft (software manufacturer)</td>
</tr>
<tr>
<td>SPS</td>
<td>Signal Processing Systems (TUE research group)</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TA</td>
<td>Temporal Abstraction</td>
</tr>
<tr>
<td>TUE</td>
<td>Eindhoven University of Technology</td>
</tr>
<tr>
<td>UK</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>UMC</td>
<td>Universitair Medisch Centrum</td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1 Introduction

In this introduction first the goals of this graduation project are mentioned. The introduction closes with an overview of the chapters in this report. The work presented in this report was carried out at the Signal Processing Systems (SPS) research group of the Eindhoven University of Technology (TUE) and supervised by dr. ir. Paul de Clercq (supervisor) and prof. dr. Erik Korsten (graduating professor). Third member of the graduation commission is prof. dr. ir. Jan Bergmans (research group chairman).

1.1 Project goals

The main goal of this project and thus the main portion of this report is about the design and implementation of a powerful anesthesia support system. Powerful as in that the system contributes significantly to the improvement of daily practice in anesthesia. The combination of the work in this project and the work of Niek Lips and Maarten van Dorst should lead to a proof of concept for such a system. The actual contents for creating a useful system have to be delivered by medical personnel and can be added to the system later on.

With the Gaston decision support system framework as a pivot, a couple of existing software tools are integrated together with some newly developed applications. The system that comes into existence can be used for both online and real time support as well as for offline training on the procedure for rare complications that can occur during anesthesia.

As a second goal, for the purpose of offline training, a simulator has to be developed that is able to run predefined complication scripts. The simulator should simulate the outputs of an anesthesia monitoring device when a patient is connected to it.

In addition to these two goals a literature study has been performed in order to come with suggestions on how to implement temporal abstractions like ‘increasing’ and ‘decreasing’ in the knowledge-based system. This will be the basis for a future project in which trend detection and data mining will be added to the system.

1.2 Chapter overview

The second chapter of this report stresses the importance of this project by explaining the seriousness of the problem of avoidable medical errors. It is also shown how this project fits in the line of previous research projects performed in the SPS research group on the topic of hospital automation.

The technical framework of the developed anesthesia support system is sketched in the third chapter. That chapter will introduce and discuss the software components that were integrated into a powerful system.

The fourth chapter uncovers the most important implementation highlights. A more in depth overview of the implementation details can be found in the appendices.

In the fifth chapter some examples are shown of how the system can be used in clinical practice in order to support medical personnel during difficult tasks.
The resulting system is discussed and suggestions for further research and improvements are given in the sixth chapter.

The seventh and last chapter reports about a literature study on the representation of abstract concepts in knowledge-bases. In medical monitoring applications it is important that the system has notice of what the meaning of terms like ‘high’, ‘slowly’, ‘increasing’ and so on is. Suggestions are given for further research on how these temporal abstractions can be represented within the developed system.
2 Background

This chapter first gives some facts about medical errors in order to stress the importance of this work. Then it focuses on the particular field of anesthesia for which the advice system is developed. After that this project is positioned in the right historical context looking at previous researches performed in the Signal Processing Systems (SPS) research group of the Eindhoven University of Technology, where this work was carried out. This chapter closes with positioning this work in line with the Medicast project.

2.1 Facts and figures about medical errors

The general public first came aware of the large number of patients suffering from (avoidable) medical injuries when the Institute of Medicine (IOM) in the United States published To Err is Human: Building a Safer Health System back in 1999. That report declares that between 44,000 and 98,000 people die in US hospitals as a result of medical errors. [1] This is more than the number of deaths caused by traffic incidents, breast cancer or AIDS. [2] The numbers suggest that more Americans get killed in US hospitals every 6 months than died in the entire Vietnam War. If the numbers are correct, the public health care system is a public health menace of epidemic proportions. [hayward2001] The numbers presented in the IOM report are doubted by some physicians because according to them a portion of the deads would have died anyway as a result of their severe illness. [3] Limitations of the mean of identifying errors as used in the researches that presented the numbers are demonstrated in a new study. [hayward2001] According to others however most of the severely ill patients did not meet the screening criteria and their data were excluded from the research. They even call it unethical to lessen the significance of the death of severely ill patients. [1]

The percentage of adverse events (unintentional outcomes with injury caused by the health system) ranges from 2.9% (Utah and Colorado, US) to 16.6% (Australia) of the hospital admissions. From these adverse events between 27.7% (France) and 51.2% (Australia) is considered to be avoidable. To what extent patients suffer from the injuries has never been examined systematically. When data from foreign studies are extrapolated to the Dutch situation, avoidable mortality in the Netherlands will be 1,500 – 6,000 patients per year. [2] The Inspectie voor de Gezondheidszorg states in their annual report over 2002 that exact figures for the Dutch situation are unknown but that there is no reason to think that the situation in the Netherlands is different from elsewhere. [4] Although the presented figures are subject of argue it might be clear that there is a need for the reduction of avoidable medical errors.

Most of the adverse events are related to surgical interventions. These events included, amongst others, technical complications, maemorrhage and wound infection. The importance of this project is stressed by the fact that also during prescribing, delivering and administering of medications a lot of adverse events occur: previous known and unknown allergic reactions, administration of the wrong drugs or the wrong dose. [2]

Besides preventable injuries adverse events have also financial consequences. The costs of increased hospital stay due to avoidable adverse events in the UK are
estimated at 1,000 million pounds per year. The total costs of avoidable adverse events in the US are estimated at 17,000 – 29,000 million dollars every year. That includes loss of income, production loss, incapacity for work and direct health care costs. [2]

The most important message of the IOM report is not subject of discussion and has broad support. It is not about the figures but about a call for understanding the cause of errors and for developing an approach using computerized and other support systems to decrease error rates. [3] Now eight years later there is still a lot to improve in patient safety. [5]

2.2 Patient safety in anesthesia

The provision of anesthesia is intrinsic and hazardous. However it is necessary for the performance of surgery and other invasive procedures. In the past, considerable efforts have been made to make anesthesia safer for patients. Mortality rate in anesthesia decreased from 1 death in every 1,561 cases to 1 death in several hundred thousand. For healthy patients anesthesia is nearly as safe as commercial aviation. As a result anesthesia is regarded as a leader in improving patient safety. [6]

However when patients becomes progressively less healthy the risk of dying from anesthesia increases to roughly 1 in every 10,000-15,000 cases. This rate has been more or less unchanged over the past 20 years. Fewer data are available to determine the trend of morbidity. It is unclear whether fewer anesthetic injuries are occurring. [6]

Former researcher found that human error was a factor in 70% of the occurrences of critical incidents. These errors are, amongst others, errors in drug administration, anesthesia machine use, airway management, fluid management and the use of intravenous equipment and monitoring device use. Medication errors continue to be common occurrences. Standards, protocols and evidence-based practices are not uniformly implemented or followed. [6] Errors in drug administration can be prevented by the use of advice systems. Also the implementation of standards and protocols can be improved by using an advice system.

Simulation is important for exposing practitioners to rare, but life-threatening, events and training in crisis management. [6]

A knowledge based system can be an expert in monitoring very large amounts of data. This can be both historical and current data from both the current and other patients. By developing smart rules (guidelines) for such a system a powerful aid can be build. One can think of intelligent advice (warning the staff to prevent complications), protocol guidance (helping the staff to follow the right procedures during a complication) and prediction (estimating the chance that a complications can occur based on similar cases in the past).

In this project the main effort is put into the advice and the guidance task but also a base is created for making predictions.
2.3 Former SPS research on improving patient safety

Already back in 1986 the SPS group made a start in improving the organization and information management in the operating room. A.P. Meijler developed a data acquisition and display system. Main goal of his research was to automatically collect and process data and a centralized representation of this data. Readability of information and the use of colors played an important role. Meijler suggested the integration of machines in order to get a centralized arrangement of control panels. From his research it could be concluded that his system helped to detect slow unwanted changes in parameter values and that experienced anesthesiologists gained better control of parameters when using the system. Meijler suggested the use of knowledge about the causes of critical incidents in order to design automated patient monitoring systems as efficient as possible. In his opinion three causes had to be considered first: administration of wrong medication, incorrect gas flows and loose connections in the respiration circuit. [7]

In 1990 J.A. Blom designed the basis of a real time patient monitoring expert system. His SIMPLEXIS was a collection of software tools that could be used to design and implement real time expert systems. Knowledge was compiled into an internal representation and checked for incorrectness in many ways. That ensured fast and safe systems. The SIMPLEXIS languages consisted of goals (what should be done?) and protocols (when should this be done?). The goals and protocols together formed rules. [8]

In the same year P.J.M. Cluitmans developed a reliable and objective method to monitor anesthetic depth. The methods used up to then where not usable anymore. The four aspects of anesthesiology (unconsciousness, repressed processing of pain stimuli, reduced autonomic reflexes and repressed muscle functions) were no longer controlled together but separately. Cluitmans investigated the correlation between neurophysiological parameters and clinical observed and measured anesthetic depth. He chose auditive evoked potential parameters (the response of the nerve system to an auditive stimulus) as an indicator for anesthetic depth. Cluitmans concluded that his methods could be used for the measurement of anesthetic depth theoretically but that the practical and technical usability of the methods should be further improved first. [9]

Three years later, in 1993, J.H.M. van Oostrom investigated the use of information about the patient and the type of operation in order to automatically set alarm limits in patient monitoring systems during anesthesia. Setting the limit manually caused too much work and therefore most of the time no limits, or default factory settings, where used. The developed system used information from a preoperative evaluation and the knowledge of the anesthesiologist about anesthesia in order to generate an advice for the alarm settings. [10]

In 1996 Nicole de Beer worked on the monitoring of anesthetic depth using spontaneous and evoked electroencephalographic (EEG) activity. A new method was necessary because new anesthetics have almost no side effects and older methods were based on the side effects of anesthetics. The research of De Beer builds on the earlier work of Cluitmans. Her method uses spontaneous brain activity and auditive evoked brain activity (AEP). The small amplitude of these signals causes them to be very sensitive to distortion due to medical equipment in the operating room. For that
reason De Beer developed a method to detect and remove these artifacts. The research proved that EEG and AEP cannot be used to predict the concentration of administered anesthetics in the blood. However it could be predicted correctly whether or not the patient would show an increase in heart rate or blood pressure on incision in 72% of the cases. Latencies in the peaks of the AEP proved to be related to level of unconsciousness and the amplitudes to the level of painlessness. This suggested that EEG and AEP can be used to judge both levels separately. [11]

Hannie van Hooff worked in close cooperation with Nicole de Beer. In 1996 she investigated whether evoked and event related potential can be used to see whether patients can sense, process and store auditive information during anesthesia. The research showed that in many cases information processing continues during anesthesia using propofol or alfentanil but this information processing does not reach the level of consciousness. Van Hooff concluded that AEP can contribute to conventional method for measuring anesthetic depth. [12]

In 1998 Philip de Graaf worked on the design of an intelligent anesthesia monitoring system. He developed the cognitive activity flow model (CAF model) to get insight in the decision process of the anesthesiologist. With this model the need for support can be specified. It proved to be possible to construct a cognitive map of relations between events and measurement data. This map can be used as basis for information management and support. De Graaf did not manage to construct a cognitive map of the relations between processes and measurement data. As a reason for this he names the difference in how different anesthetists look at the process of anesthesia. De Graaf defined four levels of need for support: 1) user interface for easy access to relevant information, 2) possibility to see supporting information for current case, 3) use of input from the anesthetist about normal and abnormal parameter values to automatically support when the situation changes and 4) full automatic support based on automatic classification and data validation without input from the anesthetist. De Graaf developed a user interface in the second level. Key features of the system are grouping of relevant information, use of a touch screen and the use of icons instead of text. For future research De Graaf suggested judgment of how effective the use of the system is in decreasing the risks for patients and how incident registration systems can contribute to the building of databases for knowledge acquisition and design of support systems. [13]

A year later, in 1999, Luc Cluitmans designed software for the collection and analysis of data on an intensive care unit. The data acquisition part (HPDATA) continuously collects the measurement data of the monitoring equipment. The HPView application shows the doctors charts with information about the patient’s health. Two applications (COPDA and DataWand) are designed to simplify the design and implementation of patient monitoring systems that use the data of HPDATA. DataWand provides one with a graphical interface to implement data processing algorithms for offline data analysis. The primary function of COPDA is the interpretation of data processing algorithms. [14]

In 2003 P.A. de Clercq presented his research on how to collect, represent and implement computer based guidelines for (medical) decision support. His research led to the development of the Gaston framework: a collection of software tools for the development and implementation of decision support systems. Such an active form of
decision support helps to improve healthcare because passive text guidelines are often ignored by medical personnel. They often think, unfounded, that their actions are conform the guidelines without checking this. De Clercq developed a new guideline representation formalism based on ontologies, primitives and problem solving methods on which he based the Gaston framework. [15]

The Gaston framework is the pivot in the anesthesia support system developed during this project.

![Figure 1, time line SPS research on patient safety improvements (Ph.D. projects)]

### 2.4 Medicast: clinical data mining and decision support systems

The Medicast project (2001-2005) aimed on the integration of data mining technology and knowledge-based systems. The participants in this project were the hospitals Catharina Ziekenhuis Eindhoven and Diakoneshuis Utrecht, the companies Dräger Medical Netherlands (hardware), INAD Medical Software, KiQ (data mining) and Medecs (medical decision support systems) and the Eindhoven University of Technology. They brought in € 5 million and were funded with another € 2.5 million from the Dutch Economic Affairs department.

The integration of data mining and expert systems was expected to lead to revolutionary results. The combination of more and more information sources (laboratory data, patient monitoring data, formalized knowledge) will increase the number of elements that can be controlled in quality care. This increases the quality of knowledge-based systems and decreases the number of (avoidable) medical errors.

Several Ph.D. projects at the TUE were carried out within the Medicast project. This resulted in new knowledge and technology on the subject of medical decision support and data mining. The graduation project described in this thesis aims on using this technology to develop a system for practical use in the field of anesthesia.
3 Technical framework

This chapter describes the software components that were used to build the system developed in this project. A general description of the components is given as well as a more in depth analysis of the parts that play a major role in the complete system and what this role is. In this chapter no details on how the system was actually implemented are given. These implementation details can be found in the next chapter.

3.1 Data acquisition: Vierkleurenpen and Chipsoft EZIS

The data that is used to generate advice using the decision support system is taken from the databases of two applications: Vierkleurenpen and CS-EZIS.

3.1.1 Vierkleurenpen

Vierkleurenpen (4kp) is a (partly) automated anesthesia recording application. It is designed and implemented by dr. Leo van Wolfswinkel from the Universitair Medisch Centrum (UMC) Utrecht.

The application uses a MS SQL Server database to store the data about the anesthesia cases. Through a server application the program acquires physiological data about the patient under anesthesia from the patient monitor (for example a Datex Anesthesia Monitor). A client application with a graphical user interface (GUI) shows these physiological data in real time. This real time access to the data is a main advantage of 4kp when compared to other (commercial) anesthesia recording applications. This GUI can also be used by the anesthetist or an anesthetic nurse to annotate the registered physiological data. The administration of medication, the connection of a pump or a drip or special events can be registered. All these annotation are shown in the GUI and in parallel stored in the database. A more detailed description of the database structure can be found in Appendix B.

In the system developed, the input function of Vierkleurenpen is replaced by a smart guided input protocol using a decision support system (DSS). This DSS stores the annotation data in the 4kp database directly. The Vierkleurenpen application still acquires the physiological data from the patient monitor and shows both physiological and annotated date (stored by the DSS) in real time.

3.1.2 CS-EZIS

The CS-EZIS system developed and exploited by Chipsoft, is a modular hospital information system. [16]

CS-EZIS is a workflow management system. It can be used to support patient focused treatment and registration processes. The software modules of the application can be divided roughly into three groups:

- modules for patient registration and logistics;
- modules for electronic patient data archiving and decision support;
- modules for financial and management information.

Information is stored in a database. In the system developed, the information from the patient registration and data archiving modules are of interest. The CS-EZIS
application itself is not used within the system. Only data from the database serves as input for the decision support system. The database is accessed read-only. This means that no data is changed in or added to the CS-EZIS database.

3.2 Decision support: Gaston

Gaston is a very powerful decision support system framework developed and implemented by dr. ir. Paul de Clercq from Eindhoven University of Technology and is now commercially exploited by De Clercq’s company Medecs BV. [15] The following text is taken from the report on a practical training project by Rico van de Vin. [17]

3.2.1 Decision support systems: a subclass of expert systems

Expert systems (or: knowledge-based systems) are systems that perform a certain task by applying rules of thumb on a symbolic representation of knowledge instead of by applying algorithmic or statistical methods. This definition of an expert system immediately shows the components of such a system:

1. a symbolic representation of knowledge (in a specific domain). This is called the ontology of the system;
2. a set of rules and guidelines (which can be applied to the knowledge);
3. a mechanism for reasoning with the knowledge.

A decision support system (DSS) is a system that supports people (or other applications) in making a decision. The main advantage of a DSS is that it can reason more consistent and more complete than a human’s brain. A DSS evaluates the specified guidelines and advises the user. In the end, the user can follow the advice or overrule it. A DSS is a support system and not a decision maker.

3.2.2 Gaston applications

Gaston consists of several applications. Gaston Execution Time Kernel (GASTET) takes care of the reasoning part at runtime. It applies user-specified (and user-created) rules and guidelines on a user-defined ontology. Both ontology and guidelines can be developed without the need for the user to know anything about knowledge-based systems. In order to do so two applications are available: Gaston Terminology Editor (or: Domain Ontology Editor, DOE Edit) and Gaston Knowledge Base Editor (GASTAT). These two are combined in an application called KATool. Both applications have a user-friendly GUI which hides all the difficult expert system details for the user. See Figure 2.
In the terminology editor the user can define the ontology for the desired (medical) domain. The user can create items (e.g., Blood) and add attributes to these items (e.g., SystolicPressure and Pulse). The items can be linked to a table in a database in order to enable access to data. The attributes can be linked to fields in the table associated with the item.

GASTAT can be used to develop guidelines using the ontology defined with DOEdit. Guidelines are graphically presented as flow charts. The building blocks for the flow charts are called primitives (e.g., Decision Step, Generate Message and Calculation). First the user has to define the structure of the flow chart (guideline). This can be done by dragging primitives from a library to the working surface. The primitives can be connected to each other in order to indicate the direction of the flow. Once the structure of the guideline is defined, the user can specify the contents of the primitives. For example for a Decision Step the attributes on which the decision should be based and the conditions which these attribute should meet can be defined (e.g., “Blood: SystolicPressure is less than 100” or “Blood: Pulse is more than 80”). The attributes can be dragged from a library into the primitive content editor and then a condition can be specified. Multiple conditions can be specified for one primitive (e.g., “Blood: SystolicPressure is less than 100” AND “Blood: Pulse is more than 80”).

After all guidelines have been developed the desired guidelines can be exported to an application. The guidelines will be stored in a format that can be read by GASTET on a location that can be found by GASTET. At runtime the guidelines are executed by the Execution Time Kernel and advices are generated and displayed.

### 3.3 Simulation: Windows application developed using Borland Delphi

The Borland Delphi 2006 programming environment was used to write the Patient Simulator application and compile it into a Microsoft Windows 32-bit executable. The Delphi environment has a large amount of build in components which ease the programming of graphical user interfaces and adding communication capabilities (e.g.
communication with other application or communication with databases) to an application.

3.3.1 Windows Messages
The Patient Simulation application has to communicate with both the 4kp client application and the 4kp MS SQL Server database. The simulator has to send the physiological parameter data to the 4kp client using Windows Messages (the build in message system of the Windows operating systems). For this purpose a build in Delphi function called `postmessage` is used. This function posts a message to the message system addressed to a certain window (in this case the 4kp client window). The message consists of a message id and an actual message (consisting of two components). The message id cannot be chosen completely free. Some ranges are reserved to be used by the operating system. [18]

3.3.2 ADO
The database communication can be handled using Microsoft’s ActiveX Data Objects (ADO). The Delphi programming environment has a library of components that can be used for this purpose. The communication of the Patient Simulator application with the database is one-way (read-only) and can be established using two components: `TADOConnection` (defines the connection with the database) and `TADOQuery` (defines the SQL query that is used to acquire data from the database).

3.3.3 XML
Delphi has also build in functionality to parse eXtensible Markup Language (xml) files. This markup language is developed to ease the interchange of data between applications. Because it is a plain text format it is platform independent. It can be used to structure texts, giving meaning to every part of the text. It uses tags to divide the text in parts. This makes the format relatively easy to be read by humans. Delphi has the `TXMLDocument` component to easily handle xml files. Because of these properties it would be a good choice to store the case description files for the Patient Simulator in this format.

3.4 Speech recognition: Dragon Naturally Speaking (DNS)
As a special feature the user interface of the decision support system is designed for use with speech recognition and touch screens. For the speech recognition part the software application Dragon Naturally Speaking from Nuance is used. This software analyzes the words spoken into a microphone and converts it into key strokes (when the cursor is in a text area) or mouse clicks (when special speech commands are given). In this way buttons and similar screen elements can be clicked or text can be typed. In fact the complete computer can be controlled using voice.
4 Major implementation issues

This chapter gives an overview of how the system was implemented in the context of the technical framework described in the previous chapter. Using flow charts, pseudo code and images, the major implementation issues and decisions are explained. Details about the implementation such as the full source code of the Patient Simulator application can be found in the appendices.

4.1 Vierkleurenpen

The Vierkleurenpen (4kp) anesthesia registration software was adapted in some manners to better fit in the developed system. These changes were made by the creator of the 4kp software dr. Leo van Wolswinkel from Universitair Medisch Centrum (UMC) Utrecht. This section gives an overview of the functionality that was added to or changed in 4kp.

4kp Has an internal simulation mode that can be used to simulate certain values for physiological parameters. To be able to run a simulation using an external application 4kp was extended with the possibility to control the simulation mode using Windows Messages. As described in the previous chapter, a Windows Message consists of an id and two message components (WParam and LParam). The id for a parameter value message in 4kp is wm_user+13. WParam is used to send the parameter id and LParam to send the new value for the corresponding physiological parameter. See the section about the Patient Simulator application for more information about how this functionality is used in an external application.

The standard storage resolution of 4kp is 1/60 per second (one sample per minute). De anesthesia monitor is sampled every 5 seconds. From the 12 samples per minute the median is taken and displayed on the screen and stored in the database. Storage per 5 seconds demands a lot of storage space, displaying per 5 seconds gives a shivery line. However for simulation purposes it would be nice to have more samples per second available in the database. Especially for simulation script debugging and demonstration purposes it can be nice to be able to run a simulation faster than real-time. Therefore the simulation mode in 4kp was adjusted in such a way that the data storage resolution can be changed to 1/30 per second (two samples per minute) or 1/12 per second (five samples per minute).

During the development of the system it appeared that data written in the 4kp database using an external application (in this case the decision support system) was not displayed on the 4kp screen. Because the system has to be used in real-time mode this however is a necessary condition. Therefore 4kp was extended with the possibility to update its screen on demand using the data in the database. Two methods are now available for this purpose. One method is broadcasting a Windows Message with id wm_user+15 to the Vierkleurenpen application. The other method uses the internal message system of 4kp and comes down on adding a row to a table in the 4kp database. For the developed application this last one is the most useful method. More details about this can be read in Appendix B.

Another problem that was discovered during development is that data written in the 4kp database table tb_case_descriptor by another application was overwritten by 4kp.
on the next screen update. This problem is solved. Now first the database data is read and when necessary and possible completed with information from the application memory.

4.2 CS-EZIS database

The CS-EZIS database is used in read-only mode only. This means that no changes are made to database itself or the data in it. The database is coupled to the system by means of the ontology of the decision support system in the way described in the next chapter.

4.3 Decision Support System

Some prework for the creation of a Decision Support System (DSS) based on the Gaston framework was done. After that Maarten van Dorst and Niek Lips from Fontys Hogeschool Eindhoven were conducted when they implemented the DSS based on this prework. Most of the text in this section is based on their work. Only a brief overview of the implementation is given. Details can be found in the report of Maarten van Dorst and Niek Lips. [19]

4.3.1 Coupling with databases

The coupling with a database in Gaston is performed in two steps. The first step is writing SQL queries. These queries are used to acquire data (using SELECT statements) from or write data (using INSERT or UPDATE statements) to the database on runtime. With this data some kind of internal database is created in the working memory of the DSS. This internal database generally uses other table and field names than the original database. In the second step these internal database table and field names are mapped to ontology items using the ontology editor of Gaston. An advantage of this two step construction is that when another database is used holding more or less the same types of information, only the queries have to be changed to restore the coupling. In general the queries used look like this:

Acquiring data:

```
SELECT FROM table_name
  old_field_name1 AS new_field_name1,
  old_field_name2 AS new_field_name2,
  ...
  old_field_nameN AS new_field_nameN
WHERE
  id_field_name = id_field_value;
```

Inserting a new record:

```
INSERT INTO table_name
  (field_name1, field_name2, ..., field_nameN)
VALUES
  (field_value1, field_value2, ..., field_valueN);
```

Modifying an existing record:

```
UPDATE table_name SET
  field_name1 = field_value1,
  field_name2 = field_value2,
  ...
  field_nameN = field_valueN
WHERE
  id_field_name = id_field_value;
```
Sometimes a query generates an error when the data type of a certain field value is inconsistent with the data type of the corresponding field in the table. Therefore it can be useful to explicitly typecast the field values in the queries. When for example field 1 is an integer and field 2 is a string the corresponding values are type casted as follows:

```csharp
field_value1.AsInteger
field_value2.AsString
```

Other often used data types are floats (`AsFloat`) and datetimes (`AsDateTime`).

### 4.3.2 Work flow

As mentioned in the previous chapter, guidelines for the DSS are created as flow charts. In this sub section a brief overview of the main chart of the developed system is given. The details about the implementation of the other (sub) charts can be found in the report of Niek Lips and Maarten van Dorst [19]. The main advantage of the use of Gaston is that the implementation can be changed or extended to fit the needs of the department where the system is used. The main chart (version of April 25th 2008) is displayed in Figure 3.

As a first step the planning for the current operating room is acquired from the CS-EZIS and 4kp databases and displayed in a popup window. The user can select the appropriate patient for the current case. The patient information for this selected patient is acquired from the EZIS database and stored in the 4kp database. All this is done in the sub chart represented by the block ‘planning’ in the figure. When no planning is available or when the patient is not on the planning it is checked whether the patient identification number is in the 4kp database. If not a popup window appears asking for this number and the number entered is stored in the 4kp database (‘IP_invoer’ block). Then the Vierkleurenpen screen is updated (‘refresh block’) to show the patient name and identification number on top of this screen.

Before the start of the induction preoperative data is acquired from the CS-EZIS database. This data can hold information that is important to determine the type of anesthesia that should be used and the doses of the medications involved (‘PreOp’ block). The ‘Inleiding’ block guides the user through the induction procedure. Based on the preoperative information and the type of anesthesia selected, medications and doses for these medications are suggested to the user. When the user confirms the medications and doses these are registered in the 4kp database.

In a similar way the user is guided through the complete anesthesia process. More features can be added at every point in the process. All steps can be registered in the 4kp database after confirmation by the user. Actions commonly performed together can be grouped and group wise registered. In this way the registration procedures can be simplified and the anesthetist can concentrate on the patient instead of on the registration. And even more important: the anesthesia process can be guarded and the user can be warned when mistakes are about to be made.
4.4 Patient Simulator

As described in the previous chapter, the Patient Simulator is build from scratch using the Borland Delphi programming environment. In the previous chapter an overview can be found of the built-in components that are used for example to communicate with Vierkleurenpen. This section of this chapter shows the work flow of the application and the routines that are used. (In pseudo code, a complete source code listing can be found in Appendix A.)
A simulation script is a collection of states, actions to be performed when a state becomes the active state and transitions between these states. Figure 4 shows a small part of the state diagram of a simulation script. When the increased_heartrate state is entered two actions are performed: temp1 and temp 2 are set to 3950 and 3850 respectively in 6 and 4 minutes. When heartrate reaches the value 110, increased heartrate becomes the active state. When this state becomes active no actions have to be performed. When meanabp reaches the value 165 in the increased_heartrate state, increased_hr_abp becomes the active state. In this state also no actions have to be performed.

**Figure 4, part of the state diagram of a simulation script**

**4.4.1 Work flow**

When the Patient Simulator application is started and a case description file is selected, the xml is parsed and an object of type TCase is created in working memory. This object contains the following attributes

- the name of the case,
- a description of the case,
- the state in which the case starts,
- an array with n objects of type TState, containing the following attributes
  - the name of the state,
  - an array with n_i events that have to be applied when the state becomes active,
  - a Boolean indicating whether or not this state is the active state,
  - an indicator for the time passed in since this state became the active state,
  - an array with n_j objects of type TTransition, containing the following attributes
    - a parameter name,
    - a comparison operator (“condition”),
    - a threshold value for the parameter,
    - the next state.
The TTransition object also contains a method Go() that should be called when the transition condition is met.
The TState object contains a method EnterState(FromState: integer) that should be called when this state becomes the active state.
The TCase object contains a method StateIndex() that return the index of the active state.
These three methods are described in pseudo code in the next part of this section.

After that the TCase object is created, a list is created with all medication that can be administered to the simulated patient. The list is read either from an xml file or from the Vierkleurenpen database. An array of objects of type TMedication is created in working memory. Each object has the following attributes

- an unique id for this medication item,
- the name of this medication item,
- the type of medication (administered at once, pump or drip),
- the unit in which this medication item is entered,
- a float containing the current dose of this medication item.

The same holds for the events that can be applied to the simulated patient. These can also be read from an xml file or from the Vierkleurenpen database. An array of object of type TEvent is created in working memory (the choice of names is somewhat unfortunate with concern to the Events attribute of the TState object). Each object of type TEvent has the following attributes

- a unique id for this event,
- the name of this event,
- a Boolean indicating whether or not the event is active (or has already been applied to the simulated patient).

Now the application is ready to start the simulation. This is done by making the default state of this case the active state. If there are events attached to this state these events are applied. Then a timer is started. Every t minutes (default is every minute) this timer triggers the following cycle

- for every of the 16 physiological parameter known by the application (see the appendices) it is checked whether or not the value has to be updated, if so this update is performed. The application keeps track of the updates to be performed in a 16x2 two-dimensional array ParameterUpdates. For every index in the first dimension (i.e. for every parameter) the first position in the second dimension holds the step size for the update and the second position holds the end value. The parameter value is changed with amount step size every cycle until the end value is reached. When the end value is reached the step size is set to 0 indicating that no update is necessary for the corresponding parameter. Every time a parameter value is changed a Windows Message is broadcasted to update the parameter value in Vierkleurenpen too;
- for every transition attached to the active state it is checked whether or not the condition for the transition is met, the first transition in the list that meets its condition is performed.
When a transition has to be performed the CurrentState attribute of the current state is set to false and the CurrentState attribute of the next state is set to true. Then the events attached to this next state are applied if there are any. For every event the following steps are performed:

- the event string is parsed;
- it is checked whether the event has to be performed at once or over a certain time span;
- if the event has to be performed at once it is performed right away;
- if the event has to be performed over a certain time span the step size is calculated (equal steps over time), the concerning parameter is added to the update list (step size and end value) and the first step is performed.

This cycle is looped over and over again until the user of the Patient Simulator application clicks the ‘stop’ or the ‘restart’ menu option. Remark: the simulation does not automatically stop when a final state of the case is reached!

When the ‘stop’ menu option is clicked the timer is stopped and the TimeInCase parameter (holding the time passed since the case was started) is set to 0. For all 16 physiological parameters its values are set to data_invalid (this simulates that that patient is disconnected from the patient monitor). The working memory is cleared (the TCase object is destroyed).

When the ‘restart’ menu option is clicked the same actions as with the ‘stop’ option are performed but then the simulation is started again from the beginning.

During the simulation medication can be administered and events can be applied to the patient in order to improve the patient’s condition (i.e. change the state). When the ‘Medication and Events’ menu option is clicked the graphical user interface of the Patient Simulator application is shown.

When a medication is selected from the ‘Medication’ list box, the current dose is shown in the upper right of the window. When a dose is entered in the textbox and the button ‘Administer’ is clicked, the dose entered is added to the current dose (the Dose attribute of the corresponding TMedication object is increased with the value entered).

When a pump is selected from the ‘Pump’ list box, the current stand of the pump is shown. When the stand is changed and the ‘Change Pump’ button is clicked, the pump stand (Dose attribute of the corresponding TMedication object) is changed to the new value.

When a drip is selected from the ‘Drip’ list box, the current volume of the drip is shown. When the volume is changed and the ‘Change Drip’ button is clicked, the drip volume (Dose attribute of the corresponding TMedication object) is changed to the new value.

When an event is selected from the ‘Event’ list box, the caption of the button below the list box is changed according to the state of the selected event. If the event is active the caption is changed to ‘Deactivate’. When the button is clicked the selected event is deactivated (i.e. the Active attribute of the corresponding TEvent object is...
set to false). If the event is not active the caption is ‘Activate’. When the button is clicked the selected event is activated (i.e. the Active attribute of the corresponding TEvent object is set to true).

4.4.2 Simulation routines

This part of this section describes the routines that perform the functionality of the Patient Simulator application as it is described in the previous part. It is not a complete source code list (this can be found in the appendices) but the most important routines are described using pseudo code.

The first routine parses an xml case description file into an object of type TCase. This routine is called ParseDescription and takes a filename as input parameter. It is a Delphi procedure so it does not have a return value.

```delphi
procedure TPatient.ParseDescription(CaseFileName: TFileName);
begin
  try
    for all child nodes of the root element do begin
      if node is information element then begin
        for all child nodes of the information element do begin
          copy each values to the corresponding attribute of the case object;
        end;
      end;
      if node is state element then begin
        create temporary state object
        for all child nodes of the state element do begin
          if node is name element then begin
            copy value to the name attribute of the temporary state object
          end
          else if node is event, sign or msg element then begin
            add an element to the events attribute of the temporary state object and copy the node value to that element
          end
          else if node is transition element then begin
            create a temporary transition object
            find the parameter name in the node value and copy to the parameter attribute of the temporary transition object
            find the (end) value for the parameter in the node value and copy to the value attribute of the temporary transition object
            find the next state in the node value and
        end;
      end;
    end;
  end;
end;
```
copy to the next state attribute of the temporary transition object

add an element to the transitions attribute of the temporary state object and copy the temporary transition object to that element end;
end;

set the CurrentState and TimeInState attributes of the temporary state object to the default values

add an element to the states attribute of the case object and copy the temporary state object to that element end;
end;
finally ...
end;
end;

The routines ParseMedicationList and ParseEventList are much alike. Therefore they are not shown here completely. In both routines a temporary object is created, values are copied from the xml file that is parsed and the temporary object is added to the medication or event object list. A major difference with the ParseDescription routine is that in the ParseMedicationList and ParseEventList routines there is also the functionality to get information from a database instead of from an xml file. The following piece of pseudo code shows how this is done in ParseMedicationList. A similar subroutine is used in ParseEventList.

compose the query to select medication from the Vierkleurenpen database

with ADOQuery1 do
begin
First;
while not Eof do
begin
 create a temporary medication object and copy the information from the database to the corresponding attributes of the temporary medication object

 add an element to the medication list and copy the temporary medication object to this element

 Next;
end;
end;

repeat for pumps and drips

The most important routines in the Patient Simulator application are the ones that perform the actual simulation. A flow chart for the simulation is shown in Figure 5. These are the Timer1Timer routine, the EnterState method of the TState object and the Go method of the TTransition object.
procedure TPatient.Timer1Timer(Sender: TObject);
begin
  update the time passed in this case
  update TimeInState for current state

  for all 16 physiological parameters do
  begin
    if an update is scheduled then
    begin
      perform the update
      if end value is reached then
      begin
        set parameter to end value
        remove parameter from update list
      end;
    end;
  end;

  for all possible transitions in this state do
  begin
    if parameter is one of the 16 physiological parameters then
    begin
      if condition is met then
      begin
        call the Go() method of this transition
        Break;
      end;
    end
    else if parameter is an event then
    begin
      if condition is met then
      begin
        call the Go() method for this transition
        Break;
      end;
    end
    else if parameter is TimeInState then
    begin
      if maximum time in state has passed then
      begin
        call the Go() method for this transition
        Break;
      end;
    end
    else
    begin
      if medication dose condition is met then
      begin
        call the Go() method for this transition
        Break;
      end;
    end;
  end;

procedure TTransition.Go();
begin
  set CurrentState and TimeInState attributes of current state
  object to false and 0 respectively

  for all states in this case do
begin
if this state is the NextState of the current state then
begin
make this state the current state
    call the EnterState() method for this state
    Exit;
end;
end;
end;

procedure TState.EnterState(FromState: integer);
begin
    for all events in this state do
    begin
        if sign then
        begin
            add this sign to the list of signs
            show a popup message with this sign
        end
        else message then
        begin
            show a popup with this message
        end
        else
        begin
            find parameter name in event string
            find parameter value in event string
            if 'over' is in the event string then
            begin
                find time span in event string
                calculate step size using current parameter value,
                end value and time span
                add step size to current value
                place parameter on update list
            end
            else
            begin
                change current parameter value to parameter value
                from event string
            end;
        end;
    end;
end;

The Patient Simulator application is designed to be used in combination with Vierkleurenpen. Every time a physiological parameter value is changed in the simulator a Windows Message is broadcasted to the Vierkleurenpen application in order to perform this update in 4kp as well. The first component of the message (WParam) contains the id for the physiological parameter, the second component (LParam) contains the new value for that parameter:

PostMessage(FindWindow('TVierkleur', nil), wm_user + 13, Parameter, Value);

These were the most important routines in the application. The other routines are mostly tools to perform simpler subtasks within the routines shown above. The
implementations of these simpler routines speak for themselves. The reader is referred to the complete source code listing in Appendix A to see these routines.

**Figure 5, simulation flow chart**

### 4.4.3 Case description file debugging and fine tuning

Whether a simulation is performed correctly depends on the case description xml file. During a simulation in normal mode it is sometimes difficult to see whether or not the simulator acts like the creator of the description file has planned. Therefore the Patient Simulator application has some case description file debug options build in. For every simulation a log file is created. In this log file every action of the simulator is logged with a time stamp. This log file can be used to debug or fine tune the case description afterwards. It is also possible to show every log entry during the simulation. When the application is running in debug mode every line that is added to the log file is also shown as a popup message. The routines that are used to add lines to the log file and show popup messages are straightforward thus they are not shown here.
5 Results
In this chapter the results of this project are illustrated using some example screen shots. As will be described in the next chapter some more work is necessary before the system can be taken into use safely. However the advice system currently runs in a test setup that simulates a hospital network and there are some basic sample case scripts available for the patient simulator.

The resulting system can be extend with more guidelines and protocols. The Gaston framework enables people without a technical background to build in these guidelines them selves. With this system theoretical research can be brought into clinical practice in order to improve patient safety in anesthesia.

5.1 Advice system proof-of-concept
Maarten van Dorst and Niek Lips [19] put their effort in setting up a proof-of-concept for an advice system that combines knowledge from the anesthesia database (4kp) and the general hospital database (CS-EZIS) with general rules and guidelines about anesthesia. This resulted in a test setup as will be described in this section.

The test setup consists of one server pc (an HP laptop) and three clients (Sevo touch screen pc’s). The server pc runs several server applications. MS SQL Server manages the database connections with a sample Vierkleurenpen database and a sample CS-EZIS database. This simulates the hospital data storage environment. Also a Gaston server runs on the server pc. In this way there is a central place where all guidelines and protocols coincide and which can be accessed from every pc in the network over the intranet. For this purpose an IIS (Internet Information Services) web server is running on the server pc. This web server is integrated in the MS (Microsoft) Windows operating systems. The web version of Gaston uses the Microsoft scripting language Active Server Pages (ASP) to create dynamic web pages (HTML). It also uses Microsoft’s .NET framework.

Figure 6, test setup advice system, an HP laptop serves as a server and three touch screen pc’s serve as clients
On every client a copy of Vierkleurenpen is installed as well as a copy of the Patient Simulator. These applications connect to the databases on the server over a simple network using a basic router. The clients also have a copy of the MS Internet Explorer internet browser installed. Using this browser the Gaston guidelines and protocols can be accessed over the intranet. See Figure 6 for a schematic overview of the test setup. Figure 7 shows some screenshots of the advice system.

Figure 7, some screenshots from the advice system (clockwise starting upper left: preoperative information, latest laboratory results, induction medication and tube setting)

5.2 Practical examples

In this section some examples of how the system can be used are shown. The system enables medical staff to use the outcomes of earlier research in the daily practice of their work. The first is an example of protocol guidance in case of a seldom occurring complication, malignant hyperthermia. The second is an example of how general knowledge about the patient can be used to better dose agents in anesthesia. The third example is about the implementation of advanced models in the advice system in
order to assist the physicians in making decisions. The first example is really implemented in the system and in the simulator. The other two are only examples and not yet functioning.

5.2.1 Malignant Hyperthermia (MH) protocol guidance

One of the goals of this project was to create a training environment to train protocols for seldom occurring complications in anesthesia. An example of such a complication is malignant hyperthermia (MH). MH is a rare life-threatening condition triggered by exposure to certain agents (drugs) used in general anesthetics. Only very few medical professionals are confronted with a case of MH during their active work lives. However when MH is not treated accurately the risk that the patient dies is severe. So it is of major importance that the medical staff can act fast and accurate when a case of MH occurs.

Signs of MH are, amongst others, increased heart rate, increased blood pressure and increased body temperature (may be late sign). The treatment protocol implemented in the advice system is taken from the MHAUS (Malignant Hyperthermia Association of the United States) treatment poster. [20]

A case script for the simulator simulates a MH by increasing certain physiological parameters. When the advice systems recognizes a severe increase in heart rate and blood pressure a warning is generated stating that there is a possible occurrence of MH. The user can choose to ignore the message, treat the MH by himself or treat the MH supported by the system. In that last case the system guides the user through the MH treatment protocol step-by-step. The steps can be performed in the simulator and the simulator adapts its current state accordingly. Vierkleurenpen registers the changes in physiological parameter values in its database. The advice system reacts on these new data by adapting the treatments steps displayed on the user’s screen. In clinical practice the place of the simulator is taken by the patient and the anesthesia monitor.

The figures on the next two pages (Figure 8 - Figure 11) show some steps in such a simulation. On the background of Figure 8 the Vierkleurenpen charts show an increasing heart rate and blood pressure. This is registered in the database and detected by the advice system. The advice system shows a warning (possible MH) and asks the user whether or not he wants to be guided during treatment. In Figure 9 an advice window with treatment steps is shown. The first step is to hyperventilate with 100% oxygen. Figure 10 shows how this is done in the simulator. Important is that the events done in the simulator must also be registered in Vierkleurenpen to be detected by the advice system (this is also the case in a situation with a real patient). Once the advice system detects that the hyperventilation step is carried out, it leaves out this step in the treatment advice (Figure 11). In this way the user is guided through the treatment protocol.
Figure 8, the advice system recognizes an increase in heart rate (red line) and blood pressure (purple dotted line) and shows a message that there is possible occurrence of MH.

Figure 9, the advice system can support the user during treatment of MH.
Figure 10, in the simulator one can activate events or administer medication.

Figure 11, after registration of an event in Vierkleurenpen the advice system shows the next step in the treatment protocol.
5.2.2 Adapting sevoflurane concentration to patient’s age and desired MAC

Sevoflurane (an anesthetic agent) is often overdosed in clinical practice. Not only is the patient unnecessarily exposed to high doses of this agent but sevoflurane is also very expensive. It is however difficult to rapidly estimate the end-expired concentration of volatile agent needed to obtain a given total MAC multiple. MAC is the minimum alveolar concentration at 1 atmosphere that prevents movements in 50% of patients exposed to surgical incision and is well established as a measure of anesthetic potency. MAC decreases with increasing age. In 2004 J. Lerou from the Radboud University Nijmegen published a nomogram that can be used for the purpose of MAC multiple calculations or, the other way around, end-expired agent concentration calculations. [21] This nomogram can be implemented in the advice system and the anesthetist can get an advice every now and then about how to maintain an optimal concentration of sevoflurane. This can also be done for other anesthetic agents.

Figure 12, nomogram relating age, total MAC and end-expired concentrations of volatile agent and nitrous oxide (figure taken from [21])

Figure 12 shows the nomogram of Lerou with an example of how to detect the sevoflurane concentration in 67% nitrous oxide for a 3-year old to obtain a total MAC of 1.3 MAC units. The dotted lines show that the concentration of sevoflurane should be 1.8% (at 1 atmosphere). Extra agents and other concentrations of nitrous oxide (or xenon) can be added.

The end-expired concentrations of sevoflurane and nitrous oxide are measured by the anaesthesia monitor and can be registered by Vierkleurenpen (remark: the simulator can simulate these values as well!). A guideline can be created in the advice system that uses these values from the 4kp database, the age of the patient (which is stored in the EZIS database) and the desired MAC to give advice about the optimal sevoflurane concentration.
Figure 13 shows a screenshot of how such an advice can look like. On the background one can see the charts of Vierkleurenpen. The charts show a current end-expired sevoflurane concentration of 2.0% (yellow line) and a nitrous oxygen concentration of 67% (blue dots which overlap with the red heart rate line in this figure). Based on these values and the patient’s age and the desired MAC multiple, the system suggests to decrease the sevoflurane concentration from 2.0% to 1.8%. On the top of the advice window (upper left) there is a link which can be clicked for background information. This link opens the Background information window (lower middle). This window shows more information about the advice. In this example it suggests to look at Lerou’s nomogram to verify the advice. The Background information window can have hyperlinks, for example to an image of Lerou’s nomogram. In that way the user can access this information really fast.

Figure 13, a screenshot of how the nomogram of Lerou can be used in clinical practice using the advice system developed in this project

5.2.3 Hypovolemia

Hypovolemia, the situation in which a patient’s blood volume is too small, is a threatening situation. The patient’s blood flow becomes too low, resulting in oxygen shortage in organs (shock). This can cause irreversible damage. A possible therapy is fluid loading. Blood or solutions are added to the circulating blood volume. However research showed that only 50% of the patient in intensive care respond positively to fluid loading. Fluid loading can result in hypervolemia in patients that are negative responders. Hypervolemia in turn can result in life-threatening conditions. [22]
Responsiveness to fluid loading mainly depends on the functioning of the heart. This is however difficult to measure. Measures are preload and stroke volume. Variations in preload result in variations in stroke volume. Which in turn result in variations in arterial blood pressure (ABP). ABP can be measured. But a clear indication of the functioning of the heart cannot be obtained from only the variations in ABP. [22]

In 2006 two students graduated within the SPS group on research to obtain models that can be used in estimating variations in preload. Sjoerd Diepen concentrated on a model for variations in blood pressure. [22] Emil Verheijen concentrated on variations in intrathoracic blood flows. [23] Their aim was to come with a system that has potential to predict a patient’s response to fluid loading. Some further research is necessary to complete their work, but a prediction system like the one Diepen and Verheijen aimed for can be integrated into the advice system presented in this thesis. In that way, if hypovolemia occurs during a case, the advice system can discourage the use of fluid loading as a therapy if the patient is aspected to respond in a negative way based on the models predictions.

5.3 SPS Ph.D. research

The results of former SPS Ph.D. research, as presented in Chapter 2.3, can be brought to clinical practice by integration in the advice system. Build on the Gaston framework of De Clercq [15], the system is a descendant of Blom’s SIMPLEXIS [8]. Due to the combination of information sources (databases) and the integration of domain knowledge (protocols and guidelines) the system is a intelligent patient monitoring system like Meijler already suggested in 1986 [7]. Information about the patient and the type of operation comes available from the EZIS database. This information can be used to automatically adapt the thresholds for advices and warnings to the specific situation of the running case. This resembles the results of the Ph.D. project of Van Oostrom [10].

Theoretically it should be possible to implement anesthetic depth measurements based on EEG activity like suggested by De Beer [11] and Van Hooff [12] by adding new information sources (EEG measurements and annotations about the time points where stimuli were applied). But it might be necessary to investigate the performance of the knowledge-based system on high frequency data like EEG signals first.
6 Conclusions and recommendations

As a result of this project and the project of Maarten van Dorst and Niek Lips [19] there is a proof-of-concept available for a powerful advice and support system for use in the operating room. The system combines knowledge about both the pre-operative and the intra-operative condition of the patient with general rules and guidelines from the field of anesthesia. Due to the use of speech and touch screen for input tasks, the registration of events during anesthesia is less time consuming. The use of some form of templates for common anesthesia tasks makes the registration process easier and at the same time more complete.

With the development of the patient simulator, a useful tool is created for training medical personnel in the use of the advice and support system before they use the system during an actual operation. In this way the staff can get used to the user interface of the system in a controlled and stress less environment. The simulator can also be used to practice the protocols for seldom occurring complications such as malignant hyperthermia.

The results of the project are very satisfactory and there are positive critiques from the field. However there are some recommendations for future work in order to further improve the system. As Niek Lips and Maarten van Dorst suggest in their report [19] the medical contents of the knowledge base should be increased before the system is taken into use in daily practice. Furthermore some research has to be done on how to integrate the system safely into the hospital network. All network traffic in the hospital network is routed by a communication server (or ‘message broker’) called Cloverleaf. For database communication the data protocol Health Level 7 (HL7) is used in the Catharina Ziekenhuis Eindhoven (CZE). They also, for good reasons, suggest that the system should be under development continuously since the field of anesthesia, and other medical fields, are changing continuously too.

In addition to these recommendations it might be useful to pay some attention to the use of the system in daily practice. When it comes to the use of speech there are some aspects that should not be overseen when the system is introduced. For example the medical staff in an operating room wear masks for hygiene reasons. Such a mask deforms the characteristics of one’s voice. This should be taken into account when the speech recognition application is trained to interpret the voice of a staff member since the characteristics of his or her voice play a major role in speech recognition. For the same reason problems may occur when a staff member has a cold. Although the system is designed in such a way that it can be controlled with a keyboard when speech recognition is failing, the integration of speech is one of the key features of the system. Due to recent advances in the field of speech recognition and hardware improvements (selective microphones) the problems can be solved. Therefore it will be worth putting effort into finding a solid and stable speech recognition solution (both software and hardware).

This report contains no quantitative measures on the improvements that the advice system brings to patient safety in anesthesia. Therefore it is recommended to perform a study to the effects of the use of the system on patient safety once the system is taken into use. Such a study can be performed to research the effects of the use of the simulator on the knowledge of medical staff also.
With the system that is now presented a nice and solid base is delivered for advanced automated support in anesthesia. Starting from this base a lot of new applications can be thought of to further extend the system. In the next and final chapter of this report a start is made on the implementation and representation of abstract concepts like ‘increasing’ and ‘decreasing’. Once we are able to represent these concepts in the knowledge-based system, trend detection can be performed. Recommendations on this subject can be found in the next chapter.

Another interesting subject is data mining. In data mining large portions of data from a large number of cases are analyzed in order to find relations between events. Since the system presented has access to the data of a large number of patients and cases, it might be worth examining how this data can be used to find such relations. Once we are able to find relations between events, predictions can be made on possibly occurring complications. This will be a revolution in medicine because such a system will be able to prevent the occurrence of particular threatening complications. And that of course will be even more useful than early detection of complications.

The worked carried out in this project is a direct result from the Medicast project. The goal of that project was to combine predictive modeling with decision support in order to be able to predict possible occurring complications. A decision support system is realized and a basis is created for trend detection. This can be useful for data- and process mining techniques, which in turn will enable prediction.
7 Temporal abstraction and trend detection

From October 2003 till October 2004 Eleni Kokkinou [24] and Eva Triantafyllou [25] researched temporal aspects of knowledge based system as an extension of the work of Paul de Clecq [15]. Eleni Kokkinou designed a temporal querying system and Eva Triantafyllou developed a temporal ontology and implemented this into the Gaston framework. In their recommendations for future work they both suggest to put effort into the representation of temporal abstractions like ‘increasing’ and ‘decreasing’ and ‘the highest dose of a medication over the last week’.

To close the presented in this thesis, a brief literature study on this subject was performed. A summery of the methods found is presented in this chapter.

7.1 Temporal abstraction

Temporal abstraction (TA) is the abstraction of (for example) clinical data (often recorded as time-stamped measurements) into higher level concepts (holding over time periods). Shafar summarized the advantages of temporal abstractions in knowledge-based systems [26]:

- data summaries have an immediate value to an human user
- abstractions support monitoring of plans during execution
- meaningful time-oriented context enable the generation of context-specific abstractions, interpretation of the same data in different contexts and certain hindsight and foresight inferences
- temporal abstractions are helpful for explanation of recommended actions in intelligent systems
- they are a useful representations for process and outcome intentions of designers of guidelines and enable critiquing (real-time and retrospective) and measurement of the quality of the application of these guidelines by care providers
- effective visualization and dynamic exploration of time-oriented medical data by care providers require domain-specific, meaningful, interval-based characterization of that data. Visualization and exploration are essential for effective decision making.

In another publication Shahar mentions the problems of the temporal abstraction task [27]:

- input values may be of several data types and various abstraction levels (e.g. ‘heart rate = 80 b.p.m.’ and ‘pulse is HIGH’)
- input data may arrive out of temporal order
- several alternate interpretations might need to be maintained and followed over time
- parameters have domain-specific and context-specific temporal properties
- acquisition and maintenance of knowledge from domain-experts should be facilitated.

7.1.1 Knowledge-based Temporal Abstraction Method

Shahar developed the knowledge-based temporal abstraction (KBTA) method, which serves as a basis for methods developed later by other researchers. The KBTA method decomposes the temporal abstraction task into five subtasks. To solve these subtasks there are five domain-independent mechanisms that depend on four domain-specific
knowledge types: structural (e.g., IS-A and PART-OF relations and parameter properties such as measurement scale and units), classification (functional; e.g., classification into LOW, HIGH, VERY HIGH and classification of temporal patterns), temporal semantic (logical; e.g., the concatenable property allows to concatenate adjacent equal intervals) and temporal dynamic (probabilistic; e.g., minimal values of a significant change). [27] The five subtasks of the KBTA method are (see Figure 14):

- temporal context restriction (creation of context intervals from a set of input abstraction goal, parameter and event intervals)
- vertical temporal inference (creation of abstractions by inference from contemporaneous parameter propositions into abstract parameter propositions)
- horizontal temporal inference (creation of parameter intervals by performance of inference on parameter propositions of the same parameter and interpretation context)
- temporal interpolation (creation of parameter intervals by joining of disjoint parameter points or parameter intervals)
- temporal pattern matching (creation of abstraction intervals by matching of patterns over possibly disjoint parameter intervals, associated with parameter propositions of various parameters)

Figure 14, Knowledge-based temporal abstraction method (figure taken from [27])

The KBTA method and the five subtask methods assume, for the underlying model of the domain, that there is a task-specific temporal abstraction ontology. The KBTA theory defines the following set of entities in the ontology:

36
• time stamps: structures (eg. dates) that can be mapped into integer amounts of predefined temporal granularity units (eg. DAY)
• time intervals (ordered pair of time stamps indicating start and end of the interval) and time points (zero-length intervals where start and end time stamp are the same)
• interpretation contexts (define how parameter values should be interpreted within a given time interval depending on the context for that time interval)
• context intervals: structures consisting of an interpretation context and the time interval for which that context holds (eg. a chemotherapy {interpretation context} changes the way in which hematological parameters should be interpreted for several weeks {time interval})
• event propositions (represent the occurrence of an external volitional action or process (eg. the administration of a drug)
• event intervals: structures consisting of a event proposition and the time interval for which the proposition holds
• parameters
  o primitive parameters cannot be inferred from other parameters
  o abstract parameters
  o constant parameters have values that are not time-dependent
• abstraction functions: functions from one or more parameters to an abstract parameter of one of several abstraction types (at least the three abstraction types state, eg. LOW, gradient, eg. INCREASING, and rate, eg. FAST, exist)
• parameter intervals: structures consisting of a parameter, a parameter value and an interpretation index (together called a parameter proposition) and a time interval
• abstractions: parameter intervals for abstract parameters
• abstraction goals: goals or intensions that are relevant for the temporal abstraction task during certain periods (eg. diagnosis)
• abstraction goal intervals: structure consisting of an abstraction goal and a time interval
• inductions of context intervals (context intervals are induced by certain event, parameter and abstraction goal propositions)

The set of all the relevant events and propositions in the domain, their attributes, and their sub events forms the domain’s event ontology. The set of all the potentially relevant contexts and sub contexts of the domain defines a context ontology for the domain. The set of all the relevant parameters and parameter propositions in the domain and their properties forms the domain’s parameter ontology. These three ontologies, together with the set of abstraction goal propositions and the set of all inductions of context intervals, define the domain’s temporal abstraction ontology.

Shahar et al. implemented the KBTA method as a computer program called RÉSUMÉ. That system generates temporal abstractions, given time-stamped data, events and the TA ontology (parameters, events and contexts) of the domain.

7.1.2 Bellazzi’s TA method
In 2000 Bellazzi et al. presented an application of an intelligent data analysis method in the domain of diabetes mellitus. [28] Part of the method are temporal abstractions. Building on the work of Shahar et al. they decompose the TA task into two subtasks.
Basic TA abstracts time-stamped data into intervals and complex TA abstract intervals into other intervals. In basic TA they distinguish state abstractions (episodes associated with qualitative levels of time-varying variables) and trend abstractions (like increase, decrease and stationarity). See Figure 15.

The method they use is described in an earlier paper by Bellazzi. [29] They exploit the notion of the abstract state, a description of the patient’s behavior over a given time period consisting of a combination of temporal abstractions that are true in that period. The abstract state allows to describe the dynamics of the patient’s evolution as a sequence of episodes detected in the data.

---

Figure 15, decomposition of the temporal abstraction task (figure taken from [29])

Bellazzi et al. state that each TA mechanism requires setting up several parameters to give a complete specification of the episode in the specific context. For a basic TA these parameters are at least the granularity (maximum allowed gap between two measurements) and the minimum extent (minimum time span of an episode in order to be considered relevant). Other parameters are specific for each mechanism (eg. the slope parameter for the increasing trend detection mechanism).

A complex TA mechanism is introduced to be able to represent higher-level concepts. The mechanism searches for temporal relationships between episodes. For that purpose the Allen algebra (Figure 16) is used. Complex TA can be used to detect complex patterns in a single time series and to search for patterns in multiple time series. For example the situation where a persistent cough and high fever occur...
simultaneously in a patient can be detected by the temporal abstraction PERSISTENT COUGH OVERLAPS HIGH FEVER.

7.1.3 Multivariable Fuzzy Temporal Profile (MFTP) model

In 2003 Otero et al. proposed the Multivariable Fuzzy Temporal Profile (MFTP) model. [30] This model enables the identification of a pattern of special significance over a set of temporal evolving parameters. An MFTP is represented by means of a network of fuzzy constraints between a set of points.

The MFTP model is an extension of the Fuzzy Temporal Profile (FTP) model that is only suited to find a pattern in a single parameter time-series. The FTP model projects a fuzzy description of the temporal evolution of the parameter onto a fuzzy constraint network between a set of significant points (unknown parameter values on unknown points in time, which, in absence of a constraint, can take every set of precise values in the time-series). The constraints can limit the fuzzy temporal duration, the fuzzy increment and the fuzzy slope between a set of significant points. A FTP is defined as a finite set of significant points and a finite set of constraints between them.

The MFTP model extents the FTP model in such a way that temporal relations between patterns in different time-series can be represented. A MFTP can be decomposed into a set of FTPs and a set of constraints between them.

The aim of a MFTP model is to identify a pattern described by an expert. The solution to a MFTP assigns precise values to the set of significant points, which satisfy the set of constraints with a degree greater than zero. Matching a MFTP as a whole may have high computational complexity. Therefore Otero et al. imitate human experts and divide the matching in as many stages as the number of levels of abstraction (three: sampled signal, history of morphological events over each parameter and multivariable pattern occurrences). The disadvantage of this method is that local optimal solutions can be found that are not part of the global optimal solution.

7.1.4 Temporal abstraction in the VIE-VENT system

W. Horn et al. presented a paper about data validation in the VIE-VENT real-time monitoring and therapy-planning system. [31] Temporal abstraction is required for the usage of qualitative descriptions for data validation. The method used in the VIE-VENT system is comparable to the point temporal abstraction task of Shahar. Horn et al. define seven regions of interest for blood-gas values: extremely below, substantially below, slightly below, target range, slightly above, substantially above and extremely above. These regions are not equal in size. The value ranges of the regions are smaller the closer to the target range.

Schemata for data-point transformation are defined for all kinds of blood-gases and are clinical context dependent. This results in different numerical ranges for the same gas in different contexts. The schemata transform each valid numerical value into one of the seven qualitative categories. This temporal abstraction process is further enhanced by smoothing of data oscillating near thresholds (avoids rapid changes of qualitative categories by leaving out changes of region that only hold for a short period of time), smoothing of data-point transformation schemata (avoids sudden changes of schemata when the context changes by gradually changing from one
Interval data are transformed to qualitative descriptions of the change of the parameter over time (trends). Four kinds of trends are defined: very short, short, medium and long. They differ from each other in duration and validity criteria. For each kind of trend the actual growth rate and the derived qualitative trend category is determined. Qualitative steps are defined in terms of the qualitative point categories. These steps express experts’ expectations about how a parameter value should change over time. ‘Normal decrease’ can be expressed by for example ‘moving one qualitative step towards target rate in 20-30 minutes’. The growth rate is transformed into one of ten qualitative trend categories which are divided in an upper and a lower region by the target range. In the upper region the categories are ‘dangerous increase’, ‘zero change’, ‘decrease too slow’, ‘normal decrease’ and ‘decrease too fast’. In the lower region the categories are ‘increase too fast’, ‘normal increase’, ‘increase too slow’, ‘zero change’ and ‘dangerous decrease’.

7.1.5 Basic Temporal Abstraction and Temporal Relation Extraction

When data are collected irregularly and over a long period of time, the basic temporal abstraction techniques cannot be applied. Therefore a novel temporal abstraction method was developed at the Japan Advanced Institute of Science and Technology in Ishikawa. This method is called Temporal Relation Extraction and is published in 2007. The research focused on the mining of hepatitis data. [32]

The framework of the Temporal Relation Extraction method is as follows:

1. For each object (patient) find all possible significant abstracted temporal basic patterns on temporal intervals in the event sequence (sequence of outcomes) of each medical test.
2. Detect all significant temporal relations in terms of temporal logic between the temporal basic patterns found. Represent each object as a graph or a transaction of temporal relations.
3. Use data mining methods to find temporal rules from the collection of graphs or transactions.

The researchers divided the selected tests in short-term and long-term changed tests. For the short-term tests they defined the basic state patterns ’normal region’, ‘high’, ‘very high’, ‘extreme high’, ‘low’, ‘very low’ and ‘extreme low’. The situation where a value is suddenly much higher than its neighbors is called a ‘peak’. In this way it possible to represent for example ’<parameter> is in extremely high state with peaks’. For the long-term tests the temporal basic patterns characterize the change of states between two state regions. In this way it is possible to represent for example ’<parameter> changes from normal to high state’.

Temporal relations between temporal basic patterns are expressed in term of temporal logic. The researchers used the temporal relations as shown in Figure 16 below. Now it is possible to represent for example ’<parameter1> changed from normal to high after <parameter2> changed from normal to low’.
Earlier the same researchers presented a method for basic temporal abstraction on the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining in 2003. [33] With this method they created the basis for the Temporal Relations Extraction method. In this earlier publication they mention also trend patterns ('stable', 'increasing', 'fast increasing', 'decreasing' and 'fast decreasing'). These patterns and the state patterns (including 'peak') mentioned above were determined based on visual analysis of various data sequences.

### 7.2 Trend detection

In 2005 Melek et al. [34] published a comparison between different state of the art trend detection methods. These methods are:

- Fuzzy approaches
  - Fuzzy Logic Course Approach
  - Noise-Rejection Clustering
- Trigg’s Statistics-based Trend Detection Algorithm
- Regression and Temporal Shape Trend Analysis (Temporal Reasoning Approach)
- Wavelet-based Trend Detection Approach

Melek et al. implemented the methods in the Matlab 6 simulation environment and used them for trend detection in experimentally collected systolic blood pressure and heart rate signals and synthesized opacity signals. They conclude that the Temporal Reasoning Approach and Fuzzy approaches should best be used when high-frequency oscillations and noisy components are present. These methods however are slower and should better not be used when the speed of computation is critical. In such cases the Trigg’s Algorithm is recommended. This method detects the direction of trends (increase or decrease) fast, but is not able to detect the shape of a trend (concave down, concave up, or linear). If a signal is extremely noisy and only the underlying trend has to be identified the Wavelet-based Approach is recommended. However this method does not always succeed in detecting the shape of a trend and sometimes even fails to find the trend.

### 7.3 Discussion

This literature study on temporal abstractions and trend detection was not the main goal of this graduation project. Most effort has been put into the development of the advice system and the simulator and in writing down how the system works, can be
used and can be extended. Therefore the study in this chapter might not be complete. However some suggestions with respect to the use of temporal abstractions (TAs) in the developed system can be given.

The study shows that most (if not all) TA methods have more or less the same basis: Shahar’s knowledge-based temporal abstraction framework. The precise implementation of temporal abstraction mechanisms depends on the application domain. What can be seen from the systems that have been studied is that for every application domain and even for different applications within the same domain dedicated state patterns and dedicated trend patterns should be defined. These sets of patterns can overlap between different applications but it is important to focus on the goals of the application when the patterns are defined.

A desirable feature for the future is prediction. When the advice system can detect temporal patterns (trends) in the data of the current case and compare this with trends in earlier cases where a certain complication occurred, an estimate can be made on the chance that that complication is going to occur in the current case. In this way early detection of complications is possible. Maybe even before the complication really occurs.

The complications that can occur in an operating room are quite diverse. For future research it might be worth to look at the possibility to create a generic temporal abstraction ontology that can be applied to the advice system presented in this work. Based on this generic ontology various specific ontologies for specific complications can be derived. Furthermore it should be investigated which trend detection methods (temporal abstraction mechanisms) best fit to the analysis of data for each specific complication.

Thus a general ontology framework should be created (see the framework of Shahar) and the exact implementation should depend on the goals of the temporal abstraction task.

If, for example, the framework is projected on the malignant hyperthermia complication of Chapter 5.2.1, it is needed to represent increasing and decreasing trends in heart rate and blood pressure data, a ‘succinylcholine administered’ event and temporal relations between these temporal patterns (see the temporal algebra of Allen, Figure 16). The following (complex) temporal pattern could then be used to detect a possible occurrence of malignant hyperthermia:

HEARTRATE IS INCREASING OVERLAPS BLOODPRESSURE IS INCREASING AFTER SUCCINYLCHOLINE_ADMINISTERED IS TRUE

Summarizing this, this (bit shallow) literature study can be taken as a basis for a more thorough research to a temporal abstraction ontology focused on the domain of anesthesia and the complications that can occur during anesthesia. For that purpose a close cooperation between technicians and the medical field might be necessary.
8 Literature


Malignant Hyperthermia treatment poster, Malignant Hyperthermia Association of the United States, www.mhaus.org


S. Diepen, Modeling respiratory induced variations in blood pressure, Graduation Thesis, Eindhoven University of Technology, 2006

E. Verheijen, Modelling respiratory induced variations in intrathoracic blood flows, Graduation Thesis, Eindhoven University of Technology, 2006

E. Kokkinou, Querying time-related data for guideline-based medical decision support, TWAIO Thesis, Eindhoven University of Technology, 2004


Appendix A. Source code patient simulator

This appendix is a complete source code listing of the Patient Simulator application. The code is written in and compiled Borland Delphi 2006.

```plaintext
unit patientsimulator;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ShellAPI, Menus, StdCtrls, ImgList, ExtCtrls, Inifiles, ComCtrls,
  xmldom, XMLIntf, maxmldom, XMLDoc, DB, ADODB;

const
  wm_icontray = wm_user + 1;
  wm_medication = wm_user + 2;

type
  TMedication = class(TObject)
  private
    //
  public
    Id: Integer;
    MedType: Integer;
    Name: String;
    Dose: Double;
    MedUnit: String;
  end;

  TEvent = class(TObject)
  private
    //
  public
    Id: Integer;
    Name: String;
    Active: Boolean;
  end;

  //This class defines a transition describing the condition (value) for a parameter
  //and the state to load if the condition is met
  TTransition = class(TObject)
    Parameter: String;
    Condition: String;
    Value: double;
    NextState: String;
  private
    //
  public
    procedure Go();
  end;

  //This class defines a state description: possible events and transitions
  //and an indicator whether or not this state is the current state in simulation
  TState = class(TObject)
    Name: String;
    Events: Array of String;
    Transitions: Array of TTransition;
    CurrentState: Boolean;
    TimeInState: integer;
  private
    //
  public
    procedure EnterState(FromState: integer);
  end;

  //This class defines the currently loaded case
  TCase = class(TObject)
    Name: String;
```
Description: String;
States: Array of TState;
DefaultStateName: String;
private
  //
  function StateIndex(): integer;
end;

// Class definition of main form
type
TPatient = class(TForm)
  PopupMenu1: TPopupMenu;
  StartSim: TMenuItem;
  StopSim: TMenuItem;
  Exit: TMenuItem;
  N1: TMenuItem;
  RestartSim: TMenuItem;
  ImageList1: TImageList;
  Timer1: TTimer;
  OpenCaseEditor: TMenuItem;
  N2: TMenuItem;
  OpenMedicationListEditor: TMenuItem;
  PageControl1: TPageControl;
  TabSheet1: TTabControl;
  TabSheet2: TTabControl;
  btnCloseSEditor: TButton;
  btnCloseMLEditor: TButton;
  lbICase: TLabel;
  cbCase1: TComboBox;
  SaveDialog1: TSaveDialog;
  lb1If: TLabel;
  cbIf: TComboBox;
  lb1Is: TLabel;
  tListBox:
  edIS: TEdit;
  lb1Then: TLabel;
  cbThen1: TComboBox;
  cbThen2: TComboBox;
  btnAddChange: TButton;
  reCase: TRichEdit;
  XMLCase: TXMLDocument;
  OpenFileDialog1: TOpenDialog;
  ShowCurrentState: TMenuItem;
  TabSheet3: TTabControl;
  cbMedication: TComboBox;
  edDose: TEdit;
  lblUnit: TLabel;
  cbPump: TComboBox;
  edStand: TEdit;
  UpDown1: TUpDown;
  btnChangePump: TButton;
  Label13: TLabel;
  cbDrip: TComboBox;
  edDrip: TEdit;
  Label14: TLabel;
  btnChangeDrip: TButton;
  lb1CurrentValueLabel: TLabel;
  lb1CurrentValue: TLabel;
  Label15: TLabel;
  cbEvent: TComboBox;
  btnChangeEvent: TButton;
  DataSource1: TDataSource;
  ADOConnection1: TADOConnection;
  ADOQuery1: TADOQuery;
  ShowSigns: TMenuItem;
  procedure ShowSignsClick(Sender: TObject);
  procedure btnChangeEventClick(Sender: TObject);
  procedure cbDripChange(Sender: TObject);
  procedure cbPumpChange(Sender: TObject);
  procedure btnChangeDripClick(Sender: TObject);
procedure btnChangePumpClick(Sender: TObject);
procedure cbMedicationChange(Sender: TObject);
procedure btnAdministerClick(Sender: TObject);
procedure btnCloseAMClick(Sender: TObject);
procedure AdministerMedicationClick(Sender: TObject);
procedure ShowCurrentStateClick(Sender: TObject);
procedure cbCase1Change(Sender: TObject);
procedure cbEventChange(Sender: TObject);
procedure btnCloseMLEditorClick(Sender: TObject);
procedure btnCloseSEditorClick(Sender: TObject);
procedure OpenMedicationListEditorClick(Sender: TObject);
procedure OpenCaseEditorClick(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure RestartSimClick(Sender: TObject);
procedure ExitClick(Sender: TObject);
procedure StopSimClick(Sender: TObject);
procedure FormDestroy(Sender: TObject);
procedure FormCreate(Sender: TObject);

private
{ Private declarations }
TrayIconData: TNotifyIconData;
Parameters: Array[1..16] of Integer;
Medication: Array of TMedication;
Event: Array of TEvent;

public
{ Public declarations }
procedure TrayMessage(var Msg: TMessage); message wm_icontray;
procedure ChangeParameter(Parameter: Integer; Value: Integer);
function GetParameter(Parameter: Integer): Integer;
procedure InitializeParameters();
procedure ResetParameters();
procedure AddMedication(var Msg: TMessage); message wm_medication;
function XML2StateDescription(): Boolean;
procedure ParseDescription(CaseFileName: TFileName);
procedure ParseMedicationList(CaseFileName: TFileName);
procedure ParseEventList(CaseFileName: TFileName);
function GetParameterId(Parameter: String): Integer;
function IsParameter(Parameter: String): Boolean;
function IsEvent(EventName: String): Boolean;
function GetMedicationIndex(MedicationName: String; MedicationId: integer = 0): Integer;
function GetEventIndex(EventName: String; EventId: integer = 0): Integer;
procedure UpdateLog(Event: String);

end;

var
Patient: TPatient;

implementation

{ $R *.dfm }

const
wm_simparameter = wm_user + 13;
sm_heartrate = 1;
sm_saturation = 2;
sm_meannibp = 3;
sm_meanabp = 4;
sm_meancvp = 5;
sm_temp1 = 6;
sm_temp2 = 7;
sm_sevo = 8;
sm_capno = 9;
sm_papmean = 10;
sm_resprate = 11;
sm_o2 = 12;
sm_n2o = 13;
sm_sysabp = 14;
sm_diaabp = 15;
sm_pmax = 16;
nm_parameters: Array[1..16] of String = ('heartrate', 'saturation', 'meannibp',
'meanabp', 'meancvp', 'temp1', 'temp2',
'sevo', 'capno', 'papmean',
'resprate', 'o2', 'n2o', 'sysabp',
diaabp', 'pmax');
data_invalid = -32767;
dv_parameters: Array[1..16] of Integer = (75, 95, 90, 100, 75, 3900, 3700, 230,
45, word(data_invalid), 14, 23, 55, 177, 62, 24);

var
fn_medicationlistfilename: TFileName;
fn_eventlistfilename: TFileName;
vCase: TCase;
ParameterUpdates: Array[1..16] of Array[1..2] of Integer; // [i][1] = update step
// [i][2] = update end value
LogFile: TextFile;
TimeInCase: integer;
DebugMode: boolean;
InformationSource: String;
Signs: String;

procedure TTransition.Go();
var
i: integer;
begin
// leave current state
vCase.States[vCase.StateIndex].CurrentState := false;
vCase.States[vCase.StateIndex].TimeInState := 0;
// find next state
for i := 0 to Length(vCase.States) - 1 do
begin
if vCase.States[i].Name = NextState then
begin
// enter next state
vCase.States[i].CurrentState := true;
vCase.States[i].TimeInState := 0;
vCase.States[i].EnterState(vCase.StateIndex);
Exit;
end;
end;
end;

procedure TState.EnterState(FromState: integer);
var
i: integer;
Pos1, Pos2: integer;
ParsedString, Parameter: String;
Value, TimeSpan, StepSize: integer;
ParameterId: integer;
Sign: String;
begin
// log action
Patient.UpdateLog('(state) ' + vCase.States[vCase.StateIndex].Name);

// 3
for i := 0 to Length(Events) - 1 do // all events
begin
Patient.UpdateLog('(event) ' + Events[i]);

// check for special event: show sign
if AnsiPos('sign', Events[i]) = 1 then
begin
Sign := Copy(Events[i], 7, Length(Events[i]) - 6);
Signs := Signs + #13#10 + '[' + FloatToStr(TimeInCase/1000) + #9 +
Signs + Sign;
ShowMessage('The patient shows the following sign/symptom:' + #13#10#13#10
+ Sign);
end;

// check for special event: show message
else if AnsiPos('msg', Events[i]) = 1 then
begin
ShowMessage('Message:' + #13#10#13#10 + Copy(Events[i], 6,
Length(Events[i]) - 5));
end
end;
end;

// find parameter name in event string
ParsedString := Copy(Events[i], 5, Length(Events[i]) - 4);
Pos1 := AnsiPos('#32', ParsedString);
Parameter := Copy(ParsedString, 0, Pos1 - 1);
//find parameter value in event string
ParsedString := Copy(ParsedString, Pos1 + 1, Length(ParsedString) - Pos1);
 Pos1 := AnsiPos('#32', ParsedString);
 ParsedString := Copy(ParsedString, Pos1 + 1, Length(ParsedString) - Pos1);
if Pos1 <> 0 then
  begin
    Value := StrToInt(Copy(ParsedString, 0, Pos1 - 1));
  end
else
  begin
    Value := StrToInt(Copy(ParsedString, 0, Length(ParsedString)));
  end;

//if 'over' is in event parameter value has to be changed in steps
Pos1 := AnsiPos('over', ParsedString);
if Pos1 <> 0 then
  begin
    //find timespan in event string
    Pos2 := AnsiPos(' minute(s)', ParsedString);
    TimeSpan := StrToInt(Copy(ParsedString, Pos1 + 4, Pos2 - (Pos1 + 4)));
    ParameterId := Patient.GetParameterId(Parameter);
    //calculate step size
    StepSize := Round((Value - Patient.GetParameter(ParameterId)) / 
          (TimeSpan + 1)));
    //perform first step
    Patient.ChangeParameter(ParameterId, Patient.GetParameter(ParameterId) 
              + StepSize);
    //add other steps to 'to-do-list': [1] = stepsize, [2] = end value
    Patient.UpdateLog('(on list) ' + Parameter);
    ParameterUpdates[ParameterId][1] := StepSize;
    ParameterUpdates[ParameterId][2] := Value;
  end
else
  begin
    //else change parameter value in just one step
    Patient.ChangeParameter(Patient.GetParameterId(Parameter), Value);
  end;
end;

function TCase.StateIndex(): integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to Length(States) - 1 do
    begin
      if States[i].CurrentState then
        begin
          //Return index of current state in case state list
          Result := i;
          Break;
        end;
    end;
end;

//Initialization of the application
procedure TPatient.FormCreate(Sender: TObject);
var
  Icon: TIcon;
  Configuration: TIniFile;
  Provider, IntegratedSecurity, PersistSecurityInfo, InitialCatalog, DataSource,
  UseProcedureforPrepare, AutoTranslate, PacketSize, WorkstationID,
  UseEncryptionforData,
  Tag: String;
  ConnectionString: String;
begin
  //disable editor menu options (temporary; features not available yet)
  OpenCaseEditor.Enabled := false;
  OpenMedicationListEditor.Enabled := false;
  ShowCurrentState.Enabled := false;
  ShowSigns.Enabled := false;
  AdministerMedication.Enabled := false;
//disable editor tabs (temporary; features noy available yet)
TabSheet1.Enabled := false;
TabSheet2.Enabled := false;

//read configuration from configuration file
Configuration := TIniFile.Create(ExtractFilePath(Application.ExeName) + 'config\config.ini');
try
  Timer1.Interval := Configuration.ReadInteger('general settings', 'timer', 60000); //cycle time of the simulation, default is one minute
  fn_medicationlistfilename := 'lists\' + Configuration.ReadString('general settings', 'medicationlistfilename', 'medication.xml'); //name of file containing medication list
  fn_eventlistfilename := 'lists\' + Configuration.ReadString('general settings', 'eventlistfilename', 'events.xml'); //name of file containing event list
  DebugMode := false;
  if Configuration.ReadString('general settings', 'debugmode', 'false') = 'true'
  then //debug mode enabled, default is false
    begin
      DebugMode := true;
      end;
  Provider := Configuration.ReadString('db server', 'Provider', 'SQLOLEDB.1');
  IntegratedSecurity := Configuration.ReadString('db server', 'Integrated Security', 'SSPI');
  PersistSecurityInfo := Configuration.ReadString('db server', 'Persist Security Info', 'False');
  InitialCatalog := Configuration.ReadString('db server', 'Initial Catalog', 'OK');
  DataSource := Configuration.ReadString('db server', 'Data Source', 'MBS-NB162');
  UseProcedureforPrepare := Configuration.ReadString('db server', 'Use Procedure for Prepare', '1');
  AutoTranslate := Configuration.ReadString('db server', 'Auto Translate', 'True');
  PacketSize := Configuration.ReadString('db server', 'Packet Size', '4096');
  WorkstationID := Configuration.ReadString('db server', 'Workstation ID', 'MBS-NB162');
  UseEncryptionforData := Configuration.ReadString('db server', 'Use Encryption for Data', 'False');
  Tag := Configuration.ReadString('db server', 'Tag with column collation when possible', 'False');

  ConnectionString := 'Provider=' + Provider + ';'
    + 'Integrated Security=' + IntegratedSecurity + ';'
    + 'Persist Security Info=' + PersistSecurityInfo + ';'
    + 'Initial Catalog=' + InitialCatalog + ';'
    + 'Data Source=' + DataSource + ';'
    + 'Use Procedure for Prepare=' + UseProcedureforPrepare + ';'
    + 'Auto Translate=' + AutoTranslate + ';'
    + 'Packet Size=' + PacketSize + ';'
    + 'Workstation ID=' + WorkstationID + ';'
    + 'Use Encryption for Data=' + UseEncryptionforData + ';'
    + 'Tag with column collation when possible=' + Tag + ';';
  ADOConnection1.ConnectionString := ConnectionString;
finally
  Configuration.Destroy;
end;

//Stop timer
Timer1.Enabled := false;

//Create case object
vCase := TCase.Create;

//Set Sign list
Signs := 'The patient has shown the following signs:' + nl + nl;

//change system tray data
with TrayIconData do
  begin
    cbSize := SizeOf(TrayIconData);
    Wnd := Handle;
  end;
uID := 0;
uFlags := nif_message + nif_icon + nif_tip;
uCallbackMessage := wm_icontray;
Icon := TIcon.Create;
try
  ImageList1.GetIcon(0, Icon);
  Icon.Transparent := true;
  hIcon := Icon.Handle;
  StrPCopy(szTip, 'Patient Simulator (stopped)');
  Shell_NotifyIcon(nim_add, @TrayIconData);
finally
  Icon.Free;
end;

//stop running simulation (normally no simulation is running yet)
StopSimClick(Sender);
end;

//Termination of the application
procedure TPatient.FormDestroy(Sender: TObject);
begin
  //stop running application
  StopSimClick(Sender);

  //destroy case object
  vCase.Destroy;

  //remove icon from system tray
  Shell_NotifyIcon(nim_delete, @TrayIconData);
end;

//Open popup menu when right mouse button click on system tray icon
procedure TPatient.TrayMessage(var Msg: TMessage);
var
  p: TPoint;
begin
  case Msg.LParam of
    wm_lbuttondown:
      begin
        //
      end;
    wm_rbuttondown:
      begin
        //open popup menu (system tray)
        SetForegroundWindow(Handle);
        GetCursorPos(p);
        PopupMenu1.Popup(p.x, p.y);
        PostMessage(Handle, wm_null, 0, 0);
      end;
  end;
end;

//Start simulation
procedure TPatient.StartSimClick(Sender: TObject);
var
  Icon: TIcon;
  CaseFile, MedicationFile, EventFile: TFileName;
  LogfileName: string;
begin
  //load case from xml file into memory
  OpenDialog1.InitialDir := 'cases';
  if OpenDialog1.Execute then
    begin //select xml file from dialog
      CaseFile := OpenDialog1.FileName;
      ParseDescription(CaseFile); //put case into memory
      ShowMessage(vCase.Name);
      ShowMessage(vCase.Description);
      //load medication list from xml file into memory
      MedicationFile := ExtractFilePath(Application.ExeName) + fn_medicationlistfilename;
      ParseMedicationList(MedicationFile); //put medication list into memory
      //load event list from xml file into memory
      EventFile := ExtractFilePath(Application.ExeName) + fn_eventlistfilename;
ParseEventList(EventFile); //put medication list into memory

// change system tray data
StrPCopy(TrayIconData.szTip, 'Patient Simulator (running)');
Icon := TIcon.Create;
try
  ImageList1.GetIcon(1, Icon);
  Icon.Transparent := true;
  TrayIconData.hIcon := Icon.Handle;
  Shell_NotifyIcon(nim_modify, @TrayIconData);
finally
  Icon.Free;
end;

// enable or disable menu options
StartSim.Enabled := false; // one cannot start a simulation while a simulation is already running
StopSim.Enabled := true;
RestartSim.Enabled := true;
ShowCurrentState.Enabled := true;
ShowSigns.Enabled := true;
AdministerMedication.Enabled := true;

// open log file
DateTimeToString(LogfileName, 'ddmmyyhhnnss', Now);
LogfileName := '..\logs\' + StringReplace(LowerCase(vCase.Name), ' ', '', [rfReplaceAll]) + LogfileName + '.log';
AssignFile(LogFile, LogfileName);
Rewrite(LogFile);
UpdateLog('(start) ' + vCase.Name);

// start simulation
vCase.States[vCase.StateIndex].EnterState(vCase.StateIndex); // Activate default state
end else begin
  ShowMessage('Error: Unable to open case directory');
end;

// stop menu option
procedure TPatient.StopSimClick(Sender: TObject);
var
  Icon: TIcon;
begin
  if Timer1.Enabled then
    begin
      UpdateLog('(stop) ' + vCase.Name);
      // close log file
      CloseFile(LogFile)
    end;
  // stop timer
  Timer1.Enabled := false;
  TimeInCase := 0;
  // set all parameters to data_invalid (simulating the measurement equipment not connected)
  ResetParameters();
  // remove case from memory
  Signs := 'The patient has shown the following signs:' + #13#10;
  vCase.Free;
  vCase := TCase.Create;
  // change system tray data
  StrPCopy(TrayIconData.szTip, 'Patient Simulator (stopped)');
  Icon := TIcon.Create;
  try
    ImageList1.GetIcon(0, Icon);
    Icon.Transparent := true;
    TrayIconData.hIcon := Icon.Handle;
Shell_NotifyIcon(nim_modify, @TrayIconData);
finally
  Icon.Free;
end;

// enable or disable menu options
StopSim.Enabled := false; // No case cannot be stopped if no case is running
RestartSim.Enabled := false; // No case cannot be restarted if no case is running
StartSim.Enabled := true;
ShowCurrentState.Enabled := false; // There is no current state if no case is running
ShowSigns.Enabled := false;
AdministerMedication.Enabled := false; // No medication can be administered if no case is running
end;

// Exit menu option
procedure TPatient.ExitClick(Sender: TObject);
begin
  Application.Terminate;
end;

// Restart menu option
procedure TPatient.RestartSimClick(Sender: TObject);
begin
  // restart equals stop -> start :-P
  StopSimClick(Sender);
  StartSimClick(Sender);
end;

// Change a parameter value
procedure TPatient.ChangeParameter(Parameter: Integer; Value: Integer);
begin
  // change parameter value in working memory
  Parameters[Parameter] := Value;

  // send new parameter value to Vierkleurenpen (anesthesia registration software)
  postmessage(FindWindow('TVierkleur', nil), wm_simparameter, Parameter, Value);

  // log action
  if Timer1.Enabled then
  begin
    UpdateLog('(update) ' + nm_parameters[Parameter] + ' := ' + IntToStr(Value));
  end;
end;

// Get a parameter value
function TPatient.GetParameter(Parameter: Integer): Integer;
begin
  // return parameter value from working memory
  Result := Parameters[Parameter];
end;

// Set all parameters to default value
procedure TPatient.InitializeParameters();
var
  i: Integer;
begin
  for i := sm_heartrate to sm_pmax do
  begin
    // set default value for all parameters
    ChangeParameter(i, dv_parameters[i]);
  end;
end;

// Simulate "monitor not connected"
procedure TPatient.ResetParameters();
var
  i: Integer;
begin
  for i := sm_heartrate to sm_pmax do
  begin
    // set value to data_invalid for all parameters (simulating that the monitor is not connected)
    ChangeParameter(i, data_invalid);
  end;
end;
procedure TPatient.AddMedication(var Msg: TMessage); //Administer medication or adjust pump or drip setting
var
  MedicationIndex: integer;
begin
  MedicationIndex := GetMedicationIndex('', Msg.WParam);
  case Medication[MedicationIndex].MedType of
  //Medication
  0: Medication[MedicationIndex].Dose := Medication[MedicationIndex].Dose + (Msg.LParam / 1000);
  //pump
  1: Medication[MedicationIndex].Dose := Msg.LParam;
  //Drip (infusion)
  2: Medication[MedicationIndex].Dose := Msg.LParam;
  end;
  UpdateLog('(medication) ' + Medication[MedicationIndex].Name + ': ' + FloatToStr(Medication[MedicationIndex].Dose) + ' ' + Medication[MedicationIndex].MedUnit);
end;

//Simulation cycle
procedure TPatient.Timer1Timer(Sender: TObject);
var
  StateIndex: integer;
  i, j: integer;
  ParsedCondition: String;
  Check: Boolean;
begin
  TimeInCase := TimeInCase + Timer1.Interval; //Update the time passed in this case
  //Simulation steps:
  //1: update parameter values if necessary
  //2: check transition conditions
  //3: go to new state if necessary
  //Update TimeInState for current state
  StateIndex := vCase.StateIndex();
  vCase.States[StateIndex].TimeInState := vCase.States[StateIndex].TimeInState + Timer1.Interval;
  Check := true;

  //1
  for i := 1 to 16 do //all parameters
  begin
    if ParameterUpdates[i][1] <> 0 then //if update scheduled
    begin //perform update
      ChangeParameter(i, GetParameter(i) + ParameterUpdates[i][1]);
      if ParameterUpdates[i][1] > 0 then //if parameter value increasing
      begin
        if GetParameter(i) >= ParameterUpdates[i][2] then //if end value reached
        begin
          ChangeParameter(i, ParameterUpdates[i][2]); //set parameter to end value
        end;
      end;
      end
    else //if parameter value decreasing
    begin
      if GetParameter(i) <= ParameterUpdates[i][2] then //if end value reached
      begin
        ChangeParameter(i, ParameterUpdates[i][2]); //set parameter to end value
      end;
    end;
  end;
  //2
for j := 0 to Length(vCase.States[StateIndex].Transitions) - 1 do //all possible transitions for this state
begin //if parameter is one of the 16 medical parameters
if IsParameter(vCase.States[StateIndex].Transitions[j].Parameter) then
begin
//Parameter
ParsedCondition := vCase.States[StateIndex].Transitions[j].Condition;
if ParsedCondition = '<' then //if less than
begin
//3
begin
vCase.States[StateIndex].Transitions[j].Go(); //perform transition
Break;
end;
end
else if ParsedCondition = '<=' then //less than or equal to
begin
begin
vCase.States[StateIndex].Transitions[j].Go(); //perform transition
Break;
end;
end
else if ParsedCondition = '=' then //equal to
begin
begin
vCase.States[StateIndex].Transitions[j].Go(); //perform transition
Break;
end;
end
else if ParsedCondition = '>=' then //greater than or equal to
begin
begin
vCase.States[StateIndex].Transitions[j].Go(); //perform transition
Break;
end;
end
else if ParsedCondition = '>' then //greater than
begin
end;
  UpdateLog('(condition) ' +
  vCase.States[StateIndex].Transitions[j].Parameter + ' ' +
  vCase.States[StateIndex].Transitions[j].Condition + ' ' +
  FloatToStr(vCase.States[StateIndex].Transitions[j].Value));
  vCase.States[StateIndex].Transitions[j].Go(); //perform transition
end;
end;
end; //if 'parameter' is an event
else if IsEvent(vCase.States[StateIndex].Transitions[j].Parameter) then begin
  //Event
  //Event is active and should be active
  if Event[GetEventIndex(vCase.States[StateIndex].Transitions[j].Parameter)].Active AND (vCase.States[StateIndex].Transitions[j].Value = 1) then begin
    UpdateLog('(condition) ' +
    vCase.States[StateIndex].Transitions[j].Parameter + ' ' +
    vCase.States[StateIndex].Transitions[j].Condition + ' ' +
    FloatToStr(vCase.States[StateIndex].Transitions[j].Value));
    vCase.States[StateIndex].Transitions[j].Go(); //perform transition
    Break;
  end;
  //Event is not active and should be 'not active'
  if not Event[GetEventIndex(vCase.States[StateIndex].Transitions[j].Parameter)].Active AND (vCase.States[StateIndex].Transitions[j].Value = 0) then begin
    UpdateLog('(condition) ' +
    vCase.States[StateIndex].Transitions[j].Parameter + ' ' +
    vCase.States[StateIndex].Transitions[j].Condition + ' ' +
    FloatToStr(vCase.States[StateIndex].Transitions[j].Value));
    vCase.States[StateIndex].Transitions[j].Go(); //perform transition
    Break;
  end;
end; //if parameter is time_in_state (time passed in current state)
else if vCase.States[StateIndex].Transitions[j].Parameter = 'time_in_state' then begin
  //Time in State
  if vCase.States[StateIndex].TimeInState > Timer1.Interval *
  vCase.States[StateIndex].Transitions[j].Value then begin
    //3
    UpdateLog('(condition) ' +
    vCase.States[StateIndex].Transitions[j].Parameter + ' ' +
    vCase.States[StateIndex].Transitions[j].Condition + ' ' +
    FloatToStr(vCase.States[StateIndex].Transitions[j].Value));
    vCase.States[StateIndex].Transitions[j].Go(); //perform transition
    Break;
  end;
else begin
  //Medication
  ParsedCondition := vCase.States[StateIndex].Transitions[j].Condition;
  if ParsedCondition = '<' then //less than
  begin
    //3

UpdateLog('(condition) ' + 
vCase.States[StateIndex].Transitions[j].Parameter + ' ' + 
vCase.States[StateIndex].Transitions[j].Condition + ' ' + 
FloatToStr(vCase.States[StateIndex].Transitions[j].Value)); 
vCase.States[StateIndex].Transitions[j].Go(); //perform transition 
Break;
end;
end;

else if ParsedCondition = '<=' then //less than or equal to 
begin 
if 
Medication[GetMedicationIndex(vCase.States[StateIndex].Transitions[j].Parameter)].Dose 
<= vCase.States[StateIndex].Transitions[j].Value then 
begin 
UpdateLog('(condition) ' + 
vCase.States[StateIndex].Transitions[j].Parameter + ' ' + 
vCase.States[StateIndex].Transitions[j].Condition + ' ' + 
FloatToStr(vCase.States[StateIndex].Transitions[j].Value)); 
vCase.States[StateIndex].Transitions[j].Go(); //perform transition 
Break;
end;
end;

else if ParsedCondition = '=' then //equal to 
begin 
if 
Medication[GetMedicationIndex(vCase.States[StateIndex].Transitions[j].Parameter)].Dose 
= vCase.States[StateIndex].Transitions[j].Value then 
begin 
UpdateLog('(condition) ' + 
vCase.States[StateIndex].Transitions[j].Parameter + ' ' + 
vCase.States[StateIndex].Transitions[j].Condition + ' ' + 
FloatToStr(vCase.States[StateIndex].Transitions[j].Value)); 
vCase.States[StateIndex].Transitions[j].Go(); //perform transition 
Break;
end;
end;

else if ParsedCondition = '>= ' then //greater than or equal to 
begin 
if 
Medication[GetMedicationIndex(vCase.States[StateIndex].Transitions[j].Parameter)].Dose 
>= vCase.States[StateIndex].Transitions[j].Value then 
begin 
UpdateLog('(condition) ' + 
vCase.States[StateIndex].Transitions[j].Parameter + ' ' + 
vCase.States[StateIndex].Transitions[j].Condition + ' ' + 
FloatToStr(vCase.States[StateIndex].Transitions[j].Value)); 
vCase.States[StateIndex].Transitions[j].Go(); //perform transition 
Break;
end;
end;

else if ParsedCondition = '>' then //greater than 
begin 
if 
Medication[GetMedicationIndex(vCase.States[StateIndex].Transitions[j].Parameter)].Dose 
> vCase.States[StateIndex].Transitions[j].Value then 
begin 
UpdateLog('(condition) ' + 
vCase.States[StateIndex].Transitions[j].Parameter + ' ' + 
vCase.States[StateIndex].Transitions[j].Condition + ' ' + 
FloatToStr(vCase.States[StateIndex].Transitions[j].Value)); 
vCase.States[StateIndex].Transitions[j].Go(); //perform transition 
Break;
end;
end;
end;
procedure TPatient.OpenCaseEditorClick(Sender: TObject);
var
  CaseFiles: TSearchRec;
begin
  //set active tab to Case Editor
  PageControl1.ActivePage := TabSheet1;
  with TabSheet1 do
  begin
    //search for available case's
    cbCase1.Clear;
    if FindFirst(ExtractFilePath(Application.ExeName) + 'cases\*.xml', faAnyFile, CaseFiles) = 0 then
      begin
        try
          repeat
            //add case to selection list
            cbCase1.Items.Add(StringReplace(CaseFiles.Name, '.xml', '', [0]));
          until FindNext(CaseFiles) <> 0;
          finally
            FindClose(CaseFiles);
          end;
        end;
      end;
    cbCase1.ItemIndex := 0;
    //Show editor
    Patient.Show;
  end;
end;

procedure TPatient.OpenMedicationListEditorClick(Sender: TObject);
begin
  //set active tab to Medication List Editor
  PageControl1.ActivePage := TabSheet2;
  //Show editor
  Patient.Show;
end;

procedure TPatient.btnCloseSEditorClick(Sender: TObject);
begin
  //Close editor
  Patient.Hide;
end;

procedure TPatient.btnCloseMLEditorClick(Sender: TObject);
begin
  //Close editor
  Patient.Hide;
end;

procedure TPatient.cbCase1Change(Sender: TObject);
var
  CaseFile: TFileName;
begin
  //Read case file (xml)
  CaseFile := ExtractFilePath(Application.ExeName) + 'cases' + cbCase1.SelText + '.xml';
  XMLCase.FileName := CaseFile;
  XMLCase.Active := true;
  //Convert case description format and show case in editor
  try
    reCase.Lines.Clear;
    XML2StateDescription();
  finally
    XMLCase.Active := false;
    end;
end;
//Function converts case from xml to "plain text"
function TPatient.XML2StateDescription(): Boolean;
var
  StateDescription, StateName, StateEvents, StateTransitions: TStringList;
  i, j: Integer;
  XMLNode: IXMLNode;
begin
  //Create object to store case description
  StateDescription := TStringList.Create;
  try
    //Parse xml
    for i := 0 to XMLCase.DocumentElement.ChildNodes.Count - 1 do
      begin
        //If state description
        if XMLNode.NodeName = 'state' then
          begin
            //Create objects to store respectively name, events and transitions
            of the state
            StateName := TStringList.Create;
            StateEvents := TStringList.Create;
            StateTransitions := TStringList.Create;
            try
              //Parse elements of state description
              for j := 0 to XMLNode.ChildNodes.Count - 1 do
                begin
                  if XMLNode.ChildNodes.Nodes[j].NodeName = 'name' then
                    begin
                      //Add state to selection list in editor
                      cbthen2.Items.Add(XMLNode.ChildNodes.Nodes[j].NodeValue);
                      //Add state name to description
                      StateName.Add(XMLNode.ChildNodes.Nodes[j].NodeValue);
                    end
                  else if XMLNode.ChildNodes.Nodes[j].NodeName = 'event' then
                    begin
                      //Add event to state event list
                      StateEvents.Add('      ' + XMLNode.ChildNodes.Nodes[j].NodeValue);
                    end
                  else if XMLNode.ChildNodes.Nodes[j].NodeName = 'transition'
                    then
                    begin
                      //Add transition to state transition list
                      StateTransitions.Add('      ' + XMLNode.ChildNodes.Nodes[j].NodeValue);
                    end;
                end;
            //Merge elements of state description
            StateDescription.AddStrings(StateName);
            StateDescription.Add('   ' + 'Events');
            StateDescription.AddStrings(StateEvents);
            StateDescription.Add('   ' + 'Transitions');
            finally
              StateName.Destroy;
              StateEvents.Destroy;
              StateTransitions.Destroy;
            end;
          end;
    //Show state description in editor
    reCase.Lines.AddStrings(StateDescription);
    Result := true;
  finally
    StateDescription.Destroy;
  end;
end;
//Function parses case file and creates a case object
procedure TPatient.ParseDescription(CaseFileName: TFileName);
var
  i, j: Integer;
begin
XMLNode: IXMLNode;
ParsedState: TState;
ParsedTransition: TTransition;
ParsedString: String;
Pos1: integer;
begin
XMLCase.FileName := CaseFileName;
XMLCase.Active := true;
try
//Parse xml
for i := 0 to XMLCase.DocumentElement.ChildNodes.Count - 1 do
begin
//If case information
if XMLNode.NodeName = 'information' then
begin
//Parse elements of information
for j := 0 to XMLNode.ChildNodes.Count - 1 do
begin
if XMLNode.ChildNodes.Nodes[j].NodeName = 'name' then
begin
//Set case name
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'description' then
begin
//Set case description
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'defaultstate' then
begin
//Set case default state
end;
end;
end;
//If state description
if XMLNode.NodeName = 'state' then
begin
//Create temporary state object
ParsedState := TState.Create;
//Parse elements of state description
for j := 0 to XMLNode.ChildNodes.Count - 1 do
begin
if XMLNode.ChildNodes.Nodes[j].NodeName = 'name' then
begin
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'event' then
begin
SetLength(ParsedState.Events, Length(ParsedState.Events) + 1);
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'sign' then
begin
SetLength(ParsedState.Events, Length(ParsedState.Events) + 1);
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'msg' then
begin
SetLength(ParsedState.Events, Length(ParsedState.Events) + 1);
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'transition' then
begin
ParsedTransition := TTransition.Create;
ParsedString := Copy(ParsedString, 4, Length(ParsedString) - 3); Pos1 := AnsiPos(#32, ParsedString);
ParsedTransition.Parameter := Copy(ParsedString, 0, Pos1 - 1);

ParsedString := Copy(ParsedString, Pos1 + 1, Length(ParsedString) - Pos1);
Pos1 := AnsiPos(#32, ParsedString);
ParsedTransition.Condition := Copy(ParsedString, 0, Pos1 - 1);

ParsedString := Copy(ParsedString, Pos1 + 1, Length(ParsedString) - Pos1);
Pos1 := AnsiPos(#32, ParsedString);
ParsedTransition.Value := StrToFloat(Copy(ParsedString, 0, Pos1 - 1));

Pos1 := AnsiPos('go to', ParsedString);
ParsedTransition.NextState := Copy(ParsedString, Pos1 + 6, Length(ParsedString) - 5);
SetLength(ParsedState.Transitions, Length(ParsedState.Transitions) + 1);
ParsedState.Transitions[Length(ParsedState.Transitions) - 1] := ParsedTransition;
end;
end;

if ParsedState.Name = vCase.DefaultStateName then
begin
ParsedState.CurrentState := true;
end
else
begin
ParsedState.CurrentState := false;
end;

ParsedState.TimeInState := 0;
SetLength(vCase.States, Length(vCase.States) + 1);
vCase.States[Length(vCase.States) - 1] := ParsedState;
end;
end;

// Function parses medication list file and creates an array of medication objects
procedure TPatient.ParseMedicationList(CaseFileName: TFileName);
var
i, j: Integer;
XMLNode: IXMLNode;
ParsedMedication: TMedication;
begin
if InformationSource = 'xml' then
begin
XMLCase.FileName := CaseFileName;
XMLCase.Active := true;
try
// Parse xml
for i := 0 to XMLCase.DocumentElement.ChildNodes.Count - 1 do
begin
// If medication
if XMLNode.NodeName = 'medication' then
begin
ParsedMedication := TMedication.Create();
// Parse elements of medication
for j := 0 to XMLNode.ChildNodes.Count - 1 do
begin
if XMLNode.ChildNodes.Nodes[j].NodeName = 'id' then
begin
// Set medication id
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'name' then
begin
// Set medication name
end
end
end
end
ParsedMedication.Name :=
XMLNode.ChildNodes.Nodes[j].NodeValue;
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'type' then
begin
//Set medication type
ParsedMedication.MedType :=
XMLNode.ChildNodes.Nodes[j].NodeValue;
end
else if XMLNode.ChildNodes.Nodes[j].NodeName = 'unit' then
begin
//Set medication unit
ParsedMedication.MedUnit :=
XMLNode.ChildNodes.Nodes[j].NodeValue;
end;
end;
SetLength(Medication, Length(Medication) + 1);
Medication[Length(Medication) - 1].Id := ParsedMedication.Id;
Medication[Length(Medication) - 1].Name := ParsedMedication.Name;
Medication[Length(Medication) - 1].MedType := ParsedMedication.MedType;
Medication[Length(Medication) - 1].Dose := 0;
Medication[Length(Medication) - 1].MedUnit := ParsedMedication.MedUnit;
ParsedMedication.Free;
end;
end;
finally
XMLCase.Active := false;
end;
end
else if InformationSource = 'db' then begin
if ADOQuery1.Active then ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Add('SELECT medicijn, unit FROM tb_drugs_dosing;');
ADOQuery1.Open;
with ADOQuery1 do begin
First;
  i := 0;
  while not Eof do begin
    ParsedMedication := TMedication.Create();
    ParsedMedication.Id := i;
    ParsedMedication.MedType := 0;
    ParsedMedication.Name := StringReplace(FieldByName('medicijn').AsString, ' ', '_', [rfReplaceAll]);
    ParsedMedication.Dose := 0;
    ParsedMedication.MedUnit := FieldByName('unit').AsString;
    SetLength(Medication, Length(Medication) + 1);
    Medication[Length(Medication) - 1].Id := ParsedMedication.Id;
    Medication[Length(Medication) - 1].Name := ParsedMedication.Name;
    Medication[Length(Medication) - 1].MedType := ParsedMedication.MedType;
    Medication[Length(Medication) - 1].Dose := 0;
    Medication[Length(Medication) - 1].MedUnit := ParsedMedication.MedUnit;
    ParsedMedication.Free;
    Next;
    Inc(i);
  end;
end;
ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Add('SELECT fluid_name, infuus_vloeistof FROM tb_infusion_fluid_bags;');
ADOQuery1.Open;
with ADOQuery1 do begin
First;
while not Eof do
begin
  ParsedMedication := TMedication.Create();
  ParsedMedication.Id := FieldByName('infuus_vloeistof').AsInteger;
  ParsedMedication.MedType := 2;
  ParsedMedication.Name := StringReplace(FieldByName('fluid_name').AsString, ' ', '_', [rfReplaceAll]);
  ParsedMedication.Dose := 0;
  ParsedMedication.MedUnit := 'ml';
  SetLength(Medication, Length(Medication) + 1);
  Medication[Length(Medication) - 1].Id := ParsedMedication.Id;
  Medication[Length(Medication) - 1].Name := ParsedMedication.Name;
  Medication[Length(Medication) - 1].MedType := ParsedMedication.MedType;
  Medication[Length(Medication) - 1].Dose := 0;
  Medication[Length(Medication) - 1].MedUnit := ParsedMedication.MedUnit;
  ParsedMedication.Free;
  Next;
end;

ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Add('SELECT fluid_name, infusion_fluid_code FROM tb_pump_fluids;');
ADOQuery1.Open;

with ADOQuery1 do
begin
  First;
  while not Eof do
  begin
    ParsedMedication := TMedication.Create();
    ParsedMedication.Id := FieldByName('infusion_fluid_code').AsInteger;
    ParsedMedication.MedType := 1;
    ParsedMedication.Name := StringReplace(FieldByName('fluid_name').AsString, ' ', '_', [rfReplaceAll]);
    ParsedMedication.Dose := 0;
    ParsedMedication.MedUnit := '(stand)';
    SetLength(Medication, Length(Medication) + 1);
    Medication[Length(Medication) - 1].Id := ParsedMedication.Id;
    Medication[Length(Medication) - 1].Name := ParsedMedication.Name;
    Medication[Length(Medication) - 1].MedType := ParsedMedication.MedType;
    Medication[Length(Medication) - 1].Dose := 0;
    Medication[Length(Medication) - 1].MedUnit := ParsedMedication.MedUnit;
    ParsedMedication.Free;
    Next;
  end;
end;

//Function parses event list file and creates an array of event objects
procedure TPatient.ParseEventList(CaseFileName: TFileName);
var
  i,j: Integer;
  XMLNode: IXMLNode;
  ParsedEvent: TEvent;
begin
  if InformationSource = 'xml' then
  begin
    XMLCase.FileName := CaseFileName;
    XMLCase.Active := true;
    try
      //Parse xml
      for i := 0 to XMLCase.DocumentElement.ChildNodes.Count - 1 do
      begin
        //Parse event
        ParsedEvent := TEvent.Create;
        ParsedEvent.Id := XMLNode[1].Value;
        ParsedEvent.Type := XMLNode[2].Value;
        ParsedEvent.DateTime := XMLNode[3].Value;
        ParsedEvent.Free;
        NextEvent;
      end;
    except
      on E: EXMLParseException do
      begin
        'Error parsing event list file.'
      end;
    end;
  end;
end;
if XMLNode.NodeName = 'event' then begin
  ParsedEvent := TEvent.Create();
  //Parse elements of event
  for j := 0 to XMLNode.ChildNodes.Count - 1 do begin
    if XMLNode.ChildNodes.Nodes[j].NodeName = 'id' then begin
      //Set event id
    end else if XMLNode.ChildNodes.Nodes[j].NodeName = 'name' then begin
      //Set event name
    end;
    SetLength(Event, Length(Event) + 1);
    Event[Length(Event) - 1] := TEvent.Create();
    Event[Length(Event) - 1].Id := ParsedEvent.Id;
    Event[Length(Event) - 1].Name := ParsedEvent.Name;
    Event[Length(Event) - 1].Active := false;
    ParsedEvent.Free;
  end;
  finally
    XMLCase.Active := false;
  end;
end else if InformationSource = 'db' then begin
  if ADOQuery1.Active then ADOQuery1.Close;
  ADOQuery1.SQL.Clear;
  ADOQuery1.SQL.Add('SELECT handeling, handel_ident FROM tb_handelingen;');
  ADOQuery1.Open;
  with ADOQuery1 do begin
    First;
    while not Eof do begin
      if not (FieldByName('handeling').AsString = '') then begin
        ParsedEvent := TEvent.Create();
        ParsedEvent.Id := FieldByName('handel_ident').AsInteger;
        ParsedEvent.Name := StringReplace(FieldByName('handeling').AsString, ',', '_', [rfReplaceAll]);
        SetLength(Event, Length(Event) + 1);
        Event[Length(Event) - 1] := TEvent.Create();
        Event[Length(Event) - 1].Id := ParsedEvent.Id;
        Event[Length(Event) - 1].Name := ParsedEvent.Name;
        Event[Length(Event) - 1].Active := false;
        ParsedEvent.Free;
      end;
      Next;
    end;
  end;
  ADOQuery1.Close;
end;
end;

function TPatient.GetParameterId(Parameter: String): Integer;
var
  j: integer;
begin
  Result := 0;
  for j := 1 to 16 do begin
    if nm_parameters[j] = LowerCase(Parameter) then begin
      Result := j;
      Break;
    end;
  end;
function TPatient.IsParameter(Parameter: String): Boolean;
var
  i, j: integer;
begin
  i := 0;
  for j := 1 to 16 do
    begin
      if nm_parameters[j] = LowerCase(Parameter) then
      begin
        i := j;
        Break;
      end;
    end;
  if i = 0 then
    begin
      Result := false;
    end
  else
    begin
      Result := true;
    end;
end;

function TPatient.IsEvent(EventName: String): Boolean;
var
  i, j: integer;
begin
  i := -1;
  for j := 0 to Length(Event) - 1 do
    begin
      if Event[j].Name = EventName then
      begin
        i := j;
        Break;
      end;
    end;
  if i = -1 then
    begin
      Result := false;
    end
  else
    begin
      Result := true;
    end;
end;

function TPatient.GetMedicationIndex(MedicationName: String; MedicationId: integer = 0): Integer;
var
  i: integer;
begin
  MedicationName := StringReplace(MedicationName, ' ', '_', [rfReplaceAll]);
  Result := 0;
  if MedicationId = 0 then
    begin
      for i := 0 to Length(Medication) - 1 do
        begin
          if Medication[i].Name = MedicationName then
            begin
              Result := i;
              Break;
            end;
        end;
    end
  else
    begin
      for i := 0 to Length(Medication) - 1 do
        begin
          if Medication[i].Id = MedicationId then
            begin
              Result := i;
              Break;
            end;
        end;
    end;
end;
function TPatient.GetEventIndex(EventName: String; EventId: integer = 0): Integer;
var
  i: integer;
begin
  EventName := StringReplace(EventName, ' ', '_', [rfReplaceAll]);
  Result := 0;
  if EventId = 0 then
    begin
      for i := 0 to Length(Event) - 1 do
        begin
          if Event[i].Name = EventName then
            begin
              Result := i;
              Break;
            end;
        end;
    end
  else
    begin
      for i := 0 to Length(Event) - 1 do
        begin
          if Event[i].Id = EventId then
            begin
              Result := i;
              Break;
            end;
        end;
    end;
end;

procedure TPatient.ShowCurrentStateClick(Sender: TObject);
begin
  ShowMessage(vCase.States[vCase.StateIndex].Name);
end;

procedure TPatient.ShowSignsClick(Sender: TObject);
begin
  ShowMessage(Signs);
end;

procedure TPatient.AdministerMedicationClick(Sender: TObject);
var
  i: integer;
begin
  //set active tab to Medication and Events
  PageControl1.ActivePage := TabSheet3;

  //load medication and events into selection box
  cbMedication.Clear;
  cbPump.Clear;
  cbDrip.Clear;
  cbEvent.Clear;
  for i := 0 to Length(Medication) - 1 do
    begin
      if Medication[i].MedType = 0 then
        begin
          cbMedication.Items.Add(StringReplace(Medication[i].Name, ' ', '_', [rfReplaceAll]));
        end
      else if Medication[i].MedType = 1 then
        begin
          cbPump.Items.Add(StringReplace(Medication[i].Name, ' ', '_', [rfReplaceAll]));
        end
      else if Medication[i].MedType = 2 then
        begin
          cbDrip.Items.Add(StringReplace(Medication[i].Name, ' ', '_', [rfReplaceAll]));
        end
    end;
end;
for i := 0 to Length(Event) - 1 do
begin
  cbEvent.Items.Add(StringReplace(Event[i].Name, '_', ' ', [rfReplaceAll]));
end;
edDose.Clear;

if Event[GetEventIndex(cbEvent.Items[cbEvent.ItemIndex])].Active then
begin
  btnChangeEvent.Caption := 'Deactivate';
end
else
begin
  btnChangeEvent.Caption := ' Activate';
end;

edDose.Text := FloatToStr(Medication[GetMedicationIndex(cbMedication.Items[cbMedication.ItemIndex])].Dose);


cbPump.ItemIndex := 0;
edDrip.Text := FloatToStr(Medication[GetMedicationIndex(cbPump.Items[cbPump.ItemIndex])].Dose);

cbDrip.ItemIndex := 0;
edDrip.Text := FloatToStr(Medication[GetMedicationIndex(cbDrip.Items[cbDrip.ItemIndex])].Dose);
procedure TPatient.cbPumpChange(Sender: TObject);
begin
  edStand.Text := FloatToStr(Medication[GetMedicationIndex(cbPump.Items[cbPump.ItemIndex])].Dose);
end;

procedure TPatient.cbDripChange(Sender: TObject);
begin
  edDrip.Text := FloatToStr(Medication[GetMedicationIndex(cbDrip.Items[cbDrip.ItemIndex])].Dose);
end;

procedure TPatient.cbEventChange(Sender: TObject);
begin
  if Event[GetEventIndex(cbEvent.Items[cbEvent.ItemIndex])].Active then
  begin
    btnChangeEvent.Caption := 'Deactivate';
  end
  else
  begin
    btnChangeEvent.Caption := 'Activate';
  end;
end;

procedure TPatient.btnChangeEventClick(Sender: TObject);
begin
  if btnChangeEvent.Caption = 'Activate' then
  begin
    //activate event
    Event[GetEventIndex(cbEvent.Text)].Active := true;
    UpdateLog('(activate) ' + cbEvent.Text);
  end;
  if btnChangeEvent.Caption = 'Deactivate' then
  begin
    //deactivate event
    Event[GetEventIndex(cbEvent.Text)].Active := false;
    UpdateLog('(deactivate) ' + cbEvent.Text);
  end;
end;

procedure TPatient.UpdateLog(Event: String);
begin
  if DebugMode then
  begin
    ShowMessage('[Debug Message]' + #13#10 + Event);
  end;
  WriteLn(LogFile, '[' + FloatToStr(TimeInCase/1000) + #9 + Event);
end;
end.
Appendix B. Reading and writing Vierkleurenpen

This appendix describes (part of) the structure of the database of the Vierkleurenpen (4kp) anesthesia recording application. It focuses on how to register events in the database and how to update the (display) window of Vierkleurenpen. It is important to keep in mind that writing to the database of 4kp makes you responsible for the validity of the information in the database! The 4kp database is a MS SQL Server database.

There are five types of events that can be registered in the database: the administration of medication, the start of a pump, the attachment of a drip, actions (like incision) and complications. Four tables play a role in the registration of events: tb_events (main table), tb_given_infuus, tb_given_pumps and tb_complicatie_nva. How each of the five types of events is registered in these four tables is described in this text.

This appendix also describes briefly how many (frequently occurring) events are predefined in the 4kp database tables.

The Vierkleurenpen application is written and maintained by dr. Leo van Wolfwinkel from the Universitair Medisch Centrum (UMC) Utrecht.

B.1. Administration of medication

This part of this text described how the administration of medication is registered in the database of 4kp. The table tb_events is used for this purpose, so first the structure of this table is described.

B.1.1. Structure of tb_events

The table tb_events contains eight fields. Each of these fields is explained below. Between brackets the data type of each field is specified.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case_id</td>
<td>int</td>
<td>unique identifier for every case in the database used to link the event to a particular case</td>
</tr>
<tr>
<td>timestamp</td>
<td>datetime</td>
<td>holds the time on which the event was added to the table</td>
</tr>
<tr>
<td>displaytime</td>
<td>datetime</td>
<td>holds the time on which the event actually occurred</td>
</tr>
<tr>
<td>medidose</td>
<td>float</td>
<td>dose of the medication administered</td>
</tr>
<tr>
<td>event</td>
<td>varchar(80)</td>
<td>description of the event</td>
</tr>
<tr>
<td>eventtype</td>
<td>varchar(1)</td>
<td>character indicating the type of event</td>
</tr>
<tr>
<td>verwijderd</td>
<td>datetime</td>
<td>holds the time on which the event was deleted</td>
</tr>
<tr>
<td>id</td>
<td>int</td>
<td>unique identifier for each event</td>
</tr>
</tbody>
</table>

B.1.2. Registering the administration of medication

To register the administration of medication in the database of 4kp add a row to the table tb_events with the following values for the fields. The field id has an auto increment value and should not be filled.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>case_id</td>
<td>[id of current case]</td>
</tr>
<tr>
<td>timestamp</td>
<td>[current time]</td>
</tr>
<tr>
<td>displaytime</td>
<td>[time of occurrence of the event]</td>
</tr>
<tr>
<td>medidose</td>
<td>[dose of medication administered]</td>
</tr>
</tbody>
</table>
B.1.3. Predefined medication

Some frequently used medication is hard coded into the GUI of 4kp. There is also a list of medication defined in the table `tb_drugs_dosing`. This table has the following structure.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>medicijn</td>
<td>name of the medication</td>
</tr>
<tr>
<td>medicijn_afko</td>
<td>common abbreviation of the medication name (allowed to have the same value as medicijn)</td>
</tr>
<tr>
<td>unit</td>
<td>unit of the doses for the medication</td>
</tr>
<tr>
<td>maxdose</td>
<td>the maximum dose for the medication</td>
</tr>
<tr>
<td>mindose</td>
<td>the minimum dose for the medication</td>
</tr>
<tr>
<td>convers</td>
<td>when the dose of the medication is outside the bounds as specified in maxdose and mindose, the dose is multiplied with the value of converse; this feature can be used for medication that can be administered in doses of both $10^{-3}$ and $10^{-6}$</td>
</tr>
<tr>
<td>drugtype</td>
<td>used to group the medication</td>
</tr>
</tbody>
</table>

4kp can recognize abbreviations of medication names when these are defined in a table called `tb_drugs_abbr`. This table has the following structure.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbr</td>
<td>unique abbreviation for the medication</td>
</tr>
<tr>
<td>expanded</td>
<td>medication name as in <code>tb_drugs_dosing</code></td>
</tr>
</tbody>
</table>

This table can be used to connect different names and abbreviations for the same medication to just one entry in `tb_drugs_dosing`.

B.1.4. Example

For example the following command is given during case 0123456789 at time ‘4/4/2008 10:22:34 AM’: ‘profol 150mg 5 minutes ago’. The name ‘profol’ can be recognized as an abbreviation of ‘propofol’ in the table `tb_drugs_abbr`. Than it can be checked whether the dose 150 is in between the two limits of the propofol dose in `tb_drugs_dosing`. The event now can be registered by adding a row containing the following values to `tb_events`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>case_id</td>
<td>0123456789</td>
</tr>
<tr>
<td>displaytime</td>
<td>4/4/2008 10:17:34 AM</td>
</tr>
<tr>
<td>medidose</td>
<td>150</td>
</tr>
<tr>
<td>event</td>
<td>propofol</td>
</tr>
<tr>
<td>eventtype</td>
<td>M</td>
</tr>
<tr>
<td>verwijderd</td>
<td>null</td>
</tr>
</tbody>
</table>
B.2. Start or change of a pump

Given pumps are registered in the table tb_given_pumps. When the stand of a pump is changed a new row is added to the table.

B.2.1. Structure of tb_given_pumps

case_id (int) unique identifier for each case used to link the pump to a particular case
timestmp (datetime) holds the time on which the pump stand was added to the table
displaytime (datetime) holds the time on which the pump stand was actually changed
infuus_vloeistof (smallint) unique identifier for each possible pump fluid
infuus_stand (float) stand of the pump
verwijderd (datetime) holds the time on which the pump stand was deleted
id (int) unique identifier for each pump stand registered
pump_id (int) unique identifier for each pump given (used to group the rows that belong to the same given pump)

B.2.2. Registering (the change of) a pump

When a pump is started or when the stand of a pump is changed, this can be registered by adding a row to the table tb_given_pumps with the following field values. The field id contains an auto increment value and should not be filled. If the pump is started a new pump_id is generated from the current timestamp using the following formula:

\[
pump_id := \text{StrToInt}('hnnsszzz', \text{now}) + \text{random}(100) - 50;
\]

‘hnnsszzz’ Means one or two positions for the hour of the day (no leading 0), two positions for the minutes (leading 0), two positions for the seconds (leading 0) and three positions for the milliseconds (leading 0 or 00). That time string is converted to an integer. Then a random integer between 0 and 100 is added. At the end 50 is subtracted from the resulting number. This ensures a unique id for every pump during a case (a case will never last longer than 24 hours so the combination of case_id and pump_id is always unique). If a pump is changed, the pump_id, of course, is the same as when that pump was started.

case_id [id of current case]
timestmp [current time]
displaytime [time of start or stand change of this pump]
infuus_vloeistof [infusion_fluid_code of the (predefined) fluid in this pump]
infuus_stand [current stand of this pump]
verwijderd null
pump_id [id of current pump]

B.2.3. Predefined pump fluids

The pump fluids that can be in the pumps are defined in the table tb_pump_fluids. This table has the following structure.

infusion_fluid_code (int) unique id for each fluid
fluid_name (varchar(30))  name of the fluid as it should be displayed in lists
fluid_abbr (char(16))  name of the fluid as it should be displayed in the chart
display_priority (int)  used to sort fluids in lists
display_color (int)  identifier of the color of the line in the chart for this fluid
fluid_description (varchar(30))  not used
epidural (smallint)  indicates on which page of the GUI of 4kp the pump can be found

B.2.4. Example
During case 0123456789 at time ‘4/4/2008 2:48:12 PM’ a pump containing ‘morfine’ is started with stand 3. In tb_pump_fluids the infusion_fluid_code of morfine can be found to be 37. At time ‘4/4/2008 2:57:56 PM’ the event is registered in tb_given_pumps with the following field values.

<table>
<thead>
<tr>
<th>case_id</th>
<th>0123456789</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>4/4/2008 2:57:56 PM</td>
</tr>
<tr>
<td>displaytime</td>
<td>4/4/2008 2:48:12 PM</td>
</tr>
<tr>
<td>infuus_vloeistof</td>
<td>37</td>
</tr>
<tr>
<td>infuus_stand</td>
<td>3</td>
</tr>
<tr>
<td>verwijderd</td>
<td>null</td>
</tr>
<tr>
<td>pump_id</td>
<td>24812614</td>
</tr>
</tbody>
</table>

When the stand of the pump is changed to 1 at time ‘4/4/2008 3:28:23 PM’ and this is immediately registered, a new row is added to tb_given_pumps with the following field values.

<table>
<thead>
<tr>
<th>case_id</th>
<th>0123456789</th>
</tr>
</thead>
<tbody>
<tr>
<td>infuus_vloeistof</td>
<td>37</td>
</tr>
<tr>
<td>infuus_stand</td>
<td>1</td>
</tr>
<tr>
<td>verwijderd</td>
<td>null</td>
</tr>
<tr>
<td>pump_id</td>
<td>24812614</td>
</tr>
</tbody>
</table>

B.3. Attachment of a drip
When a drip is attached, this has to be registered in two tables: tb_events and tb_given_infuus. The structure of tb_events is described in the section Administration of medication.

B.3.1. Structure of tb_given_infuus
<table>
<thead>
<tr>
<th>case_id (int)</th>
<th>unique identifier for each case used to link the drip to a particular case</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp (datetime)</td>
<td>holds the time on which the drip was added to the table</td>
</tr>
<tr>
<td>displaytime (datetime)</td>
<td>holds the time on which the drip was actually attached</td>
</tr>
<tr>
<td>infuus_vloeistof (smallint)</td>
<td>unique identifier for each possible drip fluid</td>
</tr>
<tr>
<td>infuus lijn_no (smallint)</td>
<td>identifier of the line of the drip in the chart</td>
</tr>
</tbody>
</table>
volume_bag (smallint) volume of the bag of the drip in ml
bag_nr (smallint) identifier of the bag of the drip
verwijderd (datetime) holds the time on which the drip was deleted
id (int) unique identifier for each drip registered

B.3.2. Registering the attachment of a drip

The attachment of a drip is somewhat more complicated than the other events. One should add a row to both the tables tb_events and tb_given_infuus. For tb_events the fields should have the following values.

case_id [id of current case]
timestamp [current time]
displaytime null
medidose 1
event [size and position of the drip]
eventtype I
verwijderd null

For tb_given_infuus the fields should have the following values. The field id contains an auto increment value and should not be filled.

case_id [id of current case]
timestamp [current time]
displaytime [time of attachment of this drip]
infuus_vloeistof [infusion_fluid_code of the (predefined) fluid in this pump]
infuus_lijn_no [identifier of the line used for this drip]
volume_bag [volume of the bag of this drip]
bag_nr [unique number for this drip in this case]
verwijderd null

B.3.3. Predefined drip fluids

The drip fluids that can be used are defined in the table tb_infusion_fluid_bags. This table has the following structure.

infuus_vloeistof (smallint) unique identifier for this fluid
fluid_type (varchar(10)) type of fluid
fluid_name (varchar(50)) name of this fluid (displayed in selection lists)
fluid_abbr (char(2)) abbreviation for this fluid (displayed on chart)
display_priority (smallint) used to sort fluids in selection lists
volume_1 … volume_5 (smallint) five possible bag volumes

The types that can be used as value for fluid_type are defined in the table tb_infusion_fluid_types. This table has the following structure.

fluid_type (varchar(10)) unique identifier for this fluid type
fluid_color_code (int) ???
fluid_color (varchar(20)) name of the color displayed on the chart
B.3.4. Example
A drip is attached during case 0123456789 at time ‘4/4/2008 4:00:26 PM’. The bag contains 500ml of the fluid voluven, size is ‘18g’ and position is ‘l hand’. No other drips were attached. The event is registered 10 minutes after occurrence. In the table tb_infusion_fluid_bags the infuus_vloeistof of voluven can be found to be 19. A row is added to tb_given_infuus with the following field values.

case_id=0123456789
timestamp=4/4/2008 4:10:26 PM
displaytime=4/4/2008 4:00:26 PM
infuus_vloeistof=19
infuus_lijn_no=1
volume_bag=500
bag_nr=1
verwijderd=null

Also a row is added to the table tb_events with the following field values.

case_id=0123456789
timestamp=4/4/2008 4:10:26 PM
displaytime=null
medidose=1
event=18g l hand
eventtype=I
verwijderd=null

B.4. Actions
All actions not covered by the descriptions above are registered in the table tb_events by adding a row with the following field values.

case_id [id of current case]
timestamp [current time]
displaytime [time of occurrence of the event]
medidose [mostly 0]
event [description of the event]
eventtype E (or T for events indicating the progress of the operation)
verwijderd null

B.4.1. Predefined actions
Some predefined actions are hard coded in the GUI of Vierkleurenpen. Others are defined in tables. tb_regionalblocks and tb_localanesthetics contain actions related to giving blocks and anesthetics. tb_menuitems contains actions indicating the progress of the operation. tb_handelingen contains all actions not covered by the other tables.

tb_regionalblocks has the following structure.

id (int) unique identifier of the block
priority (smallint) used to sort selection lists
block (varchar(80)) name of the block
**drug** (smallint) identifier of the anesthetic used

**ml_dose** (smallint) dose of the anesthetic in ml

tb_localanesthetics has the following structure.

- **priority** (smallint) used to sort selection lists
- **solution** (varchar(80)) (descriptive) name of the anesthetic
- **reg_or_epi** (varchar(10)) indicates whether the anesthetic is used regional (reg) or epidural (epi)

tb_menuitems has the following structure.

- **itemtext** (varchar(30)) menu item text
- **itemorder** (smallint) used to sort in menu
- **itemtype** (char(1)) v
- **visibletoolbar** (smallint) 0

tb_handelingen has the following structure.

- **handel_ident** (smallint) unique identifier for this action
- **handeling** (varchar(50)) descriptive name for this action
- **handel_type** (char(10)) used to group actions in the selection list
- **tegenhandeling_nr** (int) time is counted from the start of this action till start of the action with handel_ident(that) = tegenhandeling_nr(this)
- **cardio** (smallint) indicates whether or not action is displayed if or type is Cardio
- **neuro** (smallint) … if operating room (or) type is Neurochirurgie
- **algemeen** (smallint) … on all types of or
- **kind** (smallint) … if or type is Kind
- **complex** (smallint) … if or type is Complex
- **seh** (smallint)
- **recovery** (smallint) … if or type is Obstetrie
- **overig** (smallint) … if or type is Obstetrie

**B.4.2. Example**

During case 0123456789 at time ‘4/4/2008 2:59:23 PM’ the surgeon makes an incision (predefined action ‘start (incisie)’ from tb_menuitems), which is registered 1 minute later. A row is added to tb_events with the following field values.

case_id=0123456789
timestmp=4/4/2008 15:00:23 PM
displaytime=4/4/2008 2:59:23 PM
medidose=4
event=start (incisie)
eventtype=T
verwijderd=null

(The value 4 for medidose is the first number of itemorder in tb_menuitems.)
B.5. Complications

Complications are registered in the table tb_complicate_nva. This table has the following structure.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case_id</td>
<td>unique identifier of this case</td>
</tr>
<tr>
<td>timestmp</td>
<td>time on which the complication was added to the table</td>
</tr>
<tr>
<td>sjabloonid</td>
<td>identifier of the template for this type of complication</td>
</tr>
<tr>
<td>behandeling</td>
<td>code indicating the treatment for the complication</td>
</tr>
<tr>
<td>consequentie</td>
<td>code indicating the consequences of the implication</td>
</tr>
<tr>
<td>fase</td>
<td>code indicating the fase of the operation when the</td>
</tr>
<tr>
<td></td>
<td>complication occurred</td>
</tr>
<tr>
<td>ernst</td>
<td>code indicating the severity of the operation</td>
</tr>
<tr>
<td>comment</td>
<td>additional information about the complication</td>
</tr>
<tr>
<td>blijvend_belang</td>
<td>indicating whether or not the complication has long term</td>
</tr>
<tr>
<td></td>
<td>implications</td>
</tr>
<tr>
<td>bespreking</td>
<td>indicating whether or not the complication should be</td>
</tr>
<tr>
<td></td>
<td>discussed after the operation</td>
</tr>
<tr>
<td>bespreking_mailed</td>
<td>indicating whether or not the discussion is sent to</td>
</tr>
<tr>
<td></td>
<td>stakeholders</td>
</tr>
</tbody>
</table>

B.5.1. Registering a complication

To register complication in the database of 4kp add a row to the table tb_complicatie_nva with the following values for the fields. The fields behandeling, consequentie, fase and ernst contain codes that are defined in tb_complicatie_nva_subcodes.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case_id</td>
<td>[id of current case]</td>
</tr>
<tr>
<td>timestmp</td>
<td>[current time]</td>
</tr>
<tr>
<td>sjabloonid</td>
<td>[corresponding template id from tb_compli_sjabloon]</td>
</tr>
<tr>
<td>behandeling</td>
<td>[corresponding code from tb_complicatie_nva_subcodes]</td>
</tr>
<tr>
<td>consequentie</td>
<td>[corresponding code from tb_complicatie_nva_subcodes]</td>
</tr>
<tr>
<td>fase</td>
<td>[corresponding code from tb_complicatie_nva_subcodes]</td>
</tr>
<tr>
<td>ernst</td>
<td>[corresponding code from tb_complicatie_nva_subcodes]</td>
</tr>
<tr>
<td>comment</td>
<td>[comment on the complication]</td>
</tr>
<tr>
<td>blijvend_belang</td>
<td>[0 if no, 1 if yes]</td>
</tr>
<tr>
<td>bespreking</td>
<td>[0 if no, 1 if yes]</td>
</tr>
<tr>
<td>bespreking_mailed</td>
<td>[0 if no, 1 if yes]</td>
</tr>
</tbody>
</table>

The table tb_complicatie_nva_subcodes has the following structure.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtype</td>
<td>code indicating the type of subcode</td>
</tr>
<tr>
<td>subcode</td>
<td>unique identifier for this subcode</td>
</tr>
<tr>
<td>submeaning</td>
<td>short description of this subcode</td>
</tr>
</tbody>
</table>
B.5.2. Complication templates

Templates for complications are defined in the table \texttt{tb_compli_sjabloon}. This table has the following structure.

- **\texttt{sjabloonid}** (uniqueidentifier) \hspace{1cm} unique identifier for this template
- **\texttt{nva_crs_id}** (uniqueidentifier) \hspace{1cm} identifier for this complication by Nederlandse Vereniging van Anesthesisten (NVA)
- **\texttt{beschrijving_kort}** (varchar(100)) \hspace{1cm} short description of the complication
- **\texttt{subgroep_code}** (smallint) \hspace{1cm} used to group the complications
- **\texttt{volgorde}** (smallint) \hspace{1cm} used to sort the complications in selection lists

B.6. Updating the Vierkleurenpen client window

An external application can force the Vierkleurenpen client to update the information in it’s window using the information in the database. With this method it’s possible to register events using an external application and make these events visible in the 4kp client’s charts.

There are two ways to force the update. The first is by sending a Windows Message to Vierkleurenpen with id \texttt{(wm_user + 15)}. This message forces the client to perform the update.

The other way uses the internal message system of 4kp. Add an entry to the \texttt{tb_messages} with the following field values. For the use with Gaston this method has the advantage that a query can be written to perform this task. This query can be coupled to an ontology item. In this way the update can be incorporated everywhere in a guideline.

```
from_ok=system
to_ok=[number of ok for the vierkleurenpen client]
verzendtijd=[current time]
gelezen=null
inhoud=update client
```

The client polls the \texttt{tb_messages} every 30 seconds. When an entry is discovered with \texttt{inhoud=’update client’} and \texttt{gelezen=null}. The client forces itself to perform the update.

A little trick can be used to prevent the \texttt{tb_messages} from getting filled with ‘update client’ entries. Add only one entry with \texttt{inhoud=’update client’} and set its \texttt{gelezen} field to null every time you want to perform the update. For more consistency one can update the \texttt{verzendtijd} field to hold the current time simultaneously (at this moment the \texttt{verzendtijd} field is not used when performing the update but it might be used in the future).
Appendix C. Writing cases for Patient Simulator

This appendix describes how cases for the Patient Simulator testing and training application can be written. The cases are xml files specifying the behavior of the patient’s parameters during the case.

The simulator is build as an state diagram. When a state becomes active, events can be performed. Transitions enable the simulator to switch between states during the simulation. This text describes how states, their events and the transitions between states can be specified in the case’s xml file.

A xml file has to start with the following line

```xml
<?xml version="1.0" encoding="UTF-8"?>.
```

The complete remaing part of the document has to be contained in a so called root element. For a Patient Simulator case file this element is called ‘scenario’:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
</scenario>.
```

A case file should contain one element ‘information’ with information about the particular case. This information is specified using the tags ‘name’, ‘description’ and ‘defaultstate’. Within these tags respectively the name of the case, a case description and the state in which the case starts are specified:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
  <information>
    <name>Malignant Hyperthermia</name>
    <description>Malignant Hyperthermia (MH) is a rare genetic disorder of metabolism that, when triggered by drugs used in anesthesia, creates a major crisis. Once a diagnosis of MH has been made, a complex and specific treatment plan needs to be implemented quickly and efficiently to prevent fatal outcome. This scenario is designed to practice the treatment scheme as produced by the Malignant Hyperthermia Society of the United States (MHAUS; www.mhaus.org).</description>
    <defaultstate>stable</defaultstate>
  </information>
</scenario>.
```

A case file contains at least one (in general at least two) elements of type ‘state’. A state contains one tag called ‘name’ that specifies the name of the case (spaces are not allowed in the name but special characters are) and at least one tag called ‘transition’ that specifies a transition between two states (only in the final state it is allowed to have no ‘transition’ tags):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
  <information>
    <name>Malignant Hyperthermia</name>
```
Malignant Hyperthermia (MH) is a rare genetic disorder of metabolism that, when triggered by drugs used in anesthesia, creates a major crisis. Once a diagnosis of MH has been made, a complex and specific treatment plan needs to be implemented quickly and efficiently to prevent fatal outcome. This scenario is designed to practice the treatment scheme as produced by the Malignant Hyperthermia Society of the United States (MHAUS; www.mhaus.org).

Besides a ‘name’ tag and one or more ‘transition’ tags a state can contain one or more ‘event’ tags. These tags describe the events that occur when that particular state becomes active:

<?xml version="1.0" encoding="UTF-8"?>
<scenario>
  <information>
    <name>Malignant Hyperthermia</name>
    <description>Malignant Hyperthermia (MH) is a rare genetic disorder of metabolism that, when triggered by drugs used in anesthesia, creates a major crisis. Once a diagnosis of MH has been made, a complex and specific treatment plan needs to be implemented quickly and efficiently to prevent fatal outcome. This scenario is designed to practice the treatment scheme as produced by the Malignant Hyperthermia Society of the United States (MHAUS; www.mhaus.org).</description>
  </information>
  <defaultstate>stable</defaultstate>
  <state>
    <name>stable</name>
    <transition>if time_in_state > 5 minute(s) then go to start_of_mh</transition>
  </state>
  <state>
    <name>dead</name>
    <event>set heartrate to 0</event>
    <event>set meanabp to 0</event>
  </state>
</scenario>.

The transition specification should have the following structure:

1. the word if, followed by a single space character and
2. the name of a physiological parameter (see the list of parameters later on in this text) or medication, followed by a single space character and
3. a comparison operator (, <=, =, >=, >), followed by a single space character and
4. a value for the parameter entered, followed by a single space character and
5. the words then go to, followed by a single space character and
6. the name of the state to go to.

A special case is the ‘parameter’ time_in_state that can be used to specify a maximum for the time that can be resided in the particular state. (eg. When no action is taken within a certain amount of time the condition of the patient can become worse.) For this parameter the value of point 4 is followed by the word minute(s):

\[
\text{if time_in_state} > 5 \text{ minute(s)} \text{ then go to start_of_mh.}
\]

Remark: because of the use of the ‘<’ character as starting character of a tag in xml the comparison operator ‘less than’ should be written as &lt; (the so called html entity for the ‘<’ character) within a transition tag.

\[
\text{if temp2 < 3800 then go to treat_dysrhythmias.}
\]

Remark: the name of a medication must be identical to the name of that medication in the xml file or database table defining medication. The only difference is that all space characters in the name must be replaced by an underscore (_). In the popup windows of the Patient Simulator application all underscores are replaced by space characters.

\[
\text{if succinylcholine_50_mcg/kg = 0 then go to succ_stopped}
\]

The specification of an event has the following structure:

1. the word set, followed by a single space character and
2. the name of a physiological parameter (see the list of parameters later in this text), followed by a single space character and
3. the word to, followed by a single space character and
4. a value for the parameter entered.

\[
\text{set heartrate to 0}
\]

That is it for a parameter that should get the specified value at once. In general however there will be an increasing or decreasing parameter value with a certain amount in a certain time span. In that case the specification of the event is extended with:

5. a single space character, followed by the word over, followed by a single space character and
6. a value for the time span in minutes, followed by a single space character and
7. the word minute(s).

\[
\text{set heartrate to 120 over 10 minute(s)}
\]

It is also possible to let the simulated patient show symptoms and signs. During a simulation a popup window indicating a symptom is shown when a specific state becomes active. A symptom description can be specified using the ‘sign’ tag:

\[
\text{<state>}
\]
\[
\text{...}
\]
\[
\text{<sign>heavy sweating</sign>}
\]
\[
\text{...}
\]
\[
\text{</state>}
\]
It is possible to embed multiple sign tags within one state tag. The symptoms that are shown during the simulation are recorded in a list with a mark for the time of occurrence (in seconds passed since the start of the simulation). This list can be accessed during the simulation from the application’s popup menu.

It is also possible to show a message to the user when a certain state becomes active. This can be specified in the xml file by using the ‘msg’ tag:

```
<state>
  ...
  <msg>You have been to slow treating this patient. A complication occurred that is beyond the scope of this case. Please restart the case and try again.</msg>
  ...
</state>
```

It is possible to embed more than one message tags within one state tag.

A complete case description in xml could look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
  <information>
    <name>Malignant Hyperthermia</name>
    <description>Malignant Hyperthermia (MH) is a rare genetic disorder of metabolism that, when triggered by drugs used in anesthesia, creates a major crisis. Once a diagnosis of MH has been made, a complex and specific treatment plan needs to be implemented quickly and efficiently to prevent fatal outcome. This scenario is designed to practice the treatment scheme as produced by the Malignant Hyperthermia Society of the United States (MHAUS; www.mhaus.org).</description>
  </information>
  <defaultstate>stable</defaultstate>
  <state>
    <transition>if time_in_state > 5 minute(s) then go to start_of_mh</transition>
  </state>
  <state>
    <name>start_of_mh</name>
    <transition>if heartrate > 119 then go to increased_heartrate</transition>
    <transition>if meanabp > 179 then go to increased_meanabp</transition>
  </state>
  <state>
    <name>increased_heartrate</name>
    <transition>if temp1 > 3949 then go to severe_mh</transition>
    <transition>if temp2 > 3949 then go to severe_mh</transition>
  </state>
  <state>
    <name>increased_meanabp</name>
    <transition>if temp1 > 3949 then go to severe_mh</transition>
  </state>
</scenario>
```
<state>
 <name>severe_mh</name>
 <transition>if 100%_oxygen >= 10 then go to oxygen_administered</transition>
 <transition>if succinylcholine_50_mcg/kg = 0 then go to succ_stopped</transition>
 <transition>if time_in_state > 60 minute(s) then go to dead</transition>
</state>

<state>
 <name>oxygen_administered</name>
 <transition>if succinylcholine_50_mcg/kg = 0 then go to administer_dantroline</transition>
</state>

<state>
 <name>succ_stopped</name>
 <transition>if 100%_oxygen >= 10 then go to administer_dantroline</transition>
</state>

<state>
 <name>administer_dantroline</name>
 <transition>if dantrolene_IV_bolus >= 2.5 then go to decreased_mh</transition>
</state>

<state>
 <name>decreased_mh</name>
 <transition>if dantrolene_IV_bolus >= 2.5 then go to decreased_heartrate</transition>
</state>

<state>
 <name>decreased_heartrate</name>
 <transition>if meanabp < 141 then go to cool_patient</transition>
</state>

<state>
 <name>cool_patient</name>
 <transition>if temp1 < 3800 then go to treat_dysrhythmias</transition>
 <transition>if temp2 < 3800 then go to treat_dysrhythmias</transition>
</state>
The following 16 physiological parameters are known by the simulator:

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>heartrate</td>
<td>Heart rate</td>
</tr>
<tr>
<td>2</td>
<td>saturation</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>meannibp</td>
<td>Mean non invasive blood pressure</td>
</tr>
<tr>
<td>4</td>
<td>meanabp</td>
<td>Mean arterial blood pressure</td>
</tr>
<tr>
<td>5</td>
<td>meancvp</td>
<td>Mean central venous pressure</td>
</tr>
<tr>
<td>6</td>
<td>temp1</td>
<td>Temperature</td>
</tr>
<tr>
<td>7</td>
<td>temp2</td>
<td>Temperature</td>
</tr>
<tr>
<td>8</td>
<td>sevo</td>
<td>End-expired sevoflurane concentration</td>
</tr>
<tr>
<td>9</td>
<td>capno</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>papmean</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>resprate</td>
<td>Respiration rate</td>
</tr>
<tr>
<td>12</td>
<td>O2</td>
<td>Oxigen concentration</td>
</tr>
<tr>
<td>13</td>
<td>n2o</td>
<td>End-expired nitrous oxygen concentration</td>
</tr>
<tr>
<td>14</td>
<td>sysabp</td>
<td>Systolic arterial pressure</td>
</tr>
<tr>
<td>15</td>
<td>diaabp</td>
<td>Diastolic arterial pressure</td>
</tr>
<tr>
<td>16</td>
<td>pmax</td>
<td></td>
</tr>
</tbody>
</table>
Appendix D. Defining medication and events for Patient Simulator

This appendix describes how to define the medication and events that can be administered or applied to the simulated patient in the Patient Simulator application. This information can be read from the Vierkleurenpen database or defined by hand in xml files.

D.1. Reading medication and events from Vierkleurenpen

In order to be able to read the medication and events data from the Vierkleurenpen database the config.ini configuration file of the Patient Simulator application should contain the following information. (The config.ini file can be found in the ‘config’ folder that is contained in the Patient Simulator’s root folder.)

The [general settings] section of the configuration file should contain the line:

```
informationsource=db
```

Further more the should be a section [db server] containing the following lines that specify the database connection string:

```
Provider=SQLOLEDB.1
Integrated Security=SSPI
Persist Security Info=False
Initial Catalog=OK
Data Source=MBS-NB162
Use Procedure for Prepare=1
Auto Translate=True
Packet Size=4096
Workstation ID=MBS-NB162
Use Encryption for Data=False
Tag with column collation when possible=False
```

The values of the parameters should be modified according to the local situation. In general, when using the application with the MS SQL Server type database of Vierkleurenpen, only the Data Source and Workstation ID parameters have to be changed.

D.2. Specifying medication in a xml file

When specifying by hand the medication that can be administered to the simulated patient the following lines should appear in the [general settings] section of the config.ini configuration file of the Patient Simulator application. (The config.ini file can be found in the ‘config’ folder that is contained in the Patient Simulator’s root folder.)

```
informationsource=xml
medicationlistfilename=<name of the xml file containing the medication specification>
```

The xml file containing the medication specification should reside in the ‘lists’ folder that is contained in the Patient Simulator’s root folder. The following part of this text described the structure of such a xml file.
A xml file should always start with the following line

```xml
<?xml version="1.0" encoding="UTF-8"?>.
```

All the remaining content of the xml file should be contained in the so called root element. For a medication list xml file this element is called ‘medicationlist’:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<medicationlist>
</medicationlist>.
```

A medication list contains elements of the type ‘medication’ only. A medication contains four tags named ‘id’, ‘type’, ‘name’ and ‘unit’:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<medicationlist>
    <medication>
        <id>1</id>
        <type>0</type>
        <name>propofol</name>
        <unit>mg</unit>
    </medication>
</medicationlist>.
```

The ‘id’ tag contains a unique number that is used as the identification number for that particular medication. The ‘type’ tag specifies the type of medication according to the table shown below. In the ‘name’ tag the name of the medication can be specified and the ‘unit’ tag contains the unit in which the medication value is registered. The name tag should not contain any space characters. One should use an underscore (_) instead. Underscores are replaced with space characters in the popup windows of the Patient Simulator application. For a pump the unit must be ‘(stand)’, indicating that a pump has a certain stand.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<medicationlist>
    <medication>
        <id>1</id>
        <type>0</type>
        <name>propofol</name>
        <unit>mg</unit>
    </medication>
    <medication>
        <id>12</id>
        <type>1</type>
        <name>propofol_10_mg/l</name>
        <unit>(stand)</unit>
    </medication>
    <medication>
        <id>39</id>
        <type>2</type>
        <name>ringerlactaat</name>
        <unit>ml</unit>
    </medication>
</medicationlist>.
```
<table>
<thead>
<tr>
<th>type id</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Medication that is administered at once</td>
</tr>
<tr>
<td>1</td>
<td>Pump</td>
</tr>
<tr>
<td>2</td>
<td>Drip</td>
</tr>
</tbody>
</table>

**D.3. Specifying events in a xml file**

When specifying by hand the events that can be applied to the simulated patient the following lines should appear in the [general settings] section of the config.ini configuration file of the Patient Simulator application. (The config.ini file can be found in the ‘config’ folder that is contained in the Patient Simulator’s root folder.)

```plaintext
informationsource=xml
eventlistfilename=<name of the xml file containing the events specification>
```

The xml file containing the events specification should reside in the ‘lists’ folder that is contained in the Patient Simulator’s root folder. The following part of this text described the structure of such a xml file.

An xml file should always start with the following line

```plaintext
<?xml version="1.0" encoding="UTF-8"?>.
```

All the remaining content of the xml file should be contained in the so called root element. For a event list xml file this element is called ‘eventlist’:

```plaintext
<?xml version="1.0" encoding="UTF-8"?>
<eventlist>
</eventlist>.
```

An event list contains elements of the type ‘event’ only. An event contains two tags named ‘id’ and ‘name’:

```plaintext
<?xml version="1.0" encoding="UTF-8"?>
<eventlist>
  <event>
    <id>1</id>
    <name>cool_patient</name>
  </event>
</eventlist>.
```

The ‘id’ tag holds a unique number that is used as the identification number of that particular event. The ‘name’ tag holds a name for that event. The name tag should not contain any space characters. One should use an underscore (_) instead. Underscores are replaced with space characters in the popup windows of the Patient Simulator application.
Appendix E. How to use the Patient Simulator

When the Patient Simulator application is started (e.g. by double clicking on the icon of the executable) a heart with a bandage is shown in the system tray. When clicking on the system tray icon with the right mouse button, the Patient Simulator popup menu is shown. See the figure below.

All menu options are disabled except for ‘Start’ and ‘Exit’. To load a case and start the simulation click ‘Start’. A file selection dialog is shown. The dialog shows the content of the ‘scenarios’ folder residing in the Patient Simulator’s root folder. If the case file you want to run is in another folder, you can browse to the appropriate location. Select the case file you want to run and click ‘Open’. See the figure below.
The selected case is now loaded into working memory and the simulation is started automatically. First a popup message is shown with the name of the current case. After that a second popup messages shows a description of the case. Then the simulator starts running on the background. Except for the situation that the simulator runs in debug mode (see later on in this text) only symptom popup messages and case messages are shown.

If the Patient Simulator application’s icon in the system tray is not bandaged a simulation is running. During a simulation more menu options are enabled. See the figure below.

**Stop**
Stop the currently running case and wait for an user action

**Restart**
Stop the currently running case and start it again from the beginning

**Show Current State**
Show a popup message with the state the simulation is currently in
Show Signs
Show a popup message with the signs and symptoms occurred so far during the simulation.

Medication and Events
This menu option opens the graphical user interface of the application and shows a form that can be used to administer medication or apply an event to the simulated patient.
When a medication is selected from the ‘Medication’ listbox, the current dose is shown in the upper right of the window. When a dose is entered in the textbox and the button ‘Administer’ is clicked, the dose entered is added to the current dose.

When a pump is selected from the ‘Pump’ listbox, the current stand of the pump is shown. When the stand is changed and the ‘Change Pump’ button is clicked, the pump stand is changed to the new value.

When a drip is selected from the ‘Drip’ listbox, the current volume of the drip is shown. When the volume is changed and the ‘Change Drip’ button is clicked, the drip volume is changed to the new value.

When an event is selected from the ‘Event’ listbox, the caption of the button below the listbox is changed according to the state of the selected event. If the event is active the caption is changed to ‘Deactivate’. When the button is clicked the selected event is deactivated. If the event is not active the caption is ‘Activate’. When the button is clicked the selected event is activated (applied to the simulated patient).

**Exit**
The application is terminated.

Remark: the ‘Case Editor’ and ‘Medication List Editor’ menu options are disabled in all states of the current version of the Patient Simulation application. See the
recommendations in the main part of this report on how the usability of the application can be improved by implementing simple editors to write case xml files.