Hochschule Ulm

University of
Applied Sciences

Entwicklung einer HL7 Schnittstelle zur elektronischen Dokumentation von Blutgasanalysewerten

Bachelorarbeit

an der Hochschule Ulm

Fakultät Informatik

Studiengang Medizinische Dokumentation und Informatik

vorgelegt von

Tobias Bronsch

März 2010

Gutachter:

Prof. Dr. med. Tibor I. Kesztyüs

Prof. Dr. med. Jörg Lehmann

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Zitate als solche kenntlich gemacht und mit genauer Quellenangabe dargelegt habe.

Ort, Datum	Unterschrift

Danksagung

Ich möchte an dieser Stelle meinen Betreuern, Herrn Dr. Tewes, Herrn Professor Kesztyüs, sowie Herrn Kuhn und Herrn Sing für Ihre sehr gute Unterstützung bei dieser Bachelorarbeit danken.

Besonderer Dank gilt auch meiner Familie, die mir immer zur Seite stand und mir dieses Studium erst ermöglichte.

Abstract

Das Ziel dieser Bachelorarbeit war es, eine Schnittstelle zu entwickeln, welche die automatisierte, elektronische Dokumentation von Blutgasanalysewerten eines Radiometer^{TM} ABL725^{TM} Blutgasanalysators ermöglichen sollte. Die Lösung sollte die papierene Dokumentation dieser Werte ersetzen. Das Projekt fand in der EDV-Abteilung des Universitätsklinikums Ulm, Abteilung HNO, statt. Das BGA-Gerät befand sich dabei auf der Intensivstation der HNO-Klinik.

Die Lösung des Problems wurde unter Zuhilfenahme des Free Pascal Compilers, der OpenSource Entwicklungsumgebung Lazarus, den Zeos Datenbankobjekten, der LNET Netzwerkkomponenten, sowie dem Datenbankmanagementsystem MySQL erreicht. Als Betriebssystem wurde Microsoft Windows Server 2003 genutzt.

Die Arbeit beschäftigt sich mit der vorgefundenen Situation und der Aufgabenstellung, der Anforderungen an die Applikation, sowie deren Implementierung und der Diskussion der Ergebnisse. Es soll hierbei auch besonderes Augenmerk auf die möglichen Risiken beim Einsatz der Lösung gelegt werden.

Inhaltsverzeichnis

1 Einleitung

Das Pflegepersonal der Intensivstation des Universitätsklinikums Ulm, Abteilung HNO[1], nutzt zur Ermittlung der BGA[2]-Werte eines Patienten einen Blutgasanalysator der Firma Radiometer™ , Modell ABL725™.(1)

Die bei einer BGA ermittelten Werte werden derzeit zur Dokumentation auf einem geräteinternen Drucker ausgedruckt und mit in die papierene Patientenakte gelegt. Die Patientenakte wird nach der Entlassung des Patienten archiviert.

Gleichzeitig hat sich in der HNO eine umfangreiche elektronische Patientenakte, genannt ePA, etabliert. Dieses Abteilungsinformationssystem, entwickelt vom Leiter der HNO-EDV Abteilung, Herrn Dr. Siegfried Tewes, hat das Potenzial, die papierene Dokumentation in der HNO vollständig zu ersetzen, was sowohl Kosten sparen als auch Wartezeiten für Patienten reduzieren würde.

1.1 Ziele

Das Verfahren der papierenen Dokumentation der BGA-Werte soll im Rahmen dieser Bachelorarbeit abgeschafft werden. Das bedeutet, dass die Werte elektronisch aus dem Gerät ausgelesen und in einer Datenbank gespeichert werden müssen. Die Idee ist, dass die ermittelten Werte dann zeitnah in der ePA angezeigt werden, sobald die BGA abgeschlossen ist.

Es soll eine Applikation entwickelt werden, die eine Verbindung mit dem BGA-Gerät aufbaut, die ermittelten Werte ausliest und in die ePA-Datenbank speichert. Dabei ist zu beachten, dass der Patient eindeutig identifiziert werden kann.

Um diese Patientenidentifikation zu ermöglichen, soll der Arbeitsablauf auf der Intensivstation auf eine Art und Weise angepasst werden, die einerseits wenig Aufwand für das Personal bedeutet, andererseits die sichere, eindeutige Zuordnung der BGA-Werte zum Patienten ermöglicht.

Die Intensivstation verfügt über vier Patientenbetten. Die Verwechslungsgefahr von Patienten ist auch bei einer solchen, vergleichsweise kleinen Station gegeben und muss unbedingt vermieden werden. Die korrekte Ermittlung der BGA-Werte sowie die eindeutige Zuordnung zum Patienten hat also absolute Relevanz.

1.2 Nutzen für die Klinik

Die elektronische Dokumentation der BGA-Werte ist von Vorteil, um auch von ausserhalb der Intensivstation über den aktuellen Zustand des Patienten im Bilde zu sein, sowie um während

[1] HNO: Hals-, Nasen- und Ohrenheilkunde
[2] BGA: Blutgasanalyse

seines Aufenthalts und nach seiner Entlassung leicht an die Werte zu kommen, ohne dass die papierene Patientenakte vorliegen bzw. gesucht werden müsste.

Desweiteren könnten anhand des Datums der Analyse für jeden Patienten alle jemals ermittelten Werte für einen definierten Zeitraum angezeigt werden, um die Entwicklung der Werte über die Zeit des Aufenthalts leicht nachvollziehen und bewerten zu können.

In Zukunft könnte die papierene Patientenakte durch eine noch weiter gehende, elektronische Dokumentation verschiedener anderer Patientendaten von z.B. intensivmedizinischen Gerätschaften gänzlich wegfallen.

Diese Arbeit ist also als eine Art Zwischenschritt hin zu einer gänzlich elektronischen Patientenakte zu verstehen.

1.3 Blutgasanalyse

Unter einer Blutgasanalyse versteht man die in der Anästhesie und Intensivpflege routinemäßig durchgeführte Ermittlung von Blutgas- und Oxymetriewerten, aber auch von Elektrolyten- und Metabolitenkonzentrationen mit Hilfe von Blutgasanalysatoren.

Das dafür benötigte Blut kann arterieller (z.B. A. radialis[3]), venöser (z.B. über einen Katheter aus der V. cava superior[4]), gemischtvenöser (Pulmonaliskatheter[5]) oder kappillärer (z.B. über ein Ohrläppchen) Natur sein. Arterielles Blut eignet sich für eine BGA am Besten, weil sich damit zuverlässige Aussagen über den Sauerstoffstatus des Patienten in Verbindung mit seiner Hämoglobinkonzentration treffen lassen.

Beispielhafte Werte sind:

- Blutgas: Sauerstoffpartialdruck im Blut, pO_2.
- Oxymetrie: Gesamthämoglobinkonzentration, ctHb.
- Elektrolyte: Kaliumionenkonzentration, cK^+.
- Metaboliten: Glucose-Konzentration, cGlu.

Die ermittelten Parameter werden zur Diagnose und Therapie benötigt und können bspw. auf eine Störung im Säure-Basen-Haushalt hinweisen.(2)

[3] Arteria radialis: Speichenarterie
[4] Vena cava superior: Obere Hohlvene
[5] Zugang bis zur Arteria pulmonalis, der Lungenarterie

2 Material und Methoden

2.1 Verwendete Software und Technologien

Für die Lösung der Aufgabenstellung wurden verschiedene Programme und Programmiersprachen benutzt. Im Folgenden werden diese vorgestellt.

2.1.1 Free Pascal Compiler

Der Free Pascal Compiler (auch FPC genannt) ist ein Open-Source Pascal Compiler der ein hohes Maß an Delphi Kompatibilität bietet und auf verschiedenen Plattformen, inklusive Windows, Mac OS X und Linux verfügbar ist.

FPC-Programme können leicht auf unterschiedlichen Betriebssystemen zum Laufen gebracht werden. Es genügt in aller Regel ein einfaches Kompilieren auf dem Zielsystem, um z.B. aus einem Windows-Programm ein Linux-Programm zu machen. Dadurch entstand auch das Motto des FPC: „Write once, compile anywhere.“.(3) (Version: 2.2.4)

2.1.2 Lazarus Entwicklungsumgebung

Lazarus ist eine Open-Source Entwicklungsumgebung für Rapid Application Development[6] (RAD), welche auf FPC aufbaut. Lazarus enthält eine Komponentenbibliothek, die höchst kompatibel mit Delphis Visual Component Library (VCL) ist.

Beide, Free Pascal und Lazarus sind in Pascal geschrieben.(3) (Version: 0.9.28.2)

2.1.3 TCP

TCP (Transmission Control Protocol) ist ein Netzwerkprotokoll, das die Übertragung von Daten inklusive einer Überprüfung der Richtigkeit der Übertragung ermöglicht. TCP stellt eine Punkt-zu-Punkt Verbindung zwischen zwei Sockets[7] her, auf der in beide Richtungen Daten verschickt und empfangen werden können.

TCP setzt auf dem Internet Protocol (IP) auf, weshalb auch oft von TCP/IP die Rede ist(5).

[6] Rapid Application Development: Vorgehensmodell zur schnellen Umsetzung von Anforderungen in Programmcode

[7] Socket: Softwaremodul zur Verbindung mit einem anderen Socket über ein Rechnernetz hinweg(4)

2.1.4 LNET

Die LNET[8]-Komponenten sind eine Sammlung von Netzwerkkomponenten zur Kommunikation über verschiedene Netzwerkprotokolle, bspw. über TCP[9] oder UDP[10] in Lazarus. Die LNET Komponenten sind dabei eventgesteuert, d.h., dass auf Ereignisse wie das Empfangen einer Nachricht reagiert werden kann, sobald das Event ausgelöst wurde.

Die LNET Komponenten bestehen dabei aus folgenden Teilen:

- LTCPComponent
- LUDPComponent
- LTelnetClientComponent
- LFTPClientComponent
- LSMTPClientComponent
- LHTTPClientComponent
- LHTTPServerComponent
- LSSLSessionComponent

Jede dieser Komponenten behandelt das im Namen angegebene Protokoll. Die LTCPComponent regelt TCP-Verbindungen, die LUDPComponent UDP, usw. Die Komponenten LTCP- und LUDPComponent können dabei jeweils als Server- oder Clientkomponente genutzt werden.

Im Folgenden werden die einzelnen Komponenten beschrieben, wobei der *LTCPComponent* eine deutlich intensivere Beschreibung zufällt, weil sie einen Kernbestandteil der Lösung darstellt und das Verständnis der Komponente hilfreich für das Verständnis der Implementierung ist.

LTCPComponent

Die *LTCPComponent* kann einerseits als Server agieren, indem mit der Funktion *LTCPComponent.Listen(Port:Word)* ein TCP-Serversocket gestartet wird, der auf eine eingehende TCP-Verbindung auf dem genannten Port wartet. Mögliche Werte für den Port sind *0-65535*, was 16

[8] LNET: Lightweight Networking Library

[9] TCP: Transmission Control Protocol, Netzwerkprotokoll, das prüft, ob die übertragenen Daten korrekt sind. Anwendungsfall wäre bspw. das Versenden von E-Mails

[10] UDP: User Datagram Protocol, Netzwerkprotokoll, das keine Prüfung der übertragenen Daten vornimmt. Anwendungsbeispiele sind Sprachübertragung oder Videospiele

bit, also 2^{16} Möglichkeiten entspricht. In der Praxis muss allerdings garantiert werden, dass man einen Port benutzt, der anderweitig nicht benötigt wird.(6)

Andererseits kann die Komponente auch als Client fungieren, indem mit der Funktion *LTCPComponent.Connect(Server:String; Port:Word)* eine Verbindung zu einem TCP-Serversocket aufgebaut werden kann. Die Komponente kann dabei selbstverständlich mit der Auflösung von DNS[11]-Namen umgehen. Im Folgenden werden die implementierten Events beschrieben:

- *OnAccept(aSocket: TLSocket)*
- *OnConnect(aSocket: TLSocket)*
- *OnDisconnect(aSocket: TLSocket)*
- *OnError(const msg: string; aSocket: TLSocket)*
- *OnCanSend(aSocket: TLSocket)*
- *OnReceive(aSocket: TLSocket)*

Sollte ein TCP-Clientsocket eine Verbindung mit einem gestarteten Serversocket der *LTCPComponent* aufbauen, so wird das Event *OnAccept(aSocket: TLSocket)* des *LTCPComponent*-Servers aufgerufen. Als Übergabeparameter wird der betreffende Socket *aSocket* vom Typ *TLSocket* mit übergeben. Eine angemessene Reaktion auf dieses Event wäre bspw. die Information des Nutzers über die eingegangene Verbindung, indem dem Nutzer die IP-Adresse des Clients angezeigt wird. Dies wäre über die Eigenschaft *aSocket.PeerAddress* möglich, die einen String[12] mit der IP-Adresse enthält.

Das Event *OnConnect(aSocket: TLSocket)* wird aufgerufen, sobald sich die *LTCPComponent* als Client mit einem TCP-Serversocket verbunden hat. Übergabeparameter ist wiederum der betreffende TLSocket *aSocket*. Die Reaktion darauf könnte so aussehen, als dass man dem Nutzer die erfolgreiche Verbindungsherstellung aufzeigt.

OnDisconnect(aSocket: TLSocket) wird aufgerufen, wenn eine Verbindung regulär geschlossen wird, bspw. über die Prozedur *LTCPComponent.Disconnect*. Mit übergeben wird der *TLSocket aSocket*, auf dem die Verbindung geschlossen wurde. Der Nutzer könnte hier über das Schließen der Verbindung informiert werden.

[11] DNS: Domain Name System, System zur Zuordnung von Domänennamen wie *www.hs-ulm.de* zu IP-Adressen, nützlich, um sich nicht IP-Adressen merken zu müssen

[12] String: Zeichenkette, die Buchstaben, Zahlen und Sonderzeichen enthalten kann

Bei *OnCanSend(aSocket: TLSocket)* handelt es sich um ein Ereignis, das auftritt, wenn nach einem Sendefehler auf einem Socket wieder gesendet werden kann.

Bei dem Event *OnError(const msg: string; aSocket: TLSocket)* handelt es sich um ein Ereignis, welches bei TCP-Fehlern aufgerufen wird. Ein Beispiel für eine Fehlernachricht ist: *Error on connect: Connection refused.* Übergabeparameter sind einerseits die Fehlermeldung *msg*, andererseits der *TLSocket aSocket*, auf dem der Fehler auftrat. Dieses Event wird *nicht* bei Netzwerkausfällen (Zeitüberschreitung) aufgerufen.

Das für die Anwendung wichtigste Event ist jedoch das *OnReceive(aSocket: TLSocket)*, das dann gerufen wird, wenn eine Nachricht empfangen wurde. Bei einem Empfang kann diese Nachricht über die Funktion *aSocket.GetMessage(out msg:String)* abgerufen und weiter verarbeitet werden. Nach Ausführung der Funktion wird die empfangene Nachricht aus dem Zwischenspeicher im Socket gelöscht und befindet sich nunmehr nur noch im String *msg.*(7)

Sollte die empfangene Nachricht nicht sofort verarbeitet werden, so füllt sich der Speicher im Socket mit jeder weiteren empfangenen Nachricht immer weiter auf. Die empfangenen Nachrichten hängen dann direkt aneinander.

Sollte die Komponente als Client verwendet werden, so kann mit der Funktion *LTCPComponent.SendMessage(const msg:String)* eine Nachricht an den verbundenen ServerSocket verschickt werden.

Die Anwendung der *LTCPComponent* liegt in Bereichen, in denen die korrekte Übertragung der Daten garantiert werden muss, bspw. beim Versand von Laborberichten oder auch von E-Mails.

LUDPComponent

Die *LUDPComponent* verhält sich so wie die *LTCPComponent.* Die Anwendung dieser Komponente könnte im Bereich der Sprach- oder Videoübertragung liegen, da es dabei darauf ankommt, eine verzögerungsarme - da verbindungslose - Verbindung bereitzustellen.

LTelnetClientComponent

Die Komponente *LTelnetClientComponent* regelt den Zugriff auf einen Telnet-Server.

LFTPClientComponent

Die Kommunikation über FTP wird über die *LFTPClientComponent* gelöst. Hier wird für die Transfers eine *LTCPComponent* unter *DataConnection* eingetragen. Eine *LTelnetClientComponent* kann als *ControlConnection* eingebunden werden. Die Komponente unterstützt sowohl die passive als auch die aktive FTP-Transfermethode.

LSMTPClientComponent

Mit der *LSMTPClientComponent* können über das SMTP[13] Protokoll E-Mails verschickt werden. Dazu kann eine Verbindung mit einem SMTP-Server hergestellt und über die Prozedur *LSMTPClientComponent.SendMail(from, recipients, subject, msg:String)* eine E-Mail verschickt werden.
Zur Datenübermittlung wird auch hier eine *LTCPComponent* als *Connection* benötigt.

LHTTPClientComponent

Die *LHTTPClientComponent* bietet die Möglichkeit, sich mit einem HTTP[14] Server zu verbinden und Daten über das HTTP Protokoll mit ihm auszutauschen.

LHTTPServerComponent

Die HTTP Server Komponente kann Verbindungen von HTTP Clients entgegennehmen und auf Requests der Clients reagieren.

LSSLSessionComponent

Die *LSSLSessionComponent* sorgt für die SSL[15]-Verschlüsselung einer HTTP-Verbindung. Dabei muss die *LSSLSessionComponent* als *LTCPComponent.Session* angegeben werden.

(Version: 0.6.2)

2.1.5 SQL

SQL steht im allgemeinen Sprachgebrauch für „Structured Query Language“ und bezeichnet eine Datenbanksprache, welche sich an der englischen Umgangssprache orientiert und die Kommunikation mit und Manipulation von relationalen Datenbanken ermöglicht.

SQL ist laut ANSI[16]-Standard keine Abkürzung, sondern ein eigenständiger Name. Die Beze-

[13] SMTP: Simple Mail Transfer Protocol, Netzwerkprotokoll zum Versand von E-Mails
[14] HTTP: Hypertext Transfer Protocol
[15] SSL: Secure Sockets Layer, Verschlüsselungsprotokoll
[16] ANSI: American National Standards Institute, Institut für die Etablierung von Normen in der Industrie

ichnung SQL entstand aus dem vorherigen Namen SEQUEL, welcher allerdings ein eingetragener Markenname war und daher nicht weiter genutzt werden konnte.

SQL-Befehle lassen sich in drei Kategorien unterscheiden:

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DCL (Data Control Language)

Die DDL enthält Befehle zur Definition eines Datenbankschemas, welches dann mit Hilfe der DML manipuliert werden kann. Unter der Definierung des Schemas wird das Erstellen, Ändern und Löschen von Datenbankstrukturen verstanden. Die Manipulation meint das Einfügen, Auslesen, Ändern und Löschen von Daten. Die DCL wird schliesslich für die Rechteverwaltung und Transaktionskontrolle benötigt. (8)

2.1.6 MySQL

MySQL ist ein relationales Datenbankmanagementsystem, das sowohl als Open Source-, als auch als kommerzielle Version erhältlich ist. Das System gehört seit Januar 2010 der Firma Oracle. MySQL ist für verschiedene Betriebssysteme verfügbar und gilt als Grundlage vieler Webauftritte. Die Open Source Variante ist das populärste Open Source Datenbanksystem der Welt.

MySQL kann über verschiedene Möglichkeiten verwaltet werden, bspw. direkt über die Konsole oder über grafische Oberflächen wie phpMyAdmin[17].(9) (Version: MySQL Server 5.0)

2.1.7 MySQL Workbench

Die in dieser Arbeit gezeigten Datenbankmodelle sind mit Hilfe der MySQL Workbench erstellt worden. Nachdem das Modell grafisch erstellt wurde, kann es kurzerhand als SQL Skript exportiert und auf einem DBMS[18] ausgeführt werden. Die MySQL Workbench enthält alles, was ein Datenmodellierer für die Erstellung komplexer Datenbankmodelle benötigt, und stellt darüber hinaus wichtige Funktionen für die Durchführung von komplexem Änderungsmanagement und Dokumentationsaufgaben bereit, die normalerweise viel Zeit und Mühe kosten. MySQL Workbench ist für Windows, Linux und Mac OS verfügbar.(10) (Version: 5.1.18)

[17] phpMyAdmin Webseite: `http://www.phpmyadmin.net/`
[18] DBMS: Datenbankmanagementsystem

2.1.8 ZeosLib

Die ZeosLib ist eine Open Source Komponentenbibliothek, die - z.B. in Lazarus - zur Arbeit mit Datenbanken genutzt werden kann. Es werden dabei verschiedene Datenbanksysteme unterstützt. Der Wechsel zwischen den verschiedenen Systemen vollzieht sich durch einfaches Auswählen des Datenbanktyps (bspw. MySQL 5), d.h., man muss das jeweilige Programm nur minimal anpassen, wenn man von einem DBS[19]-Anbieter zu einem anderen wechseln möchte.(11)

Im Folgenden werden die für diese Arbeit benötigten Teile der ZeosLib beschrieben.

ZConnection

Die *ZConnection* ist die Komponente, mit der die Datenbankverbindung realisiert wird. Die Verbindung kann dann von anderen Zeos Komponenten genutzt werden. Für eine Verbindung mit einem DBMS[20] werden mindestens folgende Daten benötigt:

- *Hostname* des Servers
- *Benutzername*
- *Passwort*
- *Datenbank*
- *Datenbanktyp* (z.B. MySQL-5)

Anhand dieser Daten kann nun mit der *ZConnection* eine Verbindung mit dem DBMS aufgebaut werden. Dazu wird lediglich die Prozedur *ZConnection.Connect* aufgerufen.

Folgende Datenbanksysteme werden unterstützt:

- ASA / Sybase
- Firebird
- Interbase
- MSSQL
- MySQL
- Oracle

[19] DBS: Datenbanksystem, bestehend aus DBMS und der Datenbank

[20] DMBS: Datenbankmanagementsystem, Programm, das die Zugriffe auf die eigentlichen Nutzdaten eines Datenbanksystems regelt

- PostgreSQL
- SQLite

ZQuery

Die *ZQuery* ist eine Komponente, die den Umgang mit SQL-Abfragen ermöglicht. Die ZQuery benötigt dazu die *ZConnection*, um die SQL-Abfragen an das DBMS absenden zu können. Dazu wird unter der Eigenschaft *ZQuery.Connection* die *ZConnection* mit angegeben.

(Version: 6.6.6_stable)

2.1.9 UMLet

Nahezu alle Diagramme in dieser Arbeit wurden mit *UMLet* erstellt. *UMLet* ist ein leistungsfähiges Open Source Tool für die Erstellung von UML Diagrammen. Die erstellten Diagramme können anschliessend in verschiedene Formate exportiert werden. Das Programm wurde in Java entwickelt und bietet dadurch weitgehende Plattformunabhängigkeit.

Mit *UMLet* können schnell und einfach komplexe Diagramme erstellt werden, da die Bedienung teils maus- und teils tastaturbasiert ist und somit einiges an Zeit gespart wird.(12) (Version: 10.3)

2.1.10 Dia Diagrammeditor

Das in dieser Arbeit gezeigte Klassendiagramm wurde mit dem Diagrammeditor *Dia* erstellt. Dia ist für Linux / UNIX und Windows erhältlich und steht unter der GPL. Dia hat im Vergleich zu UMLet allerdings den Nachteil, dass die Diagramme nur relativ langsam erstellt werden können, da das Programm vorwiegend mit der Maus bedient werden muss und sich die Erstellung eines Diagramms dadurch vergleichsweise zeitintensiv darstellt.(13) (Version: 0.97)

2.2 HL7

2.2.1 Einführung

Die Abkürzung HL7 steht für „Health Level Seven“ und bezeichnet eine Gruppe von Standards zur Kommunikation von Daten im Gesundheitswesen. HL7 wird durch eine gleichnamige Organisation weiterentwickelt(14).

HL7 ist in den Versionen 2.x und 3 verfügbar. Die Version 2.x definiert den Aufbau ASCII[21]-

[21] ASCII: American Standard Code for Information Interchange, amerikanische Zeichenkodierung für den Informationsaustausch

basierter Textdateien, wobei hingegen in der Version 3 eine XML[22]-Struktur verwendet wird. HL7 Version 2.x ist zur Kommunikation von Gesundheitsdaten zwischen medizinischen Informationssystemen gedacht, z.B. von einzelnen Laborberichten zu einem Patienten, wohingegen die Version 3 in Richtung einer ganzheitlichen Patientenakte geht.(15)

Die aktuellste Version der Reihe 2.x ist die Version 2.6(16).

Da das BGA-Gerät die HL7 Version 2.2 unterstützt, wird im Folgenden nur noch auf diese Version eingegangen.

2.2.2 Aufbau von HL7 Version 2.x Nachrichten

Grundsätzlich besteht eine HL7-Datei aus sog. Segmenten, Feldern und darin befindlichen Daten. Ein Segment nimmt in der HL7-Datei eine ganze Zeile ein und besteht aus mehreren, durch ein Trennzeichen getrennten Feldern, welche schliesslich die zu übertragenden Daten beinhalten. Bei der Erstellung einer HL7-Nachricht werden diese Segmente dann wie in einer Art Baukastensystem zusammengesetzt. (15)

[22]XML: Extended Markup Language, Auszeichnungssprache zur Strukturierung von Daten

HL7 Nachrichten sind grundsätzlich nach folgender hierarchischer Struktur aufgebaut:

- Nachricht
  - Segment
    - Feld
      - Komponente u. Subkomponenten

Ein vom BGA verschickter Patientenbericht enthält dabei folgende Segmente:

Segment Type	**Name**
MSH	Header Segment
PID	Patient Information Segment
OBR	Observation Request Segment
{NTE}]	Notes and Comments Segment, 0 oder mehrere Kommentarcodes für die gesamte Nachricht
[{OBX	Observation Result Segment, 0 oder mehr Ergebnissegmente, jedes mit einem Parameter und Wert
[{NTE}]	Optionales NTE-Segment, das das vorherige OBX Segment kommentiert. Ein ABL700^{TM} Series Analysator kann nur 0 oder 1 NTE Segmente pro OBX Segment enthalten
}]	

Tabelle 1: Struktur eines Patientenberichts, (17, Kapitel 9-3, Patient Result - Message Structure)

Das MSH-Segment wird in jeder HL7-Nachricht benötigt und enthält bspw. Informationen über den Sender, den Nachrichtentyp, das Datum der Übermittlung, die verwendete HL7-Version, etc.

Das PID-Segment wiederum gibt Daten zum Patienten an, bspw. die PatID[23], den Namen, Anschrift, Telefonnummer, etc.

Das OBR-Segment dient als Report Header und identifiziert z.B. ein Laborergebnis. Dieses Segment kann in einer Anfrage wie auch in einem Ergebnisbericht vorkommen.

In einem OBX-Segment befindet sich ein ermittelter Wert einer Observation, also bspw. einer Laboruntersuchung, inklusive der Einheit und u.U. sogar Angaben, ob der Wert innerhalb der Norm liegt, etc. Es können bei umfangreicheren Laborergebnissen natürlich viele OBX-Segmente auftreten.

Ein NTE-Segment kommentiert eine ganze Nachricht, wenn es dem MSH Segment folgt, oder ein vorheriges OBX-Segment. Bspw. könnte ein OBX-Segment einen Wert enthalten, der durch einen Fehler in einem Laborgerät entstand. Dieser Fehler (z.B. fehlerhafte Kalibrierung) kann dann in einem nachfolgenden NTE-Segment dokumentiert sein.

Als Trennzeichen zwischen den einzelnen Feldern wird üblicherweise ein senkrechter Strich „|“ verwendet. Direkt nach dem Segmentbezeichner „MSH“ werden die benötigten Trennzeichen für Felder, Komponenten, Subkomponenten, etc. definiert.

Die Segmente werden schliesslich durch ein <cr>(Wagenrücklauf)-Zeichen voneinander getrennt.

Beispielhafte HL7-Datei

Anhand der besprochenen Segmente könnte nun eine HL7-Nachricht konstruiert werden, die den folgenden Inhalt hat:

```
1 MSH|^~\&|Sending^Application|Sending^Facility|||20100222080000||
      ORU^R01|20100222080000|P^not present|2.2
2 PID|1|||0123456789|Max^Mustermann|||U
3 OBR|1||12345^Lab Report|||||||O|||| arteriell^
4 OBX|1|ST|pO2||100|mmHg||N|||F|||20100222075930||
5 NTE|1|X|123
```

Die gezeigte Nachricht enthält u.a. Angaben zum Nachrichtentyp *ORU_R01* (*Unsolicited Observation Message*, ein unaufgefordert gesendeter Laborbericht), das Datum der Nachrichtenübermittlung im Format *JahrMonatTagStundeMinuteSekunde*(17, Kapitel 8-4), Patientendaten wie die PatID *0123456789* und den Namen *Max Mustermann*, sowie ein Laborergebnis, welches sich aus einem ermittelten Parameter, hier pO2 für Sauerstoffpartialdruck, dem Wert *100*, der Einheit *mmHg*, sowie dem Datum der Messung zusammensetzt. Das Laborergebnis ist

[23] Patientenidentifikationsnummer

mit dem nachfolgenden NTE-Segment kommentiert, welches den Kommentarcode *123* enthält.

Die Anzahl der Felder pro Segment kann je nach Implementierung variieren. Es kann sein, dass eine HL7-Implementierung z.B. nicht alle Felder im PID-Segment abbildet, die dort möglich wären. Felder, die nicht benötigt werden, können auch einfach weggelassen werden. Es muss aber garantiert werden, dass die Felder, die benutzt werden, auch dem HL7-Standard entsprechen, d.h., dass sich die benötigten Felder nicht verschieben, wenn Felder weggelassen werden. Die im HL7-Standard definierten Positionen müssen auf jeden Fall eingehalten werden.

Da der Aufbau einer HL7-Nachricht standardisiert ist, kann man davon ausgehen, dass in jedem Feld immer das steht, was dafür im HL7-Standard vorgesehen wurde. D.h., in Feld 1 steht immer der Segmentbezeichner, in Feld Nr. 6 des PID Segments steht der Name des Patienten, etc. Es kann natürlich vorkommen, dass der Name des Patienten fehlt, aber es kann nicht vorkommen, dass dort z.B. seine Telefonnummer zu sehen ist.

Neben den von HL7 standardisierten Segmenten gibt es noch die Möglichkeit, individuelle Segmente, sog. Z-Segmente, zu definieren, um nicht abgedeckte Informationen abzubilden. Bspw. könnten dies Zusatzdaten für die Krankenversichertenkarten sein, wie sie im von der deutschen HL7 Benutzergruppe e.V. entwickelten ZGK Segment implementiert sind.(18)

Die Standardisierung ermöglicht also, dass jeder die HL7-Nachrichten auslesen und verstehen kann, solange entweder keine Z-Segmente benutzt werden, oder diese vorher ausgehandelt und erklärt wurden.

2.2.3 Kommunikation der HL7-Nachrichten

Die Kommunikation von HL7-Nachrichten erfolgt vorwiegend dateibasiert, d.h., dass eine HL7-Datei vom Sender in ein für den Empfänger zugängliches Verzeichnis abgelegt wird. Der Empfänger schaut regelmäßig in dieses Verzeichnis und liest bei Vorliegen einer neuen Nachricht die HL7 Datei ein.

Grundsätzlich ist aber auch die direkte Übermittlung einer HL7-Nachricht an den Empfänger möglich, bspw. über eine TCP-Verbindung. Der Vorteil einer dateibasierten Übertragung liegt darin, dass der Empfänger nicht zwingend verfügbar sein muss, wie es bei einer TCP-Verbindung der Fall sein könnte. Sollte der Empfänger einmal ausfallen, so kann die Nachricht auch später noch eingelesen und verarbeitet werden.

2.3 Radiometer™ ABL725™ Blutgasanalysator

Der Radiometer™ ABL725™ Blutgasanalysator ist dafür bestimmt, menschliches Blut und Expirationsluft zu analysieren. Es werden dabei pH-, Blutgas- und Oxymetriewerte, sowie Elektrolyt- und Metabolitenkonzentrationen ermittelt.(19, Kapitel 1-2, Einführung)
Nachfolgend sind Abbildungen des BGA-Gerätes, Vorder- und Rückansicht, zu sehen:

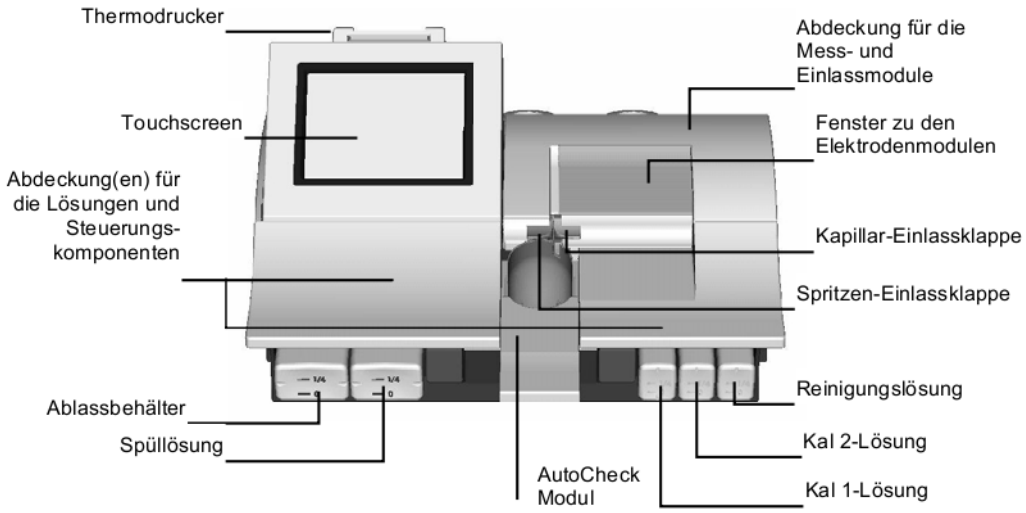

Abbildung 1: ABL725™ BGA, Vorderansicht (19, Kapitel 2, Hauptkomponenten - Vorderseite)

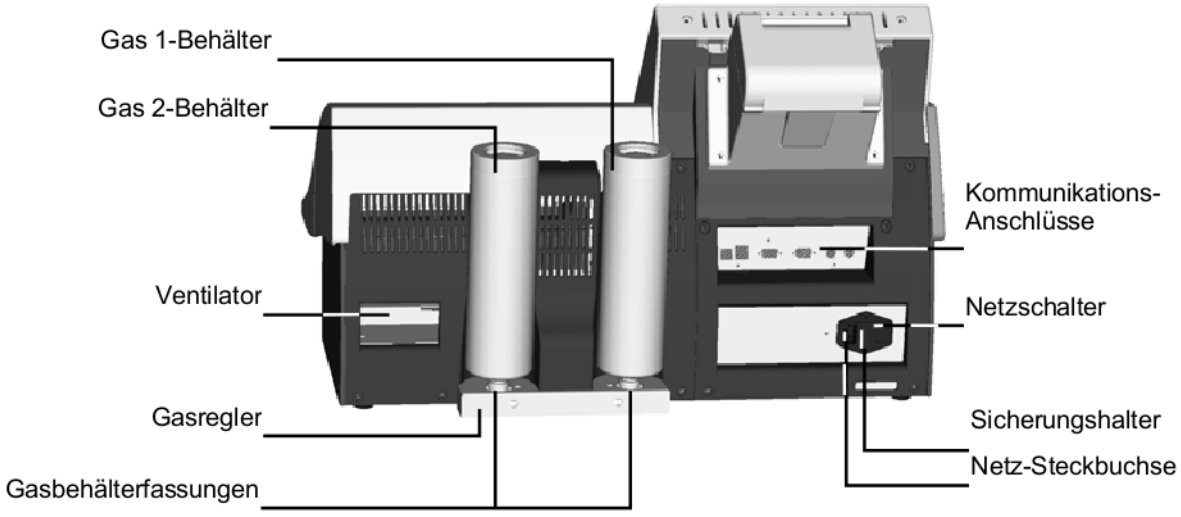

Abbildung 2: ABL725™ BGA, Rückansicht (19, Kapitel 2, Hauptkomponenten - Rückseite)

2.3.1 Ermittelbare BGA-Werte und deren Bedeutung

In der Konfiguration des BGA-Gerätes können unterschiedliche Analyseparameter aktiviert werden.

In der Expirationsluft eines Patienten können die Werte $p\mathrm{CO}_2$ (Kohlendioxidpartialdruck) und $p\mathrm{O}_2$ (Sauerstoffpartialdruck) ermittelt werden (19, Kapitel 1-2, Einführung). Die verfügbaren Parameter für die Blutgasanalyse der ABL700$^{\mathrm{TM}}$ Serie sind dagegen wie folgt definiert:

Parameter	**HL7**	**Einheit**	**Bedeutung**
pH	pH	pH Skala	pH-Wert
pH(T)	pH(T)	pH Skala	pH-Wert, Temparaturkorrektion
$p\mathrm{CO}_2$	pCO2	mmHg kPa	Kohlendioxidpartialdruck
$p\mathrm{CO}_2$(T)	pCO2(T)	mmHg kPa	Kohlendioxidpartialdruck, Temperaturkorrektion
$p\mathrm{O}_2$	pO2	mmHg kPa	Sauerstoffpartialdruck
$p\mathrm{O}_2$(T)	pO2(T)	mmHg kPa	Sauerstoffpartialdruck, Temperaturkorrektion
ctHb	tHb	g/dL g/L mmol/L	Gesamthämoglobinkonzentration
$s\mathrm{O}_2$	sO2	%	Sauerstoffsättigung des Blutes
$c\mathrm{tO}_2$	tO2	%	Gesamtsauerstoffgehalt des Blutes
$F\mathrm{O}_2$Hb	O2Hb	% Fraktion	Oxyhämoglobinfraktion des Gesamthämoglobins
FCOHb	COHb	% Fraktion	Carboxyhämoglobinfraktion des Gesamthämoglobins
FHHb	RHb	% Fraktion	Desoxyhämoglobinfraktion des Gesamthämoglobins
FMetHb	MetHb	% Fraktion	Methämoglobinfraktion des Gesamthämoglobins

fHbF	HbF	% Fraktion	Fetalhämoglobinfraktion des Gesamthämoglobins
$c\mathrm{K}^+$	K+	mmol/L meq/L	Kaliumionenkonzentration
$c\mathrm{Na}^+$	Na+	mmol/L meq/L	Natriumionenkonzentration
$c\mathrm{Ca}^{2+}$	Ca++	mmol/L meq/L mg/dL	Calciumionenkonzentration
$c\mathrm{Cl}^-$	Cl-	mmol/L meq/L	Chloridionenkonzentration
cGlucose	Glu	mmol/L mg/dL	Konzentration von Glukose
cLactat	Lac	mmol/L meq/L mg/dL	Konzentration von Lactat
ctBil	tBil	mol/L mg/dL mg/L	Gesamtbilirubinkonzentration
Temp	T	°C °F	Temperatur des Blutes
Hct	Hct	%	Hämatokritwert
BE	ABE	mmol/L	Aktueller Basenüberschuss
SBC	SBC	mmol/L	Standard Bicarbonat Konzentration ($\mathrm{HCO_3}^-$)
$p50$	p50(act)	mmHg kPa	Sauerstoffpartialdruck bei Halbsättigung
Anionenlücke, (K+)	Anion gap (K+)	mmol/L meq/L	Konzentrationsdifferenz: Anion gap (K^+) = $(cNa^+ + cK^+) - (cCl^- + c\mathrm{HCO_3}^-)$

Tabelle 2: Die BGA-Werte und deren Bedeutung, (19, Technische Daten - Gemessene Parameter)

2.3.2 Kommunikation

Das BGA-Gerät bietet verschiedene Möglichkeiten der Kommunikation an. Technisch gesehen ist die Kommunikation über die serielle Schnittstelle RS232 oder die Ethernet-Anbindung möglich. Das Gerät unterstützt dabei die Kommunikation auf Applikationsebene[24] durch HL7 Version 2.2, ASTM6xx und ASTM[25], auf der Transportebene stehen Serial, Serial (RAW) und TCP/IP zur Verfügung.

Es gibt softwaretechnisch zwei Möglichkeiten, mit dem Gerät zu kommunizieren. Auf der einen Seite bietet die Firma Radiometer^{TM} eine eigens entwickelte Software namens RADIANCE^{TM} an, welche über ein kommerzielles Protokoll mit dem BGA-Gerät kommuniziert. Auf der anderen Seite besteht die Möglichkeit, das Gerät direkt an ein KIS[26] / LIS[27] anzubinden.

Nachrichten

Grundsätzlich werden drei Gruppen von Nachrichten unterschieden, die mit dem oder vom BGA kommuniziert werden können. Es gibt

- Nachrichten, die vom BGA oder RADIANCE^{TM} an das KIS / LIS geschickt werden.
- Nachrichten, die vom BGA oder RADIANCE^{TM} vom KIS / LIS empfangen werden. können
- „Query Response“[28] Nachrichten, die vom BGA oder RADIANCE^{TM} an das KIS / LIS geschickt werden können, welche dann eine Antwort des KIS / LIS erfordern.(17, Kapitel 1-6, Message Types and Message Flow)

[24] OSI-Modell: Modell zur Veranschaulichung der Kommunikation zwischen Computersystemen auf verschiedenen Ebenen

[25] ASTM International: American Society for Testing and Materials, internationale Standardisierungsorganisation

[26] Krankenhausinformationssystem

[27] Laborinformationssystem

[28] Query Response: Eine Anfrage, die eine Antwort erfordert

Desweiteren kann RADIANCE^{TM} als Schnittstelle zwischen dem BGA-Gerät und dem KIS / LIS dienen, wie auf der folgenden Abbildung veranschaulicht:

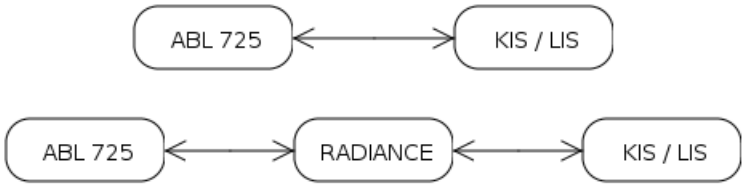

Abbildung 3: Kommunikationsmöglichkeiten ABL725^{TM} mit KIS / LIS, (17, Kapitel 1-2, Message Types and Message Flow)

Es ist also auch möglich, dass RADIANCE^{TM} vom BGA-Gerät empfangene Nachrichten an das KIS / LIS weiterleitet. Die Kommunikation zwischen BGA-Gerät und RADIANCE^{TM} ist proprietär und nicht näher beschrieben.

Eine weitere Möglichkeit ist die Kommunikation vom KIS / LIS mit dem BGA-Gerät, entweder direkt oder indirekt:

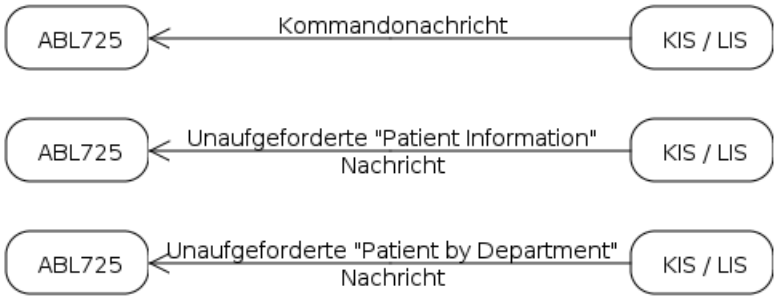

Abbildung 4: Kommunikationsmöglichkeiten KIS / LIS mit ABL725^{TM}, (17, Kapitel 1-2, Message Types and Message Flow)

Eine Kommandonachricht kann das Gerät in einen Zustand *LOCK* setzen und ihn auch durch das Kommando *UNLOCK* wieder daraus befreien.

Eine weitere Kommunikationsmöglichkeit wäre der *Patient Lookup*, d.h., dass über das BGA-Gerät ein *Patient Information Query*, also eine Anfrage zur Identifikation des Patienten, zusammen mit der Fallnummer / PatID an ein KIS / LIS geschickt werden kann, welches dann mit einer *Patient Information Response* weitere Patientendaten an das BGA-Gerät übermitteln

könnte.(17, Kapitel 1-10)

Eine *Patient by Department Query* erfrägt vom KIS / LIS eine Liste von Patienten, die zur Abteilung gehören. Der Nutzer kann dann aus dieser Liste den Patienten händisch auswählen, wenn z.B. die PatID nicht vorliegt.(17, Kapitel 8-22)

2.3.3 Kommunikationsverhalten bei KIS / LIS Verbindung

Sollte das BGA-Gerät eine bestehende Verbindung mit einem KIS / LIS aufweisen, so gibt es vier mögliche Typen von Nachrichten, die das Gerät dem KIS / LIS mitteilen kann.

Es handelt sich dabei um folgende Nachrichten (17, Kapitel 1-6, Messages Sent):

- Patientenbericht
- Qualitätskontrollbericht
- Aktivitätenbericht
- Kalibrierungsbericht

Die folgende Abbildung veranschaulicht den Kommunikationsweg der Nachrichten:

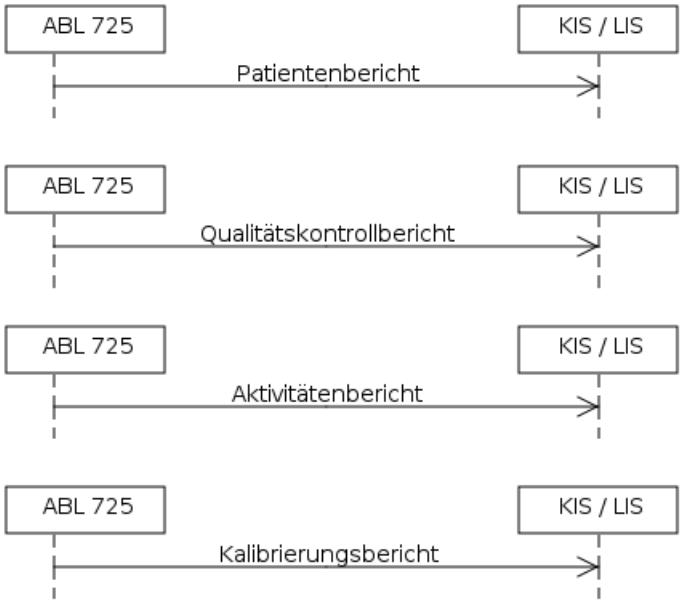

Abbildung 5: Kommunikationsweg der HL7-Nachrichten bei KIS / LIS Verbindung, (17, Kapitel 1-7, Message Flows for Sent Messages)

Diese Nachrichten werden automatisch verschickt, wenn die automatische Sendung der jeweiligen Nachrichtentypen im Gerät aktiviert wurde.(19, Kapitel 5-93, Automatische Datensendung)

Ein „Patientenbericht“ wird übermittelt, nachdem eine BGA-Analyse vorgenommen wurde. Der Bericht enthält alle ermittelten Werte mit optionalen Kommentaren, dem Datum, sowie Informationen zur Identifikation des Patienten. In den Kommentaren können sich Hinweise auf die Richtigkeit bzw. Unrichtigkeit eines ermittelten Wertes befinden.

Ein „Qualitätskontrollbericht“ enthält Daten zur Qualitätssicherung. Eine Qualitätssicherung wird in regelmässigen Abständen durchgeführt und dient zur Kontrolle der Kalibrierungslinie.

Nachrichten des Typs „Aktivitätenbericht“ werden bei Aktivitäten am Gerät erzeugt und übermittelt. Die Nachrichten beinhalten einen oder mehrere Codes, die diverse Aktivitäten spezifizieren. Die genaue Beschreibung einer Aktivität ist allerdings nur in einer lokalen Datei im Gerät hinterlegt und dient lediglich der Information der Anwender über stattgefundene Aktivitäten.

Bei einem „Kalibrierungsbericht“ handelt es sich um das Ergebnis einer Kalibrierung des Gerätes. Die empfangene Nachricht enthält also alle Kalibrierungsergebnisse für jeden Parameter.

Die genannten vier Nachrichtentypen werden jeweils nur einmal gesendet, nämlich genau dann, wenn das jeweilige Ereignis stattgefunden hat. Ein nachträgliches Auslesen des Gerätes ist mit einer KIS / LIS Verbindung nicht möglich.

Das bedeutet also, dass ununterbrochen eine Verbindung mit dem Gerät bestehen muss, um die Nachrichten abfangen und bearbeiten zu können. Ist die Verbindung unterbrochen und findet in dieser Zeit eine Aktivität statt, so sind diese Daten für das KIS / LIS verloren. Die Nachrichten werden lediglich noch lokal im Gerät vorgehalten.

Es besteht die Möglichkeit, die zuletzt stattgefundenen Messungen noch einmal abzusenden. Dazu muss man am Gerät ins Dateiarchiv wechseln und die betreffenden Berichte händisch auswählen und neu versenden. Die Liste enthält jedoch nicht alle jemals stattgefundenen Berichte, sondern nur eine relativ kleine Auswahl.

2.4 Ablauf einer BGA in der HNO-Intensiv

Das Personal der Intensivstation entnimmt dem Patienten mit einer Spritze über einen zuvor gelegten, arteriellen oder venösen Zugang eine Blutprobe, bspw. aus einer Arteria femoralis[29].

Die Spritze mit der entnommenen Blutprobe wird dann an einer Halterung am BGA-Gerät befestigt und das Analyseprogramm wird gestartet. Daraufhin fährt das Gerät eine Nadel aus, über welche das Blut aus der eingehängten Spritze angesaugt wird, um dann die benötigten Werte automatisiert zu ermitteln.

[29] Arteria femoralis: Oberschenkelarterie

Nachdem das Gerät die BGA-Werte ermittelt hat und der Patient identifiziert wurde, kann die Liste der Werte auf einem im Gerät integrierten Thermopapierdrucker inklusive der Daten zur Identifikation des Patienten ausgedruckt werden.

Am Ende der Prozedur legt das Personal derzeit den Ausdruck mit in die Patientenakte.

2.4.1 Identifikation des Patienten

Das Personal identifiziert den Patienten üblicherweise durch händisches Eintippen des Vor- und Nachnamens, oder durch die Identifikation anhand der Bettnummer des Patienten.

Diese Methode ist für das Personal einfach, stellt aber für eine Software ein schwer bis gar nicht lösbares Problem dar. Diese Arbeitsweise muss geändert werden, um die eindeutige Zuordnung des Patienten für eine Software zu ermöglichen.

2.5 Anforderungen an die Applikation

Das Programm soll auf einem Windows Server laufen (Windows Server 2003) und eine grafische Oberfläche mit Statusangaben bieten. Es soll die benötigten Einstellungen für die Datenbank- und TCP-Verbindung aus einer Konfigurationsdatei einlesen und alle nötigen Schritte absolut autonom ausführen.

Zusätzlich soll das Programm möglichst plattformunabhängig sein, um bei einem Wechsel des Betriebssystems ohne großen Aufwand eine Migration des Tools zu ermöglichen.

Die Idee ist, dass das Programm auf den Empfang einer HL7 Nachricht wartet, diese verarbeitet, lokal auf der Festplatte speichert und dann die nötigen Werte in die ePA-Datenbank einträgt, welche dann von dort in die ePA eingelesen werden können.

Der Server, auf dem das Programm läuft, sollte desweiteren eine fest zugewiesene IP Adresse haben, da das Gerät nicht mit DNS Namen und deren Auflösung umgehen kann. Sollte es also zu einer Netzwerkumstrukturierung kommen, müssen die Verbindungseinstellungen am BGA-Gerät und in der Konfigurationsdatei der Applikation angepasst werden.

2.5.1 Einbettung in den Arbeitsablauf

Die Einbettung des Tools in den Ablauf einer BGA ist wie folgt gedacht:

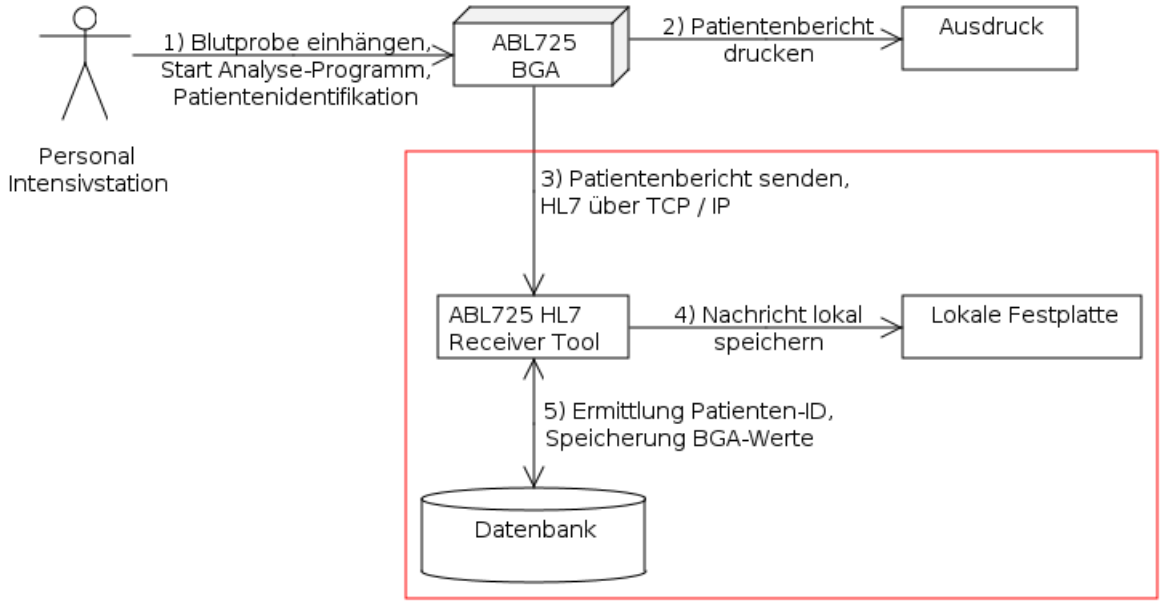

Abbildung 6: Einbettung des HL7 Tools in den Arbeitsablauf

2.5.2 Datenbankanbindung

Das Programm benötigt eine Anbindung an eine MySQL Datenbank, um den Patienten identifizieren zu können und um die ermittelten Werte in die Datenbank einzutragen.

2.5.3 Abgleich einer Liste lokaler HL7 Nachrichten mit der Datenbank

Eine weitere Anforderung ist das Abgleichen einer Liste von bereits empfangenen HL7 Nachrichten mit der Datenbank, da die Möglichkeit besteht, dass diese Nachrichten zwar empfangen wurden, aber nicht in die Datenbank eingetragen werden konnten.

Es soll möglich sein, einen lokalen Ordner mit beliebig vielen HL7 Nachrichten zu füllen, welche beim Start des Programms mit einem definierten Übergabeparameter eingelesen und abgearbeitet werden können. Es sollen dabei keine doppelten Einträge in der Datenbank auftauchen, falls diese Nachrichten schon in der Datenbank existieren sollten.

Das folgende Diagramm veranschaulicht diese Anforderung:

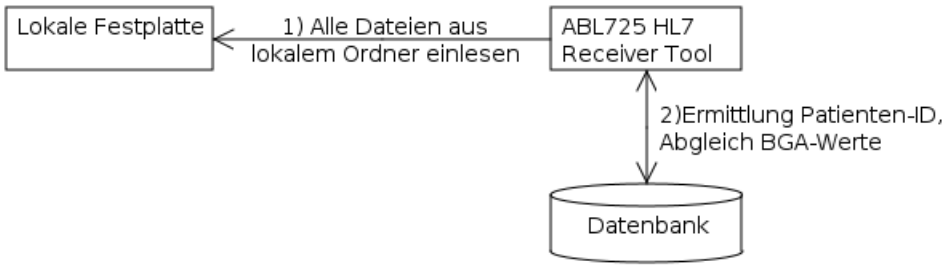

Abbildung 7: Abgleich lokaler HL7 Nachrichten mit der Datenbank

2.5.4 GUI

Die GUI[30] der Applikation soll aus zwei Anzeigefeldern bestehen, die dem Administrator die Möglichkeit geben, auf einen Blick den Status des Programms und die empfangenen Nachrichten zu sehen. Einerseits sollen alle Aktivitäten und Abläufe dargestellt, andererseits aber auch alle empfangenen Nachrichten im Klartext angezeigt werden, also so, wie sie empfangen wurden.

Das Programm soll keine aktive Benutzerinteraktion erfordern. Der Administrator soll nur kontrollieren können, wie es um den Status des Programms steht. Bei Fehlern soll dieser nur als Eintrag in der Logdatei auftauchen.

Zusätzlich soll angezeigt werden, wann das Programm gestartet wurde.

2.5.5 Netzwerkverbindung mit dem BGA-Gerät

Um vom BGA-Gerät Daten über das TCP empfangen zu können, muss ein TCP-Serversocket angeboten werden, da das BGA-Gerät als TCP-Clientsocket fungiert. Das Gerät versucht nach

[30]GUI: Graphical User Interface, grafische Benutzeroberfläche

einem definierten Intervall von n Sekunden eine Verbindung mit dem / den im Verbindungsprofil eingestellten Host/s herzustellen.

Die Verbindung muss dauerhaft gehalten werden und darf nicht zusammenbrechen, da durch die automatische Datensendung nur eine einzige Übertragung jeder Nachricht erfolgt.

2.5.6 Zugang zu IS-H*med

Um den Patienten identifizieren zu können, wird eine Tabelle in der Datenbank genutzt, die die Zuordnung von der PatID zur Fallnummer enthält. Da diese Tabelle aber nicht vollständig ist, muss ein Zugang zu IS-H*med[31] geschaffen werden. In IS-H*med sind alle PatIDs zu allen Fallnummern zugeordnet.

Über diesen Zugang soll mit Hilfe der Fallnummer die PatID ermittelt werden. Die PatID und die Fallnummer sollen dann in die Fallnummerntabelle in der Datenbank eingetragen werden. Das Programm, welches die HL7 Nachrichten empfangen soll, ermittelt dann also *anschliessend* die PatID aus der aktualisierten Fallnummerntabelle, nicht direkt über den IS-H*med-Zugang.

2.5.7 Applikationsneustart

Da es im Netzwerk zu Verbindungsausfällen kommen kann, die von der TCP-Komponente möglicherweise nicht erkannt werden, bspw. durch Neustart des Kommunikationsservers der Uniklinik, etc., soll das Programm nach einem definierten Leerlauf von bspw. 60 Minuten automatisch neugestartet werden.

Der Leerlauf wird als Indikator für eine möglicherweise zusammengebrochene TCP-Verbindung genutzt und meint das Fehlen jeglicher Vorkommnisse, wie z.B. TCP-Events, über einen definierten Zeitraum.

Beim Neustart des Programms sollen die beiden Anzeigefelder der GUI in zwei Textdateien umgewandelt und mit Zeitstempel auf der Festplatte gespeichert werden. Diese Dateien fungieren dann als Logdateien.

[31] IS-H*med: Modul von GSD-Siemens, angelehnt an ISH (SAP), bildet ein Krankenhausinformationssystem

3 Implementierung

3.1 Änderung des Arbeitsablaufs bei einer BGA

Der Arbeitsablauf in der Intensivstation muss angepasst werden, um eine eindeutige Idenfifikation des Patienten zu ermöglichen. Es ist zu beachten, dass es sich dabei nicht um eine zu große Anpassung handeln soll, da das Intensivpersonal die elektronische Dokumentation für die tägliche Arbeit nicht zwingend benötigt. Für die korrekte Behandlung des Patienten ist prinzipiell auch ein Ausdruck ausreichend, die Lösung hat also für das Personal selbst keinen direkten Zugewinn.

Eine Anpassung, die für das Personal eine zu große Hürde darstellt, bzw. eine schlichte Übergehung der Meinung des Personals soll vermieden werden, da die Lösung ansonsten schlicht abgestossen wird.

3.1.1 Methodik zur Identifikation des Patienten

Zur Identifikation des Patienten stehen im Allgemeinen folgende Ansätze zur Verfügung:

- Einscannen eines aus IS-H*med stammenden Barcodes(19, Kapitel 2-11, Strichcode-Scanner), der die Fallnummer des Patienten enthält.
- Auswahl der Bettnummer des Patienten (1-4).
- Manuelles Eintippen des Patientennamens.

Die Identifikation über einen manuell eingetippten Patientennamen ist unzureichend, da es zu Verwechslungen von gleichnamigen Patienten, oder auch zu Tippfehlern kommen kann. Auch die Identifikation über die Bettnummer ist nicht optimal, da die Datenbanktabelle, in der die Bettnummer mit der PatID verknüpft ist, nicht unbedingt korrekt sein muss. Bspw. ist die Identifikation anhand der Bettnummer inaktiv, wenn lokale HL7 Nachrichten mit der Datenbank abgeglichen werden, da in diesem Fall die Tabelle mit der Zuordnung PatID <-> Bettnummer nicht korrekt ist, da die Buchungshistorie nicht abgebildet ist.

Die einzige sichere und gleichzeitig relativ einfache Variante ist, mit Hilfe des am Gerät befindlichen Barcode Scanners einen zum Patienten gehörenden Barcode einzuscannen, welcher die Fallnummer des Patienten enthält. Anhand der ermittelten Fallnummer kann nun die PatID zugeordnet werden.

Der Vorteil der Identifikation eines Patienten anhand seines Barcodes liegt darin, dass das Personal einerseits nur eine relativ kleine Änderung im Arbeitsablauf erhält, andererseits ist die Identifikation aber auch sicher, da es nicht zu Tippfehlern kommen kann und der Patient

eindeutig identifiziert wird. Eine Verwechslung eines Patienten ist nur noch über eine Verwechslung des Barcodes oder eine fehlerhafte Zuordnung in IS-H*med oder der Fallnummerntabelle möglich.

3.2 Datenbank

3.2.1 Beschreibung der benötigten Tabellen

Es werden insgesamt vier Tabellen in der Datenbank *epa* benötigt:

- *fallnummer*: Zuordnung *PatID* mit *FallNr.*
- *pflbga*: Ziel für die Speicherung der BGA-Werte.
- *wl_stationi*: Zuordnung *Bett* mit *PatID.*
- *pflbga_fehler_st*: Zuordnung HL7 Kommentarcodes zu deren Bedeutung.

Die Tabelle *pflbga_fehler_st* existiert noch nicht und muss noch erstellt werden.

3.2.2 Notwendige Anpassungen

In der Tabelle *pflbga* müssen diverse Anpassungen vorgenommen werden, da die Tabelle nur rudimentär entwickelt wurde:

- Hinzufügen aller Parameter, die vom BGA-Gerät geliefert werden, aber noch in *pflbga* fehlen. Die Parameter werden im Attributkommentar kurz beschrieben.(20)
- Kommentierung der bereits vorhandenen Attribute (z.B. BE = Basenüberschuss).
- Hinzufügen einer Spalte *beschreibung*, um Meta-Informationen über den Datensatz speichern zu können.
- Indexierung über *PatID, Datum, Nr* und über *PatID, Datum.* Erstere, um eine schnelle Abfrage in der angegebenen Reihenfolge zu ermöglichen, zweitere um doppelte Einträge zu vermeiden.

Desweiteren wird eine Tabelle mit Kommentarcodes (19, Kapitel 12-8, Fehlersuch-Meldungen) und deren Bedeutung benötigt, um die Kommentarsegmente in einer HL7-Nachricht verstehen zu können. Diese Tabelle wurde *pflbga_fehler_st* genannt.

Die Kommentarcodes aus den NTE-Segmenten der empfangenen HL7-Nachricht werden dann durch ein Trennzeichen getrennt in der Spalte *beschreibung* der Tabelle *pflbga* im Format <Parameter>:<Kommentarcode(s)>; ... angefügt.

Die Idee hierbei ist, dass die Kommentarcodes aus der Spalte *beschreibung* für jeden Parameter ausgelesen und über die Tabelle *pflbga_fehler_st* aufgelöst werden können. Der Nutzer ist dann darüber informiert, dass z.B. der Wert Na^+ durch einen Kalibrierungsfehler zustande gekommen ist und ignoriert werden kann.

Sollte ein Parameter kommentiert sein, so wird der dabei ermittelte Wert nicht gespeichert. In dem betreffenden Feld wird dann der Wert *NULL* eingesetzt und im Feld *beschreibung* wird der / die Fehlercode/s eingefügt.

Die Tabellen für die BGA-Werte und die Kommentarcodes sehen in der überarbeiteten Version wie folgt aus:

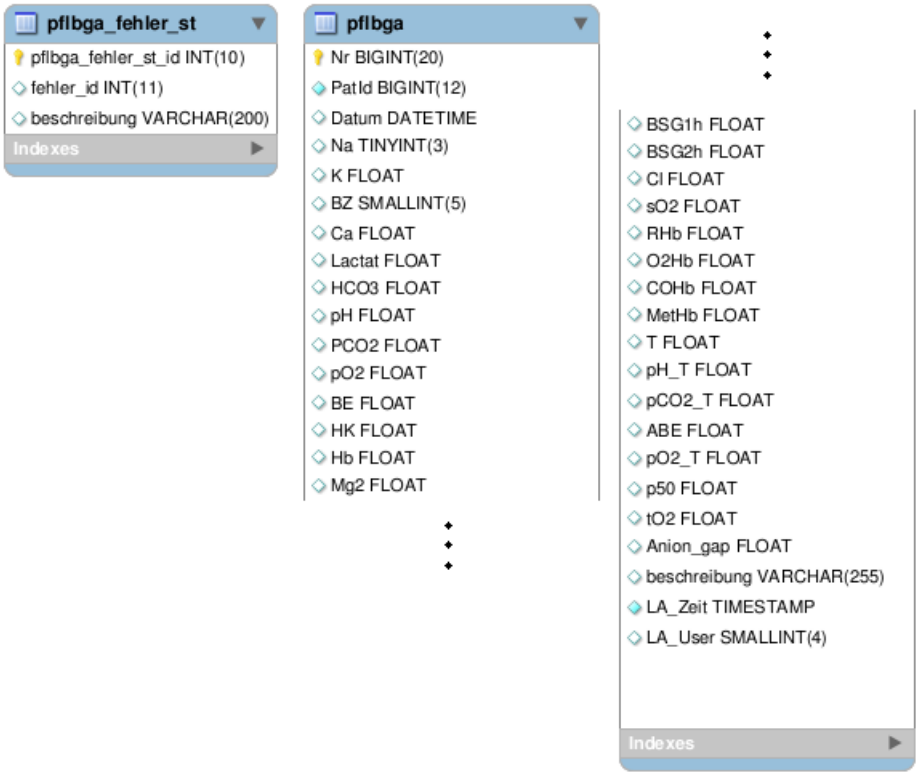

Abbildung 8: BGA Tabellen in der Datenbank *epa*

Die Werte *Mg2* (Mg^{2+}, Magnesiumionenkonzentration) *BSG1h* und *BSG2h* (Blutsenkungsgeschwindigkeit 1 Stunde und 2 Stunden) werden vom Gerät nicht ermittelt und werden ignoriert. Sie sind noch Überbleibsel eines ersten Tabellenentwurfs.

Die Umsetzung der Änderungen wird über ein SQL Skript realisiert, das kurzerhand auf dem DBMS ausgeführt wird, ohne weitergehende Interaktionen oder Anpassungen zu erfordern. Zu beachten ist, dass das Skript nur funktioniert, wenn bereits eine Tabelle *pflbga* wie folgt existiert:

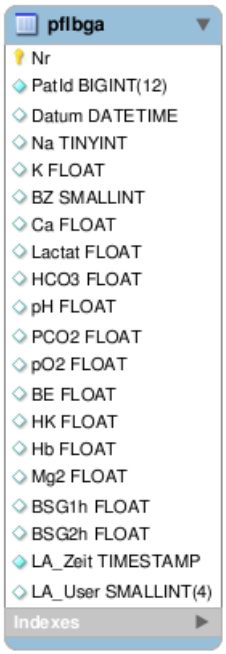

Abbildung 9: Ursprünglich vorgefundene Tabelle *pflbga* in der Datenbank *epa*

Die Zuordnung der Parameternamen in der HL7 Nachricht mit denen der Datenbanktabelle ist fest im Programmcode verankert.

3.3 Aufbau der Applikation

Die Applikation nennt sich *ABL725 HL7 Receiver* und besteht aus folgenden Klassen, die sich jeweils in einer eigenen Unit[32] befinden:

- *T_App_Controller*
- *TFo_View*
- *T_TCP_Handler*
- *T_Database_Handler*
- *T_ABL725_HL7_Reader*

[32]Unit: Sourcecode Datei, die als Modul in einer Applikation verwendet werden kann

Der *App_Controller* liest beim Start des Programms die Daten aus der *config_bga_abl725_reader.ini* ein, startet den TCP-Serversocket und reagiert auf die vom *TCP_Handler* gesendeten Events, wie z.B. *OnReceive*, und entscheidet, wie eine empfangene Nachricht weiter verarbeitet werden soll. Zunächst wird eine empfangene Nachricht jedoch erst einmal lokal auf der Festplatte gespeichert, bevor eine weitere Verarbeitung erfolgt.

Beim Start des Programms wird hier noch geprüft, ob die benötigten Tabellen *pflbga* und *fallnummer* in der Datenbank überhaupt existieren. Sollte das nicht der Fall sein, wird dieser Fehler auf dem *Memo_log* angezeigt und das Programm stoppt, da die Tabellen essentiell für die Funktionalität des Programms sind.

Sollte beim Programmstart (noch) keine Datenbankverbindung herstellbar sein, so fährt das Programm trotzdem fort, da es in diesem Moment auch kurzzeitig zum Ausfall des Datenbankservers gekommen sein könnte.

Die Klasse enthält zwei Timer vom Typ *TTimer*, einen namens *Timer_stack* für die Prüfung der globalen Komponente *StringList_stack* (Im Folgenden nur noch Stack genannt) auf neue Nachrichten und einen namens *Timer_restart_app*, welcher bei einem Leerlauf des Programms einen Neustart der Applikation durchführt.

Der Neustart der Applikation wird nur durchgeführt, wenn der Stack leer ist, denn nur dann darf neugestartet werden, da die dort gespeicherten Nachrichten sonst natürlich nicht verarbeitet würden.

Die Klasse *TFo_View* enthält die grafische Oberfläche, sowie eine Methode zum Neustart des Programms, die vom *App_Controller* nach Ablauf der eingestellten Zeit für den Leerlauf des Programms gerufen wird.

Der *T_TCP_Handler* bietet die Möglichkeit, einen TCP-Serversocket zu starten, zu stoppen und die Dauer einer Verbindung auszugeben. Die folgenden Events werden an den *App_Controller* weitergegeben:

- *OnAccept(aSocket: TLSocket)*
- *OnDisconnect(aSocket: TLSocket)*
- *OnError(const msg: string; aSocket: TLSocket)*
- *OnReceive(aSocket: TLSocket)*

Die Reaktion des *TCP_Handler* auf die oben gezeigten Events ist lediglich die Weiterleitung dieser an den *App_Controller*. Dieser entscheidet dann, wie bei welchem Event vorgegangen

werden soll. Bei den Events *OnAccept* und *OnDisconnect* wird lediglich eine Meldung auf das TMemo-Feld *Memo_log* des Formulars ausgedruckt, dass sich ein Client verbunden hat oder die Verbindung beendet wurde. Dazu übergibt der *TCP_Handler* die IP-Adresse des Clients an den *App_Controller*.

Bei *OnReceive* wird auf dem *Memo_log* eine Meldung ausgedruckt, dass eine neue Nachricht empfangen wurde. Danach wird diese Nachricht auf den Stack gelegt. Der *TCP_Handler* übergibt hier die empfangene Nachricht als String an den *App_Controller*.

Das Event *OnError* wird zwar mit übergeben, derzeit ist aber noch keine Reaktion darauf implementiert.

Das *OnConnect* ist nicht implementiert, weil es nicht benötigt wird, tritt es doch nur auf, wenn man selbst als Client fungiert, was hier nie geschieht. Dasselbe gilt für das *OnCanSend* Event.

Die Klasse *T_Database_Handler* beinhaltet eine *ZConnection* und bietet die zentrale Datenbankverbindung an. Über die Funktion *f_connect_to_database* kann unter Angabe der benötigten Parameter eine Verbindung mit einer Datenbank hergestellt werden. Da die *ZConnection* eine *public* Variable ist, also auch von anderen Klassen genutzt werden kann, kann diese im *App_Controller* von einer *ZQuery* für die Datenbankverbindung genutzt werden.

Da es zu Netzwerkausfällen kommen kann, wird eine Datenbankverbindung nur bei Bedarf hergestellt und nicht dauerhaft gehalten. Die Verbindung wird mit der Prozedur *p_disconnect_from_database* wieder geschlossen, sobald die Verbindung nicht mehr benötigt wird.

Die Klasse *T_ABL725_HL7_Reader* ist dazu da, die HL7 Nachricht zu validieren, einzulesen und um das SQL Insert-Skript zu generieren. Das SQL Insert wird dann an den *App_Controller* zurückgegeben, welcher dann eine Datenbankverbindung über den *Database_Handler* herstellt und das Skript an das DBMS abschickt.

Sollte es zu Exceptions kommen, wie z.B. *Duplicate Entry for key PatID, Datum*, also wenn der Datensatz schon in der Datenbank vorhanden ist, so wird diese Exception im *Memo_log* angezeigt. Dies wäre beim Abgleich bereits empfangener Nachrichten mit der Datenbank der Fall, wenn diese schon in der Datenbank existieren.

Das folgende Klassendiagramm veranschaulicht den Aufbau des Programms:

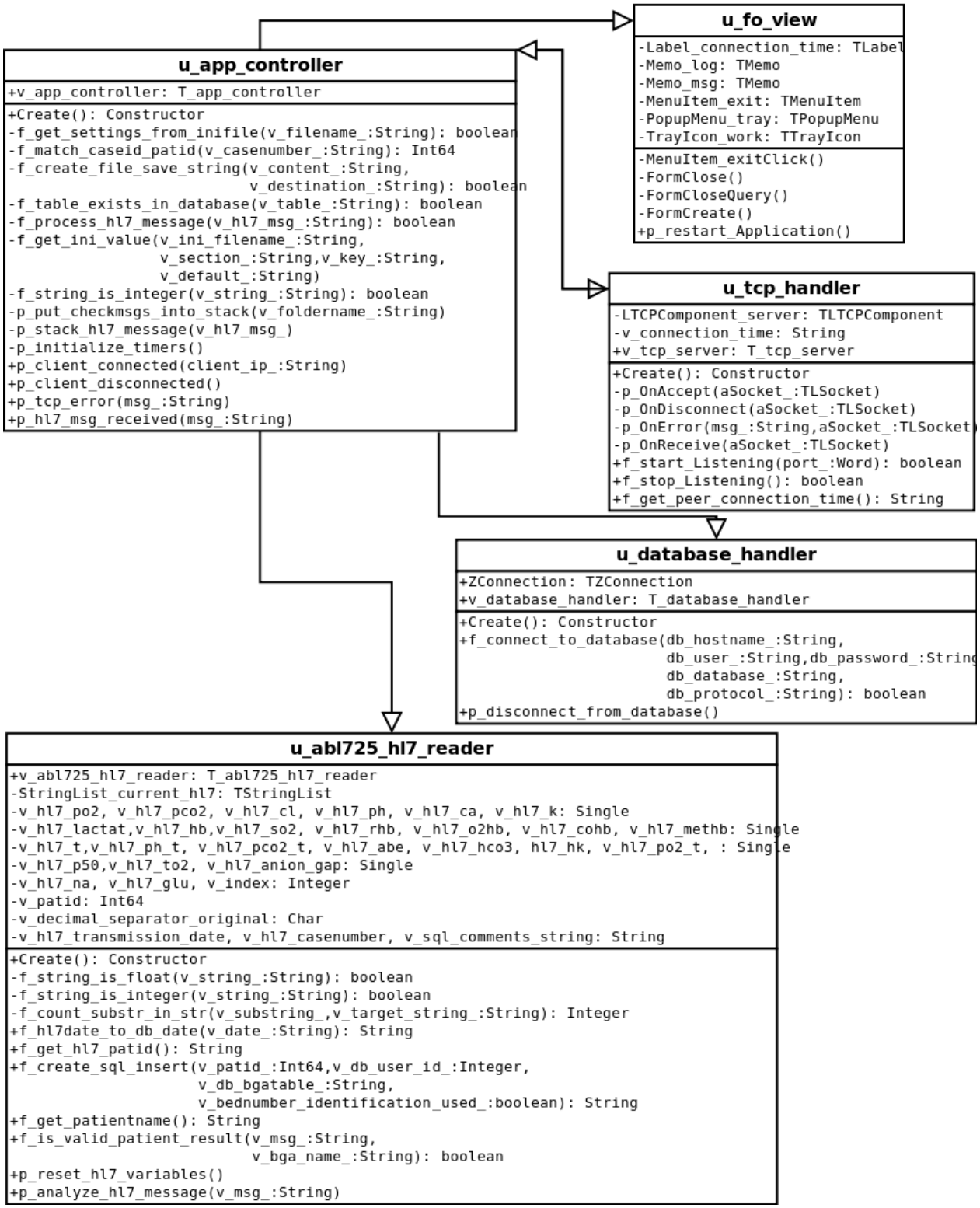

Abbildung 10: Klassendiagramm *ABL725 HL7 Receiver*

Die Klasse *T_App_Controller* enthält wie zuvor beschrieben einen Stack namens *StringList_stack* (Typ: TStringList), auf den die empfangene Nachricht gelegt wird, wenn es sich um eine valide HL7-Nachricht handelt. Ein Timer *Timer_stack* (Typ: TTimer) prüft regelmäßig den Stack auf neue Nachrichten, welche dann abgearbeitet werden.

Eine empfangene Nachricht wird bei erfolgreicher Abarbeitung aus dem Stack gelöscht. Sollte es dazu kommen, dass die Nachricht nicht abgearbeitet werden kann (z.B. Datenbankverbindung nicht möglich), so wird die Nachricht im Stack behalten. Der Stack füllt sich mit der Zeit und wird, sobald die Datenbankverbindung wieder steht, rückwärts abgebaut:

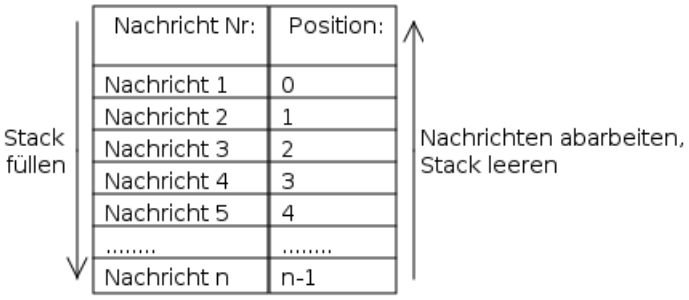

Abbildung 11: Füllen und Abarbeiten des Stacks

Solange sich etwas im Stack befindet, wird der Neustart des Programms unterdrückt. In der HNO Intensivstation fallen täglich etwa 10-15 BGAs an, das bedeutet also, dass der Stack in aller Regel nur gebraucht wird, wenn lokale HL7 Nachrichten eingelesen werden, oder wenn der Datenbankserver steht. Ansonsten ist im Stack nie mehr als eine Nachricht gleichzeitig gespeichert.

3.4 Die Konfigurationsdatei config_bga_abl725_reader.ini

Im Folgenden wird die Konfigurationsdatei *config_bga_abl725_reader.ini* dargestellt und erklärt, welche beim Start des Programms eingelesen wird:

```
1  [Database]
2  db_protocol=mysql-5
3  db_hostname=HOSTNAME
4  db_user=benutzername
5  db_password=123das_passwort
6  db_database=epa
7  db_bgatable=pflbga
8  db_casetable=fallnummer
9  db_userid=1234
10
11 [SAP]
12 sap_patid_matcher=<Pfad zu FallNr2PatId.exe>
13
14 [epa]
15 tabelle_warteliste=wl_stationi
16
17 [BGA]
18 bga_hl7_name=ABL725^
19
20 [Restart / Log Timer]
21 app_restart_interval=3600000
22
23 [BGA TCP Serversocket]
24 tcp_port=6789
```

Die Sektion *[Database]* beinhaltet alle nötigen Einstellungen für die Verbindung mit dem DBMS, sowie zur Ermittlung der PatID (*db_casetable*) und der Eintragung der Werte in die BGA Tabelle (*db_bgatable*). Die *db_userid* wird bei jedem Datensatz als *LA_User* eingetragen.

Da für die Datenbankverbindung eine *ZConnection* genutzt wird, können als *db_protocol* folgende DBS-Typen eingesetzt werden:

ASA7	ASA8	ASA9	firebird-1.0	firebird-1.5
firebird-2.0	firebirdd-1.5	firebirdd-2.0	firebirdd-2.1	interbase-5
interbase-5	interbase-6	mssql	mysql	mysql-4.1
mysql-5	mysqld-4.1	mysqld-5	oracle	oracle-9i
postgresql	postgresql-7	postgresql-8	sqlite	sqlite-2.8
sqlite-3	sybase			

Tabelle 3: Auflistung der unterstützten Datenbanksysteme

Der Vorteil liegt hier darin, dass beim Wechsel des DBS nur kurzerhand das Protokoll abgeändert werden muss. Eine Änderung des Programmcodes wäre nur dann erforderlich, wenn die SQL Queries inkompatibel würden.

Das Passwort für den Datenbankzugang enthält *Salt* (Begriff aus der Kryptologie), d.h., das hier gezeigte Passwort *123das_passwort* wurde um eine Folge von Zeichen erweitert, die nicht zum eigentlichen Passwort gehören. Das soll es einem Angreifer etwas schwieriger zu machen, sollten ihm die Zugangsdaten in die Hände fallen. Hier wurde am Anfang die Zeichenfolge *123* angefügt, die von der Applikation wiederum beim Programmstart abgeschnitten wird.

Unter *sap_patid_matcher* muss der Pfad zum Tool *FallNr2PatID* angegeben werden. Fehlt diese Angabe, so wird der Zwischenschritt über dieses Tool umgangen und die PatID wird direkt aus der *db_casetable* über die Fallnummer zugeordnet.

Bei *tabelle_warteliste* muss die Tabelle angegeben werden, in der die Bettnummer mit der PatID verknüpft ist. Fehlt diese Angabe, so wird der Patient abermals nur über die *db_casetable* identifiziert.

Der Schlüssel *bga_hl7_name* wird benötigt, um den Sender der HL7 Nachricht zu verifizieren. Nur HL7 Nachrichten, die den angegebenen Wert als Substring[33] im Feld 3 des MSH Segments (Sending Application) haben, werden in die Datenbank aufgenommen. Bsp: In der INI-Datei

[33]Substring: Teil eines Strings. Bsp: Der String *Hallo* ist Teil des Strings *Hallo Welt!*

wurde als *bga_hl7_name* der String *ABL725^* angegeben, dann wird eine Nachricht mit der Sending Application *ABL725^ICU_ENT_1* als valide erkannt.

Hintergrund ist hierbei, dass bei einer nachträglichen, manuellen Übertragung der Patientenberichte lediglich der String für den Gerätenamen als *Sending Application* eingetragen ist, ohne den individuell definierbaren Gerätenamen, der üblicherweise den Stationsnamen enthält. Dabei handelt es sich höchstwahrscheinlich um einen Fehler in der Software des BGA-Geräts. Solche Nachrichten sollen aber natürlich auch in die Datenbank, also wird dieser String hier allgemeiner gehalten, auf Kosten der eindeutigen Identifizierbarkeit des Senders.

Das *app_restart_interval* wird genutzt, um einen Neustart der Applikation zu ermöglichen. Hier muss die Zeit in Millisekunden eingetragen werden, um den Neustart nach einem Leerlauf zu erzwingen. Bsp.: Wurde als *app_restart_interval* der Wert *3600000* eingegeben, so startet die Applikation nach einem Leerlauf von einer Stunde neu und schreibt dabei die Logdateien.

Wird hier kein Wert oder der Wert *0* angegeben, so startet die Applikation nicht neu und schreibt auch keine Logdateien.

Unter dem Schlüssel *tcp_port* kann der Port angegeben werden, auf dem sich das BGA-Gerät mit dem TCP-Serversocket der Applikation verbinden soll. Dieser Port muss natürlich mit dem im Gerät angegebenen Port identisch sein.

3.5 GUI

Nachfolgend ist die grafische Oberfläche nach der erfolgreichen Verarbeitung eines Patientenberichtes zu sehen (Betriebssystem: Ubuntu 9.10 (Linux)):

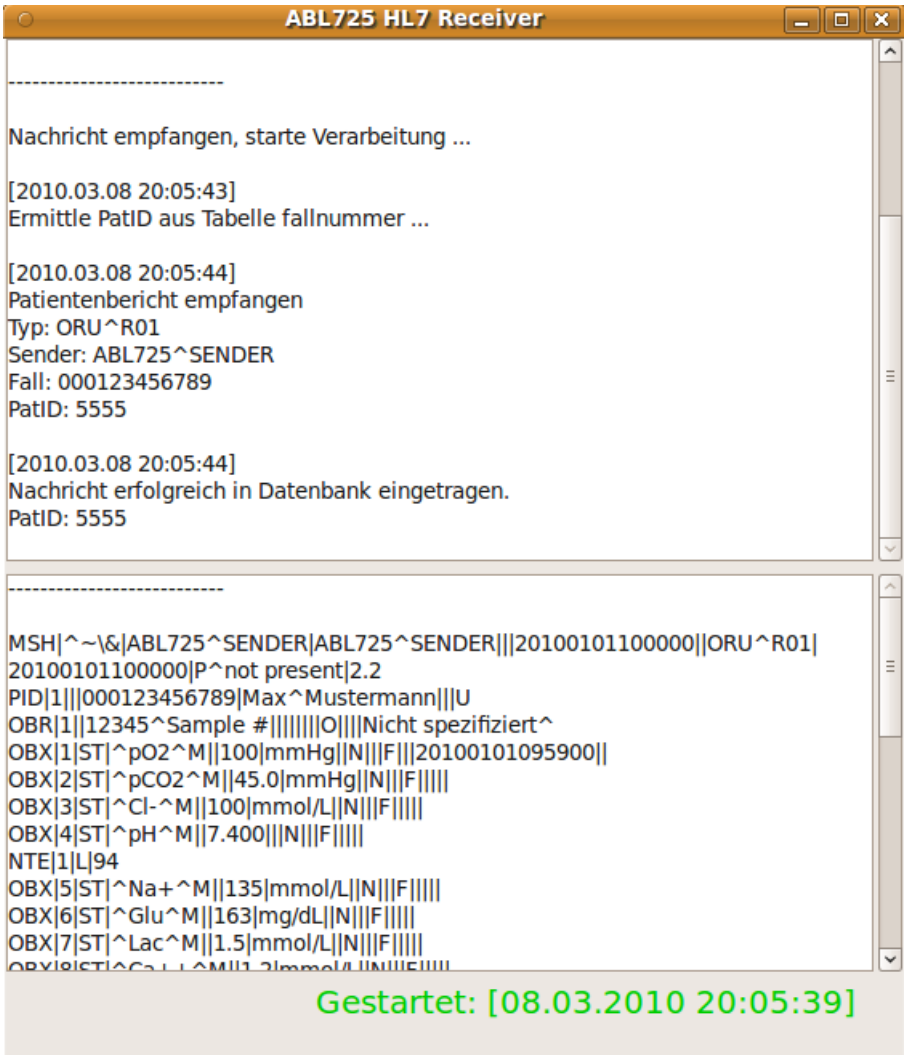

Abbildung 12: GUI nach Verarbeitung einer empfangenen HL7 Nachricht

3.6 Patientenidentifikation über IS-H*med

Um die PatID eines Patienten anhand seiner Fallnummer ermitteln zu können, muss ein Zugang zu IS-H*med eingerichtet werden, weil die Tabelle *fallnummer* unvollständig ist. Herr Dr. Tewes hatte zu diesem Zweck bereits ein Tool namens *FallNr2PatID* entwickelt, das einen Zugang zu IS-H*med zur Ermittlung der PatID anhand der Fallnummer ermöglichte.

Dieses Tool wurde nun dahingehend umgestaltet, als dass es sich unsichtbar und mit der Fallnummer als Übergabeparameter starten lässt, automatisiert eine Verbindung zu IS-H*med

herstellt, anhand der Fallnummer die PatID aus IS-H*med ermittelt und damit die Tabelle *fallnummer* aktualisiert.

Dieses Programm wird als Subprozess bei der Verarbeitung eines Patientenberichts gestartet und beendet sich nach erfolgreicher Aktualisierung der Tabelle selbst. Solange dieser Subprozess läuft, ist der *ABL725 HL7 Receiver* durch die Prozessoption *poWaitOnExit* eingefroren. Der gesamte Vorgang inklusive der Aktualisierung der Tabelle dauert jedoch lediglich einen Bruchteil einer Sekunde, dadurch fällt diese Aktion in der Regel auch nicht auf.

Es gibt für dieses Programm eine eigene Konfigurationsdatei namens *config_match_patid_sap.ini*, die dieselben Datenbankeinstellungen enthält, wie die Konfigurationsdatei des *ABL725 HL7 Receiver*.

3.7 Wichtige Funktionen und Abläufe

Im Folgenden wird anhand wichtiger Szenarien die Funktion des Programms dargestellt. Relevante Teilschritte sind dabei:

- Programmkonfiguration einlesen
- Datenbankverbindung herstellen
- Datenbank auf benötigte Tabellen prüfen
- Nachrichten über TCP empfangen
- Empfangene Nachrichten lokal speichern
- Alle wichtigen Statusmeldungen und alle empfangenen Nachrichten auf der GUI anzeigen
- Eine HL7 Nachricht validieren
- Eine valide HL7 Nachricht einlesen
- Das Programm *FallNr2PatID* mit Übergabeparameter *FallNr* als Subprozess starten
- Die PatID aus IS-H*med anhand des Übergabeparameters *FallNr* ermitteln
- Die Tabelle *fallnummer* aktualisieren
- Den Subprozess abschalten
- Über die Tabelle *fallnummer* die PatID ermitteln
- Die Applikation neustarten
- Eine Liste lokaler HL7 Nachrichten mit der Tabelle *pflbga* abgleichen

3.7.1 Programmstart, Empfang und Verarbeitung einer validen HL7 Nachricht

Das nachfolgende Ablaufdiagramm zeigt den Ablauf vom Programmstart bis hin zum Hinzufügen einer empfangenen validen HL7 Nachricht zum Stack:

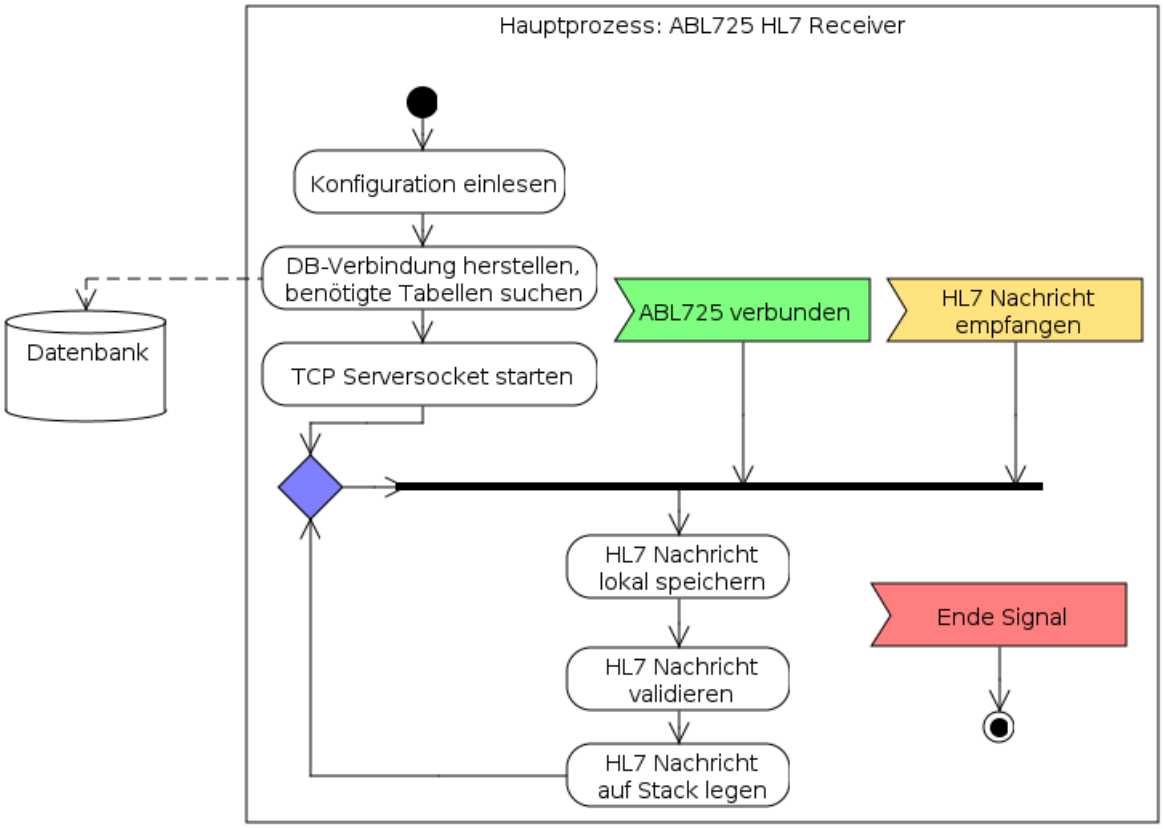

Abbildung 13: Ablaufdiagramm für den Ablauf vom Programmstart bis zum Hinzufügen zum Stack

Um eine HL7 Nachricht auf den Stack legen zu können, muss der TCP-Serversocket gestartet, ein BGA-Gerät verbunden und eine HL7 Nachricht empfangen worden sein. Sind diese drei Bedingungen erfüllt, dann wird die empfangene Nachricht lokal gespeichert und die Validierung, sowie die Speicherung der Nachricht im Stack kann durchgeführt werden.

Der Prozess wird entweder vom Benutzer oder durch das Signal des *Timer_restart_app* beendet.

Um diese HL7 Nachricht(en) nun vom Stack abzuarbeiten, wird sekündlich das Event *Timer_stack.OnTimer* gerufen und der Stack auf neue Nachrichten geprüft. Existieren neue Nachrichten im Stack, so wird die Abarbeitung wie folgt durchgeführt:

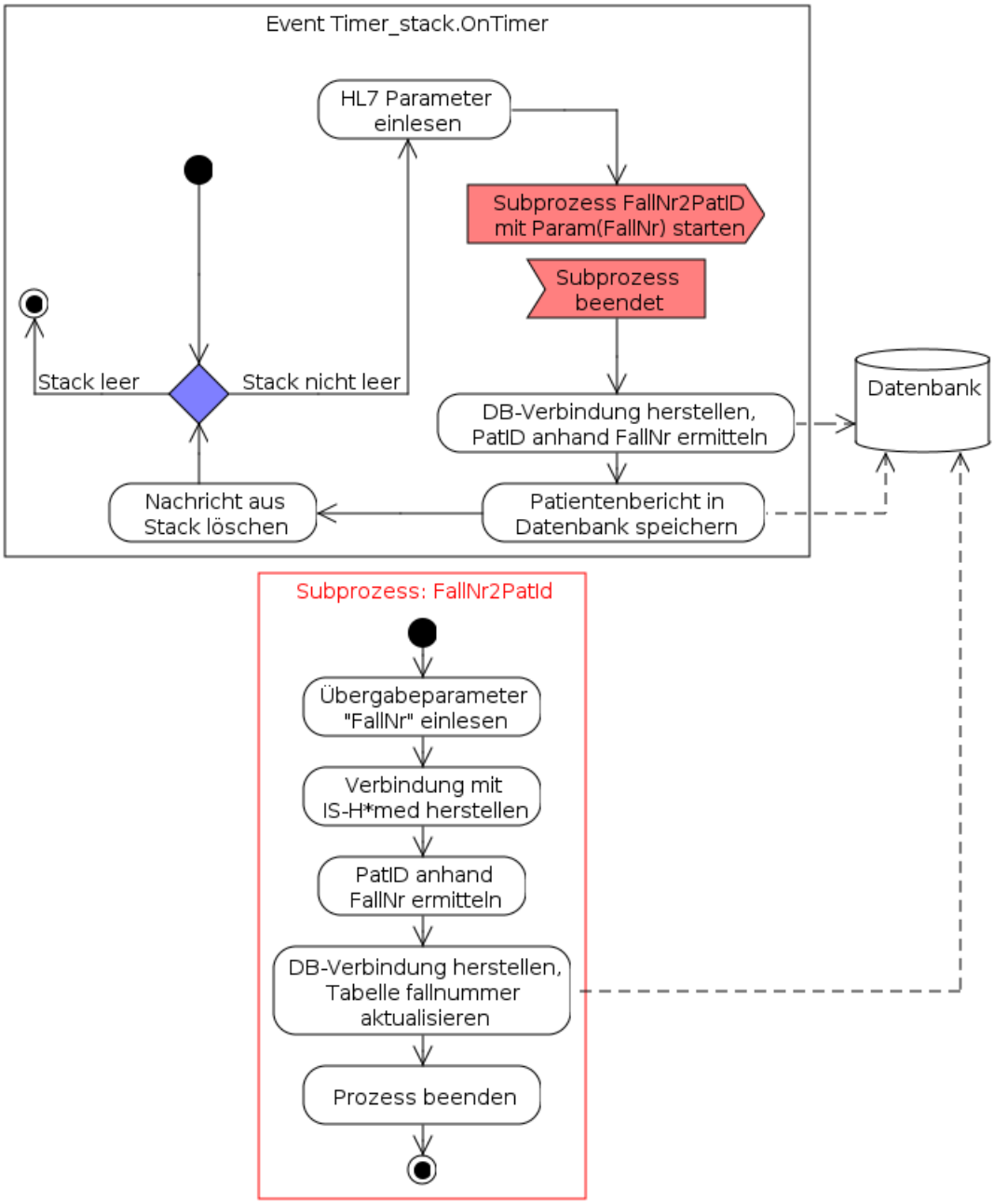

Abbildung 14: Ablaufdiagramm für die Abarbeitung des Stacks

Passend dazu veranschaulicht das folgende Sequenzdiagramm den Ablauf innerhalb des Programms *ABL725 HL7 Receiver*:

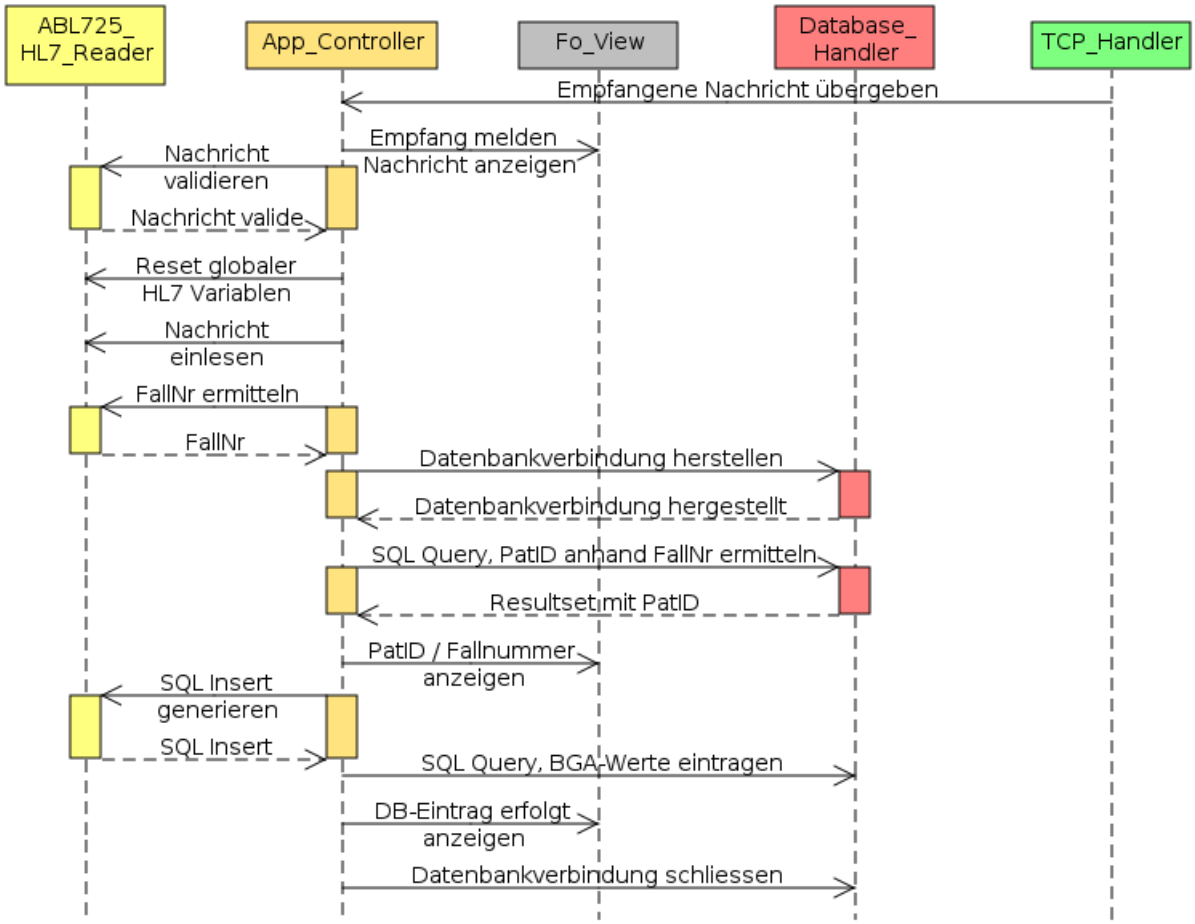

Abbildung 15: Sequenzdiagramm für den Empfang einer validen HL7-Nachricht im ABL725 HL7 Receiver

Die empfangene Nachricht wird im Ordner *msg* gespeichert. Sollte es sich um einen validen Patientenbericht handeln, so wird die Nachricht zusätzlich im Ordner *Patientenberichte* gespeichert.

Das Programm startet also, liest die Konfiguration ein, prüft die Datenbankverbindung und das Vorhandensein der Tabellen *pflbga* und *fallnummer* und startet den TCP-Serversocket auf dem angegebenen Port. Ist ein BGA-Gerät verbunden und wurde eine HL7 Nachricht übermittelt, so wird diese lokal gespeichert, validiert und anschliessend auf den Stack gelegt.

Der Stack wird mit Hilfe des *Timer_stack* sekündlich auf neue Nachrichten geprüft. Sollten eine oder mehrere Nachrichten vorliegen, so wird der Stack wie zuvor besprochen rückwärts abgearbeitet. Sollte währenddessen eine neue HL7 Nachricht empfangen werden, so wird sie

hinten an den Stack angehängt.

Eine während der Abarbeitung des Stacks empfangene Nachricht wird also erst dann abgearbeitet, wenn der Stackzeiger auf Position *0* steht und dann erkannt wird, dass doch noch etwas im Stack liegt. Dann wird der Zeiger wieder an das Ende des Stacks gesetzt und es wird wiederum rückwärts abgearbeitet.

Durch die Rückwärts-Abarbeitung werden die neu eingegangenen Nachrichten zuerst abgearbeitet, diejenigen, die eigentlich schon viel länger auf ihre Bearbeitung warten, kommen zuletzt dran. Da die Abarbeitung jeder Nachricht jedoch weniger als eine Sekunde dauert, fällt diese eigentlich falsche Vorgehensweise nicht ins Gewicht.

Da die Vorwärts-Abarbeitung zu Verwirrung mit der Zeigerposition führen kann, wird die Rückwärts-Abarbeitung angewendet. Durch das Löschen der abgearbeiteten Nachrichten fällt der Stack mit der Zeit in sich zusammen, d.h. Nachricht 2 rutscht auf Position 1, Nachricht 3 auf Position 2 usw., gleichzeitig kann aber auch eine neue Nachricht auf den Stack gelegt werden, was ihn wiederum kurz erhöht. Die rückwärtige Abarbeitung erspart das Implementieren einer Routine, die diese Verschiebungen abfangen könnte.

3.7.2 Erkennung des Leerlaufs und Durchführung des Neustarts der Applikation

Der Leerlauf wird mit Hilfe des *Timer_restart_app* wie folgt erkannt:

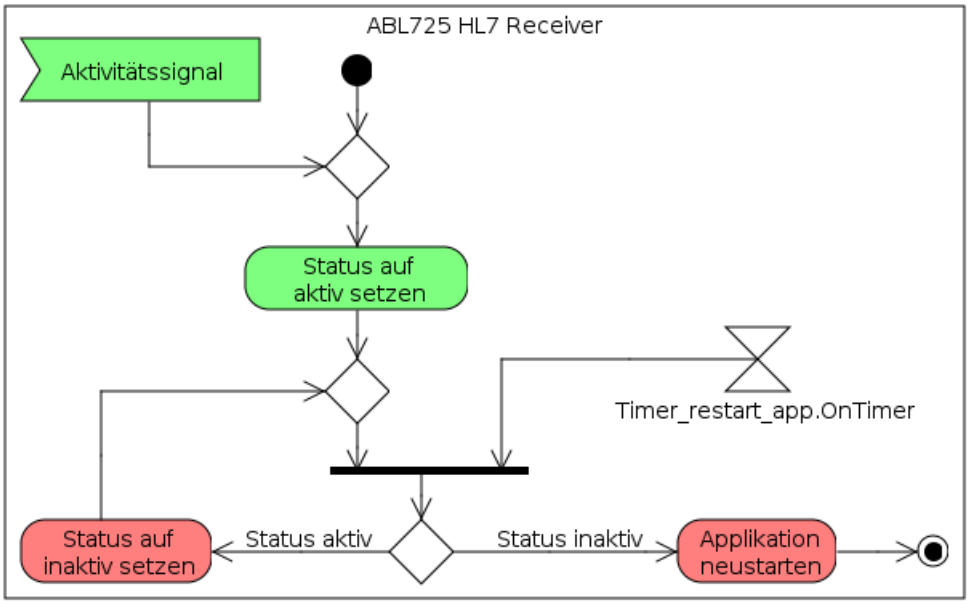

Abbildung 16: Ablaufdiagramm für die Erkennung des Leerlaufs, Neustart der Applikation

Das Aktivitätssignal tritt bei jeglichen TCP Events und bei der Abarbeitung im Stack auf,

beim Start des Programms ist ebenfalls der Status des Programms auf *aktiv*. Das Aktivitätssignal setzt den Status des Programms auf aktiv, der *Timer_restart_app* hingegen setzt den Status auf inaktiv. Aktivitätssignal und Timer arbeiten sozusagen gegeneinander.

Findet der Timer nun ein inaktives Programm vor, so startet er die Applikation neu. Kommt ihm das Aktivitätssignal jedoch zwischenzeitlich zuvor, so wartet er weiter auf die Inaktivität. Der Timer muss also *zweimal* aktiv werden, ohne dass in dieser Zeitspanne das Aktivitätssignal aufgetreten ist, damit die Applikation neugestartet wird. Das Intervall für den Timer ist wegen des benötigten doppelten Durchlaufs: $\left\lceil \frac{app\_restart\_interval}{2} \right\rceil$

Sollte der Neustart der Applikation durchgeführt werden, so geschieht dies wie folgt:

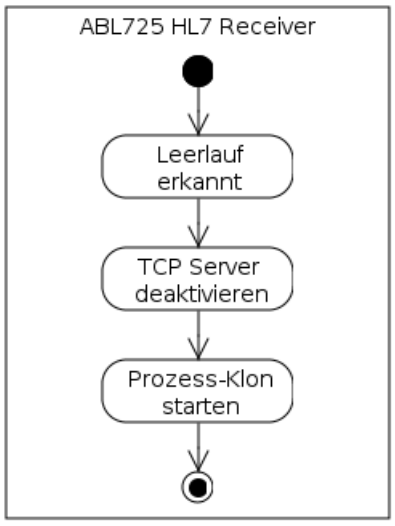

Abbildung 17: Ablauf des Neustarts der Applikation

Der Prozess-Klon erhält vom Betriebssystem eine neue Prozess-ID, die Gleichnamigkeit der beiden Prozesse ist dadurch unproblematisch.

Es ist essentiell wichtig, dass der ursprüngliche Prozess alle Aktivitäten einstellt, die dem Neuen in die Quere kommen könnten. So muss bspw. der TCP-Server unbedingt abgestellt werden, weil dieser ja vom neuen Prozess auf demselben Port gestartet wird.

3.7.3 Empfang einer validen HL7 Nachricht mit Identifikation des Patienten über dessen Bettnummer

Sollte es dazu kommen, dass das Personal den Patienten nicht anhand seines Barcodes, sondern anhand der Bettnummer 1 bis 4 identifiziert, so wird die Bettnummer dem Patienten über die Tabelle *wl_stationi* zugeordnet. Diese Identifikationsmethode ist allerdings nicht immer ein-

deutig, da diese Tabelle auch fehlerhaft sein kann, z.B. kann die Zuordnung nicht up-to-date sein. Es ist daher erforderlich, dass der Benutzer weiss, dass die Zuordnung über diese Tabelle stattgefunden hat, und nicht über die aktualisierte Fallnummerntabelle.

Zu diesem Zweck wird im Feld *beschreibung* der Tabelle *pflbga* der Inhalt von Feld 5 (Patientenname) stets mit angefügt. Dort befindet sich in diesem Fall nicht der Name, sondern die Bettnummer des Patienten im Format *ICU_ENT_1^Bett 4* oder auch *Bett 4^ICU_ENT_1*.

Diese Methode ist allerdings unzuverlässig und sollte möglichst vermieden werden. Das Mittel der Wahl zur Identifikation des Patienten ist eindeutig die Identifikation über seinen Barcode, also die Fallnummer.

3.7.4 Abgleich lokaler HL7 Nachrichten mit der Datenbank

Der *Abgleich lokaler HL7 Nachrichten mit der Datenbank* funktioniert vom Ablauf her ähnlich wie zuvor der *Programmstart, Empfang und Verarbeitung einer validen HL7 Nachricht*, nur, dass eben keine Nachricht über TCP empfangen wird, sondern n Nachrichten aus dem lokalen Ordner *msg_check* in den Stack geladen und anschliessend abgearbeitet werden.

Das folgende Ablaufdiagramm veranschaulicht die Vorgehensweise des Programms in diesem Modus:

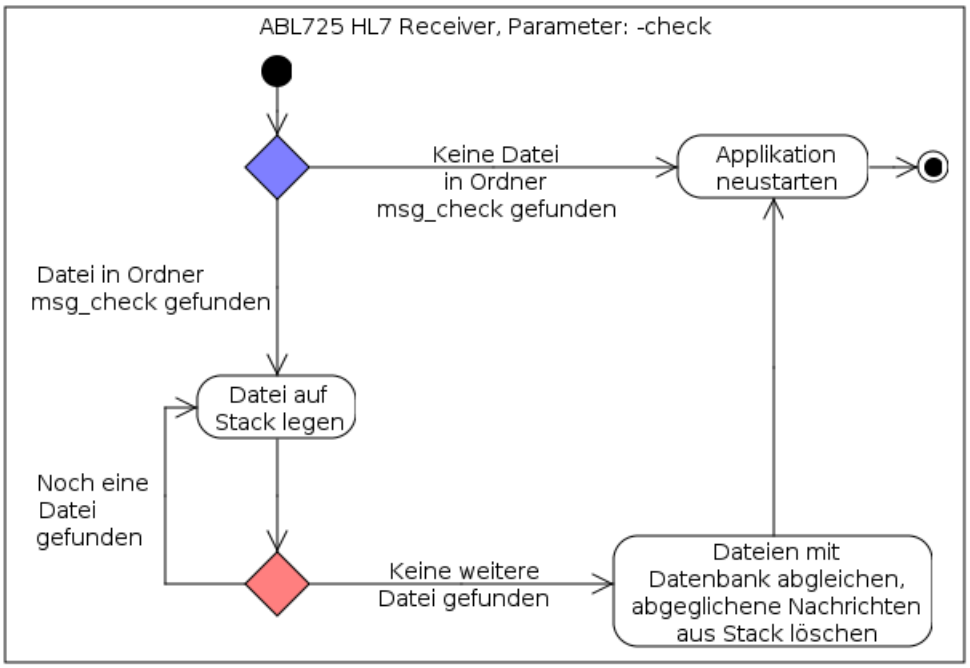

Abbildung 18: Ablaufdiagramm für das Abgleichen lokaler HL7-Nachrichten mit der Datenbank

Das Programm muss hierfür mit dem Übergabeparameter *-check* gestartet werden. Dabei

wird im Ordner *msg_check* jede vorgefundene Datei auf den Stack des *App_Controllers* gelegt, welche dann im Event *Timer_stack.OnTimer* abgearbeitet werden. Nachdem die Nachrichten abgearbeitet wurden, startet sich das Programm neu und fährt wie gewohnt fort.

Zur Sicherheit bleibt der TCP-Serversocket auch während des Abgleichs eines lokalen Ordners aktiv, da dieser Abgleich je nach Anzahl der Nachrichten länger dauern kann und in dieser Zeit ja auch eine BGA stattfinden könnte. Diese Nachricht soll dann natürlich auch gleich mitverarbeitet werden und nicht verloren gehen.

In diesem Abgleich-Modus wird die Identifikation des Patienten ausschliesslich über die jeweils mit dem Tool *FallNr2PatID* aktualisierte Tabelle *fallnummer* durchgeführt. Kann ein Patient nicht identifiziert werden, so lautet seine PatID stets: *0*. Ein Nutzer müsste dann entweder durch die Informationen aus Feld 5 des PID Segments (Name / Bettnummer) den Patienten zuordnen, oder anhand der Uhrzeit. Generell ist dieses Vorgehen aber wegen der fehlenden Eindeutigkeit als problematisch anzusehen.

3.7.5 Empfang einer invaliden HL7 Nachricht

Sollte eine Nachricht als invalide erkannt werden, so ist der Ablauf wie folgt:

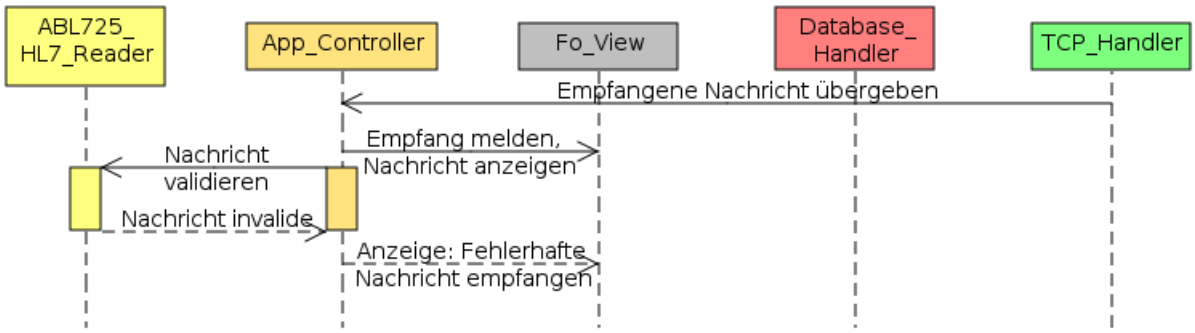

Abbildung 19: Sequenzdiagramm für den Empfang einer invaliden Nachricht

Die empfangene Nachricht wird direkt nach der Übergabe an den *App_Controller* lokal im Ordner *msg* gespeichert und bei Erkennen der Invalidität noch einmal im Ordner *fehlerhafte_nachrichten* abgelegt. Das Programm zeigt in der *Memo_log* den Fehler an und löscht dann die als fehlerhaft erkannte Nachricht aus dem Stack und wartet auf den Empfang weiterer Nachrichten.

Ein Administrator muss also später in der Logdatei des *Memo_log* sehen, dass eine invalide HL7-Nachricht empfangen wurde und muss in der dazu passenden Logdatei des *Memo_msg* diese Nachricht ausfindig machen, um den Grund der Invalidität der Nachricht zu verstehen. Kon-

trolliert ein Administrator die Logdateien nicht, so wird er auch nie erfahren, dass ein solcher Fehler stattgefunden hat, da das Programm auf Automatismus ausgelegt ist, um Administrationsaufwand zu vermeiden.

3.8 Beispielhafter Patientenbericht

Nachfolgend ist ein beispielhafter, vollständiger und anonymisierter HL7 Patientenbericht zu sehen, wie er vom ABL725^{TM} der HNO-Intensivstation nach einer BGA versendet wird:

```
1  MSH|^~\&|ABL725^ICU_NAME|ABL725^ICU_NAME|||20100101100000||
       ORU^R01|20100101100000|P^not present|2.2
2  PID|1|||000123456789|Max^Mustermann|||U
3  OBR|1||12345^Sample #|||||||O||||Nicht spezifiziert^
4  OBX|1|ST|^pO2^M||100|mmHg||N|||F|||20100101095900||
5  OBX|2|ST|^pCO2^M||45.0|mmHg||N|||F|||||
6  OBX|3|ST|^Cl-^M||100|mmol/L||N|||F|||||
7  OBX|4|ST|^pH^M||7.400|||N|||F|||||
8  OBX|5|ST|^Na+^M||135|mmol/L||N|||F|||||
9  OBX|6|ST|^Glu^M||163|mg/dL||N|||F|||||
10 OBX|7|ST|^Lac^M||1.5|mmol/L||N|||F|||||
11 OBX|8|ST|^Ca++^M||1.2|mmol/L||N|||F|||||
12 OBX|9|ST|^K+^M||4.6|mmol/L||N|||F|||||
13 OBX|10|ST|^tHb^M||10.3|g/dL||N|||F|||||
14 OBX|11|ST|^sO2^M||99.0|%||N|||F|||||
15 OBX|12|ST|^RHb^M||3.1|%||N|||F|||||
16 OBX|13|ST|^O2Hb^M||91.5|%||N|||F|||||
17 OBX|14|ST|^COHb^M||3.4|%||N|||F|||||
18 OBX|15|ST|^MetHb^M||2.0|%||N|||F|||||
19 OBX|16|ST|^T^I||37.0|Cel|||||F|||||
20 OBX|17|ST|^pH(T)^M||7.400|||N|||F|||||
21 OBX|18|ST|^pCO2(T)^M||50.0|mmHg||N|||F|||||
22 OBX|19|ST|^ABE^C||1.9|mmol/L|||||F|||||
23 OBX|20|ST|^SBC^C||24.0|mmol/L|||||F|||||
24 OBX|21|ST|^Hct^C||53.0|%|||||F|||||
25 OBX|22|ST|^pO2(T)^M||100|mmHg||N|||F|||||
26 OBX|23|ST|^p50(act)^E||26.0|mmHg|||||F|||||
27 OBX|24|ST|^tO2^C||23.0|Vol%|||||F|||||
28 OBX|25|ST|^Anion gap (K+)^C||15.6|mmol/L|||||F|||||
```

Die Nachricht enthält 25 OBX Segmente. In jedem dieser Segmente befindet sich u.a. ein Parameterbezeichner, ein Wert, sowie die Einheit des Parameters. Im ersten OBX Segment befindet sich in Feld Nr. 15 das Datum der Messung, das sich u.U. stark vom Datum der

Übermittlung unterscheiden kann.

Sollten die letzten Patientenberichte manuell vom BGA-Gerät aus noch einmal verschickt werden, so unterscheidet sich das Datum der Messung völlig vom Datum der Nachrichtenübermittlung. Das bedeutet also, dass das Datum der Nachrichtenübermittlung unzureichend ist und das Datum der Messung genutzt werden muss.

Hinter einem jeden Parameterbezeichner steht ein Zeichen, das die Entstehung des Wertes beschreibt. Bsp: ^Ca++^M. Das *M* zeigt an, dass dieser Wert gemessen (engl.: *measured*) wurde. Die Parametertypen lauten(17, Kapitel 8-17, Observation Result Segment):

- C, für *calculated*, berechneter Wert.
- D, für *default*, Standardwert.
- E, für *estimated*, geschätzter Wert.
- I, für *interpolated*, interpolierter Wert.
- M, für *measured*, gemessener Wert.
- „“, nicht spezifizierter Parametertyp.

Die Parametertypen werden in der jetzigen Programmversion ignoriert. Ein Parameter wird erkannt, wenn der jeweilige Bezeichner als Substring gefunden wird. Bsp.: ^p50(act)^E wird als *p50(act)* erkannt, wenn ^p50(act)^ Substring von ^p50(act)^E ist.

3.8.1 Validierung des Laborergebnisses

Die Nachricht wird anhand der folgenden Regeln validiert:

- MSH: Anzahl Felder ≥ 12, der Wert des *bga_hl7_name* aus der *config_bga_abl725_reader.ini* ist Substring der *Sending Application* der HL7 Nachricht (Feld 3).
- PID: Anzahl Felder ≥ 9.
- OBR: Anzahl Felder ≥ 16, „Sample“ als Substring in Feld 3 (Filler Order Number)(17, Kapitel 10-1, Tips for Programmers).
- OBX: Invalidierung eines einzigen OBX Segments bewirkt Invalidierung der ganzen Nachricht. Wenn in Feld 4 (Bezeichner) oder Feld 6 (Wert) nichts steht, oder die Anzahl der Felder nicht ≥ 17 ist, ist das OBX Segment invalide. Befindet sich im ersten OBX Segment in Feld 15 kein String von der Länge 14 Zeichen (Datum der Messung), so ist das OBX Segment ebenso invalide.

Sind diese vier Bedingungen nicht erfüllt, so wird die Nachricht nicht weiter verarbeitet und nur lokal im Ordner *msg* vorgehalten.

3.8.2 Auslesen eines Patientenberichts

Zu Beginn der Verarbeitung der Nachricht werden die nötigen globalen HL7-Variablen des Objekts *ABL725 HL7 Reader* auf Werte zurückgesetzt, die ausserhalb des Wertebereichs des jeweiligen Parameters liegen (Bsp. Blutzucker: *BZ = -1*). Vorzeichenbehaftete Werte erhalten den Wert *-100*. Die Nachricht wird dann segmentweise auf eine TStringList namens *StringList_work* gelegt. Anschliessend wird über deren Zeilen gelaufen und die relevanten Daten werden in die globalen HL7-Variablen eingelesen.

Sollte einem OBX Segment ein NTE (Notes and Comments) Segment folgen, so wird in der Datenbank für diesen Parameter der Wert *NULL* eingesetzt. Im Feld *beschreibung* wird dann lediglich der / die Kommentarcode/s des NTE Segments angefügt.

Die eingelesenen Daten aus den Segmenten lauten:

- PID: Fallnummer (Feld 5); Patientenname / Bettnummer (Feld 6).
- OBX: Ermittelter Wert eines jeden Parameters (Feld 6).
- NTE: Kommentarcode passend zum Parameter (Feld 4).

Die Einheit des Parameters wird in der jetzigen Version des Programms ignoriert. Die Zuordnung der Parameterbezeichner von der HL7 Nachricht zur Tabelle *pflbga* ist fest im Programm verankert. Bsp.: Der Parameter *Glu* für Glukose in der HL7 Nachricht lautet in der Datenbank *BZ* für Blutzucker, etc. Dadurch erfordert eine eventuelle Änderung der Attributbezeichner in *pflbga* die Anpassung des Programmcodes.

Fehlt die Fallnummer, so ist die PatID = *0*. Ist die Fallnummer < 100.000 (IS-H*med kennt nur Fallnummern ≥ 100.000), so wird bei Angabe der Tabelle *wl_stationi* versucht, den Patienten anhand seiner Bettnummer zu identifizieren. Klappt das nicht, so ist die PatID = *0*. Wenn diese Tabelle nicht angegeben wurde, so wird abermals direkt die Tabelle *fallnummer* zur Identifikation genutzt.

Die Reihenfolge der Attribute in *pflbga* spielt keine Rolle, da die Spalten über ihre Namen erkannt werden, nicht über deren Position innerhalb der Tabelle. Genauso spielt die Reihenfolge der OBX Segmente in der HL7 Nachricht keine Rolle, da die Zuordnung über den Parameterbezeichner erfolgt, nicht über die Segmentnummer 1-25.

3.8.3 Datensicherheit und Verschlüsselung

Das Programm unterstützt keinerlei Verschlüsselungsmethoden, weder für den Netzwerkverkehr, noch für die Konfigurationsdateien, die u.a. den Zugang zur Datenbank inklusive Passwort enthalten. Die Patientendaten werden im Klartext zwischen BGA-Gerät und der Applikation, sowie zwischen Applikation und Datenbank übertragen. Das BGA-Gerät selbst unterstützt von Haus aus keine Verschlüsselung der übertragenen Daten.

4 Zusammenfassung der Ergebnisse

4.1 Erreichte Ziele

Die Applikation ist in der Lage, sich selbstständig und ohne Zutun eines Benutzers um die elektronische Dokumentation der BGA-Werte des ABL725^{TM} zu kümmern und die programminternen Aktivitäten in Logdateien zu dokumentieren. HL7 Nachrichten können validiert und analysiert werden und die Eintragung der Werte samt Kommentarcodes in die Datenbank ist möglich.

Die eindeutige Identifikation des Patienten funktioniert dann, wenn das Personal dessen Barcode einscannt. Die Identifikation über die Zuordnung Bett <-> PatID oder gar nur über das Datum allein ist zwar auch möglich, allerdings unvorteilhaft und nur als Notlösung anzusehen. Die sinnvolle und eindeutige Zuordnung zum Patienten steht und fällt mit der Mithilfe des Pflegepersonals.

Um den Automatismus des Programms zu ermöglichen, muss das Tool natürlich automatisch beim Start des Betriebssystems mitgestartet werden. So fährt das Programm auch dann fort, wenn der Server neugestartet wurde.

Der Abgleich lokaler HL7 Nachrichten mit der Datenbank ist möglich, allerdings muss dabei IS-H*med zugänglich bzw. die Fallnummerntabelle aktuell sein.

Der Neustart der Applikation und die Speicherung der beiden Memos in Logdateien funktioniert wie erfordert.

4.2 Nicht erreichte Ziele

Das Ziel der Abschaffung der papierenen Dokumentation konnte binnen dieser Arbeit noch nicht erreicht werden. Um dies zu erreichen, muss geprüft werden, ob die Applikation korrekt arbeitet und ob das Pflegepersonal richtig mitarbeitet, also stets den Barcode bei der BGA einscannt.

Die Patientenidentifikation und der damit verbundene, geänderte Arbeitsablauf ist der Knackpunkt dieser Arbeit und muss sauber vonstatten gehen. Arbeitet das Intensivpersonal nicht mit, so kann die eindeutige Zuordnung der Werte zum Patienten nicht garantiert werden.

4.3 Plattformunabhängigkeit der Applikation

Das Programm wurde unter Windows XP / Server 2003 und Ubuntu 9.10 Linux ausgeführt und hat keinerlei Fehler im Verhalten gezeigt. Beim Abgleichen lokaler HL7 Nachrichten mit der Datenbank unter Ubuntu Linux werden jedoch die Inhalte der Memofelder nicht vollständig angezeigt (Client: Dell Latitude D505, Centrino 1.6 Ghz). Dabei handelt es sich jedoch um ein

grafisches Problem, die Logdateien sind anschliessend völlig korrekt. Ein Test unter Mac OS hat nicht stattgefunden.

Da das Trennzeichen für Verzeichnisstrukturen unter verschiedenen Betriebssystemen unterschiedlich ist (z.B. „/“ in Linux oder „\“ in Windows), wird dieses Zeichen über die Variable *DirectorySeparator* ermittelt. Bei der Kompilierung auf dem Zielsystem wird dann das dafür eingetragene Zeichen verwendet(21).

Der Neustart der Applikation funktioniert unter Windows genauso reibungslos wie unter Ubuntu Linux.

5 Diskussion

Im Folgenden werden die Ergebnisse dieser Arbeit diskutiert und zuweilen sinnvollere Alternativen zu einigen Punkten, sowie Ergänzungen zur Implementation dargestellt. Desweiteren soll hier ein besonderes Augenmerk auf die möglichen Risiken sowie die Sicherheit der Patientendaten gelegt werden.

Im Allgemeinen stehen folgende Probleme zur Diskussion:

- Der Neustart der Applikation kann dazu führen, dass Logdateien fehlen, wenn es bspw. zu unvorhergesehenen Fehlern beim Speichern kommt. Tritt ein solcher Fehler auf, zwingt der *Timer_restart_app* die Applikation zum Neustart und es werden keine Logdateien dokumentiert.

- Es wäre theoretisch denkbar, dass nach ein paar Wochen hunderte Instanzen des Programms gestartet sind, wenn der Applikationsneustart nicht richtig funktionieren sollte, also wenn sich der alte Prozess nicht beenden lässt, der neue aber gestartet werden konnte.

- Die Konfigurationsdateien liegen unverschlüsselt vor und können zur Änderung der Datenbank genutzt werden, wenn sie in unbefugte Hände fallen sollten.

- Die Idee, ein *Salted Password* zum Zwecke der Sicherheit benutzen, ist nicht hinreichend für den gewünschten Effekt. Ein Angreifer könnte diese Methode kennen und einfach Teile des Passworts entfernen und erneut probieren, ob der Zugang damit nun möglich ist. Diese Vorgehensweise sollte zugunsten einer verschlüsselten Konfigurationsdatei verworfen werden.

- Der Netzwerkverkehr zwischen dem BGA-Gerät und der Applikation, sowie zwischen Applikation und DBMS ist unverschlüsselt und könnte von anderen Teilnehmern im Netzwerk leicht mitgelesen werden, bspw. über ein Tool wie Wireshark.(22)

- Die Einheiten der Parameter werden auf Anforderung hin ignoriert. Die Einheit eines jeden Parameters könnte aber am BGA-Gerät abgeändert werden, was vom Nutzer oft nicht erkennbar ist (Wertebereiche) und vom Programm völlig ignoriert wird. Die Wertebereiche für Glukose in den Einheiten *mmol/L* und *mg/dL* überschneiden sich bei den Testbereichen bspw. nicht, jedoch ist das beim Sauerstoffpartialdruck (Einheiten *mmHg* und kPa) der Fall.(19, Kapitel 14-2, Gemessene Parameter und deren Wertebereiche)

- Der Wert für Bilirubin, die Werte für die Expirationsluft und die Fetalhämoglobinfraktion fehlen und sollten implementiert werden.

- Es besteht eine Bindung an die Datenbankstruktur im Programmcode. Ändert sich etwas in der Datenbank, so muss das Programm umgeschrieben werden.
- Die Strukturierung des Programms kann verbessert werden, gleichwohl könnte man den Ablauf intelligenter gestalten.
- Der Zugang zu IS-H*med sollte nicht extern über ein zusätzliches Programm stattfinden, sondern intern implementiert sein. Wie genau das funktionieren soll, ist in dieser Arbeit nicht evaluiert worden.
- Eine Validierung / Test der Software sollte durchgeführt werden, um Fehler auszuschliessen.
- Es werden keine zwei Konfigurationsdateien benötigt, die Informationen in der Datei *config_match_patid_sap.ini* sind redundant und werden nicht benötigt.
- Da in der Konfigurationsdatei die *Sending Application* unter dem Schlüssel *bga_hl7_name* sehr allgemein gehalten ist, kann beim Empfang einer HL7 Nachricht derzeit nicht zwischen verschiedenen Geräten des gleichen Typs unterschieden werden. Dies wäre aber wichtig, sollte das Programm dahingehend erweitert werden, um Daten verschiedener Geräte anderer Stationen zu dokumentieren. Die Logdateien enthalten den gesamten String der *Sending Application*.
- Kann beim Abgleichen lokaler HL7 Nachrichten der Patient nicht identifiziert werden, obwohl eine korrekte Fallnummer in der Nachricht steht, bspw. beim Ausfall von IS-H*med, so tauchen in der Datenbank doppelte Einträge auf, da durch die PatID *0* ein indexkonformer Eintrag entsteht. Bsp: Der Eintrag mit PatID *1234* und Datum *2010-01-01 10:00:00* existiert, dann existiert nach dem Abgleich ein zweiter mit PatID *0* und Datum *2010-01-01 10:00:00*.
- Es muss garantiert werden, dass beim Abgleich lokaler HL7 Nachrichten nur valide Patientenberichte genutzt werden, da die Validität der Nachrichten vorrausgesetzt wird. Sollte es sich um invalide Nachrichten oder bspw. gar Binärdateien handeln, so stürzt das Programm zwar nicht ab, in der Datenbank liegen dann allerdings fehlerhafte Datensätze vor.
- Wird eine lokale HL7 Nachricht mit der Datenbank abgeglichen, bei der die PatID über die Bettnummer ermittelt wurde, so wird natürlich keine PatID gefunden, da die Fallnummerntabelle bei der Identifikation über die Bettnummer nicht aktualisiert wird. Dadurch entsteht ein doppelter Eintrag in der Datenbank, jedoch mit der PatID *0*.

5.1 Applikationsneustart

Der Neustart der Applikation wird nur durchgeführt, weil ein Verbindungsabbruch u.U. von der TCP Komponente nicht erkannt wird. Erste Tests zeigten, dass bei der LNET TCP-Komponente tatsächlich keine Events wie *OnError* oder *OnDisconnect* auftraten, stattdessen zeigte sich aber unter dem Windows / Linux-Tool *netstat*, dass der TCP-Serversocket aktiv war und nach ein paar Minuten keine hergestellte Verbindung mehr anzeigte, sondern auf *Abhören* gestellt war. Es muss also intern zumindest ein Timeout o.ä. erkannt worden sein.

Wurde nun die Verbindung wieder hergestellt, fuhr das Programm wie gewohnt fort. Da es sich bei diesem Feature allerdings um eine Anforderung handelte, bleibt der Neustart bestehen.

Eine Alternative zu einem Neustart des Programms wäre das Schreiben der Logdateien nach einer gewissen Zeit des Leerlaufs und die zeitgleiche Neuinstanziierung des *TCP_Handler* gewesen, was den gleichen gewünschten Effekt gehabt hätte, wie der Neustart.

5.2 Datensicherheit und Verschlüsselung

Prinzipiell sollte über die Verschlüsselung des Netzwerkverkehrs und der Konfigurationsdateien nachgedacht werden. Die Möglichkeit des Abhörens bzw. Kopierens der Konfigurationsdateien ist technisch gegeben und stellt ganz generell ein Sicherheitsrisiko dar. Die Wahrscheinlichkeit, dass ein solches Ereignis eintritt, mag gering erscheinen, trotzdem aber wäre dieses möglich. Die Abhilfe durch eine Datenverschlüsselung wäre relativ einfach einzurichten.

Um eine Verschlüsselung des Datenverkehrs zwischen BGA-Gerät und Applikation zu ermöglichen, müsste ein Server direkt mit dem BGA-Gerät über ein Crossover Kabel verbunden sein, welcher dann die vom Gerät empfangenen Daten verschlüsselt und über eine zweite Netzwerkkarte an die Applikation weiterleitet. Gleichzeitig wäre damit das Problem mit der nötigen, festen IP-Adresse des Applikationsservers gelöst, da der zwischengeschaltete Server mit der Auflösung von DNS-Namen umgehen könnte. Dieses Vorgehen würde allerdings Mehrkosten durch die Anschaffung des Servers verursachen.

Ein solcher Server könnte auch gleich die Applikation selbst enthalten und nur die Verbindung mit dem DBMS verschlüsseln. Da man das Programm allerdings dahingehend erweitern könnte, dass n BGA-Geräte der selben Serie aus z.B. anderen Abteilungen integriert werden könnten, sollte man in so einem Fall die Applikation zentral für alle halten, um Administrationsarbeit zu sparen.

5.3 Aufnahme der Parametereinheiten in die Datenbank

Problematisch ist die Tatsache, dass die Einheiten der Parameter aktuell ignoriert werden. Ändert ein Mitarbeiter die Einheit des gemessenen Parameters am Gerät, so kann das vom Benutzer nur durch bspw. unmögliche Wertebereiche erkannt werden. Handelt es sich aber um Werte innerhalb der möglichen Wertebereiche, so wird die Fehlerhaftigkeit nicht erkannt.

Abhilfe würde die Implementierung einer Einheitentabelle passend zur Tabelle *pflbga* bringen. Da es sich zwischen der Tabelle *pflbga* und der fiktiven Tabelle *pflbga_units* um eine n:m-Verbindung handelt, da jeder Wert vielen Einheiten und jede Einheit vielen Werten zugeordnet werden kann, wäre noch eine n:m-Verknüpfungstabelle nötig.

Durch diese Änderung und das Auslesen der Einheiten aus der HL7 Nachricht wäre das Problem gelöst.

5.4 Zuordnung der HL7 Struktur zur Datenbankstruktur in der Konfigurationsdatei

Ein weiteres Problem besteht darin, dass die Struktur der Datenbank im Programm fixiert ist. Sollte man die Namen der Spalten in z.B. der Tabelle *pflbga* ändern wollen, so würde das Programm nach dieser Änderung nicht mehr funktionieren.

Es sollte zur Lösung dieses Problems über die Einbettung der Datenbankstruktur in die Konfigurationsdatei nachgedacht werden. Man könnte bspw. eine Liste wie die folgende in die Konfigurationsdatei mit aufnehmen:

ˆpO2ˆ=po2

ˆLacˆ=Lactat

ˆCa++ˆ=Ca

ˆNa+ˆ=Na

...

Diese Liste könnte derart eingelesen werden, als dass vor dem Gleichheitszeichen der Parameterbezeichner aus der HL7 Nachricht steht, und danach der zugeordnete Attributbezeichner aus der Datenbank. So können Bezeichner leicht geändert, hinzugefügt oder gelöscht werden, ohne das Programm anpassen zu müssen.

5.5 Fehlende Parameter

Durch die Abbildung der Datenbankstruktur in der Konfigurationsdatei könnte gleichzeitig die Implementierung der fehlenden Parameter *cBil*, *F*HbF und pO_2 / pCO_2 der Expirationsluft leicht aktiviert werden.

Die Parameter fehlen, weil sie in der HNO Intensivstation nicht ermittelt werden und für

die Lösung keine Rolle spielten, sollten aber der Vollständigkeit halber auf jeden Fall noch mit aufgenommen werden.

5.6 Überarbeitung der Programmstruktur, Test und Validierung

Das Programm ist in fünf Units nach Zuständigkeiten hin aufgeteilt und sollte nach Möglichkeit in weniger Units (oder nur eine) zusammengeführt werden, da der aktuelle Aufbau recht kompliziert ist. Es sollte eine nachträgliche Überarbeitung der Klassen geben und bspw. anstatt des Zurücksetzens globaler HL7 Variablen einfach eine Neuinstanziierung des Objekts *ABL725_HL7_Reader* stattfinden. Dasselbe gilt für den *Database_Handler*.

Die Programmaufteilung ist so geregelt, um eine Übersicht über die Verantwortlichkeiten zu behalten und um die Abläufe möglichst leicht verstehen zu können. Es gibt also ein Objekt, das sich um die Bewältigung der TCP-Verbindung kümmert (TCP_Handler), eines für die Datenbankverbindung (Database_Handler), ein Objekt für die GUI (fo_view), eines für die Analyse der HL7 Nachrichten (*ABL725_HL7_Reader*) und eines für die Reaktion auf Events und die Organisation des Ablaufs (*App_Controller*).

Das Programm bietet eine sehr gute Performance bei der Abarbeitung einer HL7 Nachricht. Die gesamte Verarbeitung inklusive Speicherung in der Datenbank dauerte dabei auf einem Testsystem (Dell Latitude D505 mit Centrino 1,6 GHz) weniger als eine Sekunde. Es könnten sich jedoch schwerwiegende, noch unentdeckte Fehler im Programm verstecken.

Denkbar wäre, dass das Programm - wenn der Neustart des Programms nicht stattfindet - mit der Zeit immer mehr Arbeitsspeicher belegen könnte, da das Programm ja dann u.U. über Monate hinweg laufen würde. Dieser Fehler ist zwar durchaus realistisch, würde zur Zeit aber nicht ins Gewicht fallen, da das Programm nach relativ kurzer Zeit ja wieder neugestartet wird.

Die Überarbeitung des Programmaufbaus könnte im Zuge eines umfassenden Tests und einer Softwarevalidierung durchgeführt werden.

Weiter sollte geprüft werden, ob das Programm die Parameter richtig einliest. Es sollten alle empfangenen HL7 Nachrichten mit der Datenbank abgeglichen werden und auch sollte ermittelt werden, ob die Zuordnung zum Patienten korrekt stattgefunden hat.

Ein Test kann mit Hilfe des mit dieser Arbeit mitgelieferten *Fake_BGA* Programms durchgeführt werden, wenn kein echtes BGA-Gerät zur Verfügung steht. Das Programm simuliert ein Radiometer^{TM} ABL725^{TM} Gerät und verschickt alle 3 Sekunden die beiliegende *patientenbericht.HL7*. Das Programm ermittelt beim Start den DNS Namen des lokalen Rechners, verbindet sich mit einem auf demselben Rechner gestarteten *ABL725 HL7 Receiver* und verschickt dann die Nachricht.

Da die Funktion *GetComputerName* aus der Unit *windows* benötigt wird, funktioniert das

Fake_BGA nur unter Windows. Der DNS Name wird statt des *localhost* genutzt, weil die TCP Verbindung damit unter Windows XP nicht funktionierte.

Man könnte hier alle erdenklichen Kombinationen von HL7 Nachrichten durchtesten, um die Fehlerfreiheit des *ABL725 HL7 Receivers* zu bestätigen.

5.7 Risiken

Es gibt verschiedenste Risiken beim Einsatz der Applikation. Folgende Punkte sollten überdacht werden und eine Validierung der Software sollte durchgeführt werden:

- Die Zuordnung der FallNr zur PatID muss in IS-H*med korrekt sein, da dieser Zuordnung blind vertraut wird.
- Sollte ein Fehler bei der Ausführung des Subprozesses *FallNr2PatID* auftreten, so hängt die Applikation, da auf die Terminierung des Subprozesses gewartet wird.
- Sollte in dem Zeitfenster vom Applikationsneustart bis zur Wiederverbindung des BGA-Geräts, in dem keine HL7 Nachrichten empfangen werden können, eine Nachricht vom BGA-Gerät abgeschickt werden, so ist diese Nachricht verloren. Das BGA-Gerät verbindet sich derzeit nach einem Intervall von *60* Sekunden mit der Applikation, wenn keine Verbindung bestehen sollte. Die Wahrscheinlichkeit, dass genau dann eine BGA stattfindet, ist klein, aber trotzdem gegeben.
- Die Logdateien müssen regelmäßig kontrolliert werden, da Fehler sonst unerkannt bleiben. Mögliches Lösungsszenario: Bei einem aufgetretenen Fehler könnte eine Mail mit einer Zusammenfassung des Vorfalls an den Administrator verschickt werden.

5.8 Alternativen zur Applikation und die wirtschaftliche Bedeutung der Open Source Entwicklungsprogramme

Die Hersteller von Laborgeräten wie dem Radiometer^{TM} ABL725^{TM} bieten eigene Softwarelösungen an, um die anfallenden Daten ihres eigenen Gerätes auslesen zu können, teils weil die Kunden es wünschen, bspw. um das Gerät auch fernwarten zu können, teils aber auch aus wirtschaftlichen Interessen. Die Kommunikationsmöglichkeiten von und mit Laborgeräten sind vorwiegend proprietär und generieren völlig unnötige Kosten für eine Klinik. Der HL7 Standard dagegen ist umfangreich und kostenlos und kann dem Klinikum einiges an Lizenzgebühren sparen.

Alternativ zu diesem hier entwickelten Programm könnte auch einfach die Software RADIANCE^{TM} des Herstellers genutzt werden. Dadurch würden aber Lizenzkosten generiert werden, die mit der hier vorgestellten Lösung schlicht entfallen. Der Funktionsumfang des *ABL725 HL7 Receivers*

ist im Vergleich zu einer solchen Hersteller-Lösung selbstverständlich sehr gering, erfüllt aber auf pragmatische Art und Weise den Zweck der elektronischen Dokumentation der Daten.

Verantwortliche in Krankenhäusern sollten dringend darauf achtgeben, dass die eingesetzten Geräte von Haus aus HL7 fähig sind, um das ohnehin schon teure Gesundheitswesen nicht noch zusätzlich zu belasten. Die in einem Klinikum eingesetzten Informatiker können sich dann ohne große Schwierigkeiten um die Dokumentation der Laborberichte kümmern.

Durch den hier gezeigten Einsatz von Open Source Entwicklungsprogrammen konnte erwiesen werden, dass die elektronische Dokumentation von Labordaten beinahe lizenzkostenfrei umgesetzt werden kann. Würde die FallNr und PatID des Patienten sofort bei dessen Aufnahme in die ePA Datenbank gespeichert werden, so wäre der Zugang zu IS-H*med für diese Lösung völlig irrelevant geworden. Durch die Plattformunabhängigkeit wäre dann auch ein Umstieg hin zu lizenzkostenfreien Betriebssystemen (bspw. Linux Server) ohne Probleme möglich.

Da sich in einem Klinikum eine ganze Reihe kommerzieller Software befindet, sollte in jedem Fall ausführlich über die Möglichkeiten der Abschaffung derselben nachgedacht werden. Der Einsatz von Programmen, die der Klinik selbst gehören, sorgt für Unabhängigkeit, Einsparungen und es kann agil auf seitens der Ärzte- und Pflegerschaft geäußerte Wünsche bezüglich Funktionserweiterungen und -änderungen reagiert werden.

5.9 Ausblick

Die Zukunft dieser Arbeit könnte darin liegen, dass man die zuvor genannten Probleme löst, eine Validierung des Tools durchführt und das Programm befähigt, Daten von mehr als nur einem BGA-Gerät aus bspw. anderen Abteilungen gleichzeitig zu dokumentieren.

Auch könnte man dieses Programm zu einem Modul für einen generalisierten HL7 Empfänger umbauen, welcher dann viele verschiedene Labor- oder intensivmedizinische Geräte anbinden könnte. Bspw. könnte dann ein anderes Modul für die Dokumentation der Werte der EKG[34]-Monitore der HNO Intensivstation, o.ä. entwickelt werden.

Sollte z.B. in einem Labor bereits ein Kommunikationsserver[35] existieren, so könnte man das Programm mit diesem verbinden und die anfallenden Daten der daran angeschlossenen Laborgeräte auf dieses Tool zwecks Dokumentation umleiten.

[34]EKG: Elektrokardiogramm, Methode zur Darstellung der elektrischen Aktivität des Herzens

[35]Kommunikationsserver: Server, der Schnittstellen für z.B. Laborgeräte anbietet, die unterschiedlichen Standards übersetzt und die Daten umleiten kann

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] radiometer.de. Radiometer Webseite. http://www.radiometer.de/, 2010. [Stand: 20. März 2010].

[2] Marco Monnig. Einführung für pflegerische Mitarbeiter , Blutgasanalyse. *intensiv - Fachzeitschrift für Intensivpflege und Anästhesie, Seiten 48-59*, 10 2002.

[3] Lazarus wiki. Free Pascal - wiki.lazarus.freepascal.org. http://wiki.lazarus.freepascal.org/Overview_of_Free_Pascal_and_Lazarus/de, 2010. [Online; Stand 18. März 2010].

[4] Wikipedia. Socket (Software) — Wikipedia, Die freie Enzyklopädie, 2010. [Online; Stand 15. März 2010].

[5] Wikipedia. Transmission Control Protocol — Wikipedia, Die freie Enzyklopädie, 2010. [Online; Stand 15. März 2010].

[6] iana.org. IANA Webseite. http://www.iana.org/assignments/port-numbers, 2010. [Stand: 20. März 2010].

[7] lazarus wiki. LNet - lazarus wiki. http://wiki.lazarus.freepascal.org/lNet, 2010. [Stand: 19. Februar 2010].

[8] Wikipedia. SQL — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=SQL&oldid=70823708, 2010. [Online; Stand 20. Februar 2010].

[9] MySQL Webseite. MySQL - Warum MySQL? http://www.mysql.de/why-mysql/, 2010. [Online; Stand 18. März 2010].

[10] MySQL Webseite. MySQL Workbench. http://www.mysql.de/products/workbench/, 2010. [Stand: 08. März 2010].

[11] firmos.at. ZeosLib Webseite. http://zeos.firmos.at/portal.php, 2010. [Stand: 19. März 2010].

[12] UMLet.com. UMLet Webseite. http://www.umlet.com/, 2010. [Stand: 08. März 2010].

[13] gnome.org. Dia Webseite. http://live.gnome.org/Dia, 2010. [Stand: 19. März 2010].

[14] HL7.org. HL7 Webseite. http://www.hl7.org, 2010. [Online; Stand 18. März 2010].

[15] HL7 Benutzergruppe in Deutschland e.V. Was ist HL7. http://hl7.de/standard/wasist_hl7.php, 2010. [Online; Stand 22. Februar 2010].

[16] HL7.org. HL7 Webseite - FAQs. http://www.hl7.org/about/FAQs/index.cfm, 2010. [Online; Stand 18. März 2010].

[17] Firma Radiometer. ABL700 Serie Protokollspezifikation. http://www.radiometer.de, Januar 2003. Code Number: 989-329.

[18] HL7 Benutzergruppe in Deutschland. Z-Segment-Register. http://www.hl7.de/link/z-segmente.php, 2010. [Online; Stand 22. Februar 2010].

[19] Firma Radiometer. ABL700 Serie Bedienerhandbuch bis Programmversion 6, April 2008. Ausgabe: 200804O.

[20] drott.at. Bedeutung der Parameter einer BGA. http://www.drott.at/medizintechnik/blutgasanalyse/parameter.html, 2010. [Stand: 22. März 2010].

[21] lazarus wiki. Multiplatform Programming Guide - lazarus wiki. http://wiki.freepascal.org/Multiplatform_Programming_Guide/de, 2010. [Stand: 17. März 2010].

[22] wireshark.org. Wireshark Webseite. http://www.wireshark.org/, 2010. [Stand: 20. März 2010].