Technische Universität Dresden

Fakultät für Elektrotechnik und Informationstechnik

Insitut für Biomedizinische Technik

Manuskript Diplomarbeit

Thema: Entwicklung von Methoden zur Analyse und Aufbereitung biomedizinischer Messdaten

Vorgelegt von: Enrico Grunitz

Betreuer: Dr.-Ing. Sebastian Zaunseder
Eng. Fernando Andreotti

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Hagen Malberg

Tag der Einreichung: XX. MONAT 2012

Selbständigkeitserklärung

Mit meiner Unterschrift versichere ich, dass ich die von mir am heutigen Tag eingereichte Diplomarbeit zum Thema

Entwicklung von Methoden zur Analyse und Aufbereitung biomedizinischer Messdaten

vollkommen selbständig und nur unter zuhilfenahme der angegebenen Quellen und Hilfsmittel erstellt habe. Zitate fremder Quellen sind als solche gekennzeichnet.

Dresden, den 24. September 2012

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abb.	Abbildung
EKG	Elektrokardiogramm
GUI	grafische Benutzeroberfläche *(graphical user interface)*
LGPL	*GNU Lesser General Public License*
PPG	Puls-(Foto-)Plethysmographie
Tab.	Tabelle
UML	*Unified Modeling Language*

1. Einleitung

1.1. Motivation

Ergebnisse automatisierter Biosignalverarbeitungsmethoden werden aus mehreren Gründen oftmals manuell nachbearbeitet. So erfordert die Entwicklung neuer Methoden häufig eine Verifikation der Ergebnisse und eine eventuelle Korrektur der automatisch generierten Ausgabe. Zusätzlich ist eine schnelle visuelle Überprüfung von Ergebnissen, um einen ersten Eindruck über den Effekt einer Änderung an einer Methode zu bekommen, ein Mittel, das in der Entwicklungphase genutzt wird. Daher besteht eine Notwendigkeit eines Werkzeugs, welches die Visualisierung übernimmt und den Entwickler beim Editieren von Messdaten und Ergebnissen der Signalverarbeitung unterstützt.

Ein solches Werkzeug kann durch die Definition und Festlegung von Ein- und Ausgabeformaten zu einer Vereinheitlichung von Datenformaten führen. Durch die Bereitstellung eines solchen Werkzeugs für Dritte kann auch die methodische Grundlage für die Kooperation verschiedener Institutionen geschaffen werden. Um solche Kooperationen zu unterstützen sollte es, aufgrund der unterschiedlichen Voraussetzungen, wenig spezialsierte Anforderungen an seine Umgebung stellen.

1.2. Zielstellung

Das Ziel dieser Arbeit ist ein Programm zu konzipieren und umzusetzen, das unterschiedliche (Bio-) Signale grafisch darstellt und dem Nutzer die Möglichkeit bietet, Zeitpunkte und -intervalle innerhalb des Signalverlaufs zu markieren und mit Kommentaren zu versehen. Hierbei soll insbesonderer die gleichzeitige Darstellung mehrerer Signale unterschiedlicher Natur und Ausprägung unterstützt werden. Die Erstellung und Bearbeitung von Markierungen soll leicht verständlich aus der grafische Benutzeroberfläche (GUI) heraus geschehen. Zudem soll eine Grundlage geschaffen werden, paralell aufgenommene Signale in einem Datensatz zu vereinen.

Zusätzlich soll eine zukünftige Erweiterung der Funktionalität ermöglicht und unterstützt werden. Daher ist eine klare Gliederung der Einzelkomponenten gefordert und die Dokumentation des Quelltextes sowie der einzelnen Programmteile fundamentaler Bestandteil der Aufgabenstellung.

Um die Erweiterbarkeit zusätzlich zu verbessern, soll die spätere Einbindung von Methoden der Signalverarbeitung vorbereitet werden. Dafür soll eine einfache Signalverarbeitungsfunktion in das Programm implementiert werden und in die Benutzeroberfläche integriert werden. Die Arbeit eines zukünftigen Entwicklers wird somit durch die beispielhafte Integration einer zusätzlichen Methode vereinfacht.

Neben der Entwicklerdokumentation soll auch eine seperate Dokumentation für die Benutzer des Programms zur Verfügung gestellt werden. In dieser Nutzerdokumentation soll dem Anwender die Funktionsweise und Bedienung des Programms verständlich gemacht werden.

1.3. Konkretisieren der Aufgabenstellung

Programmiersprache Java

Aufgabenteilung: Funktion — zukünftige Entwicklung — Benuzterschnittstelle

Herangehensweise: Anwendungsfälle — Anforderungesliste — Testcases zur Überprüfung —» bezug auf Softwareentwicklungsliteratur

2. Vorbetrachtungen

2.1. Allgemeine Softwareentwicklung

In dieser Arbeit soll der Begriff Softwareentwicklung den Prozess benennen, der alle Aktivitäten und die damit verbundenen (Zwischen-) Ergebnisse bei der Erstellung von Software umfasst. In der Literatur wird auch der Begriff Softwareprozess genutzt [16]. Obwohl verschiedene Vorgehensmodelle für das Erstellen von Software existieren, haben alle Softwareentwicklungsprozesse vier grundlegende Arbeitsaktivitäten gemeinsam [3, 16]:

Softwarespezifikation: Es müssen die konkreten Anforderungen an das zu erstellende Programm ermittelt werden. Dabei wird die Funktion der Software, aber auch die Grenzen der Benutzung definiert.

Softwareentwurf und -implementierung: Nach Analyse der ermittelten Anforderungen kann die Architektur des Softwaresystems entworfen werden. Durch die Zerlegung der Software in mehrere Subsysteme werden die Anforderungen in kleine Teilprobleme aufgeteilt. Diese Teilprobleme sind den verschiedenen Komponenten und Objekten der Software zugeordnet und können seperat gelöst werden. Diese Lösung wird durch den Objektentwurf und die schlußendliche Implementierung des jeweiligen Programmteils erreicht.

Validierung der Software: Die fertige Software muss auf ihre Funktionsfähigkeit überprüft werden. Hierbei ist auch abzusichern, dass die Software neben der gewünschten Funktionalität keine ungewollten Nebeneffekte auftreten.

Weiterentwicklung: Ein Programm muss sich im Laufe seiner Lebenszeit weiter entwickeln um den sich ändernden Nutzeranforderungen gerecht zu werden.

Der Punkte der Softwarespezifikation ist im Kapitel 3 abgehandelt. Im Kapitel 4 wird neben der Struktur und dem Aufbau des Programms auch auf die Fragen bezüglich Entwurfsentscheidungen eingegangen. Durch die Validierung der Software muss sicher gestellt werden, dass die Software

die Erwartungen des Benutzers erfüllt und den an sie gestellten Bedingungen gerecht wird. Dieser Punkt ist im Abschnitt 3.3 behandelt. Zudem muss das Programm auch schon während der Entwicklung immer wieder auf die korrekte Funktionsweise überprüft werden. Dazu wird auch auf Tests der einzelnen Komponenten in den jeweilegen Abschnitten im Kapitel 4 eingegangen. Der letzte der vier oben genannten Aspekte ist in dieser Arbeit ein nicht zu vernachlässigender Punkt. Weil gerade das Programm für die Nutzung während der Entwicklung von Signalverarbeitungsmethoden erstellt wird, werden die Anforderungen an das Programm fortlaufend wachsen. Somit soll schon von Beginn an die Erweiterbarkeit dieses Projektes unterstützt und gefördert werden. Deshalb sind auch in den folgenden Anforderungen Punkte enthalten die diesen Aspekt besonders hervorheben und explizit fordern.

Für die in dieser Arbeit zu entwickelnden Sofware wird eine Methode der inkrementellen Entwicklung genutzt. Die Herangehensweise dieser Methode ist in Abbildung (Abb.) 1 dargestellt. Nach der anfänglichen Feststellung und Definition der grundlegenden Anforderungen an die zu programmierende Software, wird die allgemeine Struktur des Programms festgelegt. Die Subkomponenten des Programms werden einzeln entworfen und in das bestehende Programm integriert. Da die neuen Subkomponenten immer Teilanforderungen des Programmes erfüllen, steigert sich somit die Funktionalität des Programms. Dieser Ansatz hat den Vorteil, dass die Software schon zeitig im Entwicklungsstadium getestet werden kann und damit auch schon Erfahrungen gesammelt werden können. Zusätzlich können gewünschte Änderungen der Bedienung oder der Funktionalität erkannt und implementiert werden.

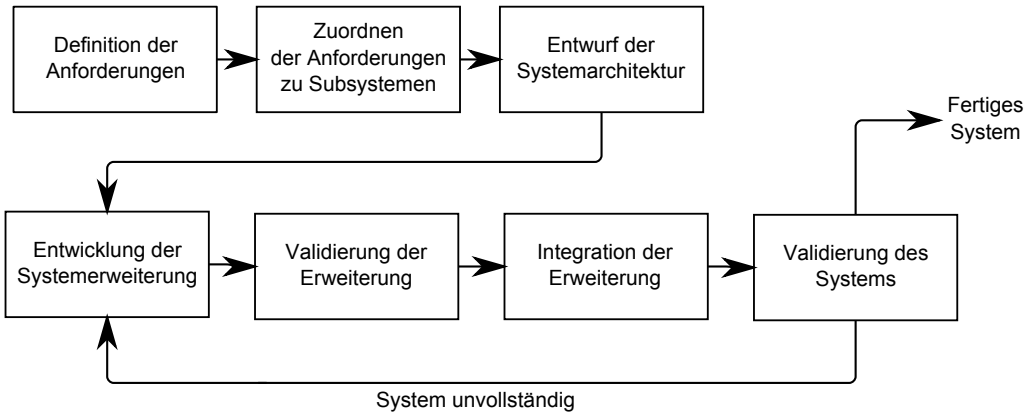

Abbildung 1.: Inkrementeller Softwareentwicklungsansatz nach [16]

2.2. [WIP]Genutzte Biosignale zur Programmvalidierung

– kurze Beschreibung für: fetale EKG-Daten, Puls-(Foto-)Plethysmographie (PPG), evtl. EKG allgemein – warum annotieren notwendig

3. Spezifikation der Programmfunktionalität

Die Spezifikation der Software ist in dieser Arbeit in zwei Etappen aufgeteilt:

- Aufstellen von Anwendungsszenarien
- Ableitung der notwendigen Anforderungen

Dabei ist das Ziel eine Liste mit konkreten Anforderungen an das zu erstellende Programm. Das Aufstellen der Anwendungsszenarien dient einerseits dem Entwickler einen Überblick über die Gesamtproblematik zu erhalten. Zusätzlich dazu wird ersichtlich, welche Arbeitsschritte notwendig sind und wie diese zeitlich zu einander ausgeführt werden. Damit wird die vom Benutzer beabsichtigte Aktion in seine fundamentalen Bestandteile aufgeteilt. Diese Bestandteile sind somit Funktionen die das Programm bereit stellen muss. Es sind die zu erfüllenden Anforderungen.

3.1. Anwendungsszenarien

Der erste Schritt stellt das Ausarbeiten von Szenarien dar, die eine Beschreibung eines Merkmals des Programmes aus Sicht des Anwenders ist. Aufgrund dieser informellen Beschreibungen häufiger Arbeitsabläufe und typischer Aufgabenstellungen wird eine Übersicht gewonnen, was die zu erstellende Software leisten soll und der Nutzer erwartet. Mithilfe dieser Erwartungen können im Anschluss die funktionalen Anforderungen an das Programm formuliert und festgelegt werden. Das Ergebnis ist somit die Beschreibung des notwendigen (Software-) Systemumfangs und der zu implementierenden Arbeitsprozesse.

Der Anwender möchte ...

a) einen Datensatzes laden. Dieser Datensatz umfasst mehrere (Bio-) Signale die sowohl mit einer konstanten Abtastrate erfasst wurden als auch Signale die nicht zu äquidistanten Zeitpunkten abgetastet wurden.

b) einen geladenen Datensatz mit allen Änderungen speichern. Hierbei sollen auch Einstellungen gespeichert werden, die die optische Präsentation wiederspiegeln.

c) sich Informationen zu dem geladenen Datensatz und seinen beinhalteten Signalen anzeigen lassen und verändern.

d) bestimmte Signale des Datensatzes auswählen und sich diese in ihrem Verlauf anzeigen lassen (Signalansicht). Hierbei möchte er Bildschirmgröße der einzelnen Ansichten verändern.

e) die Signalansicht bezüglich der Zeit- und der Amplitudenachse vergrößern und verkleinern können (Zoomen). Entlang der Zeitachse möchte er sie verschieben können (Scrollen). Signaleverläufe die parallel aufgenommen wurden, sollen auch zusammen gescrollt werden.

f) in einer Signalansicht mehrere Signale mit denselben Achsen darstellen lassen. Beispielsweise um ein Roh- und ein verarbeitetes Signal miteinander vergleichen zu können.

g) einen Amplitudenbereich eines Signals optisch hervorheben.

h) einzelne Zeitpunkte im Signalverlauf mit einer Markierung versehen und kommentieren. Diese Markierung kann sowohl für ein bestimmtes Signal gelten, aber auch für alle Signale des Datensatzes.

i) einen Zeitabschnitt markieren. Die Markierung der Abschnitte soll analog zur Markierung von Zeitpunkten erfolgen.

j) die Markierungen verändern (zeitlich verschieben, umbennen) oder löschen.

k) Markierungen gemeinsam mit dem Datensatz aber auch unabhängig vom Datensatz abspeichern.

3.2. Anforderungen an das Programm

Mithilfe der oben beschriebenen Anwendungsszenarien kann daraus die konkrete Funktionalität der Software definiert werden. Diese Definition erfolgt durch die Bestimmung konkreter Anforderungen an das Programm. Dabei beschreiben die Anforderungen die konkret umzusetzenden Funktionen und Arbeitswerkzeuge. Zusätzlich wird die in Kapitel 5.1 beschriebene Validierung der Software die in diesem Abschnitt ausgearbeiteten Definitionen als Grundlage nehmen um das erstellte Programm zu überprüfen und zu bewerten.

Die folgende List von zu erfüllenden Anforderungen ergibt sich aus den oben beschreibenen Anwendungsszenarien. Wenn mehrere Einzelanforderungen in einer Beschreibung enthalten sind, sind diese mit einer Ziffer in Klammern markiert. Das Programm ...

A) muss eine grafische Benutzeroberfläche besitzen.

B) muss ein Datensatzformat unterstützen, das äquidistant (1) und nicht äquidistante (2) abgetastete Signale speichern kann.

C) soll in der Lage sein, Daten aus einem Datensatz zu laden (1). Dem Nutzer muss es ermöglicht werden, diese Signaldaten aus einer Übersicht auszuwählen (2) und in Diagrammen darstellen zu lassen.

D) muss dem Nutzer die Möglichkeit bieten allgemeine Informationen sowohl über den Datensatz (1) als auch über die enthaltenen Daten (2) anzuzeigen.

E) muss in der Lage sein, die Signalverläufe sowohl einzeln (1) in einem Diagramm darzustellen, aber auch mehrere verschieden Signalverläufe (2) in ein und demselben Diagramm zu visualisieren. Diese Signalansichten sollten in ihrer Darstellungsgröße durch den Nutzer veränderbar sein (3).

F) soll dem Benutzer ermöglichen, seine Signalansicht frei „bewegen“ zu können. Es muss eine Vergrößerung und Verkleinerung bezüglich der Abszissen- und der Ordinatenachse unterstützen (1). Zusätzlich ist die Fähigkeit des Verschiebens der Ansicht gefordert (2). Dabei sollen mehrere Diagramme gleichzeitig Verschoben werden können (3).

G) muss in der Lage sein einen Amplitudenbereich ein oder mehrerer Signalansichten optisch hervorzuheben.

H) soll dem Nutzer ein Werkzeug zur Verfügung stellen, das ihm erlaubt Datenpunkte zu annotieren (1). Diese Annotationen sollen optisch in den Signalansichten ersichtlich sein (2) und mit einem Kommentar versehen werden können (3). Ferner ist gefordert, dass vorhandene Annotationen veränderbar sind (4).

I) soll neben der Annotation einzelner Datenpunkte auch die Markierung von Signalbereichen unterstützt werden.

J) muss Änderungen an den Signalen selbst (1) und den Annotationen (2) speichern können. Annotationen müssen unabhängig von Signalen gespeichert werden können (3). Insbesondere dürfen Annotationen sich nicht verändern, wenn sich das Ursprungssignal verändert oder nicht mehr vorhanden ist (4).

K) soll interne Einstellungen abspeichern und von einer Sitzung zur nächsten übernehmen (1). Optionen bezüglich der Darstellung von Signalen sollen in dem Datensatz mit abgespeichert werden können (2).

Die in der Aufgabenstellung geforderte Ausbaufähigkeit der Programms ist nicht durch die Anwendungsszenarien abgedeckt werden. Hierbei handelt es sich um eine nichtfunktionale Anforderung an das Programm. Daher wird die folgenden Anforderung nur auf Basis der Aufgabenstellung formuliert und nicht aufgrund der Erwartungshaltung des Benutzers:

L) Das Programm soll dem Benutzer ermöglichen eine Signalverarbeitungsmethode auf ein gewähltes Biosignal anwenden zu können (1). Dabei muss das Originalsignal unverändert bleiben (2). Der bearbeitete Signalverlauf kann als eigenes Signal im Datensatz abgespeichert werden (3). Die Implementierung dieser Anforderung soll beispielhaft für zukünftige Entwickler erfolgen um die Erweiterbarkeit zu gewährleisten.

3.3. Testszenarien

In diesem Abschnitt sollen Szenarien heraus gearbeitet werden, mit denen die Software am Ende der Implementierung validiert werden kann. Dabei soll die oben geforderte Funktionalität anhand der Behandlung von Biosignalen überprüft werden.

Folgend sollen die Testszenarien beschrieben werden die zur Validierung der Software durchgeführt werden sollen. Die Ausgangsbedingung ist eine das einfache Starten des Programmes. Somit soll der erste, nicht jeweils explizit genannte Schritt sein, das Programm zu starten. Abgeschlossen wird jedes Testszenario mit dem Speichern der geladenen Daten.

Testszenario 1 Der Benutzer lädt einen Datensatz abdominaler EKG-Daten mit sieben Aufnahmekanälen. Zusätzlich sind noch die approximierten EKG-Signale des Fetus sowie der Mutter im Datensatz gespeichert. Der Benutzer lässt sich alle verfügbaren Informationen zu dem geladenem Datensatz anzeigen. Anschließend lässt er sich einen Kanal sowohl des Rohsignals als auch der beiden abgeleiteten Signale jeweils in einer eigenen Ansicht anzeigen. Er verschafft sich durch eine geringe Zoomstufe einen Überblick über die Signalverläufe. Der Benutzer zoomt auf interessante Bereiche der Aufnahme herein und vergrößert die Ansicht der Einzelsignale. Er schließt die Ansicht des Rohsignals. Der Benutzer markiert in zwei unterschiedlichen, neu zu erstellenden Annotationskanälen die QRS-Komplexe der Mutter, sowie des Fetus (über mindestens fünf Minuten des Signalverlaufs). Zur Überprüfung der An-

notationen lässt er sich alle Signale und die gemachten Annotationen in einer Signalansicht darstellen.

Testszenario 2 Der Benutzer lädt den im Testszenario abgespeicherten Datensatz wieder in das Programm. Er überprüft ob die Ansichten und die Einstellungen aus dem ersten Testszenario übernommen wurden. Der Benutzer speichert die Einstellungen der Signalansichten. Er wählt fünf beliebige Annotationen aus versieht diese mit Kommentaren. Der Benutzer löscht jede zweite Annotation des fetalen QRS-Komplexes. Weiterhin soll er mindestens 10 Zeitbereiche markieren, wobei es auch zu Überschneidungen diese Bereiche kommen soll. Er lädt die zuvor gespeicherten Einstellungen und überprüft ob diese richtig geladen wurden.

Testszenario 3 Der Benutzer lädt den Bearbeiteten Datensatz aus dem zweiten Testszenario. Er entfernt die Kanäle der approximierten EKG-Verläufe aus dem Datensatz. Der Benutzer überprüft die gemachten Annotationen mithilfe des Rohsignals.

Testszenario 4 Der Benutzer lädt einen Datensatz mit EKG-und PPG-Signalen. Er wählt PPG und EKG Kanäle und zeigt sie sich in einer Ansicht an. Der Benutzer verschafft sich eine Übersicht durch eine geringe Zoomstufe über die Signalverläufe. Der Benutzer markiert einen Amplitudenbereich im PPG-Signal. Er wendet eine Filterfunktion auf das EKG-Signal an und speichert das Ergebnis im Datensatz ab. Er kontrolliert die Informationen des veränderten Signals. Der Nutzer entfernt das Original-EKG-Signal aus dem Datensatz.

Testszenario 5 Der Benutzer lädt den Datensatz aus dem viertem Testszenario. Er kontrolliert dabei die Einstellungen der Ansichten darauf, ob sie aus dem Szenario 4 übernommen wurden. Er annotiert die QRS-Komplexe des gefiltertem EKG-Signals für mindestens fünf Minuten des Signalverlaufs. Der Benutzer wendet eine weitere Filterfunktion auf das bereits gefilterte Signal an und speichert das Ergebnis erneut im Datensatz ab. Er kontrolliert die Annotationen mit dem erneut gefiltertem Signal.

In Tabelle (Tab.) 1 ist übersichtlich aufgelistet welche der Anforderungen durch die Testszenarien abgedeckt sind. Es ist erkenntlich, dass alle Anforderungen mindestens durch ein Testszenario überprüft wird. Durch das Bestehen der Testszenarien wird gezeigt, dass das Programm die Erwartungen des Nutzers erfüllt.

Das Testen der Funktionalität mittels bestimmter Biosignale sind aber nur spezielle Einzelfälle. Es kann durch sie nicht die absolute Fehlerfreiheit der Software gezeigt werden. Um entstehende Fehler schon während der Entwicklung abfangen und beheben zu können, wird die Software bzw. die einzelnen Programmteile auch schon einzeln daraufhin getestet, dass sie sich so verhalten, wie es

Tabelle 1.: Abdeckung der Anforderungen durch die Testszenarien (TS)

	Anforderung															
	A	B1	B2	C1	C2	D1	D2	E1	E2	E3	F1	F2	F3	G	H1	H2
TS 1	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
TS 2	✓	✓		✓	✓			✓			✓	✓	✓			✓
TS 3	✓	✓		✓	✓			✓			✓					✓
TS 4	✓	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓		
TS 5	✓	✓	✓	✓	✓			✓			✓	✓	✓	✓		✓

	Anforderung											
	H3	H4	I	J1	J2	J3	J4	K1	K2	L1	L2	L3
TS 1					✓							
TS 2	✓	✓	✓		✓			✓	✓			
TS 3						✓	✓		✓			
TS 4				✓						✓	✓	✓
TS 5				✓	✓	✓	✓	✓	✓	✓	✓	✓

der Programmierer vorgesehen hat. Speziell wird auch das korekte Verhalten im Falle eines Fehlers überprüft. Diese fortlaufenden funktionellen Test werden in den entsprechenden Abschnitten zu den einzelnen Programmkomponenten im Kapitel 4 beschrieben.

4. Programmentwurf

In diesem Kapitel möchte der Autor die Entwicklung des Gesamtprogramms erörtern. Es wird ein Überblick über das umfassende Konzept des internen Aufbaus gegeben um anschließend auf die konkrete Umsetzung der einzelnen Bestandteile einzugehen. Demzufolge sind die kommenden Abschnitte nach den Programmkomponenten gegliedert. In jedem einzelnen Abschnitt wird auf drei Punkte eingegangen:

- Grundlegende Idee und Designkonzept
- Implementierungsdetails
- Tests zur Verifikation der einzelnen Komponente

Es sei darauf hingewiesen, dass die eigentlichen Entwicklung und Implementierung nicht Komponentenweise, sondern nach entsprechenden Funktionalitäten statt findet. Das Einbinden einer bestimmten Funktionalität betrifft oft mehrere Komponenten und ihr Zusammenspiel, wodurch der Ausbau der einzelnen Bestandteile parallel geschieht. Die Gliederung dieses Kapitels nach den Programmteilen dient jedeglich der Übersichtlichkeit.

4.1. [WIP]Überblick über das Gesamtprogramm

– theoretische beschreibung des gesamtkonzeptes – komplette auflisten der subkomponenten in Tab. 2 – graphische Übersicht der Bestandteile siehe Abb. 2

4.2. Datenbehandlung

Aufgrund der Vielfalt der genutzten Datenformate zur Speicherung von Biosignalen und der Nichtexistenz eines einheitlichen Standards [15, 19, 20] muss ein Format gesucht werden, dass zur Behandlung von Datensätzen im Rahmen dieser Arbeit geeignet ist. Die Wahl des Datensatzformates stützt sich auf die vergleichende Untersuchung [15] von Dr. Alois Schlögl der Technischen Universität Graz. Das Unisens-Format besitzt eine Referenzimplementierung mit Programmierschnittstel-

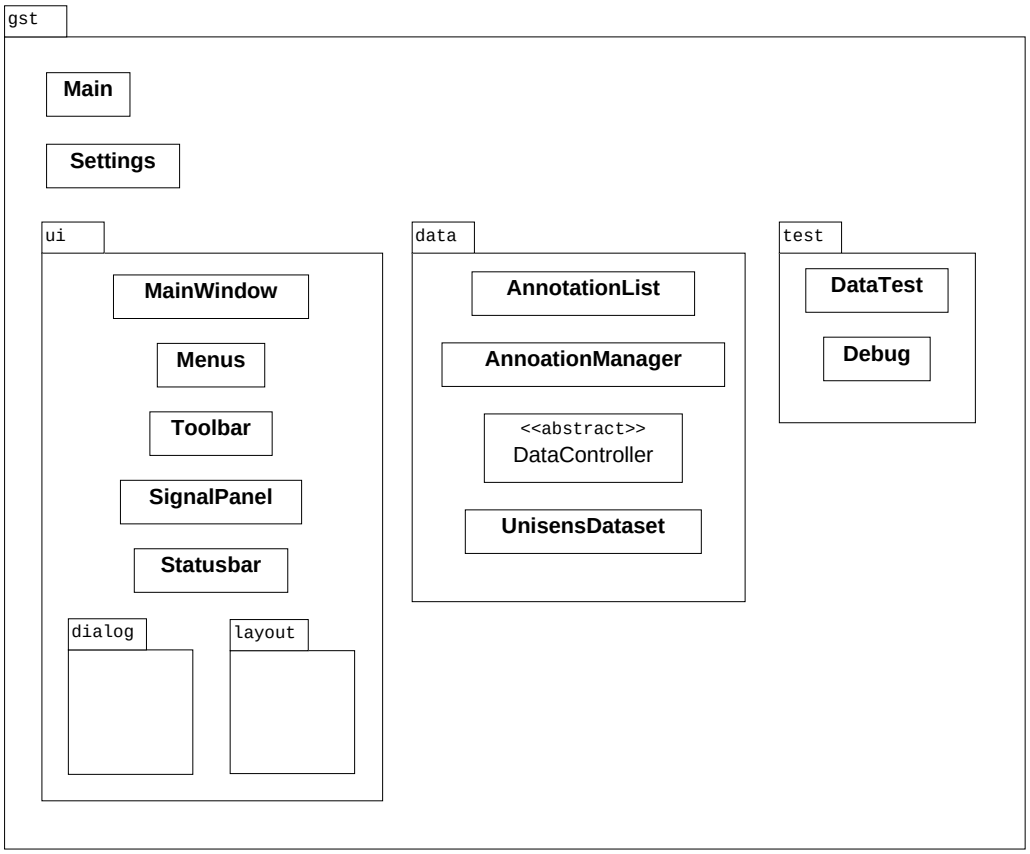

Abbildung 2.: UML-Paket-Übersicht der umgesetzten Software, nur wesentliche Klassen sind dargestellt

len sowohl für Matlab als auch für Java und bietet für diese Arbeit optimale Anwendungsvoraussetzungen. Die Organisation über eine menschlich les- und editierbare Headerdatei ist gerade in der Entwicklung interessant. Es kann ein Datensatz einfach und ohne Bearbeitungstools verändert und an die Bedürfnisse angepasst werden. Einer der Kritikpunkte nach [15] ist, dass das Format aus mehreren Dateien besteht. Da die Behandlung von Dateien und Verzeichnissen mit der aktuellen Software kaum noch Unterschiede für den Anwender darstellt, ist nach Meinung des Autors dieser Punkt nicht von hoher Priorität. Es bietet sogar die Möglichkeit Daten direkt von Sensoren (bei bekannten technischen Parametern wie z.B. Abtastrate und -auflösung) direkt in einen Datensatz integriert werden können. Zusätzlich bietet das Format durch seine Definition eine gute Erweiterbarkeit. Es kann somit an die neue Gegebenheiten und Voraussetzungen angepasst und optimiert werden.

4.2.1. Unisens

Das vom Forschungszentrum Informatik und Institut für Technik der Informationsverarbeitung der Universität Karlsruhe entwickelte Datenformat Unisens dient der Speicherung und der Doku-

Tabelle 2.: Paket- und Klassenübersicht: Paketnamen sind in `True-Type` und Klassennamen in serifenloser Schriftart geschrieben

`gst`	Main		
	Settings		
`gst.`	`data`	AnnotationController	
		AnnotationList	
		AnnotationManager	
		DataController	
		SignalController	
		UnisensDataset	
		ValueController	
`gst.`	`test`	DataTest	
		Debug	
`gst.`	`ui`	AnnotationSelectionDialog	
		DataSelectionDialog	
		MainWindow	
		Menus	
		NamedMouseAdapter	
		Sidebar	
		SignalOverview	
		SignalPanel	
		SignalView	
		SignalViewFactory	
		StatusBar	
		Toolbar	
`gst.`	`ui.`	`dialog`	DatasetSelectionDialog
			EditEventDialog
			EnterFileNameDialog
`gst.`	`ui.`	`layout`	ComponentArrangement
			MultiSplit
			SignalPanelLayoutManager
			VerticalLayoutManager

mentation von Sensordaten. Unisens ist konzipiert, Daten verschiedener Sensoren innerhalb eines Datensatzes zu speichern. Ein Datensatz ist im Dateisystem durch ein eigenes Verzeichnis und eine Headerdatei `unisens.xml` hinterlegt. In der Headerdatei werden alle Informationen über die Bestandteile des Datensatzes, deren Formatierung und ihre semantischen Zusammenhänge gespeichert. Messwerte eines Sensors werden üblicherweise in einer Datendatei innerhalb des Datensatzverzeichnisses abgespeichert. Eine solche Datendatei wird als *Entry* in dem Datensatz bezeichnet. Alle Metainformationen zu den Sensordaten werden in der Headerdatei abgspeichert, so dass die Datendateien selbst immer nur die reinen Messdaten enthalten. Als mögliche Sensordaten werden sowohl kontinuierlich abgetastete Signale als auch ereignisorientierte Daten unterstützt. Unisens unterscheidet zwischen vier Arten von Daten:

Signale *(Signal)*

Signale sind äquidistant abgetastete, numerische Messdaten. Sie zeichnen sich durch eine beliebige aber konstante Abtastrate und Abtastauflösung aus. Zudem können Signale aus

mehreren Kanälen bestehen, die alle in ein und derselben Datei abgespeichert werden. Alle Kanäle desselben Signals haben auch dieselbe Abtastrate und -auflösung.

Ereignisse *(Event)*

Ereignisse sind diskrete Zeitpunkte die mit einer textlichen Beschreibung versehen sind. (z.B. Triggersignale) Sie zeichnen sich durch einen Zeitstempel und einer kurzen Beschreibung aus. Optional können noch Kommentare zu einem Ereignis hinzugefügt werden.

Einzelwerte *(Value)*

Einzelwerte sind eine Kombination der beiden oben genannten Datenarten. Sie beinhalten numerische Werte die zu bestimmten Zeitpunkten aufgenommen wurden. Mit Einzelwerten ist es möglich Daten zu speichern, die nicht in festen Zeitintervallen gemessen werden.

Proprietäre Daten *(Custom data)*

Mit dieser Art können anwendungsspezifisch Daten gespeichert werden, die durch die drei oben genannten Arten nicht erfasst werden können. So können beispielsweise schematische Darstellungen des Messaufbaus als Bilddateien oder Patienteakten in Form von Textdateien dem Datensatz hinzugefügt werden.

Eine detailiertere Beschreibung des Formates kann der offiziellen Dokumentation [11] entnommen werden.

4.2.1.1. Details der Referenzimplementierung

In diesem Abschnitt wird kurz auf einige Details der Umsetzung des Unisens-Formates eingegangen. Das Unisens-Paket ist in Java implementiert und wird unter der GNU Lesser General Public License *(LGPL)* zur Verfügung gestellt. Die bereit gestellte Bibliothek ist auf zwei Einzeldateien aufgeteilt: `org.unisens.jar` und `org.unisens.ri.jar`. Bei der ersten Datei handelt es sich um die Definiton des Unisensformates und seiner Bestandteile als Javalassenstruktur. Diese Definition erfolgt hauptsächlich als *Interface*klassen und legt die Schnittstellen zwischen den einzelnen Bestandteilen fest. Eine Übersicht der Klassenstruktur und der von außen ersichtlichen Attribute ist in Abb. 3 auf Seite 23 dargestellt. Die vom Unisensformat unterstützten Signalarten sind auch in der Klassenstruktur in Tab. 3 erkennbar.

Signale	`SignalEntry`
Ereignisse	`EventEntry`
Einzelwerte	`ValuesEntry`
Proprietäre Daten	`CustomEntry`

Tabelle 3.: Signalarten und ihre repräsentierenden Klassen

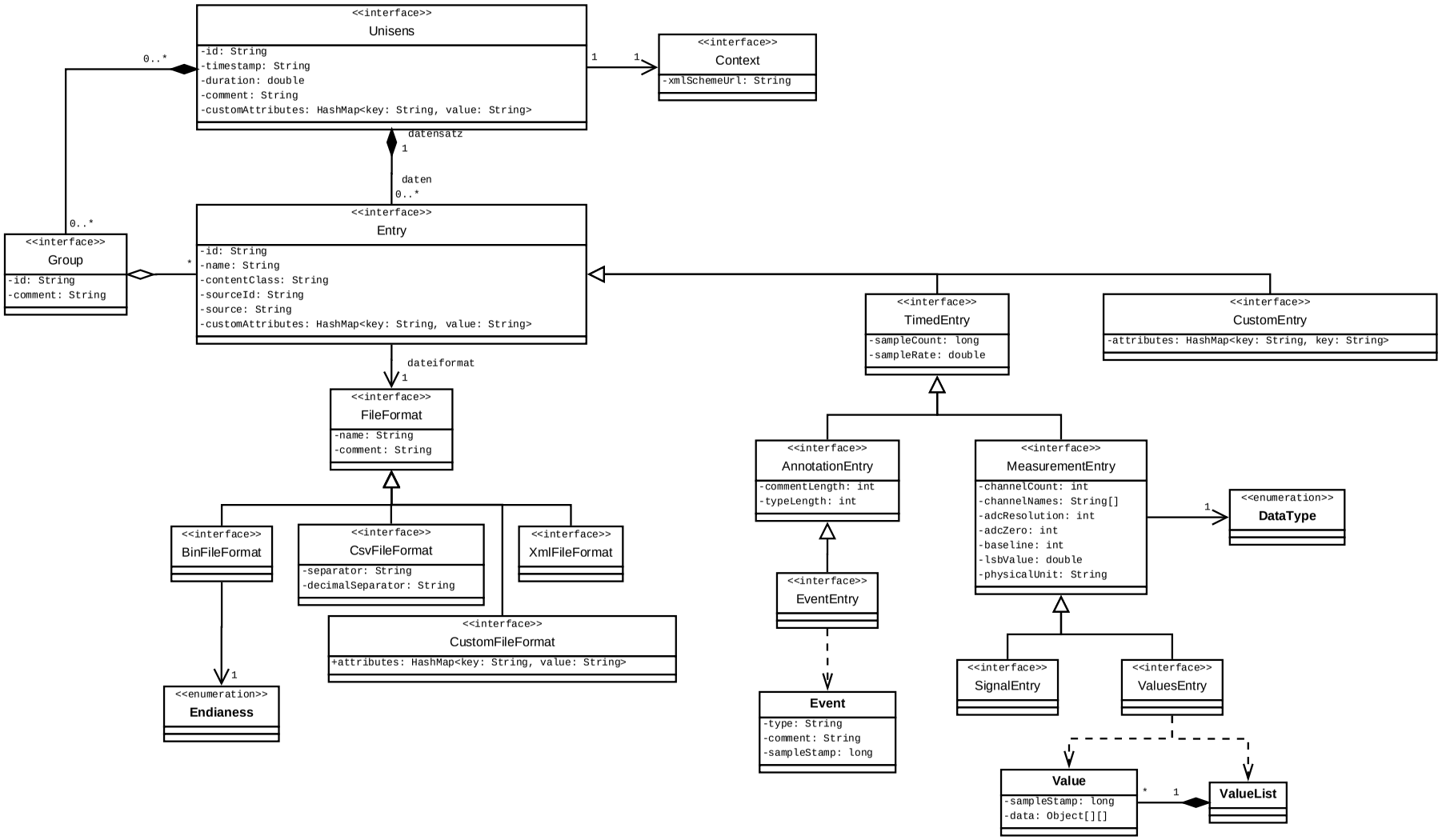

Abbildung 3.: Klassenübersicht der von Unisens definierten Schnittstellen

Aufgrund der Ableitung der Klassen `EventEntry` und `ValuesEntry` von `TimedEntry` ist ersichtlich, dass die Zeitpunkte von Ereignisdaten und Einzelwertdaten über eine virtuelle Abtastrate bestimmt werden. Der Zeitpunkt eines jeden *Event*- oder *Value*-Eintrags ist als ganzzahlige Samplenummer dieser Abtastrate gespeichert. Die Zeit eines Ereignisses, relativ zum Messbeginn, errechnet sich somit $Zeitpunkt = \frac{Samplenummer}{Abtastrate}$. Möchte man die Möglichkeit Ereignisse für jeden beliebigen Datenpunkt eines Datensatzes zuordnen zu können, dann muss die virtuelle Abtastrate als das kleinste gemeinsame Vielfache aller vorhandenen Abtastraten gewählt werden.

Die Schnittstellendefinition des Unisensformats stellt nur Methoden zum Lesen und Anhängen von Datenpunkten an den Datensatz bereit. Somit wird ein Einfügen, Löschen oder Verändern von Datenpunkten innerhalb eines Dateneintrags nicht unterstützt. Sollen diese Funktionen vorhanden sein, so muss diese Funktionalität selbst implementiert werden.

Die eigentliche Umsetzung der Funktionalität ist in der zweiten Datei abgespeichert. Im Folgenden soll sich der Begriff Referenzimplementierung auf diese funktionelle Umsetzung beziehen. Die Klassen der Referenzimplementierung bestehen aus den Klassennamen der Schnittstellendefinition und dem Suffix „Impl“ (z.B. Objekte die den Datensatz darstellen haben die Klasse `UnisensImpl`). Wenn man schon vorhandene Unisensdatensätze benutzen möchte reicht es aus, die Schnittstellendefinition zu kennen und zu nutzen. Sollen hingegen konkret Objekte erstellt werden, muss auf die Referenzimplementierung zurückgegriffen werden.

Durch einen Fehler in der Referenzimplementierung kann es vorkommen, dass beim Laden eines vorhandenen Unisensdatensatzes in dem Gruppen definiert sind eine `NullPointerException` auftritt. Insbesondere tritt dieser Fehler auf, wenn innerhalb der Headerdatei der Gruppeneintrag nicht hinter den Dateneinträgen steht.

4.2.2. Programminterne Datenstruktur

Das Paket `gst.data` ist die Schnittstelle der Programms zu dem gewählten Datensatzformat Unisens. Eine Übersicht des Paketes ist im UML-Diagramm in Abb. 4 dargestellt. Dieses Paket erfüllt zwei wesentliche Aufgaben:

- Vereinheitlichung der Behandlung der unterschiedlichen Datenarten
- Abkapselung anderer Programmteile vom Datensatzformat

Die Vereinheitlichung ist notwendig, da Unisens die Daten in drei Arten gliedert: Signale, Werte und Ereignisse. Für die Bearbeitung und Darstellung ist aber die Unterscheidung von Signalen und Werten nicht notwendig. In beiden Fällen handelt es sich um numerische Werte die zu einem bestimmten Zeitpunkt aufgenommen wurden. Somit sollen diese auch vom Programm gleichartig

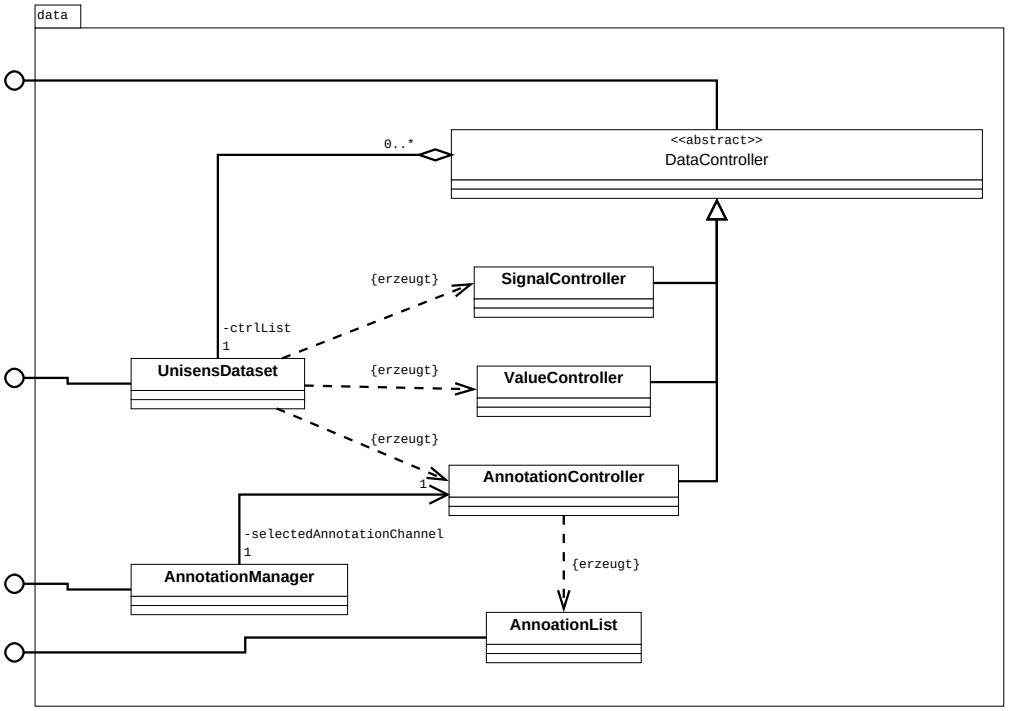

Abbildung 4.: UML-Diagramm des `data`-Paketes inklusive der bereitgestellten Schnittstellen

behandelt werden. Daher erhalten alle Datenarten die abstrakte Klasse `DataController` als Fassade. Die jeweilge konkrete Implementierung ist in den drei abgeleiteten Klassen `Signal-`, `Value-` und `AnnotationController` realisiert. Neben der Vereinheitlichung der Schnittstelle zum Datensatz werden auch die Zugriffe auf die Daten vereinfacht. Es werden notwendige Ausnahmebehandlungen (*Exceptions*) durch die Klassen des `data`-Paketes abgefangen und behandelt. Darunter fällt unter anderem die Behandlung von Fehlern, wenn Daten aufgrund von fehlenden Zugriffsrechten nicht gelesen oder geschrieben werden können. Somit ist der Zugriff auf die konkreten Daten des Datensatzes sowohl vereinheitlicht als auch vereinfacht.

Weiterhin wird mit dem `data`-Paket erreicht, dass die restlichen Komponenten des Programms nicht direkt auf das Datensatzformat zugreifen. Sie sind damit von dem gew"hlten Format abgekapselt und ein Austausch des Formates erfordert nur die Anpassung des `data`-Paketes. Der gesamte Datensatz ist durch die Wrapper-Klasse `UnisensDataset` im Programm repräsentiert. Zugriff auf den Datensatz erfolgt nur über die von der Klasse bereitgestellte Schnittstelle. Weiterhin wird der veränderde Zugriff auf Annotationen durch die Klasse `AnnotationManager` realisiert und die Behandlung von Annotationen selbst über die Klasse `AnnotationList` abgewickelt. Die zwei zu letzt genannten Klassen kapseln die von Unisens bereitgestellten Klassen (`Event` und `EventList`) von den anderen Programmteilen ab.

Neben den zwei oben genannten Aspekten wird auch der Zugriff auf die Daten selbst durch das Paket verändert. Unisens arbeitet durchgehend mit Samplenummern bei der Indizierung der einzelnen Datenpunkte. Die durch das Paket bereitgestellte Schnittstelle wandelt diese Art des Zugriffs auf eine zeitbasierte Art um. Das bedeutet, anstatt Daten über die Angabe von Samplenummern zu erhalten, bietet die Schnittstelle die Möglichkeit Daten aus einem bestimmten Zeitraum zu erhalten. Die notwendige Umrechnung zwischen Samplenummern und Abtastraten auf entsprechende Zeitpunkte wird im `data`-Paket ausgeführt. Dadurch wird die Existenz verschiedener Abtastraten der unterschiedlichen Signale vor dem restlichen Programm verborgen.

4.2.2.1. Repräsentation des Datensatzes durch die Klassen `UnisensDataset` und `DataController`

Die Klasse `UnsiensDataset` stellt die Schnittstelle des Programms zu den Datensätzen dar. Für jeden geladenen Datensatz wird im Programm ein Objekt der Klasse `UnisensDataset` erzeugt. Diese Objekte bieteten die erforderlichen Funktionen um auf die Eigenschaften des Datensatzes, die Eigenschaften der Dateneinträge und die Daten der Eintr"ge selbst zuzugreifen. Beim Laden eines Datensatzes wird für jeden Dateneintrag ein Instanz eines `DataController`s erzeugt und in einer Liste `ctrlList` im `UnisensDataset`-Objekt gespeichert. Je nach Datenart des Eintrags wird zur Laufzeit dynamisch ein Objekt der Klassen `Signal-`, `Value-` oder `AnnotationController` erzeugt. Genauer wird sogar für jeden Kanal ein eigener `DataController` angelegt. Jede Komponente des Programms die auf einen Dateneintrag zugreifen möchte, erhält vom `UnisensDataset` den entsprechenden Controller. Es wird sichergestellt, dass zu jedem Zeitpunkt jedem Dateneintrag genau ein Controllerobjekt zugeordnet ist und keine Duplizierung auftreten kann. Durch spezielle Funktionen `AddNewSignal()`, `AddNewValue()` und `AddNewAnnotation()` können einem Datensatz zus"tzliche Dateneintr"ge hinzugefügt werden. Die entsprechenden `DataController` werden dann in die Liste `ctrlList` aufgenommen und es kann durch diese anschließend auf die Daten zugegriffen werden. Der gesamte Datensatz wird bei einem Aufruf der `save()`-Funktion des Datensatzes abgespeichert werden, weil die Speicherfunktion für alle `DataController`-Objekte in `crtlList` rekursiv ausgeführt werden können.

4.2.2.2. Pufferung, Sortieren und Suchen der Klasse `AnnotationController`

Die Referenzimplementierung des Unisensformates sieht für das Lesen und Schreiben von Daten einen ungepufferten Ansatz vor. Das bedeutet sobald Daten vom Programm verändert werden, werden diese Änderungen auch sofort im Datensatz und somit auf dem Speichermedium abgespeichert. Das hat zur Folge, dass Annotationen in der Reihenfolge in der sie erstellt wurden, in den

Datendateien abgespeichert werden. Somit sind die Annotationen zeitlich unsortiert. Wenn nun eine Annotation eines bestimmten Zeitpunktes angezeigt werden soll, muss der im schlechtesten Fall der gesamte Datensatz nach dieser Annotation durchsucht werden. Dieses Verhalten ist unerwünscht, da somit die Zugriffszeit während der Laufzeit, insbesondere bei großen Datensätzen, stark schwanken können. Dieses Problem wird durch eine gepufferte Implementierung der Klasse `AnnotationController` behoben. Die Annoationen werden beim Laden des Datensatzes in den Arbeitspeicher geladen und mit einem Quick-Sort-Algorithmus nach aufsteigenden Samplenummern sortiert. Neu erstellete Annotationen werden in dieser Liste an den entsprechenden Stellen einsortiert. Es wird immer die Samplenummer der Annotation gespeichert, auf die als letztes zugegriffen wurde (sowohl Lesen als auch Schreiben). Bei der Suche nach der entsprechenden Position einer gesuchten Annotation wird dann von dieser gespeicherten Samplenummer aus gestartet. Die Suche nach einer Samplenummer erfolgt sowohl vor- als auch rückwärts. Dieser Ansatz hat den Vorteil, dass eine gesuchte Position nach wenigen Suchschritten erreicht wird, da Annotationen durch den Benutzer nur verändert werden können die auch angezeigt werden. Somit befindet sich der Index des letzten Zugriffs immer in der Nähe der gesuchten Position.

4.2.2.3. Tests des `data`-Paketes

4.3. Benutzerführung

4.3.1. Grafische Buntzeroberfläche

4.3.2. Datenvisualisierung

5. Validierung

– NOTIZ: Speicherplatzbedarf bei speicherung in `int16` und `double` – Skripte zur Wandlung Matlab <–> Unisens

5.1. Erfüllung der Anforderungen

5.2. Evaluation der Nutzeroberfläche

5.3. Validierung anhand der Annotation von fetalen EKG-Daten

5.4. Validierung mittels der Annotationüberprüfung von ...

6. Diskussion

6.1. Bewertung der Evaluation

6.2. Ausblick

6.3. Grenzen

Literaturverzeichnis

[1] Andreotti, F. : *Extraction of the Fetal ECG from Electrocardiographic Long-term Recordings*, Technische Universität Dresden, Diplomarbeit, August 2011

[2] Brügge, B. ; Dutoit, A. H.: *Object-Oriented Software Engineering. Using UML, Patterns and Java.* 2. Pearson Education, Inc., 2004

[3] Brügge, B. ; Dutoit, A. H.: *Objektorientierte Softwaretechnik. mit UML, Entwurfsmustern und Java.* 2. Pearson Education, Inc., 2004. – Überstzung des englischen Originals [2]

[4] Chlebek, P. : *User Interface-orentierte Softwarearchitektur.* Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, 2006

[5] Cooper, A. ; Reimann, R. ; Cronin, D. : *About Face 3. The Essentials of Interaction Design.* Wiley Publishing, Inc., 2007

[6] Cooper, A. ; Reimann, R. ; Cronin, D. : *About Face. Interface und Interaction Design.* Hüthig Jehle Rehm GmbH, 2010. – Übersetzung der amerikanischen Originalausgabe [5]

[7] Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J. : *Design Patterns. Elements of Reuseable Object-Oriented Software.* Addison-Wesley Publishing Company, 1995

[8] Gilbert, D. : *The JFreeChart Class Library Developer Guide.* 1.0.14. Object Refinery Limited, Februar 2007. `http://www.scribd.com/doc/82678249/jfreechart-1-0-14-A4#outer_page_868`

[9] Jacobson, I. ; Booch, G. ; Rumbaugh, J. ; Shanklin, J. C. (Hrsg.): *The Unified Software Development Process.* Addison-Wesley Longman Inc., 1999

[10] Kirst, M. ; Ottenbacher, J. ; Nedkov, R. : UNISENS - Ein universelles Datenformat für Multisensordaten. In: *Workshop - Biosignalverarbeitung* Universität Potsdam, 2008, S. 106 – 108

[11] Ottenbacher, J. ; Kirst, M. : *Unisens Dokumentation*. 2.0. Forschungszentrum Informatik und Institut für Technik der Informationsverarbeitung der Universität Karlsruhe, Februar 2010. http://unisens.org/downloads/documentation/Unisens-Dokumentation.pdf

[12] Rupp, C. ; Queins, S. ; Zengler, B. : *UML 2 glasklar*. Carl Hanser Verlag, 2007

[13] Santiago, M. C.: *Processing of abdominal recordings by Kalman filters*, Technische Universität Dresden, Diplomarbeit, April 2012

[14] Schäfer, S. : *Objektorientierte Entwurfsmethoden. Verfahren zum objektorientierten Softwareentwurf im Überblick*. 1. Addison-Wesley Publishing Company, 1994

[15] Schlögl, A. : An Overview on data formats for biomedical signals. In: Dössel, O. (Hrsg.) ; Schlegel, W. (Hrsg.): *IFMBE Proceedings 25/IV*, 2009, S. 1557–1560

[16] Sommerville, I. : *Software Engineering*. 6. Pearson Education, Inc., 2001. – Übersetzung der englischen Originalausgabe [17]

[17] Sommerville, I. : *Software Engineering*. 6. Pearson Education, Inc., 2001

[18] Starke, G. : *Effektive Software-Architekturen*. Carl Hanser Verlag, 2002

[19] Varri, A. ; Kemp, B. ; Penzel, T. ; Schlogl, A. : Standards for biomedical signal databases. In: *Engineering in Medicine and Biology Magazine, IEEE* 20 (2001), Mai/Juni, Nr. 3, S. 33 –37. http://dx.doi.org/10.1109/51.932722. – DOI 10.1109/51.932722. – ISSN 0739–5175

[20] Wang, Y. ; Liu, Y. ; Lu, X. ; An, J. ; Duan, H. : A general-purpose representation of biosignal data based on MFER and CDA. In: *Biomedical Engineering and Informatics (BMEI), 2010 3rd International Conference on* Bd. 2, 2010, S. 689 –693

[21] Zaunseder, S. ; Andreotti, F. ; Cruz, M. ; Stepan, H. ; Schmieder, C. ; Malberg, H. ; Jank, A. : Fetal QRS detection by means of Kalman filtering and using the Event Synchronous Canceller. In: *7th International Workshop on Biosignal Interpretation*. Como, Italy, Juli 2012

A. UML Dokumentation

B. Daten CD

Inhalt

./Diplomarbeit elektronische Form dieser Diplomarbeit

./Diplomarbeit/src LaTeX-Quelltext dieser Diplomarbeit

./Programm Quellcode des in dieser Arbeit umgesetzten Programms

./Literatur gesammelte Literatur