HeartRate2Go

Matthias Böffel Matrikel Nr.: 864483
Patrick Mathias Matrikel Nr.: 864089
Markus Nebel Matrikel Nr.: 864681
Janina Sauer Matrikel Nr.: 865235

2. Januar 2015

Hochschule Kaiserslautern
University of Applied Sciences

Betreuer: Prof. Dr.-Ing. Jan Conrad

Inhaltsverzeichnis

1 Einleitung

1.1 Vorstellung des Projektes

Im Projekt *HeartRate2Go* geht es darum, mit einer App für eine Android Uhr die Herzfrequenz per Pulsoxymetrie zu messen und diese Messwerte in einer passenden GUI darzustellen.
Somit soll der Anwender bei der Kontrolle seines Pulses unterstützt werden und ihm einen guten, verständlichen Überblick bieten. Dies gilt sowohl für eine Ruhemessung, als auch für eine Messung während einer Aktivität.

Ablauf

Für die Nutzung für *HeartRate2Go* sind drei Komponenten nötig:

- Android-Uhr
- Android-Smartphone und
- Computer

Der Nutzer trägt die Android-Uhr am Handgelenk und startet auf dieser die *HeartRate2Go-App*. Hier wird nun abgefragt, ob er eine Ruhemessung oder eine Aktivitätsmessung durchführen möchte. Anschließend wird die ausgewählte Messung durchgeführt. Bei einer Ruhemessung wird die Messung automatisch beendet, bei der Aktivitätsmessung muss der Benutzer die Messung manuell beenden. Nun wird der Anwender nach seiner Stimmung während der Pulsmessung gefragt, hier hat er die Auswahl zwischen gut, ok und schlecht. Außerdem wird der Durchschnittswert der soeben durchgeführten Messung gezeigt.
Die Übertragung der Messwerte von der Android-Uhr an das Android-Smartphone verläuft automatisch per Bluetooth. Auf der Smartphone-App werden die Messungen in einem Balkendiagramm angezeigt und bieten so einen ersten Überblick.
Nachdem das *HeartRate2Go-Programm* auf einem Computer gestartet wurde, kann der Nutzer über die Smartphone-App die Übermittlung der Daten zu der GUI starten. Dort werden wird die neuste Messung und auch vergangene Messungen tabellarisch und grafisch dargestellt und zwischen den zwei Messtypen unterschieden. Des Weiteren ist auch ein Ausdrucken der Messwerte möglich.

TODO: Ablaufdiagramm von Bö einfügen

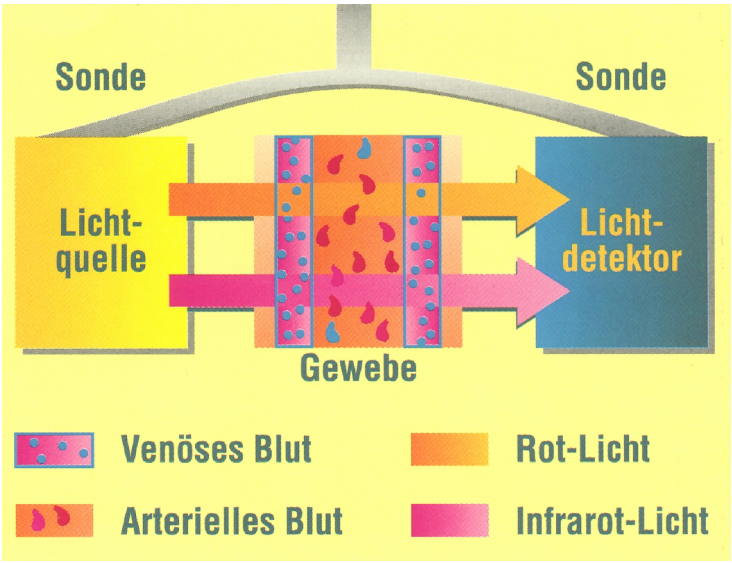
1.2 Medizinische Apps

Im Laufe der letzten Jahre wurde der Markt mit Apps, die einen medizinischen Hintergrund besitzen, überschüttet. Wenn man im deutschen iTunes-Store nach „Medizin“ sucht, erhält man mehrere hundert Einträge, dies gilt genauso für den Google-Play Store.
Im vergangen Jahr sind die Absatzzahlen von medizinischen Apps in Großbritannien, Frankreich, Niederlande und Deutschland um 42 Prozent gestiegen,

vermeldet das GfK (Marktforschungszentrum).
Diese Apps decken nahezu jeden Bereich der Medizin ab, egal ob es um die Speicherung von Vitaldaten, die Messung von Vitaldaten mit einem zusätzlichen Messgerät und die Auswertung der Daten geht. Des Weiteren sind auch viele Nachschlagewerke darunter enthalten.
Zu beachten ist allerdings, dass keiner der Apps den Arztbesuch ersetzt. Sie geben lediglich eine erste Einschätzung und sind dadurch eine große Erleichterung für den Nutzer. Allerdings ist es auch so, dass jeder Programmierer eine App mit medizinischem Hintergrund in die verschiedenen Stores hochladen darf. Diese werde nicht auf ihren Nutzen hin überprüft, so sind auch viele Apps zu finden, die mehr als Spielerei gelten.
Kaum eine App ist ein Medizinprodukt nach dem Medizinproduktegesetzt, sie gelten lediglich als Wellness- beziehungsweise Lifestyle-Apps.

1.3 Medizinische Kenntnisse - Pulsoxymetrie

Für die Messung des peripheren Pulses per Android-Uhr wird das Prinzip der Reflexions-Pulsoxymetrie genutzt.
Dieses Verfahren benötigt zwei Sensoren: zum einen eine Lichtquelle, zum anderen ein Lichtsensor. Die Lichtquelle sendet Infrarot-Lichtwellen aus, die durch die Haut dringen. Der Sensor misst die Lichtanteile, die absorbiert wurden.
Die Lichtabsorption im Blut ist abhänig von der Hämoglobinkonzentration und der Sättigung des Hämoglobins mit Sauerstoff. Oxigeniertes und desoxigeniertes Hämoglobin schwächen das Licht jeweils charakteristisch ab.
Mit diesem Prinzip ist es auch möglich, die Sauerstoffsättigung im kapillären Blut gemessen werden.



Quelle: www.edoc.hu-berlin.de

TODO: bitte Korrektur lesen Quelle einfügen Quelle: Behandlungsassistenz „... in der Arztpraxis“ von Dr. Uta Groger, Cornelsen Verlag, 1. Auflage, 2007

2 Hauptteil - Android

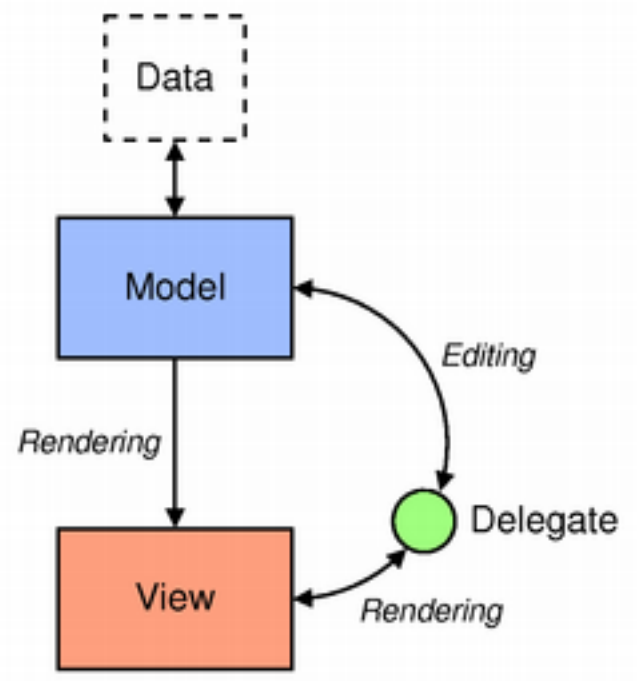
2.1 Hauptteil - Part1

Text

3 Qt-Framework Anwendung

3.1 Model-View Konzept

In der Praxis werden viele User Interfaces mit dem MVC Pattern realisiert. Dieses Pattern besteht aus der View, dem Controller und dem Model. In diesem Zusammenhang wird eine strikte Trennung der einzelnen Schichten angestrebt. Das Ziel dieses Ansatzes, liegt in dem Austausch der View, ohne eine Anpassung der internen Datenstruktur durchführen zu müssen. Eine weitere Variante dieses Konzeptes ist das Model/View Konzept. Hierbei wird die View mit dem Controller kombiniert. Die Aufgabe der View es ist mit Hilfe des Models dem Benutzer die Informationen anzuzeigen. Der Controller reagiert lediglich auf Interaktion des Benutzers mit der View. Zusätzlich kann bei einer Model/View Architektur das Konzept eines „delegates“ eingeführt werden. Dieser besitzt die Aufgabe, die einzelnen Datenelemente des Models benutzerspezifisch anzuzeigen oder auf bestimmte Veränderungen des Datenbestandes von Seiten des Benutzers auf der View zu reagieren. Das Zusammenspiel der einzelnen Komponenten wird in Abbildung 1 nochmals graphisch veranschaulicht.



In Abbildung 1 wird deutlich, dass eine Trennung zwischen der Speicherung und der Darstellung der Daten besteht. Die Aufgabe des Models besteht darin, der View und dem Delegate minimalistische Schnittstellen für die Kommunikation bereitzustellen. Die Kommunikation zwischen den einzelnen Komponenten wird in QT mit Hilfe des Signal und Slot Konzeptes realisiert. Findet eine Änderung am Datenbestand des Models statt, wird ein Signal an die View und den entsprechenden Delegate geschickt. Diese rufen die entsprechenden Slots auf und aktualisieren die View. Im umgekehrten Fall, wenn der Benutzer die Daten via View verändert, schickt diese ein Signal an das Model und den Delegate. Ein weiteres Hilfsmittel der View ist der Modelindex. Dieser Index wird

verwendet, um die einzelnen Informationen aus dem Datenbestand zu lesen.

Ein Modelindex ist eine Referenz auf einen einzelnen Datensatz des Models. Durch die Zuhilfenahme eines Delegates, kann dieser benutzerspezifisch auf der View dargestellt werden. Diesbezüglich muss erwähnt werden, dass es mehrere Möglichkeiten für die Erstellung der View unter QT existieren. Eine Möglichkeit ist die Erstellung mittels QT Widgets. Diese können durch Qt bereitgestellte Klassen erzeugt werden. Hierbei ist eine eindeutige Trennung zwischen View und Model schwer möglich. Eine andere Variante ist die Erstellung mittels QML. QML ist eine Art Beschreibungssprache für Benutzeroberflächen. Hierbei muss lediglich eine Beschreibung der GUI angegeben werden. Diese ist so allgemein gehalten, dass Designer oder UI-Entwickler ohne irgendwelche Vorkenntnisse einer Programmiersprache eine View anfertigen können. Der Vorteil hiervon liegt in der klaren Trennung der einzelnen Aufgaben. Ein Designer kann sich ausschließlich um das UI kümmern und ein Software-Entwickler um das dazugehörige Model und den Controller. Diesbezüglich existiert eine klare Trennung zwischen dem Model und der View.

In Qt sind alle Model Klassen von der Abstrakten Basisklasse QAbstractItemModel abgeleitet. Diese Basisklasse bietet eine Vielzahl an Schnittstellen für die Kommunikationen mit der View an. Um auf eine gegeben Datenstruktur besser reagieren zu können, bietet QT 3 besondere Model-Typen an. Diese Modeltypen sind: QListModel, QTableModel und QTreeModel. Mit Hilfe dieser 3 Modeltypen, kann eine Vielzahl der Anwendungsbereiche abgedeckt werden. In diesem Projekt ist ausschließlich ein QAbstractListModel verwendet worden. Zur Umsetzung des Model View/Konzeptes wurde für die Bereitstellung der UI die Beschreibungssprache QML verwendet. Für die Implementierung des Models wurde die bereits von Qt bereitgestellte Klasse QAbstractListModel verwendet. Hierbei ist eine neue Unterklasse von QAbstractListModel erzeugt und die entsprechenden Methoden für die Kommunikation mit der View neu implementiert worden. Als Datenstruktur wurde eine QList mit entsprechenden Datenobjekten gewählt. Die Datenobjekte besitzen die Aufgabe, die einzelnen Messwerte einer Messung zu kapseln. Eine Vielzahl der QML Elemente wie beispielsweise eine ListView oder TableView bieten standartmäßig eine Property „model“ an, welche die Verknüpfung mit dem entsprechenden Model realisiert. Des Weiteren kann mittels QML ein delegate Objekt erzeugt werden. Mit dessen Hilfe, können beispielsweise die Einträge in einer ListView bestmöglich auf die Wünsche des End-Benutzers angepasst werden.

Der Vorteil des QT Frameworks ist die automatische Anpassung der GUI an eine Änderung des internen Datenbestandes. Der Entwickler muss lediglich dem Model mitteilen, wann eine Änderung am Datenbestand des Models durchgeführt wurde. Infolgedessen übernimmt das Framework die komplette Aktualisierung der GUI. Der umgekehrte Fall, dass der Benutzer die Daten mittels GUI verändern kann, ist im Projektverlauf nicht implementiert worden.

Für die bestmögliche Darstellung der gesammelten Daten, mussten mehrere Diagramme erstellt werden. Hierbei bestand die Möglichkeit, alle Diagram-

me über die von QT bereitgestellten Klassen zu erzeugen, oder ein bereits vorhandenes auf QT basierendes Modul namens QCustomPlot zu verwenden. Dieses Modul kapselt die von QT bereitgestellten „Paint“ Klassen und gibt dem Benutzer eine Vielzahl an bereits vorimplementierten Diagramm-Typen. Ursprünglich wurde diese Third-Party Modul für den Einsatz mit QT Widgets konzipiert. Diesbezüglich musste eine Portierung in QML durchgeführt werden. Folglich konnten einige Funktionalitäten wie beispielsweise das Zoomen nicht in QML überführt werden. Die Portierung ist mittels der QCustomPlot Support Seite durchgeführt worden.

TODO: Quellen einfügen Quellen:
http://qt-project.org/doc/qt-4.8/modelview.html
http://qt-project.org/doc/qt-4.8/model-view-programming.html
http://sysmagazine.com/posts/181712/
http://doc.qt.io/qt-5/qtqml-cppintegration-interactqmlfromcpp.html
http://qt-project.org/doc/qt-4.8/qmlevents.html
http://qt-project.org/doc/qt-4.8/qtbinding.htmlexchanging-data-between-qml-and-c
http://www.qcustomplot.com/index.php/support/forum/172
http://www.qcustomplot.com/index.php/tutorials/settingup
The Book of QT 4.0 Daniel Molkentin

3.2 Umsetzung mittels QML

3.3 Kommunikation

TODO Themen:

- Kommunikation über TCP Verbindung [Doc14b]
- Server-Discovery per Broadcast [Doc14c]
- DataReceiver

3.4 Datenverwaltung

TODO Themen:

- SQLite
- QSqlDatabase [Doc14a]
- Datenbank Struktur
-

4 Fazit

Text

4.1 Retrospektive

text

4.2 Ausblick

Wie schon im Konzeptpapier erwähnt, besteht die Möglichkeit, das Projekt *HeartRate2Go* zu publizieren und es so anderen Anwendern zugänglich zu machen. Hierzu könnte es durch weitere Funktionen erweitert werden. Einige dieser Anwendungen finden sich schon im Konzeptpapier unter 2.b. Optionale Funktionen, zum Beispiel: das Anlegen von Benutzerprofilen.
Dies geschieht derzeit nur ansatzweise, die gesendeten Werte werden für jedes Benutzerprofil des Betriebssystems separat abgespeichert. Jedoch ist eine Anamneseabfrage noch nicht möglich. In dieser würde nach Alter, Geschlecht, Größe, Gewicht, maximaler und minimaler Pulswert für die beiden Messwerttypen gefragt werden. So wäre auch eine erste Einschätzung der gemessenen Werte möglich.
Ein anderer Punkt, ist die Berechnung des Kalorienverbrauchs. Zwar wird während einer Aktivitätsmessung die Anzahl der Schritte angezeigt, jedoch war es leider in der vorgegeben Zeit nicht möglich, der dadurch resultierende Kalorienverbrauch zu errechnen. Hierfür ist auch die Schrittlänge nötig, die mit einem Benutzerprofil einhergeht.
Des Weiteren stand zur Diskussion, ob dem Nutzer die Möglichkeit gegeben werden soll, Marker zusetzen, die eine besondere Situation kennzeichnen und in der späteren Ansicht speziell angezeigt werden. Dies ist bei einer, vom Hausarzt angeordneten, Langzeit-EKG-Messung ein wichtiger Teil, auch für die spätere Bewertung der Messung.
Da das *HeartRate2Go-Programm* auf allen Betriebssysteme läuft, wäre eine App für das iOS-Betriebssystem auch praktisch. Der Zeit existiert die *HeartRate2Go-App* nur für Android. Die Erstellung einer iOS-App war jedoch leider nicht möglich, da hierfür keine passende Apple-Komponenten zur Verfügung standen.
Die Umsetzung der genannten Punkte scheiterte an der begrenzten Zeit, die für dieses Projekt zur Verfügung stand.

Literatur

[Doc14a] Qt Documentation. Qsqldatabase class. http://doc.qt.io, 2014.

[Doc14b] Qt Documentation. Qtcpserver class. http://doc.qt.io, 2014.

[Doc14c] Qt Documentation. Qudpsocket class. http://doc.qt.io, 2014.