



Eingebettete Systeme

Entwicklung der Software für ein eingebettetes System zur kabellosen Erfassung von medizinischen Sensordaten

Zur Erlangung des akademischen Grades eines
Master of Science
- Informatik -

Fachbereich Informatik
Erstprüfer: Prof. Dr. Thomas Breuer
Zweitprüfer: Prof. Dr. Thomas Berlage
Unternehmen: Fraunhofer FIT

eingereicht von:
Daniel Pyka
Matr.-Nr. 9016660
Grantham-Allee 20
53757 Sankt Augustin

Sankt Augustin, den 31. Januar 2017

Eidesstattliche Erklärung

Ich versichere an Eides statt, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

*Sankt Augustin,
den 31. Januar 2017*

Ort, Datum

Daniel Pyka

Inhaltsverzeichnis	I
Inhaltsverzeichnis	I
Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Quelltextverzeichnis	V
Abkürzungsverzeichnis	VI
Zusammenfassung	1
1. Einleitung	2
1.1. Motivation	3
1.2. Ziele	5
2. Methode	7
3. Grundlagen	10
3.1. Stand der Technik	10
3.1.1. Kategorien mobiler Gesundheitsüberwachungssysteme	10
3.1.2. Anforderungen an mobile Gesundheitsüberwachungssysteme	12
3.1.3. Kommerzielle Produkte	13
3.1.4. Bluetooth	15
3.1.5. Weitere Funkübertragungstechniken	23
3.1.6. Programmierung eingebetteter Systeme	25
3.2. Stand der Forschung	26
4. Anforderungsanalyse	33
4.1. Funktionalität	33
4.2. Usability	33
4.3. Datenschutz und Informationssicherheit	34
4.4. Energieeffizienz	34
4.5. Zuverlässigkeit	34
4.6. Medizinische Richtlinien	34
4.7. Softwarespezifische Anforderungen	35
4.8. Anwendungsfälle (Use Cases)	35
5. Konzept und Design	37
5.1. Auswahl der Hardware	37
5.1.1. Raspberry Pi 3 und Zubehör	38
5.1.2. Herzfrequenzsensoren	40
5.1.3. SensorTag CC2650	40
5.1.4. BLE113 Entwicklungsboard	41
5.2. Mensch-Maschine-Interaktion	42
5.2.1. Interaktion mit der Sensorplattform	42
5.2.2. Interaktion mit den Sensoren	43
5.2.3. Aufzeichnen von Sensordaten	44
5.2.4. Datenbank-Interaktion	44
6. Implementierung und Softwarearchitektur	45
6.1. Bluetooth-Schnittstellen	46

6.2. Softwarearchitektur der Sensoren	48
6.2.1. Gatttool	48
6.2.2. GattLibrary	49
6.2.3. Sensorklassen	49
6.2.4. SampleCollector	52
6.2.5. AbstractSensorWrapper	53
6.3. Webanwendung der Sensorplattform	56
6.4. Datenbank	57
6.5. HardwarePlatform	58
6.6. Schnittstelle zum Mobilfunkmodul	59
6.7. Anbindung von externen Webanwendungen	61
6.8. Controller und SensorOverseer	64
6.9. Konfiguration des Betriebssystems	67
6.9.1. Autostart mit systemd	67
6.9.2. Hardware-Uhr	69
6.9.3. Benutzerverwaltung	70
6.10. Sicherheit	70
6.10.1. Sicherheitsfunktionen der Webschnittstellen	71
6.10.2. Sicherheitsfunktionen von Bluetooth-Verbindungen	73
6.10.3. Sicherheitsaspekte des Betriebssystems	79
7. Tests	80
7.1. Modultests	80
7.1.1. Modultest des Gatttools	80
7.1.2. Modultest der Datenbank	81
7.1.3. Modultest der Sensoren Polar H7 und BLE113	81
7.1.4. Modultest des Controllers und der SensorWrapperFactory	82
7.2. Testfälle (Test Cases)	85
7.3. Leistungsaufnahme der Sensorplattform	88
7.3.1. Stromverbrauch	88
7.3.2. Prozessor- und Hauptspeicherauslastung	91
7.3.3. Signalqualität der Mobilfunkverbindung	93
7.3.4. Reichweite der Bluetooth-Geräte	97
8. Diskussion und Fazit	98
9. Ausblick	102
Literaturverzeichnis	103
A. Test Cases	110
B. Modultest Polar H7	123
C. Modultest BLE113	128

Abbildungsverzeichnis

1.	RR-Intervalle beim Elektrokardiogramm [Blu15]	2
2.	Beispiel eines mobilen Gesundheitsüberwachungssystems [MOJ06]	11
3.	Herzfrequenzsensor Polar H7 mit Brustgurt [Polb]	14
4.	Philips Blutdruckmessgerät für das Handgelenk (links) [Kon], Pulsoximeter Sensor Nonin 3230 (Mitte) [Nonb], Faros eMotion 360° Dreikanalelektrokardiogramm (rechts) [Meg]	15
5.	Der Bluetooth-Protokollstack [Blub]	17
6.	Hierarchie der GATT-Profile [Bluc]	18
7.	GATT-Service für die Herzfrequenz [Blud]	19
8.	Die UMTS-Systemarchitektur [Lü01]	24
9.	Relation zwischen RSCP und E_c/I_o [Qua06]	25
10.	Modell einer Middleware für ein Wireless Sensor Network (WSN) [KFR14] .	29
11.	Systemarchitektur von Monarca [PMGM11]	30
12.	Prototyp des Zigbee-Systems [MM12]	31
13.	Architektur des Escort-Systems [TLL ⁺ 11]	32
14.	Micro-USB-Multimeter zum Messen der Stromstärke und der Spannung [Tra]	39
15.	Sensorplattform mit Surfstick, Akku (Unterseite), Messgerät (rechts) und Stromkabel	40
16.	Texas Instruments SensorTag CC2650 [RS]	40
17.	BLE113 Entwicklungsboard [Dig]	41
18.	Softwareprojekte des Gesamtsystems der Sensorplattform und deren Abhängigkeiten	46
19.	Software-Stack der Sensorplattform	48
20.	Klassendiagramm der Sensoren	51
21.	UML Aktivitätsdiagramm des HeartRateSampleCollectors	53
22.	Sequenzdiagramm für die Verarbeitung von Herzfrequenz-Samples	55
23.	Webanwendung der Sensorplattform	57
24.	UML-Klassendiagramm der Schnittstelle zur Hardware	58
25.	Klassenbeziehung der Upload-Funktionalität	63
26.	Aktivitätsdiagramm der Sensorplattform	64
27.	Sequenzdiagramm für den Start einer Aufzeichnung	66
28.	DS3231 Hardware-Uhr an den I ² C GPIO-Pins des Raspberry Pi 3	70
29.	Informationen zur Sicherheit und zum TLS-Zertifikat	72
30.	HTTP-Basic-Authentication Abfrage des Browsers	73
31.	Aktivitätsdiagramm der sicheren BLE113 Firmware	78

Tabellenverzeichnis

1.	Überblick über die Bluetooth-Versionen [Jan16, Blu16a]	16
2.	Bluetooth-Sicherheitsstufen mit den daraus resultierenden Schutzfunktionen [Nat12]	21
3.	Softwarearchitekturen eingebetteter Systeme im Vergleich	26
4.	Auswahl geeigneter ARM-Boards für die Sensorplattform [Polc, Tre, Rasb, Vara, Varb]	38
5.	Sensoren und Messwerte des CC2650 [Tex16]	41
6.	Datenbankschemata für Herzfrequenz- und Pulsoximetriedaten	57
7.	Testbarkeit der Use Cases durch Test Cases	87
8.	Stromverbrauch der Sensorplattform im Bereitschaftsmodus	89
9.	Stromverbrauch der Sensorplattform während einer Aufzeichnung	90
10.	Prozessorauslastung während einer Aufzeichnung	92
11.	Untertakte der CPU des Raspberry Pi 3	93
12.	Signalqualität der Mobilfunkverbindung und Stromverbrauch der Sensorplattform	95
13.	Latenzen beim Übertragen von Herzfrequenzdaten an WebHrs	97

Quelltextverzeichnis

1.	Enumeration für Bluetooth-Adresstypen mit roter Markierung für die Sonderregel	48
2.	Guice-Konfiguration der Sensorplattform für eine konkrete Hardware	59
3.	Udev-Regel für den Surfstick E352S-5	59
4.	Konfigurationsdatei für das Programm wvdial	60
5.	Debuginformationen des Surfsticks E352S-5	61
6.	JSON-Format der REST-Schnittstelle von TeLiPro	62
7.	Konfiguration von Guice für unterschiedliche Uploader	63
8.	Systemd Startskript für die Anwendung der Sensorplattform	68
9.	Shellskript ausführbar machen und zu systemd hinzufügen	68
10.	Erstellung des TLS-Zertifikats durch keytool	71
11.	Deklaration des GATT-Servers eines Pulsoximeters	74
12.	Einstellungen des Security-Managers von Bluetooth in BGScript	74
13.	Gespeicherte Schlüssel der Sensorplattform durch Passkey-Entry-Pairing . .	75
14.	Initialisierung einer verschlüsselten Bluetooth-Verbindung	76
15.	Initialisierung einer unverschlüsselten Bluetooth-Verbindung	77
16.	Testklasse für das Aktivieren und Deaktivieren von Bluetooth-Notifications	80
17.	Testmethode für die Sensorwerte der Bewegungsmessung	82
18.	Initialisierung des Systems für den Controller-Test	83
19.	Testmethode für die Konfiguration durch Properties	84
20.	Testmethode für die Instanziierung des Sensors Polar H7	84
21.	Testmethode für das Starten und Beenden einer Aufzeichnung	85
22.	Modultest für den Sensor Polar H7 und die Klasse AbstractHeartRateSensor	123
23.	Modultest für den Sensor BLE113 und die Klasse AbstractPulseOximeter-Sensor	128

Abkürzungsverzeichnis

AAL	Ambient Assisted Living
AES	Advanced Encryption Standard
API	Application Programming Interface
APT	Advanced Packaging Tool
ASU	Arbitrary Strength Unit
ATT	Attribute Protocol
BAN	Body Area Network
BLE	Bluetooth Low Energy
BSI	Bundesamt für Sicherheit in der Informationstechnik
CBC	Cipher Block Chaining
CCM	Counter with CBC Mode
CDMA	Code Division Multiple Access
CPICH	Common Pilot Channel
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DNS	Domain Name System
DHCP	Dynamic Host Configuration Protocol
DHS	Department of Homeland Security
EDR	Enhanced Data Rate
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIPS	Federal Information Processing Standard
FIT	Fraunhofer-Institut für Angewandte Informationstechnik
GATT	Generic Attributes
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HBRS	Hochschule Bonn-Rhein-Sieg
HDMI	High Definition Media Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HRM	Heart Rate Monitor
HS	High Speed
HSPA+	Evolved High Speed Packet Access
HSQldb	Hyper Structured Query Language Database
IKT	Informations- und Kommunikationstechnologie
IoT	Internet-of-Things
IP	Internet Protocol
IPC	Inter-Process Communication
ISM	Industrial, Scientific and Medical
IRK	Identity Resolving Key
JDK	Java Development Kit
JNI	Java Native Interface
JSON	Javascript Object Notation
LE	Low Energy
LTE	Long Term Evolution
LTK	Long Term Key
LSO	Least Significant Octet
MITM	Man-In-The-Middle
MLC	Multi-Level Cell

MQTT	Message Queue Telemetry Transport
MSO	Most Significant Octet
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
OTG	On-The-Go
PCI	Peripheral Component Interconnect
PIN	Personal Identification Number
PPP	Point-to-Point Protocol
PS	Personal Server
RAM	Random Access Memory
REST	Representational State Transfer
RNDIS	Remote Network Device Interface Specification
RSSI	Received Signal Strength Indicator
RSCP	Received Signal Code Power
RTOS	Real-time Operating System
RTSP	Real Time Streaming Protocol
SATA	Serial AT Attachment
SBC	Single Board Computer
SIG	Special Interest Group
SLC	Single-Level Cell
SoC	System-on-Chip
SoM	System-on-Module
SSH	Secure Shell
SSP	Secure Simple Pairing
TCP	Transmission Control Protocol
TLS/SSL	Transport Layer Security/ Secure Sockets Layer
UMTS	Universal Mobile Telecommunications System
UTRAN	UMTS Terrestrial Radio Access Network
UUID	Universally Unique Identifier
USB	Universal Serial Bus
VmRSS	Virtual Memory Resident Set Size
WHMS	Wearable Health Monitoring System
WLAN	Wireless Local Area Network
WoT	Web-of-Things
WSN	Wireless Sensor Network
WWBAN	Wearable Wireless Body/Personal Area Network
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Zusammenfassung

Der demographische Wandel führt zu einer steigenden Lebenserwartung der Menschen, sodass der relative Anteil an älteren Leuten zunimmt [Bun06]. Auch das Gesundheitssystem wird durch diese Entwicklung vor neue Herausforderungen gestellt, weil die Kosten für die Behandlung vieler Menschen steigen. Mobile Gesundheitsüberwachungssysteme können eine Lösung sein, um Kosten im Gesundheitswesen zu senken und trotzdem eine hohe Qualität der gesundheitlichen Versorgung zu gewährleisten [MOJ06]. Ein solches System besteht aus kleinen am Körper getragenen Sensoren zur Messung von Vitalparametern, einem mobilen Gerät als Zwischenspeicher und einer Webinfrastruktur, zu der die Sensordaten übertragen werden. Durch die Verwendung von Algorithmen, bspw. für maschinelles Lernen, soll dabei die Diagnose und die Therapie von Krankheiten computergestützt und automatisiert möglich gemacht werden [Frab]. Für den Einsatz am Menschen umfassen derartige Geräte nicht nur Themenfelder der Informatik und Elektrotechnik, sondern stellen darüber hinaus eine Vielzahl überfachlicher Anforderungen.

Diese Arbeit behandelt die Softwareentwicklung für ein solches mobiles Gesundheitsüberwachungssystem, das im Folgenden *Sensorplattform* genannt wird. Die Software basiert auf offenen Schnittstellen, um mit Sensoren zu kommunizieren und diese einheitlich anzusteuern. Die Sensorplattform unterstützt zunächst fünf Sensoren des Funkübertragungsstandards Bluetooth Low Energy (BLE), wobei die Implementierung der Anwendung die freie Konfiguration und Erweiterbarkeit der Sensorplattform gestattet, sodass diese für vielfältige Krankheitsbilder mit unterschiedlichen Sensoren verbunden werden kann. Im Rahmen dieser Arbeit wird ein Demonstrator konzipiert, der aus kommerziell verfügbaren Hardwarekomponenten besteht. Das Gerät basiert auf einem Single Board Computer (SBC) mit einer Linux-Distribution als Betriebssystem, das die Basis für die in der Programmiersprache Java geschriebene Anwendung und den Bluetooth-Stack BlueZ 5 bildet. Die Mensch-Maschine-Interaktion zwischen Anwender und Sensorplattform erfolgt über eine Webanwendung, durch die auch der Zugriff auf die Datenbank möglich ist. Die Informationssicherheit an den Funk- und Netzwerkschnittstellen der Sensorplattform wird in dieser Arbeit ebenfalls behandelt: am Beispiel einer individuellen Sensorfirmware wird Verschlüsselung und Authentifizierung für eine Bluetooth-Verbindung eingesetzt. Außerdem verwendet die Sensorplattform zwei mögliche externe Webserver zum Hochladen von Sensordaten, von denen einer speziell für den Einsatz mit der Sensorplattform konzipiert ist. Diese Masterarbeit umfasst damit alle drei Teilbereiche eines mobilen Gesundheitsüberwachungssystems.

Mit dem Demonstrator der Sensorplattform werden auch verschiedene Tests durchgeführt. Zum einen wird damit sichergestellt, dass die Software die geforderte Funktionalität aus der Anforderungsanalyse besitzt, zum anderen ergeben sich dadurch verschiedene Parameter für die Beurteilung der Performanz des Systems. Unter anderem wird die Leistungsaufnahme und Hardwareauslastung der Sensorplattform gemessen, denn bspw. die Energieeffizienz der Hardwarekomponenten ist für ein mobiles System von signifikanter Bedeutung. Abschließend werden Verbesserungsmöglichkeiten sowohl für die Software, als auch für die Hardware der Sensorplattform diskutiert und ein Ausblick für die zukünftige Entwicklung von mobilen Gesundheitsüberwachungssystemen gegeben.

1. Einleitung

Im Bereich Life Science Informatik des Fraunhofer-Instituts für angewandte Informationstechnik FIT werden neue technische Ansätze entwickelt, um krankheitsspezifische und patientenindividuelle Informationen zu erheben und diese für Diagnose und Therapie von Krankheiten zu nutzen.

Ein Ansatz ist die längerfristige Erfassung (≤ 30 Tage) von medizinischen Sensordaten direkt am menschlichen Körper, bei dem die Sensoren und gegebenenfalls weitere Geräte so angebracht werden, dass der Alltag des Patienten nur geringfügig beeinflusst wird. Der Patient ist damit während der Diagnose nicht an einen Ort (z. B. ein Krankenhaus) gebunden, sondern kann weiterhin den meisten alltäglichen Aktivitäten nachgehen. Nebenbei werden bestimmte Vitalparameter (medizinische Messgrößen des menschlichen Körpers) wie bspw. Herzfrequenz, RR-Intervalle, Körpertemperatur, Bewegung und SpO₂ (Sauerstoffgehalt im Blut) aufgezeichnet. Abbildung 1 zeigt RR-Intervalle als Beispiel für einen Vitalparameter. Ein RR-Intervall ist die Zeit zwischen zwei aufeinander folgenden Herzschlägen, die von einigen Herzfrequenzsensoren gemessen werden können (siehe Kapitel 3.1.3).

Mithilfe solcher Daten und durch Verwendung von bspw. Algorithmen für maschinelles Lernen in der Cloud, sollen in Zukunft computergestützt Krankheiten diagnostiziert und therapiert werden [Frab]. Um dies zu realisieren, wird ein mobiles Gesundheitsüberwachungssystem benötigt, das den Datentransfer vom menschlichen Körper zu einer Webanwendung umsetzen kann. Das Ziel dieser Arbeit ist die Softwareentwicklung für ein solches System, das im Folgenden *Sensorplattform* genannt wird.

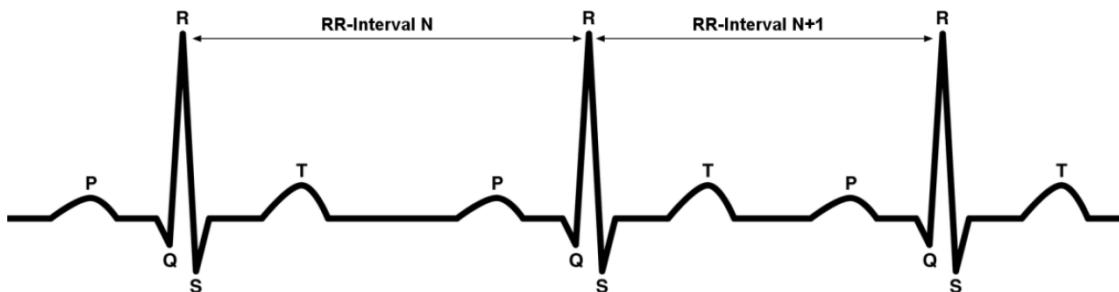


Abbildung 1: RR-Intervalle beim Elektrokardiogramm [Blu15]

Die allgemeine Vorgehensweise für die Masterarbeit ist in Kapitel 2 beschrieben. Anschließend werden die technischen und wissenschaftlichen Grundlagen in Kapitel 3 wiedergegeben. Dies gewährt zunächst einen Überblick über kommerzielle Systeme auf dem Markt. Die Produkte bieten auch bereits Beispiele für Funkübertragungstechniken, die die Kommunikation zwischen unterschiedlichen Geräten und Sensoren ermöglichen. Nachfolgend werden Informationen zu jüngsten Entwicklungen und Forschungsprojekten zusammengestellt. Dazu gehören unter anderem zwei Forschungsprojekte Polycare und MAS des Fraunhofer FIT, in denen ebenfalls mobile Gesundheitsüberwachungssysteme Anwendung finden. Die Anforderungsanalyse als Grundlage für das Konzept der Sensorplattform befindet sich in Kapitel 4 und die Hardware wird in Abschnitt 5 beschrieben und ausgewählt. Das Kapitel 6 umfasst die Implementierung der Anwendung der Sensorplattform und das Kapitel 7 beinhaltet die Tests der Implementierung. Die Bedeutungen der Ergebnisse aus den vorherigen Kapiteln werden danach in Abschnitt 8 diskutiert. Abschließend erfolgt in Kapitel 9 der Ausblick für die zukünftige Entwicklung von mobilen Gesundheitsüberwachungssystemen.

1.1. Motivation

Der demographische Wandel führt dazu, dass die Lebenserwartung der Menschen steigt und der relative Anteil an älteren Leuten zunimmt. Es wird prognostiziert, dass der Anteil der über 60-Jährigen an der Gesamtbevölkerung in Deutschland von 25 Prozent im Jahr 2006 auf 37 Prozent im Jahr 2050 steigen wird [Bun06]. Diese bisher beispiellose Entwicklung stellt auch das Gesundheitssystem vor neue Herausforderungen, weil die gesundheitliche Versorgung vieler Menschen zu steigenden Kosten führt. Mobile Gesundheitsüberwachungssysteme können eine Lösung sein, um das Gesundheitssystem auch nachhaltig finanziert zu halten und trotzdem eine hohe Qualität der gesundheitlichen Versorgung zu gewährleisten. Derartige Systeme stellen neue Möglichkeiten für die Diagnose und Therapie von Krankheiten in Aussicht. Ebenfalls verändert sich die Art und Weise, wie mit Erkrankungen umgegangen wird. Der Schwerpunkt verschiebt sich mit diesen Geräten von der reaktiven Behandlung von Krankheitsbildern zu einer proaktiven Prävention und Früherkennung von Krankheiten. Auf diese Weise sollen entstehende Kosten im Gesundheitswesen reduziert werden. Die Sensorplattform ist ein mobiles Gesundheitsüberwachungssystem, das für zukünftige Forschungsprojekte am Fraunhofer FIT konzipiert wird. [MOJ06].

Der Markt für Computerhardware ist nicht nur sehr umfangreich, sondern verändert und entwickelt sich auch ständig weiter. Komponenten werden immer kleiner, leistungsfähiger, energieeffizienter und ebenfalls häufig billiger. Während früher häufig Computersysteme mit x86-Architektur verwendet wurden, kommen heutzutage energieeffiziente ARM-Prozessoren in vielen Geräten zum Einsatz. Ein Beispiel stellen Single Board Computer (SBC) dar, zu denen die Sensorplattform ebenfalls zugeordnet werden kann. Diese Entwicklung ermöglicht außerdem den effektiven Einsatz von Computern in immer neuen Anwendungsfeldern, die zuvor nur schwer zu erschließen waren, zu denen unter anderem auch die Medizintechnik gehört. [PG10].

Die Grundlage für mobile Gesundheitsüberwachungssysteme sind die Sensoren, mit denen unterschiedliche Vitalparameter direkt am menschlichen Körper erfasst werden können. Die Messungen können automatisiert und längerfristig durchgeführt werden, erfolgen jedoch immer nicht-invasiv. Während dieser Zeit ist weder medizinisches Fachpersonal noch ein stationärer Aufenthalt, z. B. in einem Krankenhaus, erforderlich. Die Sensoren können durch besonders energiesparende Funkübertragungstechniken die Messwerte an die Sensorplattform übermitteln, wodurch sich mehrere Vorteile ergeben. Der Patient ist nicht lokal durch die Messungen eingeschränkt und kann seinem gewohnten Tagesablauf nachgehen. Außerdem begrenzt die Kombination aus kleinen, drahtlosen Geräten keine Bewegungsabläufe. Auf diese Weise soll eine hohe Qualität der gesundheitlichen Versorgung des Patienten erreicht werden und es ist das Ziel, durch das Entfallen von Betreuung und Aufenthalt des Patienten im Krankenhaus, Kosten zu sparen [Frab]. Zusätzlich wäre es vorteilhaft, wenn sich die Sensorplattform flexibel mit unterschiedlichen Sensoren verbinden ließe. Dies ermöglicht sowohl die Anpassung des Systems an individuelle Krankheitsbilder, als auch die Verwendung von Sensoren aus unterschiedlichen Systemen.

Die Sensordaten können über die Sensorplattform durch das Internet zu einem Webserver in Echtzeit übertragen werden. Der Begriff Echtzeit wird in diesem Zusammenhang in abgeschwächter Form verwendet. Zum Beispiel werden Latenzen von einigen Sekunden bei den Funkübertragungen im Rahmen der Echtzeitfähigkeit von mobilen Gesundheitsüberwachungssystemen toleriert. Mit Algorithmen, bspw. für maschinelles Lernen, kann der Computer in der Cloud die Daten auswerten. Dafür sind in der Regel große Datenmengen und Rechenleistung erforderlich, sodass diese Algorithmen nicht auf der Sensorplattform selbst ausgeführt werden können. Zukünftig sollen Computer in der Lage sein, Diagnose- und Therapieverfahren für Patienten zu entwickeln. Kurzfristig sind einfachere Aufgaben

vorstellbar, um die Arbeit des Arztes zu unterstützen. Zum Beispiel könnte der Computer unbrauchbare oder verrauschte Messwerte herausfiltern, bevor der Arzt diese Daten auswertet. Die Aufzeichnung von Sensordaten in Echtzeit ermöglicht außerdem die Reaktion auf kritische Werte, bspw. in Form eines Alarmsignals, das direkt an Rettungskräfte weitergeleitet werden kann.

Aus dieser Skizzierung ist bereits erkennbar, dass die Sensorplattform mit Computerhardware, Sensoren und Funkübertragungstechniken viele Aspekte der Informatik und Elektrotechnik kombiniert. Technische Systeme, die direkt am menschlichen Körper verwendet werden, stellen aber noch umfangreichere Anforderungen. Medizinische Richtlinien und Usability sind bspw. zwei Schwerpunkte, die es bei der Konzeption der Sensorplattform ebenfalls zu berücksichtigen gilt. Es ist außerdem erforderlich, Barrieren bei der Annahme des Geräts durch den Patienten zu überwinden und medizinisches Fachpersonal mit dem Umgang mit der Sensorplattform zu schulen [Frab]. Letztlich sind auch geringe Kosten für ein solches System von großer Bedeutung. Bei der Sensorplattform werden viele Anforderungen berücksichtigt, jedoch auch einige Einschränkungen gemacht (siehe Anforderungsanalyse in Kapitel 4).

Eingebettete Systeme für die medizinische Datenerfassung gibt es bereits käuflich zu erwerben und auch die Forschung greift solche Konzepte immer wieder auf (siehe Kapitel 3.2). Wie sich diese verschiedenen Lösungen unterscheiden, welche Vor- und Nachteile sie bieten und ob diese überhaupt für die Sensorplattform in Frage kommen, ist nicht direkt ersichtlich. Auch gibt es verschiedene Standards und Spezifikationen, z. B. die Funkübertragungsstandards Bluetooth und Ant+, die Schnittstellen und Interaktion mit Hardwaredkomponenten definieren. Diese Abstraktionsstufen werden in diesem Kontext ebenfalls berücksichtigt, um ein besseres System zu realisieren. Durch eine individuelle Implementierung und den entsprechenden Tests lassen sich weitere Parameter und Funktionen bestimmen, die für die Sensorplattform relevant sind. Beispiele für solche Parameter sind der Stromverbrauch von Hardwaremodulen und die Signalqualität der Mobilfunkverbindung.

Das Fraunhofer FIT hat in der Vergangenheit bereits viele Forschungsprojekte im Bereich Medizintechnik abgeschlossen. Dabei wurden auch Systeme genutzt, die Ähnlichkeiten mit der Sensorplattform aufweisen. Die Lösungen waren aber bisher projektspezifisch für einen Anwendungsfall konzipiert und konnten nicht projektübergreifend für ähnliche Nutzungszenarien verwendet werden. Durch die bereits beschriebene technische Entwicklung sind zukünftig aber noch weitere Anwendungsmöglichkeiten absehbar. Für kommende Projekte besteht im Institut daher das Interesse an einer allgemeineren Lösung für eine Sensorplattform, die bei zukünftigen Projekten zum Einsatz kommt. Die Ergebnisse dieser Masterarbeit können dann als Leitfaden für die Konzeption einer Sensorplattform genutzt werden.

Die Sensorplattform stellt in Verbindung mit den Sensoren und einem Webserver ein verteiltes System dar. Dadurch wird in dieser Arbeit auch die Betrachtung von Schnittstellen notwendig, die die Kommunikation zwischen unterschiedlichen Geräten ermöglichen. Eine Teilmenge dieser Geräte stellen z. B. die Sensoren dar, die ebenfalls einen Forschungs- und Entwicklungsbereich am Fraunhofer FIT bilden. Die Erkenntnisse bezüglich der Schnittstellen zu der Sensorplattform aus dieser Arbeit können zum Teil auch für die Konzeption von Sensoren verwendet werden, um die Kommunikation mit der Sensorplattform zu ermöglichen.

Das Konzept der Sensorplattform ist außerdem nicht nur auf die Medizintechnik beschränkt. Eine abstrakte Betrachtung des technischen Systems offenbart eine typische Anwendung des Internet-of-Things (IoT). Zum Beispiel könnten Sensoren auch die Lufttemperatur, -Druck und Feuchtigkeit messen und diese Daten per Bluetooth zunächst

an einen mobilen Computer als Wetterstation senden. Dieser überträgt die Sensorwerte anschließend über Mobilfunk an einen Webserver. Bestimmte technische Lösungen der Informationssicherheit, z. B. die Verwendung von Verschlüsselung und Authentifizierung zur Absicherung von Funkverbindungen, gelten für fast alle Geräte. Damit lassen sich die Ergebnisse dieser Arbeit auch für eine Vielzahl von Anwendung des Internet-of-Things (IoT) nutzen.

Die heterogenen Systeme des Internet-of-Things (IoT) stellen Anwendungsentwickler jedoch auch vor neue Herausforderungen. Unterschiedliche Geräte können die gleiche Funktion erfüllen, sich aber aus technischer Sicht deutlich voneinander unterscheiden, sodass für den Softwareentwickler ggf. tiefgreifende Kenntnisse des Systems erforderlich sind. Das Konzept Web-of-Things (WoT) umfasst allgemeine Ansätze und Software-Patterns, um Geräte in das World Wide Web zu integrieren, zu denen auch ein mobiles Gesundheitsüberwachungssystem wie bspw. die Sensorplattform gehört. Diese Abstraktion der Anwendungsebene soll die Arbeit für Entwickler erleichtern und letztlich auch die Usability für den Benutzer verbessern. Das Web-of-Things (WoT) ist auch ein Teilgebiet der Forschung (siehe Kapitel 3.2).

Die Sensorplattform vereint damit eine Vielzahl von aktuellen Themenbereichen aus der Forschung und Technik. Es ist absehbar, dass mobile Gesundheitsüberwachungssysteme für die Entwicklung am Fraunhofer FIT zukünftig eine wichtige Rolle darstellen werden, weshalb ein Leitfaden für die Konzeption einer Sensorplattform erforderlich ist.

1.2. Ziele

Das Ziel der Masterarbeit ist die Entwicklung der Software für ein eingebettetes System, das als Sensorplattform genutzt wird. Der Schwerpunkt dieses mobilen Gesundheitsüberwachungssystems liegt auf der flexiblen Konfigurierbarkeit und Erweiterbarkeit durch unterschiedliche Sensoren. Durch Verwendung von spezifizierten Schnittstellen wie beispielsweise Bluetooth Low Energy (BLE) soll es bei der Sensorplattform möglich sein, Kategorien von Sensoren allgemeingültig anzusteuern. Ein neues Gerät kann auf diese Weise schnell in die Sensorplattform integriert werden, weil sich die Steuerung auf Anwendungsebene nicht von den bisherigen Sensoren unterscheidet. Die dafür notwendigen Veränderungen oder Ergänzungen an der Software der Sensorplattform fallen dann nur gering aus. Die Formulierung *Integration in die Sensorplattform*, die im Folgenden wiederholt genutzt wird, bedeutet, dass die Sensorplattform eine Verbindung zum Sensor aufbauen kann und Messwerte empfängt. Es bedeutet nicht, dass der Sensor physisch in das Gerät der Sensorplattform eingebaut wird, sondern die *Integration* erfolgt ausschließlich auf Software-Ebene. Vor der Softwareentwicklung werden einige Randbedingungen für die zugrundeliegende Hardware und Software festgelegt, die unter anderem die Anzahl von Schnittstellen einschränkt (siehe Anforderungsanalyse in Kapitel 4). Diese Masterarbeit lässt sich in drei Teilziele aufgliedern.

Das erste Teilziel ist die Auswahl geeigneter Hardware für einen Demonstrator der Sensorplattform (siehe Kapitel 4 und 5), die ausschließlich für kommerziell verfügbare Hardwarekomponenten konzipiert ist. Eine individuelle Hardware- (und damit einhergehende Software-) Entwicklung würde den Umfang der Masterarbeit deutlich überschreiten und ist daher nicht umsetzbar. Das System muss über bestimmte kabel-gebundene und drahtlose Schnittstellen verfügen, um alle in der Software vorgesehenen Funktionen zu ermöglichen. Die Leistung und die Hardwareausstattung soll außerdem für die Sensorplattform angemessen sein. Die bei der Auswahl der Hardware getroffenen Entscheidungen bilden die Grundlage für die Implementierungsphase.

Das zweite Teilziel ist die Softwareentwicklung der Sensorplattform. Die Anwendung soll Messdaten von Sensoren abrufen können, diese speichern und an einen externen Webserver

übertragen. Die konkreten Funktionen sind als Use Cases in der Anforderungsanalyse in Kapitel 4 formuliert, für die auch einige Einschränkungen in Bezug auf das Betriebssystem, die Programmiersprache und die Sensoren gemacht werden (siehe Kapitel 4.7). Die Erweiterbarkeit der Software ist dabei ein entscheidendes Kriterium für die Sensorplattform. Während der Masterarbeit ist nur eine begrenzte Auswahl von Sensoren vorhanden, die mit der Sensorplattform kommunizieren werden. Es soll aber möglich sein, durch die Nutzung von bestehenden Schnittstellen, neue Sensoren systematisch in die Sensorplattform zu integrieren. Aus dem Erreichen dieser beiden Teilziele geht der Demonstrator der Sensorplattform hervor.

Die Tests der Sensorplattform bilden das dritte Teilziel. Durch die Tests werden zunächst Fehler erkannt und in iterativen Prozessen behoben. Durch Messungen geeigneter Parameter wird das Gesamtsystem anschließend bewertet und in Kapitel 8 diskutiert. Die Energieeffizienz des Systems ist dabei ein Schwerpunkt.

Die Ausarbeitung der Masterarbeit bildet abschließend einen Leitfaden für die Konzeption und Implementierung einer Sensorplattform. Für den zukünftige Einsatz solcher oder ähnlicher Systeme für Projekte am Fraunhofer FIT kann auf die Ergebnisse dieser Arbeit zurückgegriffen werden.

2. Methode

Die Entwicklung der Sensorplattform erfolgt als Einzelarbeit, wodurch eine hohe Flexibilität bei der Konzeption und Implementierung gewährleistet ist, weil keine Absprache mit anderen Entwicklern notwendig ist. Das Konzept der Sensorplattform bildet die Grundlage für alle weiteren Entscheidungen während der Implementierungsphase, wofür zunächst eine Anforderungsanalyse erstellt wird, um alle fachlichen und überfachlichen Themenfelder, die die Sensorplattform kombiniert, festzustellen (siehe Kapitel 4). Danach werden Use Cases angefertigt (siehe Kapitel 4.8), um die grundlegende Funktionalität des Geräts zu definieren. Die Anwendungsfälle sind nicht als bindende Ziele zu verstehen, sondern bieten gerade am Anfang einen ungefähren Eindruck für den Umgang mit der Sensorplattform und werden im Laufe der Bearbeitung präzisiert. Einige technische Einzelheiten in den Use Cases werden aber bereits genannt, bspw. die Nutzung einer Webanwendung für die Mensch-Maschine-Interaktion. Die Diskussion und Absprache bestimmter Funktionen mit den Betreuern am Institut findet wiederholt statt, wodurch sich iterative Prozesse ergeben, mit deren Hilfe Details zur Sensorplattform festgelegt werden, um die gewünschte Lösung für das Projekt zu erreichen. Unter anderem werden die möglichen Schnittstellen für Sensoren auf Bluetooth Low Energy (BLE) eingeschränkt, weil dieser Funkübertragungsstandard am weitesten verbreitet ist und deshalb den Schwerpunkt für die Schnittstellen der Sensorplattform darstellt. Durch die Test Cases in Anhang A lässt sich feststellen, ob die Anwendung der Sensorplattform alle Funktionen aus der Anforderungsanalyse besitzt.

Aus der Anforderungsanalyse und den Use Cases gehen bereits konkrete Hinweise zur Hardware hervor. Im Kapitel 5.1 werden alle Anforderungen zusammengefasst und anschließend eine Auswahl getroffen, welches Board die Basis der Sensorplattform bildet. Es werden hierbei ausschließlich Leiterplatten berücksichtigt, die Linux als Betriebssystem nutzen. Dies ist eine weitere Einschränkung, damit die Software unabhängig von (großen) Softwareunternehmen ist (siehe Kapitel 4.7). Einige Sensoren sind bereits aus anderen Projekten am Institut vorhanden, andere Sensoren werden hinzu gekauft. Die Auswahlkriterien sind die Bluetooth-Spezifikationskonformität oder die ausführliche Dokumentation der Software durch den Hersteller. Die Messgrößen beziehen sich dabei nicht ausschließlich auf Vitalparameter. Die zusätzlichen Details durch unterschiedliche Sensoren und Messwerte gilt es bei der Softwarearchitektur ebenfalls zu berücksichtigen, um die Implementierung zu verbessern (siehe Kapitel 6.2). Ein Beispiel ist der Sensor CC2650 von Texas Instruments, der einige herstellerspezifische Datenfelder in der Bluetooth-Schnittstelle nutzt. Außerdem wird ein Bluetooth-Entwicklungsboard eingesetzt, um spezielle Funktionen (z. B. die Verschlüsselung) zu testen.

Nach der Auswahl der Hardware erfolgen Überlegungen zur Mensch-Maschine-Interaktion, da das System für den Anwender in der Praxis leicht benutzbar sein soll (siehe Kapitel 5.2). Diese umfassen bereits einige konkrete Details zur technischen Umsetzung der Sensorplattform. Dabei wird unter anderem deutlich, dass das medizinische Fachpersonal über eine Webanwendung mit der Sensorplattform interagieren soll. Es gilt dabei auch herauszufinden, welche Informationen in der Sensorplattform angegeben werden müssen, damit sich diese mit einem Sensor verbinden kann. Das daraus resultierende Konzept wird als Grundlage für die Implementierung nach einer Vorgehensweise mit Ähnlichkeiten zu *Extreme Programming* genutzt.

Die Implementierungsphase lässt sich in die Arbeitspakete Bluetooth-Schnittstelle, Datenbank, Schnittstellen zum externen Webserver, Linux-Konfiguration und Webanwendung der Sensorplattform unterteilen.

Allgemein wird bei der Softwarearchitektur versucht, für die grundlegenden Funktionen Treiber, Bibliotheken und Frameworks zu nutzen, um funktionierenden und getesteten Code als Basis der Sensorplattform zu gewährleisten. Zudem würde es den Aufwand deut-

lich überschreiten, bspw. den Bluetooth-Stack neu zu implementieren. Auch die verwendete Programmiersprache und die Vorgabe des Betriebssystems haben erheblichen Einfluss auf die erstellte Softwarelösung. Auf Grund fehlender oder nicht ausgereifter Programmierschnittstellen für Bluetooth Low Energy (BLE) in Java sind Kompromisse bei der Softwareentwicklung erforderlich (siehe Kapitel 6.1), wobei eine Implementierung in Java zwei wesentliche Vorteile bietet. Die Portierbarkeit der Software auf andere Linux-Boards ist verhältnismäßig einfach gewährleistet. Außerdem passt eine weit verbreitete Programmiersprache wie beispielsweise Java zu der bisherigen Unternehmensstrategie des Fraunhofer FIT bezüglich Softwareentwicklung.

Die Softwareentwicklung erfolgt unter Berücksichtigung von Design-Patterns, um Struktur, Erweiterbarkeit und Testbarkeit der Sensorplattform zu gewährleisten. Einzelne Funktionen werden zunächst separat und ohne Interaktion zueinander entwickelt. Angefertigte Modultests stellen die korrekte Funktionsweise von einzelnen Softwarekomponenten schon während der Entwicklungsphase sicher, um danach miteinander verknüpft zu werden. Die Implementierung wird auf diese Weise in kleinen Schritten inkrementell um neue Funktionalität erweitert, bis hin zur fertigen Software. Durch Koordination mit den Betreuern am Institut werden Teilergebnisse besprochen und iterativ die nächsten Teilziele für der Softwareentwicklung festgelegt. Am Ende der Implementierungsphase werden Testfälle erstellt, mit denen alle Use Cases für die Sensorplattform getestet werden können (siehe Tabelle 7). Die Anweisungen aus den Test Cases umfassen reproduzierbare Szenarien beim Umgang mit der Sensorplattform, damit ein Tester die korrekte Funktionsweise der Software verifizieren kann. Die Modultests behandeln nur einzelne Softwareelemente ohne geräteübergreifende Interaktion und sind deshalb nicht ausreichend. In den Test Cases ist zum Beispiel das Abschalten eines Sensors durch den Tester vorgesehen, um das Verhalten der Sensorplattform in diesem Fehlerfall zu überprüfen. Das automatisierte Ausführung solcher Tests ist auf Grund der notwendigen und komplizierten Testumgebung erschwert. Weiterführend ist es möglich, dass virtuelle Testdaten anstatt der konkreten Sensorhardware genutzt werden. In Kapitel 7.1.1 wird ein möglicher Ansatz dafür vorgestellt. Eine dynamische Konfiguration der Sensorplattform, entweder mit realen oder virtuellen Sensoren ist aber für die Masterarbeit zu umfangreich, weil beispielsweise asynchrone Bluetooth-Benachrichtigungen simuliert werden müssen. Als Ausbaustufe wäre es aber sinnvoll, diesen virtuellen Testbetrieb zu ermöglichen, damit die Software dann auch auf unterschiedlichen Betriebssystemen ausgeführt werden kann. Zudem wird dadurch das Testen vereinfacht, weil keine Sensoren mehr vorhanden sein müssen. Dies ist jedoch nicht mehr Bestandteil der Arbeit.

Die Sensorplattform setzt wie bereits erwähnt auf verschiedenen Bibliotheken und Treibern (z. B. BlueZ für Bluetooth) auf. Befinden sich Fehler innerhalb des Software-Stacks, kommen diese wahrscheinlich auch in der Sensorplattform vor. Detaillierte Tests für diese Bibliotheken und Treiber können in der gegebenen Zeit nicht entwickelt werden. Im Bereich Informationssicherheit werden etablierte Verfahren und Algorithmen genutzt sowie Richtlinien und Empfehlungen vom Bundesamt für Sicherheit in der Informationstechnik (BSI) und vom National Institute of Standards and Technology (NIST) berücksichtigt.

Die Software für die Sensorplattform ist als Maven-Projekt strukturiert. Die Verwaltung von Bibliotheken und insbesondere die Erstellung der Dokumentation sind durch dieses Build-Werkzeug bereits gegeben [Apa]. Systemnahe Software, z. B. Treiber, werden in Form von Paketen und Shell-Skripts auf der Sensorplattform installiert. Die Verteilung einer solchen Installation kann bspw. mit einer Image-Datei erfolgen. Die Firmware für das BLE113 Entwicklungsboard als Sensor wird in einem BGScript-Projekt bereitgestellt (siehe Kapitel 6.10.2).

Schließlich werden weitere Tests mit der Sensorplattform durchgeführt, um wichtige Para-

meter zu bestimmen, die nicht direkt in Zusammenhang mit der Implementierung stehen. Diese können bspw. Kenngrößen zum Bestimmen der Signalqualität der Mobilfunkverbindung sein und der daraus resultierende Stromverbrauch beim Übertragen von Sensordaten an externe Webserver. Ein weiteres Beispiel für einen derartigen Parameter ist der Schreibintervall der Datenbank auf den Flash-Speicher. Die aus den Tests gewonnenen Erkenntnisse werden genutzt, um die Software der Sensorplattform in einigen Iterationen zu verbessern und Funktionalität zu ergänzen.

Das System wird außerdem unter realistischen Bedingungen an Personen getestet. Die Erkenntnisse aus diesen Tests sollen bei der Evaluation und Verbesserung der Software mit einfließen. Außerdem wird die Performanz der Sensorplattform zu einem Großteil durch die verbundenen Sensoren (und deren Abtastraten) bestimmt (siehe Kapitel 7.3). Auf welche Parameter die Sensoren außerdem Einfluss haben, wird ebenfalls herausgestellt. Dies bezieht sich unter anderem auf die Überlegung, wie die gewonnenen Erkenntnisse des Demonstrator der Sensorplattform zukünftig für mobile Gesundheitsüberwachungssysteme genutzt werden können.

3. Grundlagen

Die Computerhardware und auch die Funkübertragungstechniken haben sich in den letzten Jahren stark verändert und weiterentwickelt, sodass viele technische Geräte kleiner, leistungsfähiger, energieeffizienter und auch häufig billiger geworden sind. Diese Entwicklung bietet neue Anwendungsgebiete für derartige Technologien. Gerade in der Medizintechnik resultieren daraus viele unterschiedliche Konzepte für mobile Gesundheitsüberwachungssysteme, zum automatisierten Erfassen von Sensordaten am menschlichen Körper [PG10, MOJ06]. Die Sensoren können beispielsweise Herzfrequenz, Blutdruck, Sauerstoffgehalt im Blut, Glukose, Temperatur oder Bewegungen nicht-invasiv messen. Derartige Messwerte heißen auch Vitalparameter.

Das Kapitel der Grundlagen umfasst den Stand der Technik und den Stand der Forschung. Daraus ergibt sich ein konkreter Überblick über die Thematik mobiler Gesundheitsüberwachungssysteme, die im Folgenden auch Wearable Health Monitoring System (WHMS) genannt werden.

3.1. Stand der Technik

Zunächst wird ein Überblick über die Kategorien mobiler Gesundheitsüberwachungssysteme gegeben. Anschließend erfolgt eine genauere Kategorisierung der bestehenden Systeme, um die Sensorplattform zuordnen zu können. Danach wird eine Auswahl von kommerziell verfügbaren Systemen des Verbrauchermarktes zum Messen von Vitalparametern vorgestellt und die geläufigen Funkübertragungstechniken dieser Systeme sind im Anschluss daran erklärt. Der Bluetooth-Standard wird ausführlich behandelt, weil diese Schnittstelle zu den Sensoren die Grundlage für die Sensorplattform bildet (siehe Anforderungsanalyse in Kapitel 4).

3.1.1. Kategorien mobiler Gesundheitsüberwachungssysteme

Die Informationen für die Kategorisierung basieren auf den Papern *Wireless sensor networks for personal health monitoring: Issues and an implementation* von Aleksander Milenkovic, Chris Otto und Emil Jovanov und *A Survey on Wearable Sensor-Based Systems for Health Monitoring and Prognosis* von Alexandros Pantelopoulos und Nikolaos G.Bourbakis [MOJ06, PG10]. Diese Paper werden im Kontext von WHMSs häufig zitiert, weshalb beide Paper mit einer hohen Aussagekraft bewertet werden.

Ein WHMS zeichnet über einen längerfristigen Zeitraum in Echtzeit Vitalparameter am menschlichen Körper auf. Der Schwerpunkt bei diesem Konzept liegt auf der proaktiven Behandlung, auf der Früh-Diagnose und Prävention von Krankheiten. Allgemein besteht ein WHMS aus drei Stufen, die in Abbildung 2 dargestellt sind. Die Basis (Stufe 1) besteht aus verschiedenen am Körper platzierten Sensoren, die durch Verwendung von Funkübertragungstechniken ein Wearable Wireless Body/Personal Area Network (WWBAN) bilden. Damit werden Vitalparameter gemessen und über die Funkschnittstelle für andere Geräte zugänglich gemacht, wofür sich z. B. die Standards Bluetooth und Zigbee eignen. Ein Personal Server (PS), der als zentrale Einheit mit dem Funknetzwerk kommuniziert, sammelt zunächst diese Sensordaten wie bei einem Zwischenspeicher. Der Personal Server (PS) ist ebenfalls in der Lage, die Netzwerkkonfiguration für die Sensoren im WWBAN vorzunehmen. In Abbildung 2 stellt ein Smartphone bzw. ein Personal Computer die Stufe 2 dar, wobei in dieser Arbeit die am Körper getragene Sensorplattform dieses Gerät bildet. Diese sendet die Sensordaten über WLAN oder Mobilfunk an die Stufe 3. Eine Webinfrastruktur speichert und verarbeitet die Sensordaten, sodass dadurch eine Reaktion auf kritische Messwerte in Echtzeit möglich ist. Die Alarmierung eines Krankenwagens oder ein Hinweis für den Arzt wären mögliche Reaktionen. Der Webserver kann auch genutzt werden, um mit Hilfe von Algorithmen die Daten auszuwerten, um so Ansätze für Diagnose

und Therapie einer Krankheit zu erstellen. [MOJ06].

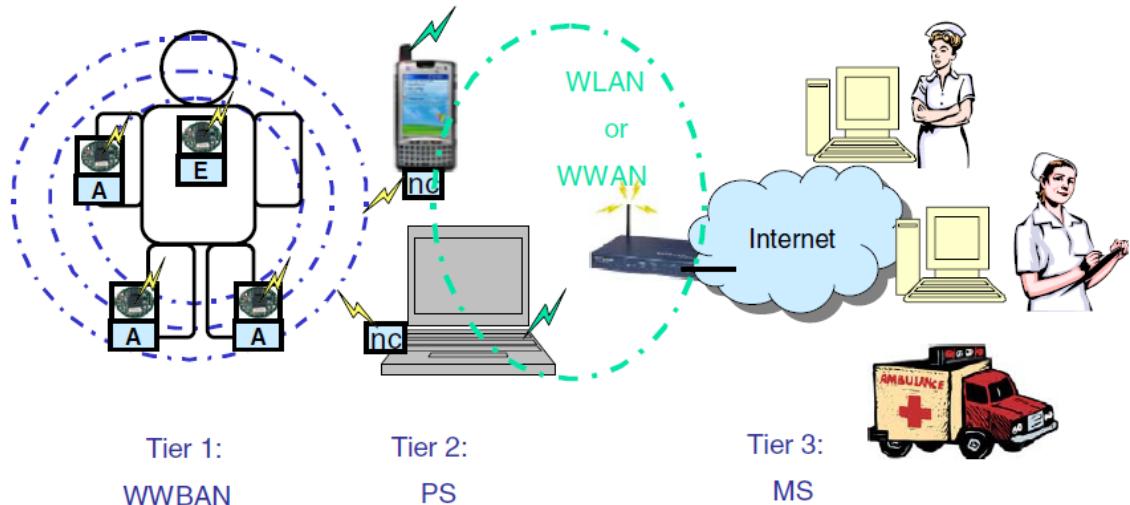


Abbildung 2: Beispiel eines mobilen Gesundheitsüberwachungssystems [MOJ06]

Wie bereits im vorherigen Abschnitt erwähnt, stellt die Sensorplattform den Personal Server (PS) aus Stufe 2 dar. Im Rahmen dieser Arbeit werden alle drei Stufen sowie deren Schnittstellen behandelt (siehe Kapitel 6). Zum Beispiel berücksichtigt die Verwendung des BLE113 Entwicklungsboard bestimmte Anforderungen der Informationssicherheit an die Schnittstelle zwischen dem Wearable Wireless Body/Personal Area Network (WWBAN) und der Sensorplattform (siehe Kapitel 3.1.2 und 6.10.2). Ebenfalls kommt für die Sensorplattform auch ein individueller externer Webserver zum Einsatz, zu dem die Herzfrequenzdaten hochgeladen werden (siehe Kapitel 6.7). Diese Kommunikation bildet die Schnittstelle zwischen der Stufe 2 und Stufe 3 aus Abbildung 2.

Die drei vorgestellten Stufen sind zunächst allgemein definiert und beinhalten noch keine technischen Details. Durch die unterschiedlichen eingesetzten Techniken und Hardwarekomponenten auf jeder der drei möglichen Stufen, kann das allgemein definierte Konzept flexibel auf konkrete Anforderungen einer Anwendung modelliert werden. Daher wird im Folgenden eine Kategorisierung vorgestellt, die bereits einige technische Details berücksichtigt [PG10].

Die erste Kategorie umfasst Mikrocontroller-Boards als zentrale Einheit. Die Softwareentwicklung erfolgt sehr nah an der Hardware und die Sensoren werden in der Regel mit Kabeln (serielle Schnittstellen oder Bus-Systeme) angeschlossen [PG10]. Das Gesamtsystem ist häufig für einen bestimmten Anwendungsfall konzipiert und lässt sich nur schwer erweitern. Die Datenauswertung erfolgt beispielsweise auf einem Personal Computer anstatt in einer Webanwendung.

Systeme auf Basis von Textilien bilden die zweite Kategorie. Sensoren sind mittlerweile so klein, dass sie auch in Kleidung eingearbeitet werden können. So ein System besteht dann aus einer Weste oder einem T-Shirt mit einer Auswahl integrierter Biosensoren. Das bietet den Vorteil, dass das Gesamtsystem kleiner und unauffälliger ist. Es wird aber ebenfalls eine zentrale Einheit benötigt, die die Sensordaten sammelt. [PG10].

Die dritte Kategorie bezeichnet am Körper getragenen Sensoren, die in Kombination ein Body Area Network (BAN) (ein Äquivalent zu einem Wearable Wireless Body/Personal Area Network (WWBAN)) bilden. Solche Netzwerke lassen sich beispielsweise mit Funkübertragungstechniken wie Zigbee oder Bluetooth realisieren. Des Weiteren ist dafür eine Basisstation notwendig, zu der die Sensordaten übertragen werden. Die Signalstation wird in der Regel nicht am Körper getragen, sondern kann bspw. in der Wohnung

platziert werden. Diese limitiert dann aber den Arbeitsbereich des Systems und somit die Bewegungsfreiheit des Patienten. In dieser Kategorie wird auch erstmals die Möglichkeit vorgestellt, Sensoren zu implantieren, deren Daten anschließend von der zentralen Einheit gesammelt werden können. [PG10].

Die vierte und letzte Kategorie umfasst kommerziell verfügbare Systeme. Besonders in den letzten Jahren hat sich der Markt von Sensoren für Vitalparameter mit den dazugehörigen Anwendung rasant vergrößert. In der Regel gibt es zu den Geräten die entsprechende App für das Smartphone, mit der die Sensorsdaten gesammelt und dargestellt werden. Häufig ist aber auch die direkte Verbindung mit einem Cloud-Dienst an die App gekoppelt, um die Daten langfristig zu speichern. [PG10].

Auffällig ist, dass sich die Sensorplattform in keine der vier Kategorien eindeutig zuordnen lässt, sondern Aspekte aus den Kategorien 1, 3 und 4 kombiniert und somit eine allgemeinere Lösung eines Wearable Health Monitoring System (WHMS) definiert. Diese Tatsache wird so gedeutet, dass sich die Ideen und Konzepte bezüglich Wearable Health Monitoring System (WHMS) auch in den letzten Jahren kontinuierlich weiterentwickelt haben. Die Kategorisierung von Alexandros Pantelopoulos und Nikolaos G.Bourbakis aus dem Jahr 2010 ist daher nicht mehr ausreichend. Im Folgenden werden die allgemeinen Anforderungen an WHMS aufgeführt.

3.1.2. Anforderungen an mobile Gesundheitsüberwachungssysteme

Die Anforderungen beziehen sich insbesondere auf das Wearable Wireless Body/Personal Area Network (WWBAN) (Stufe 1) und auf die Sensorplattform (Stufe 2), bei denen die technischen Ressourcen begrenzt sind und die trotzdem eine Vielzahl von Aspekten erfüllen sollen. Der Webserver (Stufe 3) ist weniger kritisch, weil dieser auf einer stationären IT-Infrastruktur basiert. Die Anforderungen beziehen sich auf Tragbarkeit, Zuverlässigkeit, Sicherheit und Kompatibilität. Außerdem ist die Echtzeitfähigkeit des Systems von Bedeutung. [PG10, MOJ06].

Die Tragbarkeit ist der wichtigste Aspekt der Sensoren. Diese sollen so klein und leicht wie möglich sein, um unauffällig und kontinuierlich Messdaten zu erfassen. Maßgeblich für die Größe und Gewicht des Sensors ist die Batterie). Ein Kompromiss zwischen Abmessung des Sensors und dessen Betriebszeit ist dabei häufig notwendig. Es ist absehbar, dass sich dieser Bereich zukünftig weiter verbessern wird und kleinere und sparsamere Sensoren hervorbringt. [PG10, MOJ06].

Die Zuverlässigkeit bezieht sich auf die im Wearable Wireless Body/Personal Area Network (WWBAN) genutzte Funkübertragungstechnik. Eine stabile Verbindung zwischen den Hardwarekomponenten ist die Grundlage für die korrekte Funktionsweise des WHMS. Auf Grund der begrenzten Batteriekapazität sind ebenfalls Kompromisse bei der Häufigkeit des Datenaustausches und bei der Bandbreite der Verbindung einzugehen. [MOJ06].

Der Datentransfer zwischen allen Komponenten sollte nach Möglichkeit verschlüsselt vorgenommen werden, damit die Daten nicht durch passive Lauschangriffe aufgezeichnet und ausgewertet werden können. Außerdem muss durch Authentifizierungsmethoden die Integrität des Systems gewährleistet sein. Die Authentifizierung stellt sicher, dass kein Man-In-The-Middle (MITM) am System partizipiert. [MOJ06].

Das WWBAN soll auch flexibel konfigurierbar sein, damit unterschiedliche Sensoren in Kombination mit dem Personal Server (PS) genutzt werden können. Durch die Verwendung offener Standards bei den Funkübertragungstechniken kann dies gewährleistet sein. [MOJ06].

Letztlich ist auch die Echtzeitfähigkeit für ein WHMS wichtig. Werden die Daten zeitnah an einen Webserver übertragen und dort ausgewertet, kann beispielsweise eine Reaktion

auf kritische Messwerte erfolgen. Auch das Gerät selbst könnte bei Überschreitung von Grenzwerten Rückmeldung direkt an den Patienten geben. [PG10].

Diese Anforderungen stehen zum Teil in Konkurrenz zueinander, sodass bisher keine optimale Lösung für ein WHMS absehbar ist, weil bspw. die Batterie eines Sensor nicht gleichzeitig eine große Kapazität besitzen und gleichzeitig so klein wie möglich sein kann. Stattdessen gehen immer unterschiedliche Kompromisslösungen hervor. Das gilt sowohl für Sensoren im WWBAN (Stufe 1) als auch für Geräte in Stufe 2, z. B. die Sensorplattform. Das folgende Kapitel stellt einige kommerzielle Systeme vor, die auch in Verbindung mit der Sensorplattform genutzt werden sollen.

3.1.3. Kommerzielle Produkte

In diesem Kapitel werden kommerzielle Produkte vorgestellt, die sich in Kombination mit der Sensorplattform verwenden lassen. Der Markt der *Fitness Tracker* ist noch relativ neu, jedoch gewinnt dieser insbesondere durch die Weiterentwicklung von Smartphones zunehmend an Bedeutung. Ein Smartphone stellt häufig ein zentrales System dar, das mit externen Sensoren oder Geräten verbunden ist. Außerdem verfügt das Smart-phone selbst über eine Vielzahl von integrierten Sensoren (bspw. um die Beschleunigung und damit die körperliche Aktivität zu messen). Eine Software, die die Geräte miteinander verknüpft, wird als App zur Verfügung gestellt, worüber sich anschließend Sensordaten erfassen, speichern und anzeigen lassen. Häufig gibt es dazu einen Cloud-Dienst, der für die langfristige Datenspeicherung und zum Erheben von Statistiken genutzt wird. Im Folgenden werden einige kommerziell verfügbare Systeme vorgestellt, die sich für die mobile Erfassung von Vitalparametern eignen. Diese beschränken sich zunächst auf den Funkübertragungsstandard Bluetooth, weil auch die Sensorplattform zu Beginn diese Schnittstelle zur Verfügung stellt (siehe Anforderungsanalyse in Kapitel 4). Eine Teilmenge dieser Sensoren wird im Rahmen der Arbeit in die Sensorplattform integriert. Eine zukünftige Erweiterung des Systems durch die anderen vorgestellten Sensoren ist möglich.

Die Grundlage am Verbrauchermarkt bilden Fitnessuhren, Herzfrequenzsensoren, Pulsoximeter und Smartphones. Die im folgenden Abschnitt betrachteten Beispiele beziehen sich auf Geräte der Hersteller Polar, Adidas und TomTom [Pola, adi, Tom].

Die Fitnessuhren verfügen über einen Bewegungssensor (Gyroskop, Beschleunigungsmesser oder Magnetometer) und manchmal auch über ein Pulsoximeter, um den Puls (und gegebenenfalls SpO₂) am Handgelenk zu messen. Darüber hinaus besitzen die teureren Modelle einen GPS-Chip. Ergänzend zu den Uhren gibt es Herzfrequenzsensoren, die mit einem Brustgurt am Körper getragen werden. Abbildung 3 zeigt den Herzfrequenzsensor Polar H7 mit Brustgurt. Diese Sensoren können die Herzfrequenz messen und bei einigen Modellen auch die RR-Intervalle oder den Energieverbrauch. Die mobile Stromversorgung der Sensoren ist durch Knopfzellen-Batterien und bei der Fitness-uhr durch einen integrierten Akku gewährleistet. Außerdem gibt es von jedem Hersteller eine App für das Smartphone, die alle gesammelten Daten in Bezug zueinander setzt und einem Cloud-Dienst der jeweiligen Hersteller hochlädt. Eine Webanwendung kann anschließend beispielsweise Statistiken zur Trainingsintensität aufbereiten. Diese Geräte sind nicht als Medizinprodukte zugelassen [Rat93]. Preislich liegen die Herzfrequenzsensoren bei ungefähr 40 Euro und die Fitnessuhren bei mindestens 100 Euro. [Pola, adi, Tom].

Die allgemeine Funktionsweise dieser Systeme ist ähnlich. Auf der Fitnessuhr oder auf dem Smartphone werden bestimmte Trainingsprogramme, bspw. Sprungtraining, ausgewählt. Mit Hilfe der Bewegungssensoren wird das Ausführen der Übung in der Software nachvollzogen und die Herzfrequenz des Benutzers während der Übung aufgezeichnet. Verringert sich die Herzfrequenz bei Wiederholung des Trainings in den nächsten Tagen oder Wochen, so ist der Anwender leistungsfähiger geworden. Dann bieten weitere Programme in



Abbildung 3: Herzfrequenzsensor Polar H7 mit Brustgurt [Polb]

der Software bspw. eine Erhöhung der Trainingsintensität, mit der das Training fortgeführt wird. Die Statistiken zu den Sensordaten und Übungen werden häufig auch grafisch in einer Webanwendung aufbereitet.

Das Konzept der Sensorplattform beschreibt ein System, das über offene Schnittstellen mit vielen verschiedenen Geräten kommunizieren kann (siehe Anforderungsanalyse in Kapitel 4). Diese Schnittstelle ist bei den Produkten von Polar, Adidas und TomTom durch GATT-Profilen aus der Bluetooth-Spezifikation definiert (siehe Kapitel 3.1.4). Im Rahmen dieser Arbeit soll die Sensorplattform mit den drei Herzfrequenzsensoren kommunizieren können (siehe Kapitel 5.1.2).

Als Zwischenstand lässt sich festhalten, dass die bisher vorgestellten Sensoren die folgenden Parameter messen können: Herzfrequenz, RR-Intervalle, Energieverbrauch, Puls, SpO₂, Beschleunigung und die Position (GPS). Die Messwerte dieser Sensoren können durch eine Bluetooth Low Energy (BLE) Verbindung ausgelesen werden. Auch in der Medizintechnik und im Rahmen von Wearable Health Monitoring System (WHMS) werden derartige Parameter für bestimmte Krankheitsbilder erfasst. Einige Hersteller, bspw. Philips und Jawbone, konzipieren derartige Geräte neuerdings auch als Medizinprodukte für den Verbrauchermarkt [Ver16, Rao16]. Jawbone hat zuvor mit der Serie *UP* eigene Fitness Tracker für das Handgelenk entwickelt [Jaw]. Um diesen jüngsten Trend zu berücksichtigen, werden nachfolgend weitere Geräte vorgestellt, die als Medizinprodukte zugelassen sind. Diese Systeme sind von unterschiedlichen Herstellern und sollen den in dieser Arbeit vorgestellten Verbrauchermarkt der Fitness Tracker vervollständigen.

Zunächst wird in Abbildung 4 (links) das Blutdruckmessgerät für das Handgelenk von Philips vorgestellt. Dieses erfasst den systolischen und diastolischen Wert des Blutdrucks und die Herzfrequenz. Die Daten werden am Display angezeigt und können über Bluetooth Low Energy (BLE) auch an das Smartphone übertragen werden. Auf die Bluetooth-Spezifikation kann bei diesem Sensor zurückgegriffen werden, um das Gerät in Verbindung mit der Sensorplattform zu nutzen. [Kon].

Ein weiteres Medizinprodukt ist der Sensor Nonin 3230 von Nonin Medical Inc., dargestellt in Abbildung 4 (Mitte). Dieser Sensor ist ein Pulsoximeter, das die Messungen des Puls und SpO₂ am Finger vornimmt. Der Sensor kann intern 2200 Messpunkte speichern. Die Stromversorgung erfolgt durch zwei AAA-Batterien. Der Hersteller erwähnt explizit, dass sich das Gerät durch die Schnittstelle Bluetooth Low Energy (BLE) einfach in andere Systeme integrieren lässt. Für den Einsatz in Verbindung mit der Sensorplattform wäre dieser Sensor somit ebenfalls geeignet. [Nona].

Ein Dreikanal-Elektrokardiogramm stellt der Sensor Faros eMotion 360° dar. Abbildung 4 (rechts) zeigt den Sensor ohne angeschlossene Elektroden und deren Kabel. Die Abtastrate ist bis 1000 Hz konfigurierbar. Zusätzlich wird die Beschleunigung, Temperatur und Atmung erfasst. Dieser Sensor verfügt über die Schnittstelle Bluetooth im sogenannten Dual-Mode (Bluetooth Classic und Bluetooth Smart). Die Kommunikation erfolgt über ei-

ne serielle Verbindung die *BlueRadios Serial Port* heißt. Bei Bluetooth Low Energy (BLE) werden dabei herstellerspezifische (GATT)-Characteristics als eine serielle Schnittstelle genutzt [Blua]. [Meg].

Weil sich diese Funktionsweise von der üblichen Nutzung von Bluetooth Low Energy (BLE) GATT unterscheidet, wird dieser Sensor vorerst nicht in die Sensorplattform integriert, denn die Implementierung wäre zu zeitaufwendig. Zudem ist der Faros eMotion 360° der einzige bekannte Sensor, der diese Technik nutzt.



Abbildung 4: Philips Blutdruckmessgerät für das Handgelenk (links) [Kon], Pulsoximeter Sensor Nonin 3230 (Mitte) [Nonb], Faros eMotion 360° Dreikanalelektrokardiogramm (rechts) [Meg]

Der Stand der Technik stellt eine Auswahl von kommerziellen Produkten des Verbrauchermarkts vor. Neuerdings konzipieren einige Hersteller ihre Geräte als Medizinprodukte, wobei die Sensoren ausschließlich nicht-invasiv messen. Alle Sensoren verwendet Bluetooth Low Energy (BLE), sodass sich diese für die Integration in die Sensorplattform eignen. Um die Sensoren auch mit einem herstellerfremden System wie bspw. der Sensorplattform nutzen zu können, sind definierte und offene Schnittstellen erforderlich. Eine solche Schnittstelle stellt die Bluetooth-Spezifikation durch die GATT-Profile des Low Energy Standards zur Verfügung. Nachfolgend werden deshalb die für die Sensorplattform relevanten Details von Bluetooth erläutert. Im Anschluss daran sind weitere geläufige Funkübertragungstechniken für mobile Gesundheitsüberwachungssysteme vorgestellt.

3.1.4. Bluetooth

Bluetooth ist ein Funkübertragungsstandard mit einer Reichweite von 10 bis 100 Metern. Dieser nutzt das ISM-Band 2.4 GHz und stellt Profile für bestimmte Anwendungsmöglichkeiten (z. B. Audio- und Medienwiedergabe, Telefonie oder Eingabegeräte wie Tastaturen und Mäuse) bereit. Bluetooth wurde im Jahr 1999 veröffentlicht und hat seitdem eine Vielzahl von Erweiterungen und Verbesserungen erhalten, wie Tabelle 1 in einer Übersicht zeigt.

Mittlerweile gibt es über 8,2 Milliarden Bluetooth-Geräte und die Bluetooth Special Interest Group (SIG) hat über 30000 Mitglieder [Blu16b]. Bluetooth hat sich bereits auf dem Markt etabliert und wird in Zukunft gerade im Bereich Internet-of-Things (IoT) wahrscheinlich weiter an Bedeutung gewinnen. Die für die Sensorplattform relevanten Informationen dieser Spezifikation sind im Folgenden erläutert.

Grundsätzlich sind alle Bluetooth-Versionen abwärtskompatibel. Zum Beispiel lässt sich ein Bluetooth 4.2 Modul mit einem anderen Gerät, das Bluetooth 4.0 unterstützt, verbinden. Für den Low Energy Standard wird jedoch ein Chip verwendet, der sich von vorherigen Versionen unterscheidet. Viele Bluetooth-Controller verfügen über den sogenannten Dual-Mode. Dabei sind zwei Chips auf einem Modul integriert, sodass alle Bluetooth-Versionen auf dem Gerät genutzt werden können. Für die Sensorplattform ist die Spezifikation Low Energy der Versionen 4.0, 4.1 und 4.2 relevant. Bluetooth 4.1 beinhaltet

Jahr	Version	Beschreibung
1999	Version 1.x	Viele Probleme, insbesondere in Bezug auf Kompatibilität
2004	Version 2.0 EDR	Enhanced Data Rate (EDR) bis 2,1 Mbit/s
2007	Version 2.1 EDR	Secure Simple Pairing (SSP)
2009	Version 3.0 HS	High Speed (HS) bis 24 Mbit/s (via Wifi)
2010	Version 4.0	Bluetooth Low Energy
2013	Version 4.1	Weitere Low Energy Verbesserungen bezüglich Energieeffizienz und Netz-Topologie
2014	Version 4.2	Verbesserungen für die Sicherheit von Low Energy, Internet of Things
2016	Version 5.0	Eine Vielzahl von Verbesserungen, die die Bedürfnisse des Internet of Things adressieren

Tabelle 1: Überblick über die Bluetooth-Versionen [Jan16, Blu16a]

nur einige Verbesserungen, die keinen Einfluss auf die Funktionsweise der Sensorplattform haben. In der Version 4.2 gibt es mit Numeric Comparison unter anderem ein neues Verfahren, um Bluetooth-Geräte miteinander zu paaren. Zusätzlich nutzen die Pairing-Verfahren, die Authentifizierung bieten, einen Algorithmus, der eine höhere Sicherheit gewährleistet. Die neueste Version 5.0 wurde erst am 06.12.2016 veröffentlicht, jedoch gibt es für diesen Standard noch keine Hardware, die für die Sensorplattform geeignet ist.

Alle Bluetooth-Geräte besitzen eine 48-Bit lange Adresse, über die der Verbindungsauflaufbau erfolgt (z. B. B8:27:EB:8D:1E:8D), wobei es drei verschiedene Adressstypen gibt.

Bei der öffentlichen Bluetooth-Adresse (public) sind die ersten 24 Bit herstellerspezifisch. Das heißt, anhand dieser Werte lassen sich für eine bestimmte Hardware, Rückschlüsse auf den Hersteller ziehen. Die verbleibenden 24 Bit sind zufällig vergeben, aber in der Regel innerhalb einer Produktserie einzigartig. Der Adressstyp ist zur Laufzeit des Moduls unveränderlich. Die Bluetooth-Adresse 00:07:80:6A:F2:51 des BLE113 Entwicklungsboards lässt sich beispielsweise dem Unternehmen Bluegiga Technologies zuordnen.

Die statische Bluetooth-Adresse (static) ist ähnlich wie die öffentliche Adresse definiert, allerdings sind alle 48 Bit zufällig vergeben und besitzen somit keine Referenz zu bestimmten Herstellern. Meistens wird eine zufällige Adresse nach dem Start der Firmware erzeugt, sodass das Gerät z. B. nach einem Batteriewechsel eine neue Bluetooth-Adresse generiert.

Als dritten Typ gibt es zufällige Bluetooth-Adressen (random), die in einem festgelegten Zeitraum zufällig neu erzeugt werden (bspw. bei vielen Apple-Geräten alle 15 Minuten). Durch vorhergehendes Pairing können sich zwei Geräte identifizieren, auch wenn sich die Bluetooth-Adresse geändert hat. Dieser Adressstyp erschwert das Verfolgen eines bestimmten Bluetooth-Geräts anhand der Adresse. Im weiteren Verlauf des Kapitels wird noch mal auf das *Tracking* Bezug genommen.

Ab Bluetooth Version 4.0 (auch Bluetooth Low Energy oder Bluetooth Smart) ist die Funkübertragungstechnik besonders energieeffizient umgesetzt, allerdings verringert sich auch die Datenrate. Dieser Standard ist für Anwendungsszenarien konzipiert, bei denen einen kontinuierliche, energieeffiziente Übertragung mit niedriger Datenrate erforderlich ist. Die wichtigste Neuerung sind die Generic Attributes (GATT), die nicht mit den Adopted Profiles des vorherigen Bluetooth-Standards zu verwechseln sind. Ein Profil definiert eine hierarchische Datenstruktur, dessen Inhalte von verbundenen Geräten abrufbar sind [Bluc]. GATT basiert auf dem Attribute Protocol (ATT), dargestellt in Abbildung 5 und definiert somit eine Standardisierung, wie unterschiedliche Bluetooth-Geräte miteinander kommunizieren können. In diesem Kontext stellt die Sensorplattform den Client

dar und die Bluetooth-Sensoren sind die (GATT)-Server. Die Sensorplattform als Anwendung verwendet ausschließlich die GATT-Profile. Kenntnisse über die zugrunde liegenden Protokolle und Schichten aus Abbildung 5 sind nicht erforderlich. Damit eine Bluetooth-Verbindung bestimmte Sicherheitsfunktionen (z. B. Verschlüsselung und Authentifizierung) nutzen kann, ist außerdem der Security Manager erforderlich. Bei einigen in dieser Arbeit verwendeten Sensoren ist der Security Manager jedoch nicht Bestandteil des Bluetooth-Stacks der Firmware.

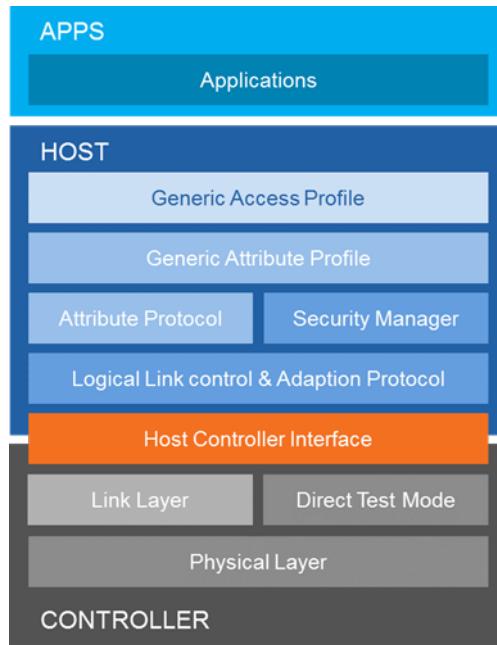


Abbildung 5: Der Bluetooth-Protokollstack [Blub]

Ein GATT-Profil beschreibt einen Use Case, die Benutzerrollen und das allgemeine Verhalten bezüglich der GATT Funktionalität. Services sind Sammlungen von Characteristics und Beziehungen zu anderen Services, die bestimmtes Verhalten für ein Gerät definieren. Ein Characteristic besteht aus den Datenfeldern Value, Descriptor und Properties, dargestellt in Abbildung 6. Über eine 16 Bit große Handle-Adresse lassen sich die einzelnen Datenfelder der Services und Descriptors auslesen oder beschreiben. Außerdem verfügt jeder Eintrag in diesem Service über eine UUID, die im Rahmen der Bluetooth-Spezifikation eindeutig einem Datenfeld zugeordnet werden kann. [Blub].

Die einfachste Möglichkeit an die Daten im GATT-Server des Sensor zu gelangen, ist das Senden einer Leseanfrage einer Characteristic. Ein Client (Sensorplattform) würde dann regelmäßig entsprechende Abfrage an den GATT-Server des Sensors senden und die Antwort empfangen. Besser ist es, Benachrichtigungen (Notifications) zu nutzen, um den Datentransfer so energieeffizient wie möglich zu gestalten. Eine Markierung (Flag) für Benachrichtigungen wird üblicherweise vom Client auf dem Server geschrieben. Jedes Mal wenn sich der Wert in einer Characteristic des GATT-Servers ändert, wird automatisch der neue Wert über die Bluetooth-Verbindung an den Client übertragen. Dies geschieht so lange, wie der Sensor aktiv ist oder bis der Client die Markierung wieder löscht. Auf diese Weise entfällt ein Teil der Kommunikation zwischen den Geräten, weil während der Datenaufzeichnung nur Daten vom Sensor zur Sensorplattform übertragen werden. Eine Bestätigung auf eine Notification erfolgt jedoch nicht, weshalb es möglich ist, dass diese Nachricht den Client nie erreicht. Eine Alternative sind Indications, die Empfangsbestätigung beinhalten. Indications werden aber nicht bei allen Characteristics unterstützt. Mit der Sensorplattform werden nur Sensoren verbunden, die lediglich Notifications ermöglichen. Manche

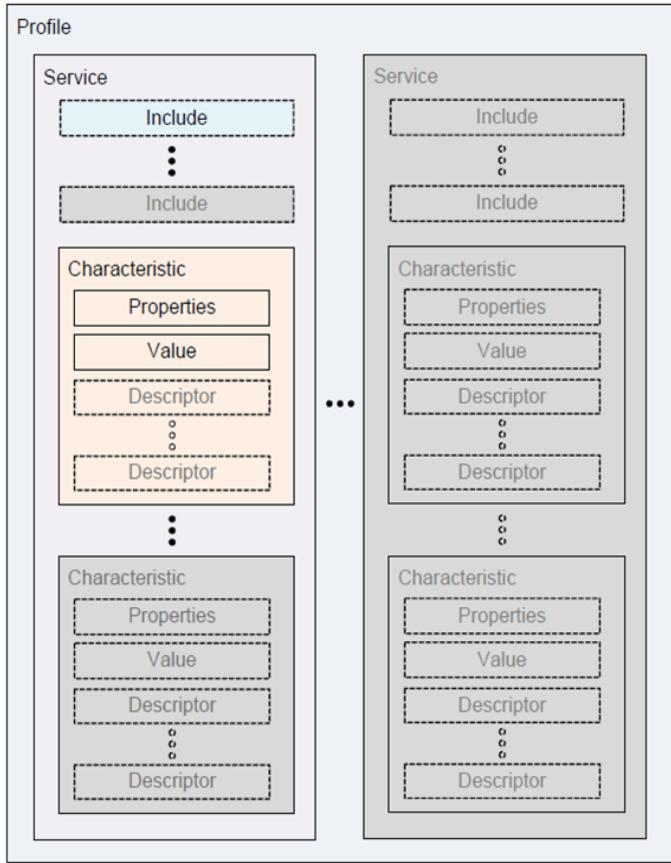


Abbildung 6: Hierarchie der GATT-Profile [Bluc]

Characteristics lassen sich gar nicht direkt auslesen, sondern der Datentransfer geschieht ausschließlich durch Notifications (z. B. durch die Herzfrequenz-Characteristic).

Der GATT-Server stellt für unterschiedliche Systeme eine Schnittstelle für die Anwendungsebene zur Verfügung. Die Signalaufbereitung und -verarbeitung des Sensors, bspw. das Filtern unerwünschter Frequenzen bei den Herzfrequenzsensoren, geschieht in dessen Firmware und ist von der Funktionalität der Sensorplattform vollständig abgeschirmt. Die Anwendung kann daher ohne Kenntnisse über die Hardware der Sensoren implementiert werden.

Abbildung 7 zeigt die Datenstruktur für den Herzfrequenz-Service. Dieser beinhaltet das Datenfeld für die eigentliche Herzfrequenz (Heart Rate Measurement) und beispielsweise Metadaten wie Body Sensor Location. Alle Informationen, die durch den GATT-Server bereitgestellt werden, müssen nicht verwendet werden. Für die Sensorplattform würde es ausreichen Heart Rate Measurement zu nutzen, während Body Sensor Location keinem Mehrwert zuzuordnen ist.

Hersteller können zudem auch individuelle Services erstellen. Dies wird insbesondere für Sensordaten, die noch nicht durch Services aus der Bluetooth-Spezifikation erfasst sind, gemacht. Um diese Daten korrekt zu interpretieren, ist dann die Dokumentation des Herstellers erforderlich. Der SensorTag CC2650 von Texas Instruments ist ein Beispiel für die Nutzung von gerätespezifischen Services (siehe Kapitel 5.1.3).

Im Allgemeinen gibt es eine Vielzahl von Angriffsmöglichkeiten durch Drittakteure auf Funkverbindungen, so auch bei Bluetooth Low Energy (BLE). Diese sind allgemein anwendbar und nicht spezifisch gegen die Sensorplattform gerichtet. Durch die Reichweite von ca. 10 Metern ist es sinnvoll, Vorkehrungen zu treffen, um mögliche Kompromittierung

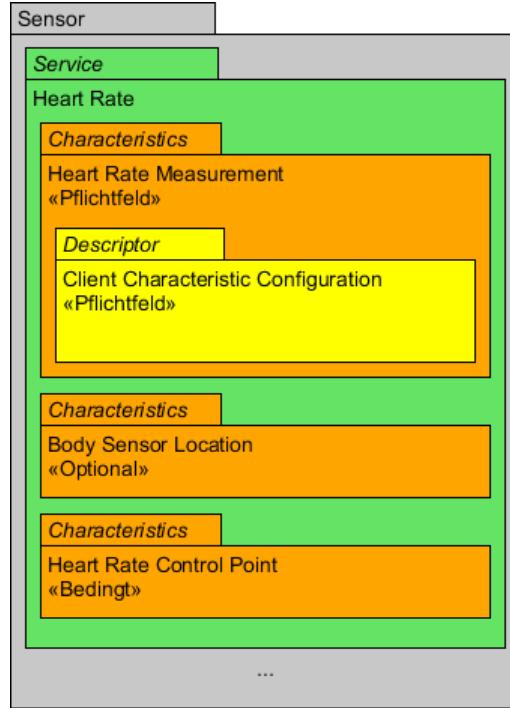


Abbildung 7: GATT-Service für die Herzfrequenz [Blud]

des Systems so schwer wie möglich zu machen. Im Kontext der Sensorplattform ist es auch vorstellbar, dass der Patient selbst Angriffe auf die Bluetooth-Kommunikation initiiert. Im Folgenden werden die Angriffe sowie adäquate Gegenmaßnahmen beschrieben.

Beim Lauschangriff befindet sich ein Angreifer in Reichweite des Patienten mit der Sensorplattform. Die Kommunikation zwischen dem Wearable Wireless Body/Personal Area Network (WWBAN) und der Sensorplattform lässt sich mit einem bluetoothfähigen Gerät und geeigneter Software aufzeichnen. Werden die Datenpakete korrekt interpretiert, lässt sich beispielsweise die Herzfrequenz des Patienten auslesen. Als Gegenmaßnahme wird das symmetrische Verschlüsselungsverfahren Advanced Encryption Standard (AES) eingesetzt. Daraus ergibt sich jedoch das Problem des Schlüsselaustauschs über einen ungesicherten Kanal, das aber Mittels Pairing gelöst werden kann. Im weiteren Verlauf dieses Kapitels wird das Pairing noch genauer betrachtet.

Der Man-In-The-Middle (MITM) ist ein klassischer Angriff, anwendbar bei fast allen Netzwerkverbindungen unterschiedlicher Standards. Der MITM verbündet sich zunächst mit dem Bluetooth-Sensor und gibt der Sensorplattform vor, er selbst sei dieser Sensor. Die Sensorplattform verbündet sich daraufhin nicht mit dem Sensor, sondern mit dem MITM, weil das Gerät diese beiden nicht unterscheiden kann. Der MITM befindet sich auf diese Weise in der Kommunikationsstrecke zwischen dem Sensor und der Sensorplattform und kann nach belieben Daten abfangen oder manipulieren. Um sicherzustellen, dass der Sensor, zu dem die Sensorplattform eine Verbindung aufbaut, tatsächlich ein Sensor des Systems ist, wird Authentifizierung benötigt. Dadurch lassen sich zu einem späteren Zeitpunkt ebenfalls einzelne Datenpakete einer Verbindung signieren. Bestimmte Pairing-Verfahren ermöglichen dies.

Ein weiterer Angriffsvektor ist die Verfolgung von Bluetooth-Geräten und deren Benutzern anhand der Bluetooth-Adresse. Dafür ist nur ein anderes Bluetooth-Gerät als Scanner erforderlich, von dessen Verwendung sich beispielsweise Bewegungsprofile von Personen ergeben. Durch den Einsatz mehrerer Bluetooth-Empfänger und besserer Antennen war es bereits möglich, Bluetooth-Geräte unter Laborbedingungen bis zu einer Reichweite von

zwei Kilometern zu orten [Bun11]. Es ist dabei unerheblich, ob die Bluetooth-Sichtbarkeit eines Geräts aktiviert oder deaktiviert ist. Als Gegenmaßnahme werden zufällige (random) Bluetooth-Adressen genutzt, die sich in festen Zeitintervallen ändern.

Die letzte Möglichkeit ist die Dienstblockade (engl. Denial of Service) eines Bluetooth-Geräts. Hat ein anderes Gerät bereits eine Verbindung mit einem Sensor hergestellt, wird die Sensorplattform nicht mehr in der Lage sein, sich mit diesem Sensor zu verbinden. Auf Grund der begrenzten Reichweite der Bluetooth-Verbindungen von nur ungefähr 10 Metern hat dieser Angriff in der Praxis geringe Bedeutung. Eine entsprechende Gegenmaßnahme stellt ebenfalls das Pairing dar.

Wie bereits erwähnt, können alle Daten einer bestehenden Verbindung zwischen zwei Geräten verschlüsselt werden. Dafür wird der Algorithmus AES-128 im CCM-Modus genutzt [Blu16c], der eine Erweiterung des AES für Verschlüsselung mit Authentifizierung darstellt. Dieses symmetrische Verschlüsselungsverfahren erfordert jedoch, dass beide Geräte den gleichen geheimen Schlüssel besitzen. Der Schlüsselaustausch muss zuvor über eine ungesicherte Verbindung erfolgen; ein klassisches Problem der Informationssicherheit. Bluetooth bietet durch Pairing bzw. Bonding verschiedene gute Lösungen unter anderem dafür an. Während Pairing für eine einmalige Verbindung zwischen zwei Geräten durchgeführt wird, bestimmt das Bonding eine langfristige gegenseitige Bekanntmachung zweier Geräte. Für zukünftige Verbindungen werden die gespeicherten Schlüssel des einmaligen Bonding-Vorgangs wiederverwendet. Im Kontext von Bluetooth Low Energy impliziert Pairing auch immer Bonding. Häufig werden daher beide Begriffe äquivalent zueinander genutzt, so auch im weiteren Verlauf dieser Arbeit. Die Sensorplattform muss demnach einmalig mit einem Sensor gepaart werden, um alle Sicherheitsfunktionen in dieser Verbindung nutzen zu können.

Für das Pairing wird in der Literatur der Begriff *Secure Simple Pairing (SSP)* verwendet (Bluetooth-Versionen 4.0 und 4.1). Ab Bluetooth-Version 4.2 wird Secure Simple Pairing zu *Legacy Pairing* umbenannt. Ab Bluetooth 4.2 heißen die Pairing Verfahren *LE Secure Connections* und nutzen zum Schlüsselaustausch einen Algorithmus, der auf elliptischen Kurven von Diffie-Hellman basiert. Die Intention der Bluetooth SIG ist es, Algorithmen zu verwenden, die entweder NIST-konform oder durch FIPS anerkannt sind, um ein höheres Maß an Sicherheit zu bieten. Das National Institute of Standards and Technology (NIST) empfiehlt auf Basis von Testprogrammen unter anderem bestimmte Kryptoalgorithmen [Nat96]. Für die Validierung von kryptographischen Modulen ist insbesondere FIPS 140-2 - nicht nur in den USA, sondern auch international - von großer Bedeutung. [Blu16c, Nat01].

Zunächst werden die wichtigsten Schlüssel betrachtet, die für unterschiedliche Sicherheitsfunktionen von Bluetooth generiert und verwendet werden.

- Der *Short Term Key* ist ein temporärer Schlüssel, der zum Aushandeln der nachfolgenden Schlüssel genutzt wird und danach keine Verwendung mehr findet.
- Der *Long Term Key* ist der Schlüssel für den AES-Algorithmus im CCM-Modus.
- Für die Signatur von Bluetooth-Paketen wird der *Local/ Remote Signature Key* genutzt.
- Damit sich Geräte wiedererkennen können, die die Bluetooth-Adresse des Typs random nutzen, wird der *Connection Signature Resolving Key* genutzt.

Diese Schlüssel werden bei jedem erfolgreichen Pairing generiert und gespeichert. Jedoch resultieren verschiedene Pairing-Verfahren in unterschiedlichen Kombinationen von Sicherheitsfunktionen: Verschlüsselung, Authentifizierung und MITM-Schutz. Welche Pairing-Methode konkret zum Einsatz kommt, wird hauptsächlich durch die Eingabe/Ausgabemöglichkeiten der Geräte bestimmt. Diese Beschreibung dafür nimmt ein sogenannter

Agent vor, der z. B. KeyboardDisplay heißen kann. Bei den Eingabe/Ausgabemöglichkeiten handelt es sich um Bildschirme, Tastaturen und Ja/Nein-Knöpfe auf den Geräten. Der Agent muss vor dem Pairing geladen werden. Eine vollständige tabellarische Auflistung aller Kombinationen stellt die Bluetooth SIG bereit [Blu16c]. Es werden aber nur die beiden Verfahren Just Works und Passkey Entry betrachtet, die auch in der Pulsoximeter-Firmware des BLE113 Development Boards implementiert sind (siehe Kapitel 6.10.2).

Das einfachste Pairing-Verfahren ist Just Works, weil es keine Interaktion mit dem Benutzer erfordert. Der Short Term Key ist dabei immer auf den Wert 0 festgelegt. Daraus resultiert ein geheimer Long Term Key für die Verschlüsselung (und auch die anderen Schlüssel werden generiert), aber der Sensor wäre gegenüber der Sensorplattform nicht authentifiziert und es existiert schon gar kein MITM-Schutz. Wird der Pairing-Vorgang von einem Angreifer aufgezeichnet, ist es verhältnismäßig einfach, daraus weitere kryptographische Schlüssel zu ermitteln.

Das Verfahren Passkey Entry ist ein komplizierteres Pairing-Verfahren, da es Interaktion vom Benutzer erfordert. Außerdem lässt es sich auf geringfügig unterschiedliche Weisen umsetzen. Entweder muss auf beiden Geräten der gleiche PIN eingegeben werden, oder auf dem einen Gerät wird der PIN angezeigt, und auf dem anderen eingetippt. Um die Hardware-Anforderungen auf Seite des Sensors gering zu halten, wird ein PIN auf dem Display des Sensors dargestellt, der in der Sensorplattform eingegeben werden muss. Knöpfe oder eine Tastatur sind demzufolge an dem Gerät (noch) nicht erforderlich. Ist die Eingabe korrekt und der PIN demnach auf beiden Geräten gleich, ist mit sehr hoher Sicherheit gewährleistet, dass es sich (aus Sicht der Sensorplattform) tatsächlich um den Sensor handelt. Damit bietet das Passkey Entry-Pairing neben der Verschlüsselung auch Authentifizierung und MITM-Schutz. Ein wesentlicher Teil der Sicherheit bei Bluetooth Low Energy (BLE) ist deshalb durch die Eingabe-Ausgabe-Möglichkeiten der Geräte bestimmt. Die korrekte Benutzung der Pairing-Verfahren liegt außerdem im Verantwortungsbereich des Benutzers.

Die Bluetooth-Spezifikation definiert bestimmte Sicherheitsmodi, die sich wiederum in unterschiedliche Stufen einteilen lassen, wie Tabelle 2 zeigt. Diese Einteilung bestimmt, welche Sicherheitsfunktionen bei einer Bluetooth-Verbindung vorhanden, bzw. welche damit nicht vorhanden sind. Je nach Pairing-Verfahren ergeben sich verschiedene Bluetooth Sicherheitsmodi bzw. Stufen. Aus der Pairing-Methode Just Works resultiert der Low Energy Sicherheitsmodus 1 Stufe 2 und durch Passkey Entry wird Modus 1 Stufe 3 realisiert (die höchste gemeinsame Bluetooth-Version der Geräte ist 4.0). Die Verwendung von Hardware, die auch Bluetooth 4.2 unterstützt, würde durch Passkey Entry den Modus 1 Stufe 4 ergeben. Die Änderungen am Schlüsselaustausch passieren aber intern im Bluetooth-Stack BlueZ und würden die Anwendung der Sensorplattform nicht betreffen. [Nat12].

Sicherheitsstufe	Verschlüsselung	Authentifizierung	MITM-Schutz
LE Sicherheitsmodus 1 Stufe 1	Nein	Nein	Nein
LE Sicherheitsmodus 1 Stufe 2	Ja	Nein	Nein
LE Sicherheitsmodus 1 Stufe 3	Ja	Ja	Ja
LE Sicherheitsmodus 1 Stufe 4 (Bluetooth 4.2)	Ja	Ja	Ja
LE Sicherheitsmodus 2 Stufe 1	Nein	Ja	Nein
LE Sicherheitsmodus 2 Stufe 2	Nein	Ja	Ja

Tabelle 2: Bluetooth-Sicherheitsstufen mit den daraus resultierenden Schutzfunktionen [Nat12]

Die Verschlüsselung ist erforderlich, um die Bluetooth-Kommunikation zwischen zwei

Geräten gegen passives Abhören abzusichern. Durch die Authentifizierung (mit MITM-Schutz) ist gewährleistet, dass es sich tatsächlich um das Gerät und nicht um einen aktiven Angreifer handelt. Aus Tabelle 2 wird damit deutlich, dass es sich bei Low Energy Sicherheitsmodus 1 Stufe 3 und 4 um die sichersten Betriebsarten handelt, da sowohl Verschlüsselung als auch Authentifizierung mit MITM-Schutz vorhanden ist. Auch das NIST spricht eine Empfehlung für die Nutzung von Modus 1 Stufe 3 aus. Stufe 4 kommt in der NIST-Publikation nicht vor, da der Bluetooth Standard 4.2 erst später veröffentlicht wurde. Weitergehend empfiehlt das NIST auf Modus 1 Stufe 1 komplett zu verzichten, weil eine Verbindung, die weder Verschlüsselung noch Authentifizierung nutzt, ungeschützt ist. Der Modus 2 unterstützt Authentifizierung durch Signatur von Bluetooth-Paketen mit oder ohne MITM-Schutz. Verschlüsselung wird dabei nicht verwendet. Der Modus 1 ist gegenüber Modus 2 immer zu bevorzugen, weil in Modus 2 kein Schutz gegen das Abhören von Bluetooth-Paketen vorhanden ist. Authentifizierung und MITM-Schutz sind separat für den LE Sicherheitsmodus 2 aufgelistet. Dabei ist es möglich, Pakete mit einem Schlüssel zu signieren, der durch das Just-Works-Verfahren erzeugt wurde (LE Sicherheitsmodus 2 Stufe 1). [Nat12].

Während das NIST konkrete Empfehlungen für und gegen bestimmte Modi ausspricht, bietet das Bundesamt für Sicherheit in der Informationstechnik (BSI) lediglich allgemeinere Hinweise bezüglich Bluetooth-Sicherheit: Bluetooth-Geräte sollen bspw. nur in abhörsicherer Umgebung gepaart werden. Das BSI differenziert auch nicht zwischen Bluetooth Classic und Bluetooth Low Energy (BLE). Für die Konzeption der Sensorplattform werden daher hauptsächlich die Informationen des NIST verwendet. Ursprünglich definiert das NIST nur für die USA geltende Standards. Allerdings hat es insbesondere im Bereich Informationsicherheit auch international große Bedeutung, so auch bspw. das Validierungsprogramm FIPS 140-2 für kryptographische Module. [Bun11].

Die für die Sensorplattform genutzten Herzfrequenzsensoren Polar H7, Adidas und TomTom sowie der CC2650 nutzen ausschließlich Bluetooth Modus 1 Stufe 1 (siehe Kapitel 5.1) Sicherheitsfunktionen werden von deren Firmware nicht unterstützt (der CC2650 bietet speziell für Windows ein Pseudo-Pairing an, da Pairing bei neueren Windows-Systemen zwingend erforderlich ist). Wie bereits erwähnt, wird für das BLE113 Entwicklungsboard das Passkey Entry-Verfahren angewendet, damit die Sensorplattform zumindest mit einem Sensor mit allen Sicherheitsfunktionen getestet werden kann.

Aus der Anforderungsanalyse in Kapitel 4 geht außerdem hervor, dass Linux als Betriebssystem für die Sensorplattform genutzt wird. Der offizielle Bluetooth Stack für Linux heißt BlueZ. Seit der Hauptversion 5 werden auch GATT-Profilen von Bluetooth Low Energy (BLE) unterstützt. Für die Sensorplattform kommt die im Oktober 2016 aktuelle Version 5.43 zum Einsatz. Das BlueZ-Projekt ist in Kernel-Space und User-Space unterteilt. Für Geräte mit Bluetooth-Chip wird der User-Space nachinstalliert, der Kernel-Space ist bereits standardmäßig integriert. Das Projekt ist nicht abgeschlossen, sondern befindet sich kontinuierlich in Entwicklung, weshalb es sinnvoll ist, immer die aktuellste Version von BlueZ zu nutzen, um Bugfixes usw. zu erhalten. Das Betriebssystem der Sensorplattform, Raspbian, wird mit BlueZ 5.23 verteilt. Eine manuelle Aktualisierung des Bluetooth Treibers ist somit erforderlich. Um alle BLE-Funktionen von BlueZ nutzen zu können, wird mindestens der Linux Kernel mit Version 3.5 benötigt [Hed12]. Diese Anforderung ist durch Raspbian erfüllt. Für einige alte Kernelversionen gibt es auch Backports von BlueZ und für bestimmte Android-Versionen gibt es eine vollständige Ersetzung des Originaltreibers Bluedroid. [Jan16].

Die Bluetooth-Spezifikation als offene Schnittstelle eignet sich damit als Grundlage für die Sensorplattform. Durch die Verwendung von GATT-Profilen können unterschiedliche Sensoren und Geräte einheitlich miteinander kommunizieren. Das nachfolgende Kapitel bietet

zur Vollständigkeit einen kompakten Überblick über weitere Funkübertragungstechniken, die bei mobilen Gesundheitsüberwachungssystemen verwendet werden.

3.1.5. Weitere Funkübertragungstechniken

Über Bluetooth Low Energy (BLE) hinaus gibt es weitere Funkübertragungstechniken, die ähnliche technische Eigenschaften aufweisen, bspw. Ant oder Zigbee. Diese eignen sich ebenfalls besonders für kurze Distanzen und damit für ein Wearable Wireless Body-/Personal Area Network (WWBAN). In einigen mobilen Gesundheitsüberwachungssystemen finden diese Techniken bereits Anwendung, sodass eine zukünftige Erweiterung der Sensorplattform für entsprechende Schnittstellen naheliegend ist. Im Folgenden soll ein kurzer Überblick über die alternativen Funkübertragungstechniken gegeben werden.

Ant bzw. Ant+ ist ein proprietärer Funkübertragungsstandard, dessen Ursprünge in das Jahr 2003 zurückreichen [Dyna]. Genau wie bei BLE ist Ant besonders energieeffizient. Ant nutzt die Frequenzen 2400 MHz bis 2524 MHz mit Ausnahme der Frequenz 2457 MHz, die für Ant+ Geräte reserviert ist [Dynb]. Die Reichweite beträgt bis zu 30 Metern. Ant+ definiert eine Vielzahl von Profilen für bestimmte Anwendungen, bspw. *Heart Rate Monitor* oder *Blood Pressure*, durch die verschiedene Ant+ Geräte miteinander kommunizieren können [Dync]. Ant+ ist dabei besonders auf Anwendungen mit Sensoren spezialisiert. Üblicherweise wird die Hardware für Ant+ als USB-Stick verkauft, sodass Einplatinenrechner mit integrierten Ant+ Chips selten sind. Die Ähnlichkeiten zu Bluetooth Low Energy (BLE) bestehen, aber noch ist nicht absehbar, ob sich dieser Standard langfristig auf dem Markt behaupten wird. Auch deswegen ist die Implementierung für Ant+ nicht Teil dieser Arbeit, aber es besteht die Option, die Sensorplattform in Zukunft für diese Technik zu erweitern.

Der Zigbee Standard bietet ebenfalls einen energieeffizienten Funkübertragungsstandard der häufig bei Hausautomationen und bei Sensornetzwerken verwendet wird. Im Batteriebetrieb kann ein entsprechendes Gerät mehrere Monate oder Jahre betrieben werden. Dabei sind flexible Netztopologien wie zum Beispiel ein Stern, ein Baum oder verschiedene Maschen möglich. Zigbee nutzt 16 Kanäle des 2,4 GHz ISM-Bands. Die Reichweite beträgt üblicherweise bis 75 Meter. Genau wie Bluetooth unterstützt Zigbee Sicherheitsfunktionen für Verschlüsselung (AES) und Authentifizierung. Für den Einsatz am menschlichen Körper kann mit verschiedenen Zigbee-Sensoren ein Body Area Network (BAN) erstellt werden, über das anschließend die Daten zu einer zentralen Einheit (die Sensorplattform) übertragen werden. [PG10].

Zigbee und Ant sind für die Kommunikation von unterschiedlichen Geräten über geringe Entfernung geeignet. Für die Sensorplattform wird aber auch eine Internetverbindung durch Mobilfunk erforderlich. Schließlich soll der Patient nicht lokal durch die Reichweite eines möglichen Zugangspunkts wie bspw. bei WLAN beschränkt sein. Genau wie ein Smartphone ist die Sensorplattform direkt mit dem Mobilfunknetz verbunden.

Die technischen Möglichkeiten für die Umsetzung der Mobilfunkverbindung sind begrenzt. Häufig werden Module mit USB- und M.2- bzw. PCI-Express Anschlüssen genutzt. An die Sensorplattform ist ein Surfstick an einem USB-Port angeschlossen (siehe Kapitel 5.1.1). Die Tests in Kapitel 7.3.3 stellen heraus, dass das verwendete Mobilfunkmodul einen wesentlichen Einfluss auf die Energieeffizienz der Sensorplattform hat. Daher werden einige Parameter erläutert und in der Software genutzt, um das Hochladen von Daten durch die Sensorplattform zu regulieren. Diese Parameter ermöglichen Rückschlüsse auf die Signalqualität der Mobilfunkverbindung.

Abbildung 8 zeigt die Systemarchitektur von Universal Mobile Telecommunications System (UMTS) und Global System for Mobile Communications (GSM). Die UMTS-Infrastruktur baut zunächst auf dem Vermittlungsteilsystem (Core Network) auf, das bereits

von den GSM-Funknetzen vorhanden ist. Erst bei weiteren Ausbaustufen von UMTS werden auch Änderungen am Core Network vorgenommen, die für die Sensorplattform jedoch nicht von Bedeutung sind. Die relevanten UMTS-Komponenten sind als UMTS Terrestrial Radio Access Network (UTRAN) zusammengefasst. Ein Node B versorgt in dessen Reichweite eine Vielzahl von mobilen Geräten mit einer Funkverbindung. Die Sensorplattform stellt in Abbildung 8 das Mobile Equipment (ME) dar. Die im Folgenden vorgestellten Parameter beziehen sich auf die Kommunikationsstrecke zwischen mobilem Endgerät und der UMTS-Basisstation (Node B). [Lü01].

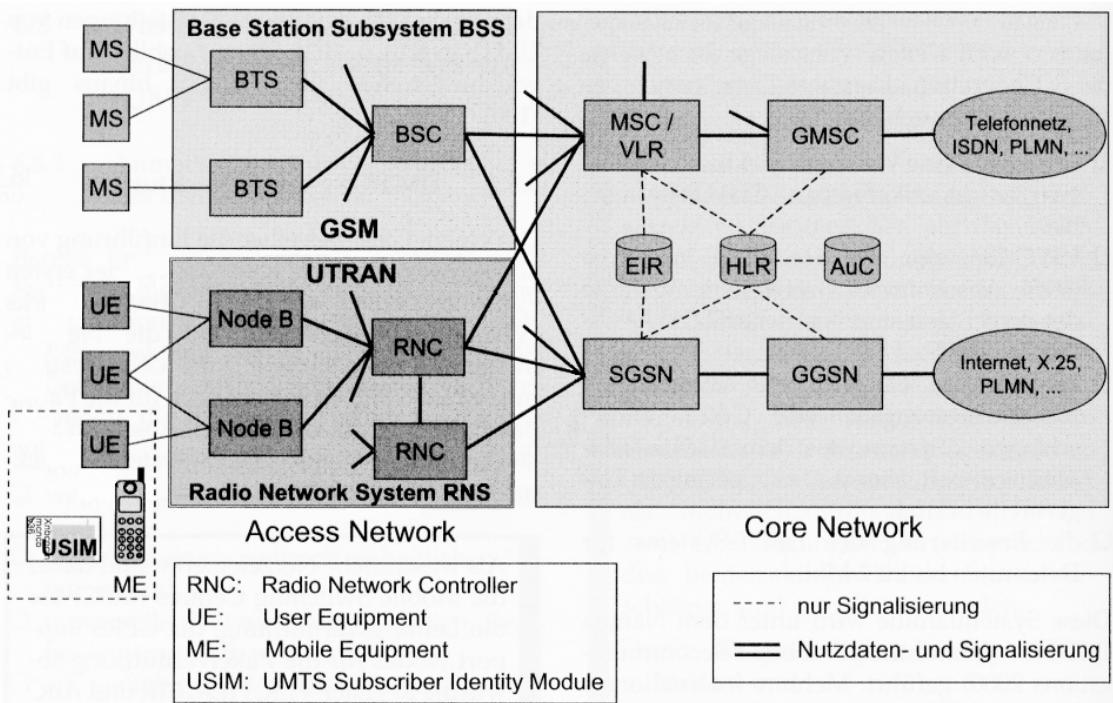


Abbildung 8: Die UMTS-Systemarchitektur [Lü01]

Die Sensorplattform nutzt drei Parameter zur Bestimmung der Signalqualität der Mobilfunkverbindung. Diese heißen Received Signal Strength Indicator (RSSI), Received Signal Code Power (RSCP) und E_c/I_o . Jede UMTS-Basisstation besitzt einen Downlink-Kanal mit einer festen Sendeleistung und Bitrate für alle Mobilgeräte, der Common Pilot Channel (CPICH) genannt wird. Die vom Endgerät empfangene Signalleistung dieses Kanals heißt Received Signal Code Power (RSCP). Das mobile Endgerät nutzt unter anderem diesen Wert für die Auswahl eines Node Bs für die Mobilfunkverbindung. Diese Kenngröße beinhaltet jedoch noch keine Informationen über die Interferenz von anderen Basisstationen. Der Parameter E_c/I_o bezeichnet das Verhältnis zwischen Energie pro Chip und der Interferenz bzw. des Rauschens, falls die Interferenz gleich Null ist (Signal-Rausch-Verhältnis). Durch eine Folge von Chips wird ein Nutzdatenbit vor der Übertragung gespreizt (siehe Bandspreizverfahren z. B. CDMA). Die dafür notwendigen Spreizcodes (Chipsequenzen) unterscheiden sich für jede einzelne Verbindung. Auf diese Weise ist es möglich, mehrere Datenverbindungen auf der gleichen Übertragungsfrequenz zu betreiben und anschließend die Nutzdaten durch Entsprenzen wiederherzustellen. Abbildung 9 zeigt die Relation zwischen RSCP und E_c/I_o (bzw. E_c/N_o) bei unterschiedlicher Last im Netz und Anzahl von Basisstationen. Verschlechtert sich der Wert für E_c/I_o , ist dies ein Hinweis auf erhöhte Aktivität von anderen Node Bs. Der Wert RSCP bleibt dann allerdings unverändert. Der Parameter Received Signal Strength Indicator (RSSI) bezeichnet die gesamte empfangene Signalleistung, der allerdings kein geeigneter Indikator ist, um

die Signalqualität abzuschätzen. Zum Beispiel können 10 schwache Basisstationen einen hohen RSSI-Wert ergeben, die tatsächliche Datenrate ist jedoch trotzdem sehr gering, weil es keine dominante Basisstation gibt, wodurch das Signal-zu-Interferenz-Verhältnis dann entsprechend schlecht ist. Dieser Effekt heißt Pilot Pollution und wird bei der Planung von UMTS-Netzen nach Möglichkeit vermieden. [Qua06].

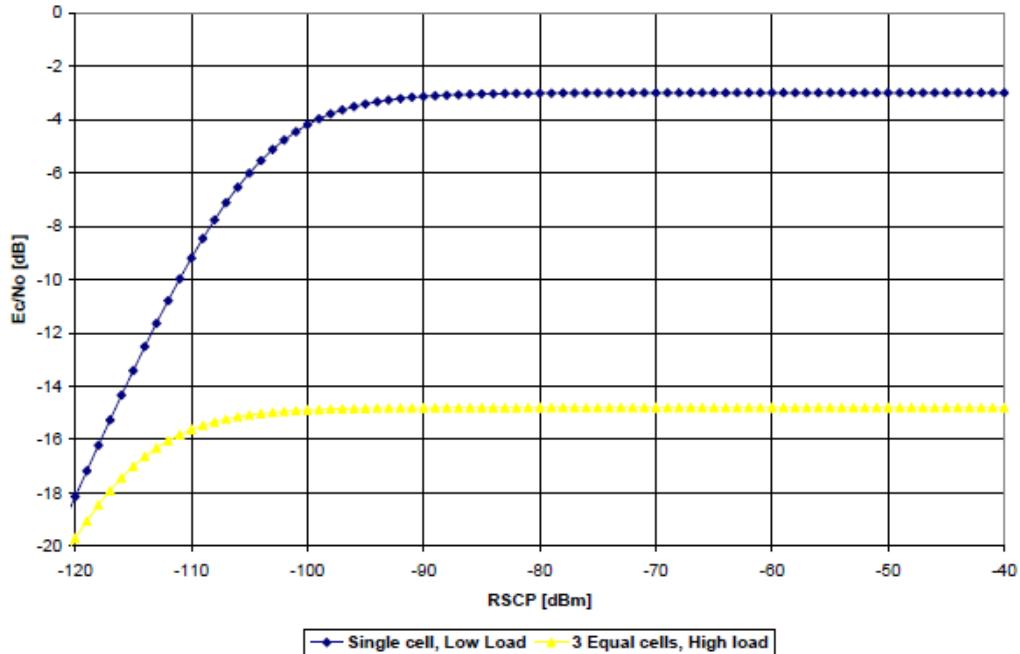


Abbildung 9: Relation zwischen RSCP und E_c/I_o [Qua06]

Die Formel für RSSI ergibt sich aus den anderen zwei Parametern wie folgt: $RSSI[dBm] = RSCP[dBm] - E_c/I_o[dB]$. Das Mobilfunkmodul der Sensorplattform, das in Kapitel 5.1.1 vorgestellt wird, gibt automatisch und regelmäßig den Parameter RSSI an dessen serieller Schnittstelle aus. [Ele07].

In Kapitel 6.6 ist erklärt, wie die Anwendung die Werte für RSSI, RSCP und E_c/I_o erhält. Die Tests in Kapitel 7.3.3 setzen die Signalqualität der Mobilfunkverbindung in einen Zusammenhang zum Stromverbrauch der Mobilfunkmodems. In einer Iterationsphase wird die Anwendung entsprechend angepasst, die dann die Hochladefunktion von Samples an einen Webserver durch die Auswertung der Signalqualität reguliert.

3.1.6. Programmierung eingebetteter Systeme

Aus der Anforderungsanalyse in Kapitel 4 geht hervor, dass die Sensorplattform eine Linux-Distribution als Betriebssystem nutzt und die Implementierung in der Programmiersprache Java erfolgt. Für die Softwareentwicklung für eingebettete Systeme gibt es jedoch noch einen anderen Ansatz. Die Anwendung kann auch direkt auf einem Mikrocontroller ausgeführt werden. Die wesentlichen Unterschiede sind in Tabelle 3 aufgelistet. Die Spalte Mikrocontroller bezeichnet die direkte Hardwareprogrammierung ohne den Gebrauch eines Real-time Operating System (RTOS). Die Spalte Betriebssystem stellt die Verwendung eines Single-Board-Computers mit einer Linux-Distribution dar.

Da es sich bei der Sensorplattform um einen Demonstrator handelt, ist die Portierbarkeit der Software das wichtigste Kriterium. Schließlich soll die Anwendung für zukünftige Projekte, mit gegebenenfalls anderer Hardware, genutzt werden. Die Sensorplattform kann auf jeden anderen Single-Board-Computer mit einem Linux-Betriebssystem und BlueZ als Bluetooth-Stack und Java portiert werden. Ebenfalls ist es damit möglich, einen Webserver mit einem komplexen Frontend für die Mensch-Maschine-Interaktion auf der Sensorplatt-

	Mikrocontroller	Betriebssystem
Echtzeitfähig	Ja	Nein
Rechenleistung und Speicherausstattung	Gering, für eine Anwendung auch nicht erforderlich	Hoch, erforderlich für Betriebssystem und unterschiedliche Funktionen
Programmiersprachen	C oder C++	Verschiedene, z. B. Java
Energieeffizienz	Hoch	Gering
Portierbarkeit	Schwierig, da eine enge Verknüpfung mit dem Hardware-Layout besteht	Gut für jeden Single-Board-Computer mit einer Linux-Distribution mit BlueZ und Java
Softwarefunktionen	Viele Funktionen müssen im Bedarfsfall in der Anwendung implementiert sein	Viele Funktionen sind durch das Betriebssystem bereits vorhanden, z. B. Scheduling für Prozesse und Threads oder Interprozesskommunikation

Tabelle 3: Softwarearchitekturen eingebetteter Systeme im Vergleich

form bereitzustellen. Die Echtzeitfähigkeit hat im Kontext von mobilen Gesundheitsüberwachungssystemen eine geringe Bedeutung, da es durch die Verwendung der Mobilfunkverbindung sowieso zu Latenzen im Gesamtsystem kommt. Eine Reaktionszeit des Systems innerhalb einiger Sekunden ist ausreichend. Die geringere Energieeffizienz bei dieser Lösung ist jedoch problematisch, wie sich in den Tests in Kapitel 7.3 herausstellt.

Die Darstellung der technischen Grundlagen für die Sensorplattform ist damit abgeschlossen. Es ist deutlich geworden, dass es bereits eine Vielzahl von Sensoren auf dem Verbrauchermarkt gibt, die sich mit der Sensorplattform kombinieren lassen. Die Schnittstellen zum WWAN werden durch die Funkübertragungsstandards Bluetooth Low Energy (BLE), Ant bzw. Ant+ oder Zigbee realisiert. Die Kommunikation zwischen Sensorplattform und dem Webserver in der Cloud geschieht über eine Mobilfunkverbindung und die Implementierung erfolgt auf einem Einplatinenrechner mit einem Linux-Betriebssystem.

3.2. Stand der Forschung

In diesem Kapitel werden Forschungsprojekte zu mobilen Gesundheitsüberwachungssystemen vorgestellt. Die Auswahl umfasst Systeme, die für sehr unterschiedliche Krankheitsbilder eingesetzt werden. Dies verdeutlicht die Wichtigkeit des Kriteriums einer flexiblen Konfiguration und Erweiterbarkeit der Sensorplattform. Dazu werden technische Möglichkeiten genannt, wie sich die Sensorplattform für einige dieser speziellen Anwendungen adaptieren ließe. Ebenfalls ergibt sich dadurch ein Ausblick auf angrenzende Forschungsbereiche wie beispielsweise *Middlewares* für Sensoren und *Home Hospitalization*.

In dem Forschungsprojekt MAS des Fraunhofer FIT ist die Grundidee einer Sensorplattform entstanden. Das Ziel dieses Forschungsprojekts ist die Entwicklung eines Ambient Assisted Living (AAL)-Assistenz-System, mit dem kardiovaskuläre Krankheiten überwacht werden sollen. Durch Früherkennung und spezifische Reaktion kann bspw. der Verlauf eines Herzinfakts abgemildert oder sogar verhindert werden. Bisher gibt es jedoch nur wenige eindeutige Parameter für die Vorhersage bestimmter Krankheitsausbrüche. Für die Diagnostik ist die Kombination von unterschiedlichen Messwerten erforderlich, die in einer mobilen Anwendung (App) auf dem Smartphone gesammelt werden. Absehbar ist aber auch die Verwendung einer Sensorplattform, um bspw. das Assistenzsystem kostengünstig zu erweitern und zu erproben. [Fraa].

Im Rahmen dieses Projekts werden die Vitalparameter Sauerstoffsättigung im Blut, Puls und Herzrhythmus-Werte erfasst. Dafür werden auch Fluoreszenz- und elektrochemische Sensoren zur Messung von Herzkreislaufmarkern eingesetzt. Anstatt der Sensorplattform wird jedoch ein Smartphone inklusive App verwendet. Die Datenübertragung zwischen Sensor und Smartphone erfolgt über eine Bluetooth-Funkverbindung und die Daten werden zentral zu einem Medizinzentrum hochgeladen und analysiert. Darüber hinaus sind auch Benachrichtigungen zwischen Patient und Arzt berücksichtigt, bspw. als Reaktion auf kritische Messwerte. [Fraa].

Das zweite Projekt des Fraunhofer FIT heißt Polycare. „POLYCARE (POLY-stakeholders integrated CARE for chronic patients in acute phases) ist ein von der Europäischen Kommission im Forschungs- und Innovationsprogramm Horizont 2020 gefördertes Projekt. Ziel von POLYCARE ist die Entwicklung und Erprobung einer integrierten, patientenzentrierten Pflege-Lösung, die mit Hilfe von modernen IKT-Systemen und Services die Betreuung und das Monitoring älterer chronisch kranker Patienten zu Hause auch in akuten Phasen ermöglichen soll.“ [Frab].

Bei diesem Projekt soll eine Vielzahl von Zielen erreicht werden. Das wichtigste Ziel ist die Entwicklung eines integrierten Versorgungssystems, für die häusliche gesundheitliche Versorgung von chronisch erkrankten Patienten. Die Anwendung soll dabei für den Patienten personalisierbar sein, um eine individuelle Versorgung zu ermöglichen. Das System erfasst kontinuierlich Daten aller wichtigen Vitalparameter und schränkt den Patienten gar nicht oder nur geringfügig ein. Außerdem kann der Patient diese Werte einsehen. Auch die Zusammenarbeit der beteiligten Akteure (Patienten, Angehörige, Freunde, medizinisches Personal) gilt es mit Polycare zu verbessern. Durch geeignete Ausbildungsprogramme für das Personal sollen Barrieren bei der Annahme der technischen Lösung gering gehalten werden. Datenschutz und Privatsphäre sind zu gewährleisten, da es sich bei Vitalparametern um sensible Daten handelt. Mit Hilfe von maschinellem Lernen sollen genauere Datenanalysen umgesetzt werden, um die gesundheitliche Versorgung bei unerwarteten Problemen zu verbessern. Letztlich ist auch der Kostenfaktor von Bedeutung. Mit Polycare soll der stationäre Aufenthalt von Patienten im Krankenhaus verkürzt und damit Kosten eingespart werden. [Frab].

Die Aufgabe des Fraunhofer FIT ist die Erstellung einer Middleware, die die Sensoren mit den verschiedenen Komponenten und Diensten des verteilten Systems verknüpft. Durch offene Schnittstellen am Polycare-System soll es möglich sein, verschiedene andere Geräte und Systeme zu integrieren. Außerdem entwickelt das Institut algorithmische Komponenten für das POLYCARE-System. [Frab].

Die Ziele und Voraussetzungen, um diese technisch umzusetzen, überschneiden sich in vielen Punkten mit dem Konzept der Sensorplattform (siehe Anforderungsanalyse in Kapitel 4). Neu ist aber der Begriff der Middleware im Kontext von Sensoren im Internet-of-Things (IoT). Das grundsätzliche Konzept dieser Middleware ist es, unterschiedliche Hardwaresensoren einheitlich in Form einer definierten Programmierschnittstelle anzusteuern. Middlewares bieten mit dieser Abstraktion das Potential bei der Entwicklung viel Zeit zu sparen, da sich Anwendungsentwickler nicht mehr mit hardwarespezifischen Details auseinandersetzen müssen [KFR14]. Es wäre möglich, die Schnittstelle der Sensorplattform zu den Sensoren separat als Middleware zu implementieren. Die eigentliche Anwendung würde dann nur über definierte Schnittstellen mit der Middleware kommunizieren. Eine Möglichkeit für die einheitliche Ansteuerung von Bluetooth Low Energy (BLE) Geräten ist bspw. das Framework Web-Bluetooth bzw. die Generic Sensor API des World Wide Web Consortium (W3C) [Worb, Wora]. Es gibt jedoch Gründe, derartige Frameworks im Rahmen dieser Arbeit nicht zu verwenden (siehe Kapitel 6.1).

Wirtschaftlich betrachtet ist es vorteilhaft, wenn neue Systeme wie bspw. bei Polycare, die

alten Geräte nicht ersetzen, sondern diese ebenfalls für die Nutzung in das neue System integriert werden können [Frab]. Dafür sind offene und spezifizierte Schnittstellen notwendig. Neue Innovationen lassen sich dann auch initial kompatibel zu bestehenden Systemen konzipieren. Auf Grund vieler verschiedener Sensoren, Geräte und Hersteller auf dem Markt ist diese Standardisierung alles andere als trivial, durch die stetige Entwicklung des Internet-of-Things (IoT) jedoch von zunehmender Bedeutung. Sensor-Middlewares stellen mittlerweile einen eigenen Forschungsbereich dar, der im Folgenden zusammengefasst wird.

Ein Wireless Sensor Network (WSN) ist in der Regel für eine bestimmte Anwendung konzipiert. Die Menge aller Anwendungen, in denen ein WSN eingesetzt werden kann, ist jedoch insbesondere durch die Weiterentwicklung des Internet-of-Things (IoT) sehr groß. Eine Middleware ist die Software-Abstraktionsstufe für Gerätehardware: hardwarespezifische Details sind dadurch vor dem Softwareentwickler versteckt. Die Herausforderung bei einer Middleware ist die Entkoppelung von Netzwerken, Hardware und Software. Aus der Perspektive der Anwendung soll es keinen Unterschied machen, ob diese mit der Middleware oder direkt mit einem Gerät interagiert. Weitere Aspekte wie beispielsweise Energieeffizienz und Informationssicherheit müssen ebenfalls durch die Middleware berücksichtigt werden. [KFR14].

Der Einsatz einer Middleware wäre auch bei der Sensorplattform möglich. Abbildung 10 zeigt das Modell einer Middleware. Die Software der Sensorplattform würde der Anwendungsschicht angehören und interagiert mit der Middleware anstatt direkt mit den Sensoren. Diese stellt verschiedene Dienste zur Verfügung, bspw. die Suche nach Geräten und natürlich die Messdaten. Nur die Middleware kommuniziert direkt mit den Hardwaresensoren. Bei der Sensorplattform ist die Schnittstelle zu den Sensoren zunächst auf Bluetooth Low Energy (BLE) beschränkt. Die Middleware kann auch unbekannte Geräte erkennen, um diese auf Basis bestimmter Eigenschaften (z. B. GATT-Profile) zu klassifizieren. Eine zukünftige Erweiterung für Ant, Zigbee oder andere Funkübertragungstechniken ist ebenfalls vorstellbar. Die neuen Schnittstellen ermöglicht dann die Integration von neuen Sensoren in das System. An der Anwendung der Sensorplattform müssten dann keine (oder nur geringfügige Änderungen) durchgeführt werden, da Sensoren zwischen Anwendungsebene und Middleware einheitlich angesteuert werden. Es ist absehbar, dass die Verwendung einer Middleware die gesamte Softwarearchitektur komplexer gestaltet, weil es mehr Stufen im Software-Stack gibt. Da dies im Rahmen dieser Arbeit zu umfangreich ist, wird die Anwendung direkt auf dem Bluetooth-Treiber aufgesetzt und keine weitere Schnittstelle unterstützt (siehe Kapitel 4.7).

Die Konzeption von Middlewares ist ein an mobile Gesundheitsüberwachungssysteme angrenzender Forschungsbereich. Die Sensorplattform stellt eine Anwendungsmöglichkeit dar, um über eine Middleware mit einem Wireless Sensor Network (WSN) zu kommunizieren. Im Folgenden werden weitere mobile Gesundheitsüberwachungssysteme für unterschiedliche Krankheiten vorgestellt.

Das Forschungsprojekt Monarca umfasst ein mobiles Gesundheitsüberwachungssystem, das nicht nur physische Vitalparameter aufzeichnet, sondern darüber hinaus auch Aktivitätsdaten sozialer Interaktion misst, die Rückschlüsse auf den psychischen Zustand des Patienten ermöglichen. Bei diesem Projekt werden mit Hilfe eines Smartphones und verschiedenen Sensoren Patienten mit bipolarer Störung untersucht. Bei einem Menschen mit dieser Erkrankung treten abwechselnd Phasen von Depression und Manie auf. Das Ziel von Monarca ist es, den Wechsel frühzeitig zu erkennen und die Intensität dieser Erscheinungen durch medikative Behandlung abzuschwächen. Während bei den bisher vorgestellten Projekten nur physische Parameter (Vitalparameter) erhoben wurden, erfasst Monarca zusätzlich noch Daten, die Rückschlüsse auf den psychischen Zustand des Patienten geben

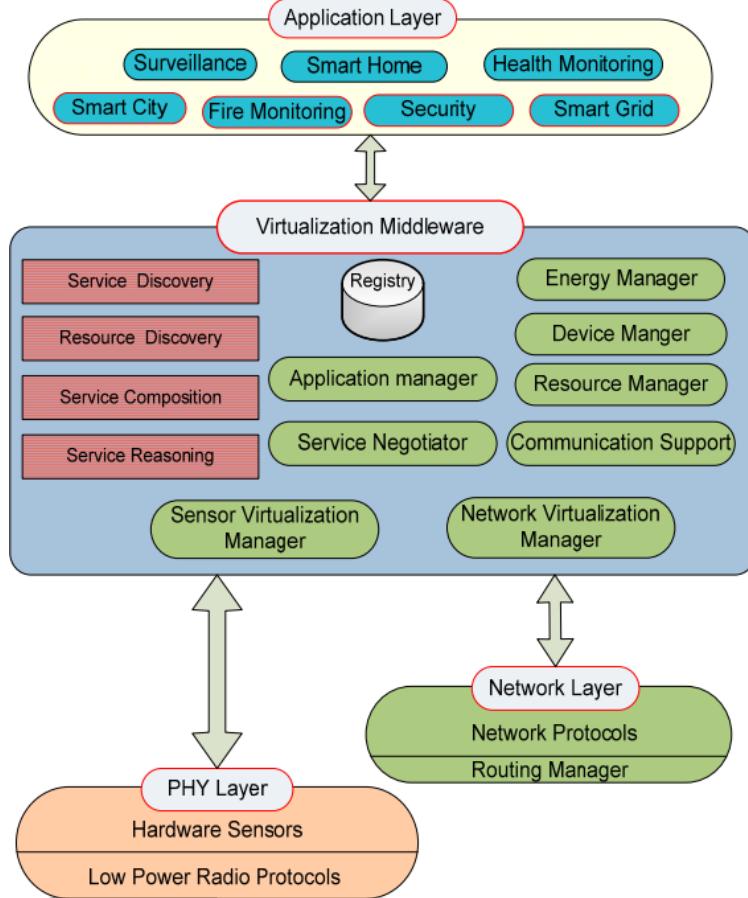


Abbildung 10: Modell einer Middleware für ein Wireless Sensor Network (WSN) [KFR14]

sollen, z. B. die Häufigkeit von Anrufen und SMS. Diese Parameter sind Indikatoren für die manische Phase. [PMGM11].

In diesem Projekt werden mehrere Sensoren für Vitalparameter verwendet. Abbildung 11 zeigt die Architektur des Monarca-Systems. Ein Gyroskop und ein Beschleunigungssensor sind in einer Uhr für das Handgelenk integriert, mit denen körperliche Aktivität gemessen wird. Zusätzlich trägt der Patient eine Socke mit integriertem Pulsoximeter sowie einem Sensor für elektrodermale Aktivität. Diese beiden Module sind in Abbildung 11 als Sensorblock zusammengefasst. Auf dem Smartphone wird die Position durch GPS und die Stimme des Patienten bei Telefonaten sowie Informationen über die Nutzung von SMS und Email (interne Sensoren), aufgezeichnet. Alle Sensordaten werden an das Smartphone gesendet und von da an die Monarca-Webserver übertragen, wodurch es anschließend möglich ist, die Informationen an ein Krankenhaus zu übertragen. [PMGM11].

Ein wesentliches Ergebnis ist, dass die Akkulaufzeit des Smartphones nur etwa 12 Stunden beträgt, wenn alle Sensoren kontinuierlich aufzeichnen. Als Gegenmaßnahme werden Einstellungen vorgenommen, dass die Sensoren nur bei Bewegung des Patienten Daten sammeln und ansonsten im Standby-Zustand sind, um Strom zu sparen. Auch werden bestimmte Bewegungen wie z. B. das Auto fahren erkannt, um in dieser Zeit Teile des Systems zu deaktivieren. Das deutet darauf hin, dass die Energieversorgung im Allgemeinen ein kritischer Bereich dieser Systeme ist. [PMGM11].

In Bezug auf Sicherheit und Datenschutz wird eine unkonventionelle Lösungen geboten: Der Patient und der Arzt einigen sich auf einen AES-Schlüssel, den nur diese beiden Akteure kennen. Mit diesem Schlüssel werden alle Daten von Monarca verschlüsselt. Bei bestehenden Protokollen wie beispielsweise HTTPS erfolgt eine doppelte Verschlüsselung.

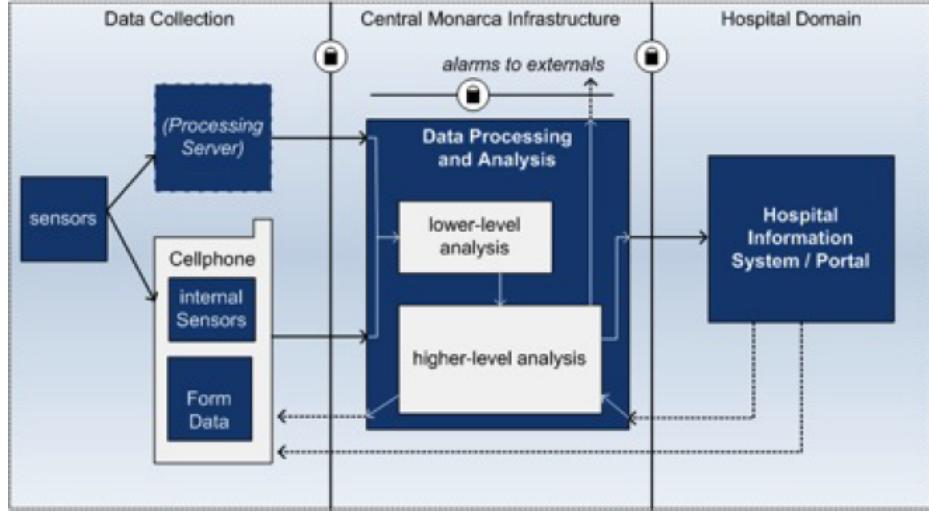


Abbildung 11: Systemarchitektur von Monarca [PMGM11]

Mit Hashes werden außerdem Daten zur Anzahl der SMS oder Emails gespeichert. Diese Informationen werden am Ende des Tages an einen Webserver hochgeladen. Eine wesentliche Funktion der Informationssicherheit liegt damit im Verantwortungsbereich des Arztes und des Patienten. [PMGM11].

Wie bereits erwähnt, besitzt die Sensorplattform im Rahmen dieser Arbeit keine Möglichkeit, um psychische Parameter zu erfassen. Allerdings lässt sich diese Funktionalität auch mit der Sensorplattform verhältnismäßig einfach realisieren. Möglich wäre dafür eine App auf dem Smartphone, die einen GATT-Server mit Services und Characteristics für bspw. die SMS- und Email-Nutzung erstellt. Die Sensorplattform kann sich anschließend mit dem Smartphone wie mit einem Sensor verbinden und die Daten abfragen. Sicherheitsfunktionen bei der Übertragung mit Bluetooth sind bereits berücksichtigt. Die gesammelten Daten müssen natürlich im Umfang begrenzt werden, da beispielsweise ein Characteristic nur höchstens 20 Bytes lang sein darf. Die gesamten Audiodaten der Telefonate ließen sich nicht effizient über Bluetooth Low Energy (BLE) übertragen.

Im Folgenden wird ein weiteres mobiles Gesundheitsüberwachungssystem gezeigt, das in dem Paper „A Zigbee-Based Wearable Physiological Parameters Monitoring System“ von K. Malhi und S. C. Muhkopadhyay präsentiert wird. Die Architektur dieses Systems unterscheidet sich geringfügig von der Sensorplattform und anstatt Bluetooth kommt der Funkübertragungsstandard Zigbee zum Einsatz. Es werden ebenfalls nur nicht-invasive Messungen vorgenommen. Das System ist für den stationären Gebrauch in einem Haus oder in einer Wohnung konzipiert. [MM12].

Das am Körper getragene System, dargestellt in Abbildung 12, setzt sich aus drei Hardwarekomponenten zusammen: Ein Pulsoximeter für das Handgelenk bildet die erste Komponente. Das zweite Gerät wird am Handgelenk getragen und besitzt einen Temperatursensor und einen Beschleunigungssensor auf zwei Achsen. Durch Verarbeitung dieser Indikatoren soll das System die Atmung überwachen und Stürze erkennen. Die dritte Hardwarekomponente ist ein eingebettetes System als Zwischenspeicher mit einem Zigbee-Funkmodul. Mit einer 9V-Batterie kann das System 25 Stunden betrieben werden. Alle drei Geräte sind jedoch durch Kabel und Steckverbindungen aneinander angeschlossen (siehe Abbildung 12). Mehrere dieser Systeme können in einem Zigbee-Netzwerk mit einem zentralen Zugangspunkt, dem Zigbee-Koordinator, verbunden sein. Dieser leitet die empfangenen Daten an einen lokalen Computer weiter, auf dem diese in einer speziellen Anwendung angezeigt werden. Die Computersoftware kann über eine Internetverbindung gegebenenfalls

auch ein Alarmsignal weiterleiten. [MM12].

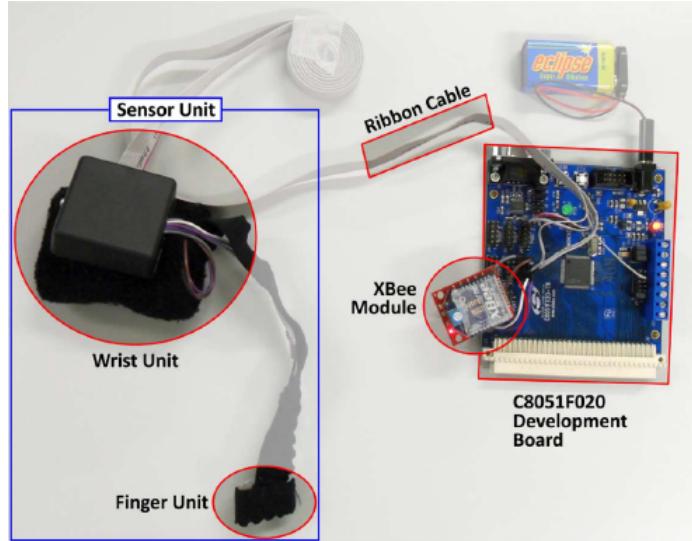
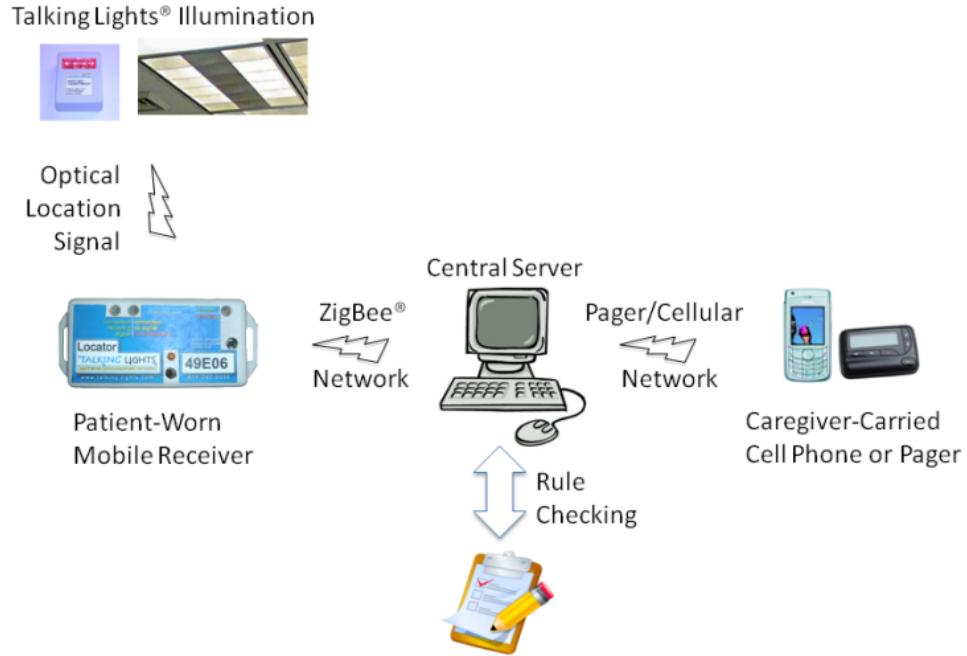


Abbildung 12: Prototyp des Zigbee-Systems [MM12]

Ein weiteres mobiles Gesundheitsüberwachungssystem, das sich stärker von den bisher vorgestellten Systemen unterscheidet, ist das „Escort System: A Safety Monitor for People Living with Alzheimer’s Disease“. Die an Alzheimer erkrankten Patienten neigen oftmals dazu, ziellos umherzuwandern. Auf Grund häufig vorkommender Stürze stellt dieses Verhalten ein großes Problem dar. Abbildung 13 zeigt einen Überblick über das Escort-Gesamtsystem. Bei diesem tragen die Patienten ein eingebettetes System (ein sogenanntes *Badge* oder Abzeichen) mit einem optischen Empfänger, das auch über eine Anbindung an ein Zigbee Netzwerk verfügt. Das Gerät zeichnet jedoch keine (Vital)-Daten auf. In jedem Raum befindet sich ein Beacon, sodass das Abzeichen die Position eindeutig ermitteln kann. Im Bedarfsfall sendet das Gerät eine Benachrichtigung an einen stationären, zentralen Server. Das System erfordert eine feste Umgebung, weil es für die Beurteilung einer kritischen Situation des Alzheimerpatienten auf vordefinierte Regeln zurückgreift. Der zentrale Server leitet die Alarmierung anschließend an das Pflegepersonal weiter. Deinen Geräte verfügen über drei Knöpfe, mit denen die Datensammlung unterstützt wird. [TLL⁺11].

Die Gemeinsamkeit der Sensorplattform und dem Abzeichen des Escort-Systems ist, dass keine Mensch-Maschine-Interaktion des Patienten mit dem mobilen Gerät vorgesehen ist. Die umfangreiche Konfiguration der Umgebung durch Lichtsender und Zigbee Repeater ist bei der Sensorplattform nicht vorgesehen. Grundsätzlich wäre es aber möglich, zumindest ein Zigbee Modul in die Sensorplattform einzubauen. Eine optische Methode zur Bestimmung der Position, wie sie beim Escort-System genutzt wird, ist bei der Sensorplattform kaum möglich, weil die Hardware gegebenenfalls unter der Kleidung des Trägers verdeckt ist. Die Verwendung eines Empfängers für das europäische Satellitennavigationssystem Galileo, das über eine höhere Genauigkeit verfügt als GPS für die zivile Nutzung, könnte jedoch eine Alternative zum optischen Verfahren darstellen [Deu]. Dennoch wären für die Datenauswertung nach wie vor viele vordefinierte Kenntnisse über das Gebäude nötig, in dem das System eingesetzt werden soll. Zum Beispiel durch die Verwendung eines Bluetooth-Sensors zum Messen der Bewegung (Lage, Beschleunigung oder Ausrichtung am Magnetfeld der Erde), könnte die Sensorplattform auch für diesen Use Case genug Daten bereitstellen. Ein lokales Zigbee-Netzwerk im Gebäude oder im größeren Nutzungsbereich eine Mobilfunkverbindung ermöglichen das Weiterleiten von Alarmsignalen der Sensorplattform an das Pflegepersonal.

Abbildung 13: Architektur des Escort-Systems [TLL⁺¹¹]

Durch den Einsatz mobiler Gesundheitsüberwachungssysteme mit drahtlosen Sensornetzwerken ergibt sich auch ein neuer Forschungsbereich *Home Hospitalization* bzw. Telemedizin. Wie bereits bei Polycare deutlich geworden, geht es unter anderem um die Reduzierung von Kosten, die durch einen stationären Aufenthalt des Patienten im Krankenhaus entstehen. Damit geht auch die Verbesserung der gesundheitlichen Versorgung einher. Die kontinuierliche Beobachtung von Vitalparametern kann auch die Häufigkeit von Fehlern bei der medikativen Behandlung verringern. Durch die geringere Anzahl an Patienten verringert sich außerdem die Belastung von medizinischem Pflegepersonal. Gerade in ländlichen Gebieten mit weit entfernten Krankenhäusern kann dadurch die gesundheitliche Versorgung verbessert werden, weil Symptome schon früh erkannt werden und auf die eine Reaktion möglich ist, bevor eine kritische Situation eintritt. Ältere Menschen akzeptieren solche Systeme mittlerweile zunehmend, wenn diese dafür ein unabhängiges Leben gewährleisten. [Var07].

Der Stand der Forschung hat gezeigt, dass es viele unterschiedliche Anwendungsmöglichkeiten für mobile Gesundheitsüberwachungssysteme gibt. Jedes System setzt den Schwerpunkt geringfügig anders, weshalb die vielfältige Konfigurierbarkeit und Erweiterbarkeit der Sensorplattform von entscheidender Bedeutung ist, um das System auf verschiedene Krankheitsbilder einzurichten.

4. Anforderungsanalyse

Zu Beginn werden die Anforderungen an die Sensorplattform als mobiles Gesundheitsüberwachungssystem festgestellt, die sich zu einem Großteil aus Erfahrung von bereits abgeschlossenen Projekten am Fraunhofer FIT ergeben. Die Sensorplattform kombiniert eine Vielzahl unterschiedlicher Anforderungsbereiche, die auch überfachliche Aspekte beinhalten, weshalb für den zeitlichen Rahmen der Masterarbeit auch bestimmte Einschränkungen festgelegt werden.

Die Anforderungen an die Sensorplattform lassen sich in die Bereiche Funktionalität, Usability, Datenschutz und Informationssicherheit, Energieeffizienz, Zuverlässigkeit und medizinische Richtlinien einteilen. Dazu kommen einige softwarespezifische Anforderungen, die von Seiten des Instituts gestellt werden.

4.1. Funktionalität

1. Die Sensorplattform ist ein eingebettetes System zum Erfassen von Sensordaten.
2. Die Sensorplattform analysiert die Daten nicht, sondern leitet diese an einen Webserver weiter.
3. Das System wird aus kommerziell erhältlichen Hardwarekomponenten zusammengestellt.
4. Die Sensoren werden durch Funkübertragungstechnologien mit der Sensorplattform verbunden.
5. Es werden (nicht-invasive) Sensoren für Vitalparameter oder andere Werte wie zum Beispiel Bewegung (Lage, Beschleunigung und Ausrichtung am Magnetfeld der Erde) für die Sensorplattform genutzt.
6. Die Sensorplattform soll erweiterbar in Bezug auf neue Sensoren sein.
7. Die Sensorplattform soll frei konfigurierbar sein, sodass auch verschiedene Kombinationen von Sensoren verbunden werden können.
8. Die Sensorplattform bietet Netzwerkverbindungen durch Kabel (Ethernet), WLAN und Mobilfunk.
9. Die Datenübertragung zwischen Sensoren und Sensorplattform geschieht in Echtzeit.
10. Die Sensordaten werden anschließend ebenfalls in Echtzeit an einen Webserver übertragen. Dies bietet die Möglichkeit, auf kritische Werte zeitnah reagieren zu können, bspw. durch Alarmierung eines Notarztes.
11. Der Zugriff auf die Sensorplattform erfolgt von einem PC über einen Webbrower. In der Webanwendung kann das Gerät konfiguriert, gesteuert und ausgelesen werden.

4.2. Usability

1. Die Sensorplattform wird am Körper getragen und soll den Alltag des Patienten nur geringfügig beeinflussen.
2. Das System soll nach Möglichkeit keine Kabelverbindungen haben, um die Bewegungen des Patienten nicht einzuschränken.
3. Es findet keine Mensch-Maschine-Interaktion zwischen Patient und Sensorplattform statt.
4. Eine LED signalisiert den internen Status der Sensorplattform, bspw. ob das Gerät gerade Daten aufzeichnet oder die Verbindung zu Sensoren verloren hat.
5. Ein Display auf dem Gerät, um z. B. bestimmte Messwerte für den Patienten anzuzeigen, ist zunächst nicht vorgesehen.

4.3. Datenschutz und Informationssicherheit

Bei dem Demonstrator finden keine Veränderungen an der Hardware statt, daher könnte die Debug-Schnittstelle auf der Leiterplatte dafür genutzt werden, den Demonstrator zu kompromittieren. Schon allein durch die Tatsache, dass das Betriebssystem des Demonstrators auf einer austauschbaren micro-SD-Karte installiert ist, sind derartige Überlegungen überflüssig, weshalb die Sensorplattform alle Daten unverschlüsselt auf dem Speichermedium ablegt. Eine Validierung der Sensorplattform als kryptographisches Modul ist zwar möglich, in dieser Arbeit aber nicht erforderlich. Es werden lediglich Sicherheitsfunktionen der Funkübertragungs- und Netzwerkschnittstellen zum System betrachtet.

1. Die Datenübertragung über die Funkschnittstellen zwischen Sensoren und Sensorplattform muss nach Möglichkeit verschlüsselt erfolgen.
2. Um die Integrität des Systems zu gewährleisten, sollen Verfahren zur Authentifizierung von Sensoren zum Einsatz kommen.
3. Die Daten zwischen Sensorplattform und externem Webserver werden verschlüsselt übertragen.
4. Außerdem ist eine Autorisierung der Sensorplattform gegenüber dem externen Webserver obligatorisch.
5. Die Webanwendung der Sensorplattform für die Mensch-Maschine-Interaktion nutzt ebenfalls Verschlüsselung.
6. Um die Webanwendung der Sensorplattform aufzurufen, ist eine Autorisierung des Benutzers erforderlich.

4.4. Energieeffizienz

1. Die Stromversorgung der Sensorplattform erfolgt über ein Netzteil oder über einen Akku bei der mobilen Verwendung des Systems.
2. Der Richtwert für die Aufzeichnungsdauer der Sensorplattform beträgt ungefähr 16 Stunden mit einer Akkuladung. Dies entspricht dem Einsatz am Tag und der Wiederaufladung über Nacht.

4.5. Zuverlässigkeit

1. Der Datenverlust bei plötzlichem Stromausfall soll so gering wie möglich gehalten werden.
2. Die Sensorplattform versucht automatisch, sich erneut mit Sensoren zu verbinden, falls es während des Betriebs zu Verbindungsabbrüchen kommt.
3. Alle Sensordaten werden immer in der lokalen Datenbank der Sensorplattform gespeichert, unabhängig davon, ob diese Daten zu einem externen Webserver hochgeladen wurden oder nicht. Dies ist nicht zwangsläufig erforderlich, bietet aber weitere Informationen beim Testen (siehe Kapitel 7.3.3).
4. Das Hochladen der Sensordaten kann vom medizinischen Fachpersonal über die Webanwendung der Sensorplattform nachgeholt werden, falls dies zuvor nicht (immer) geklappt hat, z. B. auf Grund von schlechter Signalqualität der Mobilfunkverbindung.

4.6. Medizinische Richtlinien

1. Es gibt die EU-Richtlinie 93/42/EWG für Medizinprodukte, die es gegebenenfalls bei der Konzeption der Sensorplattform zu beachten gilt [Rat93].
2. Die EU-Richtlinie für elektromagnetische Verträglichkeit 2014/30/EU ist ebenfalls für einige Hardwarekomponenten der Sensorplattform relevant [Rat14].

Bei der Entwicklung des Demonstrators wird auf Richtlinien für Medizinprodukte keine Rücksicht genommen. Die dafür notwendige Dokumentation und Tests würden ansonsten den Schwerpunkt der Arbeit verschieben.

Da ausschließlich kommerzielle Hardware zum Einsatz kommt, die innerhalb der EU erworben werden kann, ist gewährleistet, dass die Richtlinien für elektromagnetische Verträglichkeit eingehalten sind (das Vorhandensein des CE-Siegels auf den Komponenten).

4.7. Softwarespezifische Anforderungen

1. Die Sensorplattform muss auf einem Linux-Betriebssystem basieren. Dies gewährleistet die Portierbarkeit und ermöglicht die Unabhängigkeit des Systems von (großen) Softwareherstellern.
2. Das Echtzeitverhalten ist auf Grund des Betriebssystems genau genommen nicht möglich. Für sicherheitskritische Anwendung wäre dies nicht ausreichend. Der Echtzeitbegriff wird im Kontext der Sensorplattform so definiert, dass das System in einigen Sekunden auf Ereignisse reagieren kann.
3. Die Verwendung der Programmiersprache Java wird aus unternehmensstrategischen Gründen des Fraunhofer FIT vorgegeben.
4. Zunächst werden nur Bluetooth Low Energy (BLE) Sensoren mit der Sensorplattform genutzt. Die Spezifikation soll so verwendet werden, dass neue Bluetooth-Sensoren ebenfalls mit der Sensorplattform kommunizieren können. Durch die flexible Kombination von Sensoren soll die Sensorplattform für vielfältige Krankheitsbilder konfigurierbar sein.
5. Gegebenenfalls kann die Sensorplattform zukünftig für andere Schnittstellen, bspw. Zigbee oder Ant, erweitert werden.

4.8. Anwendungsfälle (Use Cases)

Die allgemeinen Anforderungen gewähren einen Eindruck, was die Sensorplattform leisten soll. Um die Anforderungsanalyse zu konkretisieren, werden Anwendungsfälle (Use Cases) definiert, die den Funktionsumfang der Software beschreiben. Es gibt zwei Akteure, die mit der Sensorplattform interagieren: das medizinisches Fachpersonal (im Folgenden auch Benutzer genannt) und der Patient. Allerdings soll die Sensorplattform so konzipiert sein, dass für den Patienten so wenig Interaktion wie möglich mit der Sensorplattform erforderlich ist. Im Idealfall signalisiert eine LED dem Patienten den Status des Geräts, ansonsten sind für diesen keine Kenntnisse über das System relevant.

Der Begriff *Aufzeichnung* beschreibt in diesem Kontext, dass die Sensorplattform in einem bestimmten Zeitraum mit mindestens einem Sensor verbunden ist und Sensordaten sammelt und verarbeitet. Das Verarbeiten umfasst das Speichern in der Datenbank und gegebenenfalls das Hochladen zu einem externen Webserver. Die folgenden Use Cases werden für die wichtigsten Funktionen der Sensorplattform formuliert:

1. Das medizinische Fachpersonal startet die Sensorplattform und verbindet diese mit einem Netzwerk (Ethernet-Kabel). Über einen Browser kann das medizinische Fachpersonal die Webanwendung der Sensorplattform aufrufen.
2. Für die Webanwendung ist eine Autorisierung (Benutzername und Passwort) des Benutzers erforderlich.
3. Das Webinterface zeigt allgemeine Informationen über den Status der Sensorplattform an, bspw. zu welchem externen Webinterface die Daten hochgeladen werden oder ob das Gerät gerade Daten sammelt.

4. Durch Eingaben in der Webanwendung startet der Benutzer eine Aufzeichnung der Sensorplattform.
5. Bei falschen oder unvollständigen Angaben gibt die Webanwendung eine Fehlermeldung aus und startet keine Aufzeichnung.
6. Die Webanwendung bietet dem Benutzer die Möglichkeit, eine zuvor gestartete Aufzeichnung abzubrechen.
7. Über das Webinterface kann ebenfalls nach neuen Bluetooth Low Energy Geräten gesucht werden. Die gefundenen Geräte (Name und Bluetooth-Adresse) werden aufgelistet.
8. Der Patient trägt Sensoren und Sensorplattform für einige Stunden am Körper. Für das kontinuierliche Aufzeichnen von Daten ist keine Interaktion des Patienten mit dem System mehr notwendig.
9. Eine LED signalisiert anhand von Blinkmustern den internen Status des Systems, bspw. ob die Sensorplattform gerade Sensordaten aufzeichnet oder ob die Verbindung zu (mindestens) einem Sensor verloren gegangen ist.
10. Ist ein Sensor beim Start einer Aufzeichnung nicht verfügbar (z. B. durch Eingabe einer falschen Bluetooth-Adresse), meldet die Sensorplattform den Fehler an die Webanwendung und bricht den Start ab.
11. Während einer Aufzeichnung wird die Stromversorgung der Sensorplattform unterbrochen und anschließend wiederhergestellt. Liegt das Ende der Aufzeichnung noch in der Zukunft, zeichnet die Sensorplattform für die verbleibende Zeit weitere Sensordaten auf.
12. Während einer Aufzeichnung wird die Stromversorgung der Sensorplattform unterbrochen und anschließend wiederhergestellt. Liegt das Ende der Aufzeichnung bereits in der Vergangenheit, wird die Messung nicht weitergeführt und die Sensorplattform ist im Bereitschaftsmodus.
13. Während einer Aufzeichnung wird die Verbindung zu einem Sensor zwischenzeitlich unterbrochen (z. B. außer Reichweite). Die Sensorplattform erkennt dies und versucht kontinuierlich, sich wieder mit dem Sensor zu verbinden.
14. Die aufgezeichneten Sensordaten werden auf der Sensorplattform persistent in einer Datenbank gespeichert. Der Benutzer kann über die Webanwendung die Daten auslesen und gegebenenfalls auch löschen.
15. Falls die Sensorplattform zwischenzeitlich keine Mobilfunkverbindung hat, werden einzelne Samples mit einer Markierung versehen und in der Datenbank gespeichert.
16. In der Webanwendung ist erkennbar, wie viele Sensordaten noch nicht hochgeladen wurden. Das medizinische Fachpersonal kann diese Daten zu einem späteren Zeitpunkt nachträglich hochladen.
17. Es kann immer nur eine Aufzeichnung aktiv sein. Falls das medizinische Fachpersonal versucht, eine zweite Aufzeichnung zu starten während die erste noch aktiv ist, erscheint eine Fehlermeldung.

Während der Implementierungsphase gelten die Anwendungsfälle als Teilziele, die es zu erreichen gilt. In Kapitel 7.2 wird zu jedem Use Case mindestens ein Test Case vorgestellt, mit denen die Funktionalität verifiziert werden kann.

5. Konzept und Design

In diesem Kapitel werden kommerziell verfügbare Hardwarekomponenten vorgestellt und ausgewählt, die in Kombination den Demonstrator der Sensorplattform bilden. Aus zeitlichen Gründen findet für die Sensorplattform keine Hardwareentwicklung statt. Als Sensoren kommen auch Geräte des Verbrauchermarkts zum Einsatz. Diese Auswahl beinhaltet zudem konzeptionelle Entscheidungen, die Einfluss auf die Implementierung nehmen.

5.1. Auswahl der Hardware

Zunächst geht es um die Auswahl eines geeigneten Single-Board-Computers als Basis für die Sensorplattform. Das wichtigste Kriterium ist dabei die Konnektivität. Durch vielfältige Anschlussmöglichkeiten und Funkschnittstellen sollen viele unterschiedliche Sensoren unterstützt werden und ebenfalls soll das System einfach in eine bestehenden Netzwerkinfrastruktur integrierbar sein. Insbesondere Einplatinenrechner und System-on-Chips sind für die Sensorplattform gut geeignet. Bei diesen sind alle Hardwarebauteile auf einer Leiterplatte eingebettet. Diese nutzen hauptsächlich Mikroprozessoren mit ARM-Architektur und einer Linux-Distribution als Betriebssystem.

Es ist zu erwarten, dass der Bedarf an Rechenleistung (CPU) und Hauptspeicher (RAM) der Sensorplattform Anwendung nicht hoch ausfällt, weil die Abtastraten der Sensoren deutlich niedriger sind als die Taktfrequenz des Prozessors und somit kaum Last für das System entstehen sollte, wie auch die Tests in Kapitel 7.3 bestätigen. Deshalb sind der Prozessor und der Hauptspeicher für die Auswahl der Hardware von geringer Bedeutung. Ein bis zwei Gigabyte Hauptspeicher sind gebräuchlich und für die Anwendung der Sensorplattform bereits überdimensioniert. Der Flash-Speicher (nicht flüchtiger Speicher) ist entweder auf die Leiterplatte gelötet oder wird als micro-SD-Karte bereitgestellt. Dabei gibt es den Unterschied zwischen Single-Level Cell (SLC) und Multi-Level Cell (MLC), der sich hauptsächlich auf Geschwindigkeit und Haltbarkeit auswirkt. In einer MLC werden zwei Bits gespeichert, sodass verschiedene Signalpegel existieren und ausgewertet werden, um den Zustand zu ermitteln. Das erfordert komplexere Schaltungen, die anfälliger und langsamer sind. Die Haltbarkeit ist ungefähr um den Faktor 10 geringer als bei SLC.

Im Vordergrund steht die Konnektivität der Sensorplattform. Das Gerät soll durch Bluetooth Low Energy (BLE) und Ant bzw. Ant+ mit Sensoren kommunizieren können. Während die Bluetooth-Chips (EDR/ LE) in der Regel auf der Platine integriert sind, soll die Ant/ Ant+ Schnittstelle als Erweiterung über einen USB-2.0-Port (Ant Stick) möglich sein. Speziell Bluetooth muss mindestens in der Version 4.0 unterstützt werden, da es sich ab dieser Versionsnummer um den Bluetooth Low Energy Standard handelt (siehe Tabelle 1). Ideal wäre aber die neueste Spezifikation der Version 4.2 (bzw. 5.0), für den sichereren Algorithmus bei den Pairing-Verfahren (siehe Kapitel 3.1.4). Vorzugsweise ist das Modem für Mobilfunk bereits auf der Leiterplatte eingebettet. Eine solche integrierte Lösung ist jedoch selten. Deshalb wird zusätzlich ein weiterer USB-Port benötigt, an dem ein Surfstick angeschlossen werden kann. Auf USB-Geräte wird nach Möglichkeit verzichtet, da diese durch den Anschluss und das Gehäuse deutlich größer ausfallen, als jeweils ein integriertes Bauteil. Um die Nutzung der Sensorplattform für das medizinische Fachpersonal zu vereinfachen, sollte eine RJ-45 Buchse vorhanden sein, damit das Gerät mit einem bestehenden lokalen Netzwerk verbunden werden kann. Ein Chip für WLAN ist in der Regel bei allen Single-Board-Computern fest verbaut. Auf dem Board muss ebenfalls eine LED vorhanden sein, die über eine Programmierschnittstelle angesprochen werden kann. Über verschiedene Blinkmuster kann dem Benutzer der Status des Systems angezeigt werden. Anschlüsse für Audio, SATA und Grafik (z. B. HDMI) sind nicht erforderlich, aber trotzdem meistens vorhanden. Zusätzlich ist ein niedriger Stromverbrauch des Boards wünschenswert. Dieser Wert ist jedoch nur selten bei den Produkten angegeben und erst durch Tests ermittelbar.

(siehe Kapitel 7.3).

Letztlich ergibt sich noch eine weitere Anforderung, die durch den Bluetooth-Stack BlueZ 5 gestellt wird. Das Betriebssystem muss, wie in Kapitel 3.1.4 beschrieben, mindestens die Kernel Version 3.5 nutzen. Generell wird ein aktueller Kernel auf Grund der häufig besseren Unterstützung von Treibern bevorzugt.

Bei der Suche nach geeigneter Hardware hat sich herausgestellt, dass Kompromisse eingegangen werden müssen. Tabelle 4 zeigt die engere Auswahl von Boards, die für die Sensorplattform in Frage kommen. In den Spalten sind die gängigen Hardware-Parameter, Verbindungsmöglichkeiten und die unterstützte Kernelversion der Linux-Distribution aufgeführt. Der Einplatinenrechner Cubieboard 3 erfüllt in Bezug auf die Kernelversion nicht die Mindestanforderung, ist aber bereits am Fraunhofer FIT vorhanden. Gegebenenfalls könnte ein Backport von BlueZ genutzt werden. Die beiden System-on-Chips DART-MX6 und VAR-SOM-MX6 haben jeweils eine Steckverbindung und werden auf Trägerboards mit den verschiedenen Anschlüssen für USB usw. gesteckt (ähnlich wie bei PCI). Beide fallen damit wesentlich kleiner aus als normale Einplatinenrechner. Die dazu gehörigen Evaluationsboards mit den Anschlüssen haben jedoch in der Regel so umfangreiche Hardware integriert, dass diese sehr groß ausfallen. Deshalb sind diese für mobile Geräte weniger geeignet, bspw. für die Sensorplattform. Ein individuell hergestelltes Trägerboard durch einen Drittanbieter, das nur die benötigten Anschlüsse aufweist, wird als Möglichkeit herangezogen. Die Kosten für die Herstellung und für die Tests sind allerdings zu hoch, weshalb diese Lösung nicht weiter verfolgt wird.

Name	CPU	RAM	Speicher	BT	Wifi	3G	RJ-45	USB	Kernel
Cubie-board3	Cortex A7 x2	2GB	8GB	4.0	✓	✗	✓	2x Host, 1x OTG	3.4.79
PixiePro	Cortex A9 x4	≤2GB	2x SD-Karte	4.2	✓	✓	✗	2x Host, 1x OTG	4.2
RPI 3	Cortex A53 x4	1GB	1x SD-Karte	4.1	✓	✗	✓	4x Host	4.4
DART-MX6	Cortex A9 x2-4	≤1GB	≤64GB	4.1	✓	✗	✓	1x Host, 1x OTG	4.1.15
VAR-SOM-MX6	Cortex A9 x1-4	≤4GB	≤64GB	4.0	✓	✗	✓	1x Host, 1x OTG	4.1.15

Tabelle 4: Auswahl geeigneter ARM-Boards für die Sensorplattform [Polc, Tre, Rasb, Vara, Varb]

Wie bereits erwähnt, wird auch bei Sensoren auf kommerzielle Hardware zurückgegriffen. Eine Auswahl von Sensoren ist bereits am Institut vorhanden, einige werden für den Einsatz mit der Sensorplattform hinzugekauft. Ergänzt wird diese Auswahl durch ein Bluetooth-Entwicklungsboard, das insbesondere für die Anwendung von Sicherheitsfunktionen der Bluetooth-Verbindung genutzt wird. Im Folgenden wird der ausgewählte Einplatinenrechner sowie das Zubehör der Sensorplattform vorgestellt.

5.1.1. Raspberry Pi 3 und Zubehör

Ein Raspberry Pi 3 stellt die Basis für den Demonstrator der Sensorplattform dar. Die neueste Version, die erst seit dem 29. Februar 2016 verfügbar ist, besitzt erstmalig einen integrierten Bluetooth- und WLAN-Chip. Durch das kompakte Layout der Leiterplatte eignet sich der Raspberry Pi 3 besser für ein mobiles System als die Alternativen in Tabelle 4. Der einzige Nachteil ist das Fehlen eines integrierten Mobilfunkmoduls. Dieses wird durch einen USB-Surfstick bereitgestellt. Außerdem bietet der Raspberry Pi mit

Raspbian ein stabiles Betriebssystem mit einem aktuellen Kernel (Version 4.4) an. Damit ist zu erwarten, dass bspw. die Treiber-Unterstützung für den Surfstick verhältnismäßig gut ausfällt.

Der Raspberry Pi 3 wird durch einen Akku ergänzt, damit auch eine mobile Stromversorgung gewährleistet ist. Dieser hat eine Kapazität von 3000 mAh und stellt einen guten Kompromiss zwischen Kapazität und Abmessung dar. Ursprünglich wurde dieser Akku für Smartphones konzipiert, passt auf Grund der Abmaße jedoch auch hervorragend zum Gehäuse des Raspberry Pis. Die Stromstärke und Spannung zwischen Akku und Sensorplattform wird mit einer micro-USB-Steckverbindung gemessen (siehe Abbildung 14). Dieses Multimeter zeigt die Spannung und die Stromstärke an abwechselnd an. Die Anzeige der Messung ist auf zwei Dezimalstellen (d. h. auf 10 mA bzw. 10 mV) genau. Alle 2 Sekunden wird automatisch zwischen den beiden Messgrößen umgeschaltet.



Abbildung 14: Micro-USB-Multimeter zum Messen der Stromstärke und der Spannung [Tra]

Eine Mobilfunkverbindung wird durch den Surfstick E352S-5 von Huawei ermöglicht, der über einen USB-2.0-Port an die Sensorplattform angeschlossen wird. Der integrierte Chip unterstützt alle 2G- und 3G-Übertragungstechniken (bis einschließlich HSPA+). Eine LTE- bzw. 4G-Verbindung ist damit jedoch nicht möglich. Der Provider der eingelegten SIM-Karte ist die deutsche Telekom. Für geringe Datenraten der Sensorplattform ist eine GSM- oder GPRS-Verbindung ausreichend (siehe Kapitel 7.3.3). Es ist zu erwarten, dass die Herzfrequenzdaten inklusive Metadaten und Header nur einige Hundert Bytes pro Sekunde in der Summe ergeben. Sollten zukünftig zusätzliche Informationen hochgeladen werden, sind immer noch nur wenige Kilobytes pro Sekunde im Upload erforderlich.

In Abbildung 15 ist das Gesamtsystem der Sensorplattform dargestellt. Das System hat mit dem Surfstick eine Abmessung von 16,5 cm × 12 cm × 3,7 cm (Länge, Breite und Höhe). Ohne den Surfstick wäre das System nur 9,5 cm lang, ohne das Kabel nur 7 cm breit. Das Gewicht beträgt 216 Gramm. Viele Smartphones wiegen zum Vergleich ungefähr 150 Gramm, daher ist die Sensorplattform um ca. das 1,5-fache schwerer. Hinzu kommen die Sensoren, die jeweils zwischen 15 Gramm (CC2650) und 26 Gramm (Polar H7) wiegen.

Nachfolgend werden die Hardwarekomponenten vorgestellt, mit denen die Sensorplattform initial kommunizieren kann. Alle Sensoren nutzen die Funkübertragungstechnik Bluetooth Low Energy (BLE), jedoch unterscheiden sich diese in einigen Details. Die Unterschiede werden bei der Softwareentwicklung berücksichtigt (siehe Kapitel 6.2). In Bezug auf die Erweiterung der Sensorplattform für neue Sensoren gehen daraus Möglichkeiten und Grenzen hervor.



Abbildung 15: Sensorplattform mit Surfstick, Akku (Unterseite), Messgerät (rechts) und Stromkabel

5.1.2. Herzfrequenzsensoren

Die in Kapitel 3.1.3 vorgestellten Herzfrequenzsensoren von Polar, TomTom und Adidas sollen in Kombination mit der Sensorplattform verwendet werden. Diese nutzen BLE Version 4.0 und deren GATT-Server orientieren sich an der Bluetooth-Spezifikation. Dies ist ein wesentliches Kriterium für die Integration der Sensoren in die Sensorplattform. Die Dokumentation der Bluetooth SIG. ist damit ausreichend, wodurch es keine Informationen der Hersteller mehr erfordert.

5.1.3. SensorTag CC2650

Der SensorTag CC2650 von Texas Instruments, dargestellt in Abbildung 16, vereint mehrere Sensoren auf einer kleinen Leiterplatte und stellt deren Messwerte in einem GATT-Server bereit. Der Sensor nutzt Bluetooth-Version 4.2. Die Hardware ist nicht speziell für den medizinischen Einsatz konzipiert, sondern für Anwendung des Internet-of-Things [Tex]. Tabelle 5 listet die integrierten Sensoren zusammen mit deren Messwerten auf.



Abbildung 16: Texas Instruments SensorTag CC2650 [RS]

Einzelne Sensoren lassen sich auch separat durch den GATT-Server aktivieren und deaktivieren. Außerdem kann die Abtastrate auf einen Wert zwischen 0,4 und 3,3 Hz festgelegt werden. Auf Grund des geringen Preises von ungefähr 35 Euro und der sehr guten Dokumentation ist der Sensor geeignet, um in die Sensorplattform integriert zu werden. Zum Beispiel könnten durch die Daten des Bewegungssensors mögliche Stürze des Patienten gemessen werden. Der GATT-Server entspricht nicht der Bluetooth-Spezifikation. Somit wird für die Integration des Sensors in die Sensorplattform die Dokumentation von Texas Instruments benötigt, die aber sowohl ausführlich, als auch öffentlich zugänglich ist [Tex16].

Sensor	Hersteller	Messung	Messdaten
TMP007	Texas Instruments	Temperatur	Chiptemperatur und Objekttemperatur (Infrarotmessung)
HDC1000	Texas Instruments	Luftfeuchtigkeit	Lufttemperatur und Luftfeuchtigkeit
BMP280	Bosch Sensortec	Luftdruck	Lufttemperatur und Luftdruck
OPT3001	Texas Instruments	Umgebungslicht	Umgebungslicht
MPU-9250	Invensense	Bewegung	Magnetfeld, Lage und Beschleunigung auf jeweils drei Achsen

Tabelle 5: Sensoren und Messwerte des CC2650 [Tex16]

5.1.4. BLE113 Entwicklungsboard

Das BLE113 Entwicklungsboard von Bluegiga, dargestellt in Abbildung 17, wird zum Testen von Funktionen von Bluetooth 4.0 genutzt, die bei der bisherigen Auswahl von Sensoren nicht möglich sind. Dazu gehört unter anderem die Verschlüsselung von Bluetooth-Verbindungen und Authentifizierung der Geräte und Bluetooth-Pakete. Der GATT-Server lässt sich beliebig programmieren und auch die Firmware ist (bis auf wenige Einschränkungen) frei konfigurierbar. Die Programmierung erfolgt über die Sprache BGScript. Das Board wird mit einem USB-Anschluss mit einem Rechner verbunden und darüber geflasht. Für das BLE113 Entwicklungsboard wird die Firmware so programmiert, dass das Gerät die Eigenschaften eines Pulsoximeters besitzt (siehe Kapitel 6.10.2). Die Messwerte, die in der entsprechenden Characteristic im GATT-Server bereitgestellt werden, sind hartkodiert. Zudem verwendet die Firmware einige Sicherheitsfunktionen für die Bluetooth-Verbindung. Die Hardwaresensoren auf dem Board, bspw. zum Messen der Beschleunigung, werden nicht genutzt, da diese bereits der Sensor CC2650 liefert (siehe Kapitel 5.1.3). Zusätzlich bietet das Board einige Knöpfe und ein Display, die für das Pairing nützlich sind.

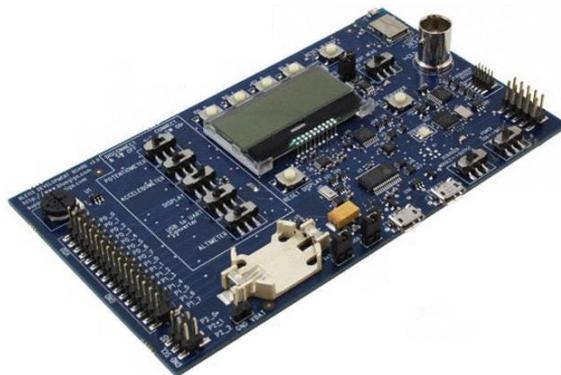


Abbildung 17: BLE113 Entwicklungsboard [Dig]

BGScript wird für mehrere Bluetooth-Entwicklungsboards genutzt. Diese Programmiersprache weist syntaktisch Ähnlichkeiten zu C auf. Funktionen werden jedoch hauptsächlich in Form von Events deklariert. Die Einstellungen für den GATT-Server erfolgen über eine XML-Datei. Ein spezieller Compiler kann aus diesen Dateien ein Programm für die Hardware des Entwicklungsboards generieren. BGScript wird auch bei einigen anderen Bluetooth-Entwicklungsboards verwendet. Der große Vorteil dieser Programmiersprache ist, dass schnell eine funktionierende Implementierung erstellt werden kann, ohne genauere Kenntnisse über die Hardware zu haben. Außerdem lassen sich die meisten in BGScript geschriebenen Programme auch auf andere Geräte portieren. [Sil].

Das BLE113 Entwicklungsboard kann zukünftig auch dazu genutzt werden, andere Senso-

ren zu simulieren. Dafür braucht im Wesentlichen nur die Konfiguration des GATT-Servers durch andere Services und Characteristics angepasst werden.

5.2. Mensch-Maschine-Interaktion

Im Kapitel Stand der Technik der Grundlagen 3.1 werden unter anderem Funkschnittstellen zwischen den Geräten (Sensoren, Sensorplattform und Webserver) vorgestellt. Durch die entsprechende Software lassen sich die einzelnen Geräte zu einem verteilten System verbinden. Bisher nicht berücksichtigt ist die Mensch-Maschine-Interaktion, was jedoch im Folgenden Abschnitt nachgeholt wird. Die vorgestellten Kategorien behandeln einen Großteil der Interaktion mit der Sensorplattform und werden in der Implementierungsphase berücksichtigt. Grundsätzlich gilt, dass die Bedienung der Sensorplattform so einfach wie möglich für das medizinische Fachpersonal sein soll.

5.2.1. Interaktion mit der Sensorplattform

Das medizinische Fachpersonal benötigt Zugang zu einer grafischen Oberfläche, um die Sensorplattform einfach bedienen zu können. Über diese kann beispielsweise eine Aufzeichnung mit bestimmten Sensoren gestartet und gestoppt werden. Für die Umsetzung gibt es unterschiedliche Möglichkeiten mit verschiedenen Vor- und Nachteilen.

Die Sensorplattform kann prinzipiell wie jeder Desktop-PC genutzt werden, da sich Monitor, Tastatur und Maus direkt an das Board anschließen lassen. Erforderlich sind dann aber ein HDMI-Ausgang und einige USB-Anschlüsse auf der Platine. Bei dem Demonstrator der Sensorplattform (Raspberry Pi 3) sind alle Anschlüsse vorhanden, allerdings erhöht sich dadurch auch der Stromverbrauch des Boards. Für eine zukünftige individuelle Hardwareentwicklung ist es daher sinnvoll, auf diese Anschlüsse zu verzichten und eine andere Bedienmöglichkeit zu wählen.

Die zweite Möglichkeit ist die Verbindung von Sensorplattform und Computer durch ein USB-Kabel. Dafür ist ein USB-OTG-Port an der Sensorplattform erforderlich. Ein USB-Host-Anschluss ist dafür nicht ausreichend. Außerdem sind ein entsprechender Treiber und Bedienungssoftware für die gängigen Betriebssysteme Windows, Linux und Mac OS notwendig. Die Software für die Sensorplattform und vor allem die Treiber sind dann gegebenenfalls an ein Betriebssystem gebunden. Das erhöht den Entwicklungsaufwand deutlich, falls alle gängigen Betriebssysteme unterstützt werden sollen. Gegebenenfalls entstehen Kosten durch Treiber-Signaturen für Windows. Beim Raspberry Pi 3 lässt sich diese Möglichkeit allerdings nicht nutzen, da nur die USB-Leitungen für die Stromversorgung bei dem USB-OTG-Port angeschlossen sind [Ras16]. Die Datenleitungen sind nicht mit dem Prozessor verbunden.

Die Bedienung der Sensorplattform kann als dritte Möglichkeit auch über eine Webanwendung erfolgen. Der entsprechende Webserver wird automatisch mit der Sensorplattform gestartet. Der Vorteil dieser Lösung ist, dass keine Software mehr auf einem lokalen Computer installiert werden muss, um die Sensorplattform zu bedienen. Mit jedem gängigen Browser kann die Steuerung für die Sensorplattform aufgerufen werden. Allerdings sind gegebenenfalls Einstellungen für das lokale Netzwerk erforderlich. Die Sensorplattform kann nur entweder über eine IP-Adresse und einen Port oder über einen Host-Namen erreicht werden. Ein Beispiel aus der internen Netzwerkkonfiguration des Fraunhofer FIT mit einer statischen IP-Adresse lautet wie folgt: <https://129.26.160.38:8080> bzw. <https://sensorplatform.fit.fraunhofer.de:8080>. Am Einplatinenrechner ist dann ein Netzwerkanschluss erforderlich, an dem die Sensorplattform eine IP-Adresse zugewiesen bekommt. Für den Demonstrator der Sensorplattform ist dies die bevorzugte Lösung, um mit dem System zu interagieren. Der Zugang zum Webserver der Sensorplattform erfolgt ausschließlich über Kabelnetzwerk (Ethernet) und WLAN. Um eine höhere

Sicherheit des Systems zu gewährleisten, ist ein Verbindungsauflauf über die Mobilfunkverbindung nicht möglich. Der Webserver ist dadurch nicht direkt möglichen Angreifern im Internet ausgesetzt.

Die vierte Möglichkeit kombiniert die Vorteile der zweiten und dritten Option. Dabei wird ebenfalls ein USB-OTG-Anschluss und ein Kabel für die Verbindung mit dem Computer genutzt. Mit einem Remote Network Device Interface Specification (RNDIS) Treiber wird auf Basis der USB-Verbindung eine virtuelle Netzwerkverbindung erzeugt. Aus der Perspektive des Betriebssystems ist dies eine Netzwerkschnittstelle, wie sie auch beim Kabelnetzwerk (Ethernet) definiert ist. Auf der Sensorplattform ist ein DHCP-Server installiert, der automatisch einem angeschlossenen Computer eine IP-Adresse zuweist. Der Webserver der Sensorplattform ist dann durch die USB-Verbindung über eine feste IP-Adresse oder einen Namen erreichbar. Es ist keine Netzwerkkonfiguration mehr durch den Benutzer erforderlich. Im emulierten Netzwerk gibt es höchstens zwei Geräte: die Sensorplattform und einen weiteren Computer. Für RNDIS ist ein Treiber mit einigen Konfigurationen im Linux-Kernel integriert, damit Betriebssysteme (insbesondere Windows) das Gerät korrekt als RNDIS-Gadget erkennen und ebenfalls automatisch den passenden Treiber installieren.

Der USB-OTG-Anschluss ist bei allen Boards vorhanden, bei denen das Betriebssystem im integrierten Speicher abgelegt ist. Beim Raspberry Pi wird es auf eine micro-SD-Karte gespeichert, was der Grund sein könnte, warum kein USB-OTG-Port mit verbundenen Datenleitungen verfügbar ist. Hat das Board der Sensorplattform zusätzlich zum USB-OTG-Port eine Netzwerk-Buchse, kann der Webserver prinzipiell auch über beide Schnittstellen aufgerufen werden. Dies bietet eine sehr flexible Konfiguration falls die Sensorplattform auf anderen Geräten genutzt wird.

5.2.2. Interaktion mit den Sensoren

Das medizinische Fachpersonal verbindet die Sensorplattform mit dem Netzwerk und ruft die Webanwendung auf. Danach gilt es, eine Aufzeichnung mit verschiedenen Sensoren zu starten. Allerdings sind dafür zunächst zwei Angaben über die Sensoren notwendig, die in der Regel nicht am Gerät selbst abgelesen werden können. Die Sensoren aus dieser Arbeit stellen Blackboxen dar. Aus den Beschriftungen der Sensoren geht lediglich das Bluetooth-Symbol hervor, um diese als Bluetooth-Geräte zu identifizieren. Zunächst ist es daher wichtig zu wissen, wie der konkrete Sensor aus dem Energiesparmodus geweckt wird. Bei den Herzfrequenzsensoren geschieht dies durch Hautkontakt mit den beiden Elektroden. Beim CC2650 muss eine von zwei Tasten gedrückt werden. Die Sensoren sind dann für eine in der Firmware definierte Zeit für andere Bluetooth-Geräte sichtbar. Dieser Zeitraum beträgt 15 Sekunden bei den Herzfrequenzsensoren und 2 Minuten beim CC2650. Sind die Sensoren sichtbar, kann über die Webanwendung nach Bluetooth Low Energy (BLE) Geräten gesucht werden, die anschließend dargestellt werden.

Aus der Suche nach Bluetooth-Geräten gehen zwei Informationen hervor: der Name des Geräts und die (in der Regel einzigartige) Bluetooth-Adresse (siehe Kapitel 3.1.4 für eine genaue Klassifizierung von Bluetooth-Adresse). Es ist zu erwarten, dass der Name einfach zu einem Gerät zugeordnet werden kann, allerdings ist dem nicht so. Der Herzfrequenzsensor von TomTom verwendet bspw. den Namen *HRM (Ver0.4)*. Daher ist es sinnvoll, eine Funktion in die Sensorplattform zu implementieren, die den Anzeigenamen in eine benutzerfreundliche Darstellung umwandelt. Eine mögliche Lösung dafür ist in Kapitel 6.2.3 vorgestellt. Die Kenntnis der Bluetooth-Adresse ist für die Verbindung zu einem konkreten Gerät obligatorisch. Mit dem Namen des Sensors und dessen Bluetooth-Adresse, kann das medizinische Fachpersonal die Sensoren für eine neue Aufzeichnung in der Webanwendung auswählen.

Auch bei einer aufgebauten Bluetooth-Verbindung unterscheiden sich die Herzfrequenzsensoren vom CC2650. Nach ungefähr 15 Sekunden fehlendem Hautkontakt kehren die Herzfrequenzsensoren in den Energiesparmodus zurück und unterbrechen eine vorhandene Bluetooth-Verbindung zur Sensorplattform, auch wenn gerade Notifications aktiviert sind. Die Sensorplattform erhält keine Benachrichtigung vor dem Abbruch der Verbindung durch den Sensor. Der Verbindungsabbruch wird erst erkannt, wenn die Sensorplattform explizit einen neuen Befehl an den Sensor schickt. Der CC2650 kehrt nur in den Energiesparmodus zurück, wenn keine aktive Verbindung mehr vorhanden ist. Es muss daher eine Art Keep-Alive-Kommunikation zwischen beiden Geräten stattfinden. Die gesendeten Pakete lassen sich jedoch nicht mit dem Debug-Programm btmon (siehe Kapitel 6.1) nachvollziehen. Bei dem Einsatz dieser Sensoren am Patienten muss dieses Funktionsweise berücksichtigt werden. Es ist vorstellbar, dass die Sensoren für kurze Zeit abgelegt werden. Die Sensorplattform muss anschließend automatisch die Verbindung zu den Sensoren wiederherstellen, wenn diese erneut verfügbar sind, da keine Interaktion des Patienten mit dem System vorgesehen ist.

5.2.3. Aufzeichnen von Sensordaten

Zum Start einer Aufzeichnung wird unter anderem eine Sensorkonfiguration benötigt. Diese beschreibt welche Sensoren für eine Aufzeichnung genutzt werden und darüber hinaus, mit welchen Einstellungen ein Sensor messen soll. Bei den Herzfrequenzsensoren ist eine Konfiguration nicht möglich. Bei dem Sensor CC2650 von Texas Instruments lässt sich definieren, welche der bis zu fünf möglichen Messwerte erfasst werden und wie hoch jeweils die Abtastrate ist. Weitere sensorspezifische Einstellungen, auch für neue Sensoren, können ergänzt werden. Zusätzlich zur Sensorkonfiguration sind einige Metainformationen anzugeben, z. B. der Vor- und Nachname des Patienten. Mit diesen Daten lassen sich die Messwerte zu einem späteren Zeitpunkt einfacher zuordnen. Das medizinische Fachpersonal gibt außerdem die Dauer einer Aufzeichnung an und die Sensorplattform errechnet daraus den Endzeitpunkt der Aufzeichnung. Alle Eingaben erfolgen über die Webanwendung der Sensorplattform und Aufzeichnungen können durch Buttons gestartet und gestoppt werden. Fehlermeldung, bspw. wenn ein Sensor wieder im Standby-Modus und nicht mehr verfügbar ist, können durch die Webanwendung dem medizinischen Fachpersonal mitgeteilt werden.

5.2.4. Datenbank-Interaktion

Die Webanwendung der Sensorplattform bietet für das medizinische Fachpersonal außerdem die Möglichkeit, mit der Datenbank zu interagieren. Die gespeicherten Messwerte können in Textform angezeigt werden. Wechselt das Gerät den Patienten, ist es sinnvoll, dass alle gespeicherten Sensordaten gelöscht werden, was ebenfalls über die Webanwendung möglich sein. Vorstellbar wäre auch, dass das Hochladen der Messwerte zu einem externen Server nachgeholt werden kann, wenn dies zuvor nicht geklappt hat, bspw. wenn zwischenzeitlich kein Mobilfunk vorhanden war. Weiterführend wäre eine Funktion nützlich, die Sensordaten in Form einer Datei in einem gängigen Format (z. B. JSON oder XML) herunterzuladen. Das medizinische Fachpersonal könnte diese Daten dann zum Auswerten auf einem lokalen Rechner nutzen. Diese Funktion ist bei der Sensorplattform jedoch nicht implementiert.

6. Implementierung und Softwarearchitektur

In diesem Kapitel wird die Implementierung und die zugrunde liegende Softwarearchitektur der Sensorplattform erläutert. Während dieser Phase werden auch einige Modultests angefertigt, die bestimmte Klassen testen. In Kapitel 7 sind diese separat erläutert. Für Sicherheitsfunktionen (z. B. die Verschlüsselung von Bluetooth-Verbindungen) sind in diesem Kapitel bereits Möglichkeiten vorgestellt, um deren korrekte Verwendung in der Software zu verifizieren.

Das wichtigste Kriterium für die Software der Sensorplattform ist die freie Erweiterbarkeit und Konfigurierbarkeit des Systems. Während zu Beginn nur fünf Sensoren unterstützt werden, soll die Sensorplattform so konzipiert sein, dass in Zukunft jeder Bluetooth Low Energy (BLE) Sensor integriert werden kann. Die GATT-Profiles der Bluetooth-Spezifikation bilden bereits eine gute Grundlage für die Implementierung einer Schnittstelle zwischen Sensor und Sensorplattform. Das heißt, dass sich bspw. Herzfrequenzsensoren unterschiedlicher Hersteller auf GATT-Ebene einheitlich ansteuern und auslesen lassen. Verwendet ein Gerät spezifische Services, ist die Dokumentation des Herstellers erforderlich, um die Characteristics korrekt zu interpretieren. Durch die Nutzung von Interfaces und abstrakten Klassen lässt sich die Softwarearchitektur so gestalten, dass in Zukunft neue Sensoren mit wenig Aufwand integriert werden können. Für die Bluetooth-Profile Herzfrequenz und Pulsoximetrie wird jeweils eine gemeinsame Codebasis implementiert, die für alle Sensoren dieser Art genutzt werden soll. Die Entwicklung einer vollständigen Programmierschnittstelle ist im Rahmen dieser Arbeit jedoch zu umfangreich, deshalb liegt der Schwerpunkt der Implementierung auf den Funktionen der Bluetooth-Notifications.

In den Anforderungen in Kapitel 4.7 ist bereits erwähnt, dass Linux als Betriebssystem und Java als Programmiersprache zum Einsatz kommen. Diese Entscheidungen haben weitreichenden Einfluss auf die gewählte Implementierung. Schließlich ist es in dem Zeitraum der Arbeit weder möglich noch sinnvoll, den gesamten Bluetooth-Stack neu zu implementieren. Stattdessen wird versucht, bestehende Funktionalität in Form von Bibliotheken zu verwenden, die auch in den meisten Fällen bereits getestet sind.

Abbildung 18 zeigt eine Übersicht über alle Softwarekomponenten auf der CD und deren Abhängigkeiten. Das Maven-Projekt *SensorplatformParent* verwaltet die Anwendung der Sensorplattform, die modular in weitere Projekte unterteilt ist. Zum Laden der Software in eine Entwicklungsumgebung (z. B. eclipse) reicht es aus, das Projekt *SensorplatformParent* auszuwählen. Anschließend werden alle dazugehörigen Projekte automatisch importiert. Das Projekt *SensorplatformApp* beinhaltet alle Komponenten der Sensorplattform, außer der Sensorabstraktion, die sich im *SensorplatformSensor*-Projekt befindet. Das Parent-Projekt beinhaltet unter anderem die Deklarationen für Bibliotheken und einige Anweisungen für die Installation. *WebHrs* ist ein separates Projekt für einen Webserver, zu dem Herzfrequenzdaten hochgeladen werden können (siehe Kapitel 6.7). Des Weiteren gibt es ein BGScript-Projekt für die Firmware des BLE113 Entwicklungsboards (siehe Kapitel 6.10.2) und einige Shellskripts für das Betriebssystem der Sensorplattform (Kapitel 6.9), die zur einfacheren Übersicht in der Abbildung als Projekt zusammengefasst sind. Die beiden zuletzt genannten Komponenten befinden sich im Unterverzeichnis *Resources/Firmware/BLE113* bzw. *Resources/Firmware/RaspberryPi3*. Die Startseite der Dokumentation für die Anwendung ist in der Datei *Resources/Documentation/Sensorplatform/SensorplatformParent/index.html* und die Java-API-Dokumentation ist in der Datei *Resources/Documentation/Sensorplatform/SensorplatformParent/apidocs/index.html*. Beide Dateien können jeweils im Browser geöffnet werden.

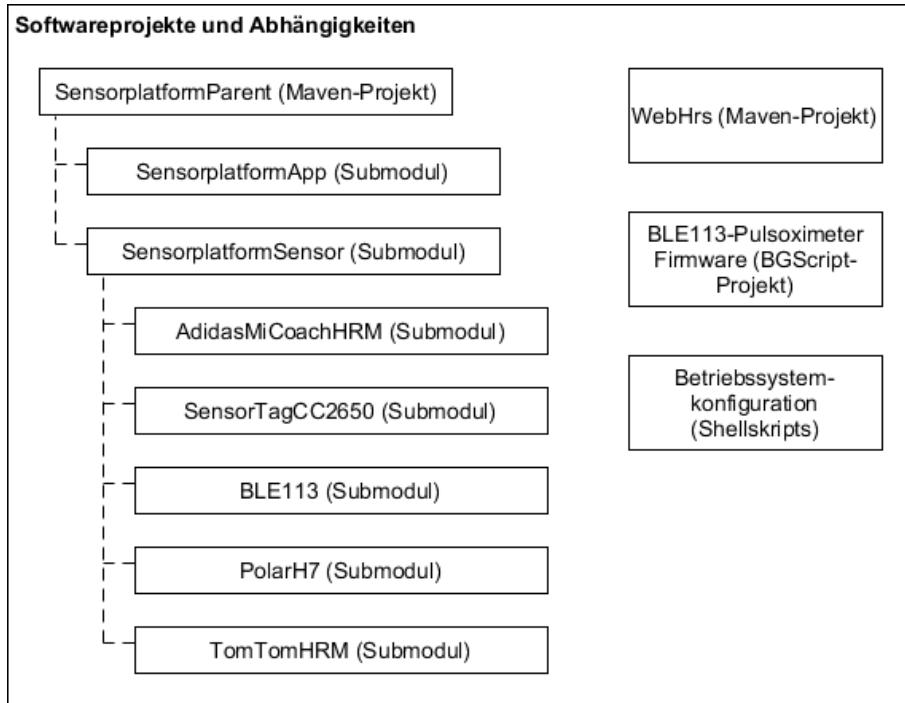


Abbildung 18: Softwareprojekte des Gesamtsystems der Sensorplattform und deren Abhängigkeiten

6.1. Bluetooth-Schnittstellen

Zu Beginn der Implementierung wird insbesondere für die Interaktion mit den GATT-Profilen eine einheitliche Softwareschnittstelle für die Bluetooth-Sensoren benötigt. Während bspw. für Android bereits eine ausgereifte Programmierschnittstelle für Bluetooth vorhanden ist, fehlen Java-Bibliotheken für native Linux-Betriebssysteme. Der von den Entwicklern von BlueZ bevorzugte Ansatz für die Schnittstelle zum Bluetooth-Treiber ist D-Bus [Jan16]. Dabei handelt es sich um ein für Linux-Betriebssysteme konzipiertes Framework für Interprozesskommunikation, das bereits bei vielen Programmen verwendet wird [frea]. Seit Version 5.42 ist die D-Bus-API von BlueZ offiziell freigegeben, die sich zuvor noch in einem experimentellen Stadium befand [Hed16]. Von der Java virtuellen Maschine kann nicht direkt auf die D-Bus-Objekte zugegriffen werden. Für D-Bus gibt es Programmierschnittstellen für verschiedene Sprachen. Die Java-Bibliothek wird jedoch seit 2009 nicht mehr aktualisiert, weshalb diese Lösung für die zukünftige Verwendung der Sensorplattform nicht sinnvoll ist [freb].

Alternativ kann durch Java Native Interface (JNI) auf die in C implementierte D-Bus-API von BlueZ zugegriffen werden. Das Projekt TinyB vom Intel IoT Development Kit bietet eine Programmierschnittstelle für Bluetooth Low Energy Funktionen für Java und C++, auf Basis von BlueZ und D-Bus an [Gitb]. Allerdings befindet sich dieses Projekt in einer frühen Phase der Entwicklung, sodass es für die Sensorplattform noch als ungeeignet bewertet wird. JNI kommt auch bei Android zum Einsatz. Android definiert dabei eine Programmierschnittstelle, die die darunterliegende Treibersoftware implementieren muss [And]. Eine solche Hardware-Abstraktionsschicht für die Sensorplattform zu implementieren würde den Umfang der Masterarbeit jedoch überschreiten bzw. den Schwerpunkt verschieben.

Die dritte Möglichkeit ist die Nutzung der Terminalprogramme von BlueZ. Diese sind standardmäßig in der Installation des BlueZ User-Space enthalten und ermöglichen dem Benutzer einen einfachen Zugang zu grundlegenden Bluetooth-Funktionen. Diese Programme haben keine grafische Oberfläche, sondern werden in der Konsole aufgerufen. Die folgenden vier Programme sind für die Sensorplattform relevant:

- Das Programm *gatttool* wird für die Interaktion mit einem GATT-Server auf einem anderen Bluetooth-Gerät verwendet. Damit lassen sich bspw. Werte von einer Characteristic auslesen und schreiben.
- Für allgemeine Einstellungen der lokalen Bluetooth-Hardware, für das Pairing und für weitere Interaktion mit Bluetooth-Geräten kommt die Anwendung *bluetoothctl* zum Einsatz.
- Zum Suchen der anderen Bluetooth-Geräte eignet sich das Programm *hcitool*. Für einen bestimmten Zeitraum werden damit Bluetooth Low Energy Geräte in der Nähe gefunden.
- Um die korrekte Funktionsweise einer Bluetooth-Verbindung zu prüfen (z. B. die Verschlüsselung) wird das Debug-Programm *btmon* verwendet. Damit werden zusammen mit einigen Zusatzinformationen für Entwickler gesendete und empfangene Bluetooth-Pakete dargestellt.

Das vom W3C entwickelte Framework Web-Bluetooth und speziell die Generic Sensor API eignet sich ebenfalls für die Interaktion mit GATT-Profilen der Bluetooth-Spezifikation [Worb, Wora]. Im Google Chrome Browser ist diese Schnittstelle bereits experimentell integriert, bei Firefox von Mozilla jedoch noch nicht. Alternativ kann auch ein NodeJS-Server auf der Sensorplattform dafür genutzt werden. Mit Web-Bluetooth lassen sich in der Browser-Umgebung Bluetooth Smart Geräte in Javascript ansteuern. Zur Zeit ist diese Schnittstelle ebenfalls noch in einem frühen Entwicklungsstadium. Auf Grund des weitreichenden Einflusses des W3C ist jedoch zu erwarten, dass dieses Framework in Zukunft häufig bei der Anwendungsentwicklung von Geräten des Internet-of-Things (IoT) genutzt wird.

Die Schnittstelle zu dem Bluetooth-Stack BlueZ wird durch Interprozesskommunikation realisiert. Für die GATT-Funktionen eines Bluetooth-Geräts wird ein Gatttool-Prozess für jeden genutzten Sensor von der Sensorplattform-Anwendung erzeugt. Durch den Input-, Output- und Errorstream können Nachrichten zwischen den Prozessen ausgetauscht werden. Abbildung 19 zeigt den daraus resultierenden Software-Stack. So wie der Benutzer einen Befehl zum Verbinden oder zum Auslesen eines Handles in die Konsole eintippen und mit Enter bestätigen kann, ist es auf gleiche Weise möglich, die Eingabe in Software durch einen anderen Prozess durchführen zu lassen. Die Anwendung hat dabei keine Kenntnisse über die internen Funktionen von BlueZ, die durch das Gatttool aufgerufen werden. Das Programm bietet verschiedene Befehle, um mit einem GATT-Server interagieren zu können, bspw. zum Auslesen einer Characteristic oder zum Aktivieren von Notifications. Die Ergebnisse eines Befehls oder die Fehlermeldung inklusive Fehlercode werden von Gatttool in Textform ausgegeben.

Diese Vorgehensweise bietet eine schnelle Implementierung der benötigten Bluetooth-Schnittstelle, weil keine Einarbeitungszeit in eine umfangreiche API wie bspw. D-Bus erforderlich ist. Für die Masterarbeit ist es daher die bevorzugte Lösung. Die Interprozesskommunikation mit den Gatttools bringt aber auch einen Nachteil mit sich. Mehrere Prozesse, die gestartet werden (jeweils ein Gatttool pro Sensor), erzeugen mehr Overhead im System. Daraus kann beispielsweise mehr Last für den Prozessor entstehen. Ob diese Implementierung tatsächlich quantifizierbar mehr Leistung benötigt, wird in den Tests in Kapitel 7.3 überprüft.

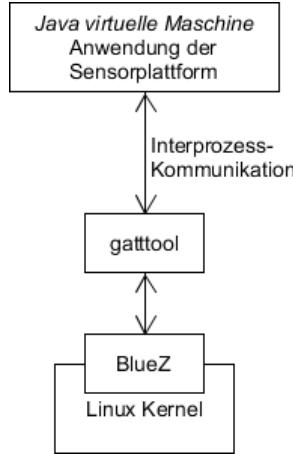


Abbildung 19: Software-Stack der Sensorplattform

6.2. Softwarearchitektur der Sensoren

Die Kernfunktionalität der Sensorplattform stellt die Bluetooth-Schnittstelle dar. Die Softwarearchitektur berücksichtigt die GATT-Profile der Bluetooth-Spezifikation, sodass die Sensoren allgemeingültig angesteuert werden. Gleichzeitig ist damit die Erweiterbarkeit der Sensorplattform für neue Sensoren gewährleistet. Die Implementierung lässt sich in mehrere Klassen unterteilen, die im Folgenden vorgestellt werden.

6.2.1. Gatttool

Wie bereits in Kapitel 6.1 erläutert, wird das Terminalprogramm Gatttool (in der Konsole gatttool) verwendet, um mit den Sensoren zu kommunizieren. Der Befehl `gatttool -I -t public -b 11:22:33:44:55:66` startet das interaktive Programm (Parameter `-I`) für jeweils einen Sensor. Mit dem Parameter `-t` wird ein Typ der Bluetooth-Adresse angegeben. Während die Bluetooth-Spezifikation drei Typen definiert (siehe Kapitel 3.1.4), erwartet das Gatttool von BlueZ nur die Typen `public` und `random`. Für statische Adressen wird ebenfalls `random` als Parameter verwendet. Diese Inkonsistenz ist bei der Benutzung von Gatttool zu berücksichtigen und wird auch im Programmcode entsprechend deklariert (siehe Quelltext 1). Die String-Präsentation des Enumeration-Feldes `STATIC` lautet `random` (rot markiert), wenn diese Angabe in den Stream zum Gatttool geschrieben wird. Mit dem letzten Parameter `-t` wird die Bluetooth-Adresse angegeben.

```

1  public enum AddressType {
2      PUBLIC("public"), RANDOM("random"), STATIC("random");
3
4      private final String type;
5
6      private AddressType(String type) {
7          this.type = type;
8      }
9
10     @Override
11     public String toString() {
12         return this.type;
13     }
14 }
  
```

Quelltext 1: Enumeration für Bluetooth-Adresstypen mit roter Markierung für die Sonderregel

Aus den resultierenden Bluetooth-Verbindungen ergibt sich eine Stern-Netzwerktopologie mit der Sensorplattform als Mittelpunkt. Theoretisch wären auch verschiedene Maschen-

Netzwerke mit Bluetooth möglich (ab Version 4.1). Die größte Flexibilität bietet aber die Stern-Netzwerktopologie, da sich die Sensoren nicht untereinander kennen müssen, sondern direkt mit der Sensorplattform verbunden sind. Ein weiterer Vorteil ist, dass sämtliche Aktionen immer von der Sensorplattform initiiert werden. Im Fehlerfall, bspw. bei einem Abbruch der Bluetooth-Verbindung, ist auch nur genau ein Sensor betroffen.

Im Wesentlichen wird in der Anwendung - neben dem Hauptthread - noch ein weiterer Thread pro Gatttool benötigt. Da die gesamte Kommunikation mit BlueZ asynchron verläuft, wird der Hauptthread dafür genutzt, Kommandos zu senden, was dem Schreiben in den OutputStream entspricht. Der zweite Thread versucht kontinuierlich, solange der jeweilige Prozess existiert, zeilenweise vom InputStream zu lesen. Die Benennung der Richtung der Streams erfolgt immer aus Sicht der Java-Anwendung. Mit Hilfe von regulären Ausdrücken kann die Anwendung der Sensorplattform auf bestimmte Nachrichten, die das Gatttool meldet, reagieren (z. B. auf Notifications). Mittels des Observer-Patterns wird dann die Notification an die Klasse SensorWrapper weitergeleitet. Bevor die Funktionsweise des SensorWrappers vorgestellt wird, sind zunächst jedoch einige weitere Softwaremodule erforderlich.

6.2.2. GattLibrary

Um einen neuen Sensor zu integrieren, ist es sinnvoll, den GATT-Server des Geräts auszulesen. Informationen dazu werden in der Klasse GattLibrary gesammelt. Ein Universally Unique Identifier (UUID) kann einer bestimmten Funktion zugeordnet werden, Daraus ergibt sich ein Bild über den gesamten Funktionsumfang eines Bluetooth Low Energy Geräts. Beispielsweise zeigt die UUID 0x2A37 an, dass es sich um die Daten einer Herzfrequenzmessung handelt. Um die UUIDs zu ermitteln, sind beide Terminalprogramme Gatttool und Bluetoothctl (bluetoothctl) geeignet. Das Auslesen des GATT-Servers ist vergleichbar mit einer Tiefensuche in einem Baum, wobei die Services Knoten und die Characteristics, Descriptor und Properties die Blätter darstellen. Allerdings nutzt das Gatttool bei vielen Eingaben hexadezimale Handle-Adressen (z. B. 0x0123), nicht aber die eindeutigen UUIDs. So wird bspw. von Notifications nur die Handle-Adresse mitgeliefert, wodurch die Referenz zu der UUID verloren geht. Die GattLibrary der Sensorplattform enthält demnach sensorspezifische Informationen und wird zusätzlich zur Verbesserung der Lesbarkeit des Codes erstellt. Dabei wird eine Semantik für Handle-Adressen definiert, z. B. HANDLE_HEART_RATE_MEASUREMENT = 0x003a. Die Sensorklassen, die im Folgenden vorgestellt werden, nutzen die in der GattLibrary deklarierten Handle-Adressen für die Steuerung des Sensors.

6.2.3. Sensorklassen

Das Ziel bei der Implementierung ist es, die GATT-Profile aus der Bluetooth-Spezifikation allgemeingültig für Kategorien von Sensoren zu verwenden. Besitzt der GATT-Server gerätespezifische Services, ist dies jedoch nicht mehr möglich. Abbildung 20 zeigt das Klassendiagramm für Sensorklassen, die unterschiedliche Services nutzen. Das Interface SensorCommands definiert zunächst vier Methoden, die die Sensorplattform verwendet, um mit Bluetooth-Sensoren zu kommunizieren. Später kann dieses Interface um neue Methoden erweitert werden. Zu Beginn lassen sich damit allgemein Benachrichtigungen für Messungen des Sensors aktivieren/deaktivieren und den Batteriestatus auslesen.

Die Interfaces HeartRateSensor und PulseOximeterSensor definieren jeweils alle Methoden, mit denen die Daten aus den jeweiligen Bluetooth-Services verarbeitet und berechnet werden können. Diese Methoden sind in abstrakten Klassen implementiert und bilden für jeden Bluetooth-spezifikationskonformen Sensor die gleiche Codebasis. Zum Beispiel erben alle drei Herzfrequenzsensoren aus der gleichen Klasse AbstractHeartRateSensor.

Ebenfalls in Abbildung 20 erkennbar, erbt der CC2650 Sensor direkt aus der Klasse Ab-

stractSensor, weil die Services von diesem Sensor nicht der Bluetooth-Spezifikation entsprechen. Weil diese auch nur in dem einen Sensor vorkommen, ergibt es keinen Sinn, dafür eine abstrakte Oberklasse zu definieren, da niemals mehr als genau ein Sensor von dieser erben wird. Die Berechnungen sind folglich alle direkt in der Klasse CC2650 deklariert und dieser Sensor erbt aus der allgemeinsten Sensorklasse, nämlich AbstractSensor.

Die Klassennamen in der Implementierung der konkreten Sensoren entsprechen nicht zwangsläufig den Namen, die auch bei der Suche nach Bluetooth-Geräten angezeigt wird. Ein Beispiel wird in Kapitel 5.2.2 genannt. Es ist offensichtlich, dass in den meisten Programmiersprachen Leerstellen, Klammern und Punkte keine gültigen Zeichen für Klassennamen sind. Reflection, die indirekte Instanziierung eines Programmierobjekts durch den Namen der Klasse als String, kommt daher nicht als Möglichkeit in Frage. Daher muss im Programm auf andere Weise von einem String auf eine bestimmte Klasse verwiesen werden. Die Sensorplattform definiert dafür eine Enumeration, um eine Form von Typ-Sicherheit zu bieten. Der Sensorname - sofern gültig - wird zuerst zur Enumeration umgewandelt. Ein Switch-Case-Block prüft anschließend, welches Sensorobjekt erzeugt wird. Der Benutzer muss jedoch wissen, welcher Anzeigename aus der Suche über die Webanwendung mit welchem physischen Sensor übereinstimmt. Anstelle von *HRM (Ver0.4)* muss dieser beispielsweise *TomTomHRM* in der Sensorkonfiguration eingegeben. In der Praxis könnte das zu Verwirrung des Benutzers führen, da Geräte über die Suche nach Bluetooth-Geräten unter Umständen nicht direkt identifiziert werden können. Für die Instanziierung eines bestimmten Sensorobjekts könnte alternativ auch das Factory-Pattern genutzt werden.

Bei den Sensorklassen (GattLibrary, AbstractSensor, SensorWrapper) wird außerdem Gebrauch von Generics gemacht. Damit werden sensorspezifische Klassen sehr früh und allgemeingültig in dem Vererbungsbaum mit Platzhaltern deklariert, aber erst bei der Instanziierung eines konkreten Objekts zur Laufzeit erfolgt die Zuweisung eines bestimmten Datentyps. Auf diese Weise kann das Konzept der Vererbung genutzt werden, auch wenn die Sensoren sehr unterschiedlich voneinander sind. Ohne den Einsatz von Generics würde es zu Code-Duplikierungen kommen. Diese Zugriffe auf die Sensorklassen können nicht durch Polymorphie realisiert werden, da die Sensoren untereinander zu verschieden sind.

Falls ein Sensor einen GATT-Server mit mehreren Bluetooth-Services aus der Spezifikation besitzt, ist es erforderlich, aus einer abstrakten Klasse zu erben und die Interfaces der anderen Services zu implementieren. Schließlich ist das Erben aus mehreren Klassen nicht möglich.

Die Sensorobjekte stellen während der Aufzeichnung Factories dar, die mit gegebenen Daten der Notifications als Eingabe, Sample-Programmierobjekte (Datenbank-Entities für Hibernate) erzeugen und zurückgeben (siehe Kapitel 6.4).

Die Abstraktion für die physischen Sensoren ist mit dieser Vererbungshierarchie abgeschlossen. Sowohl die Berechnungen für die Bluetooth-Characteristics als auch die individuelle Ansteuerung der unterschiedlichen GATT-Server der Sensoren ist mit diesen Klassen abgedeckt. Nachfolgend wird der Kontrollfluss der Messdaten erläutert.

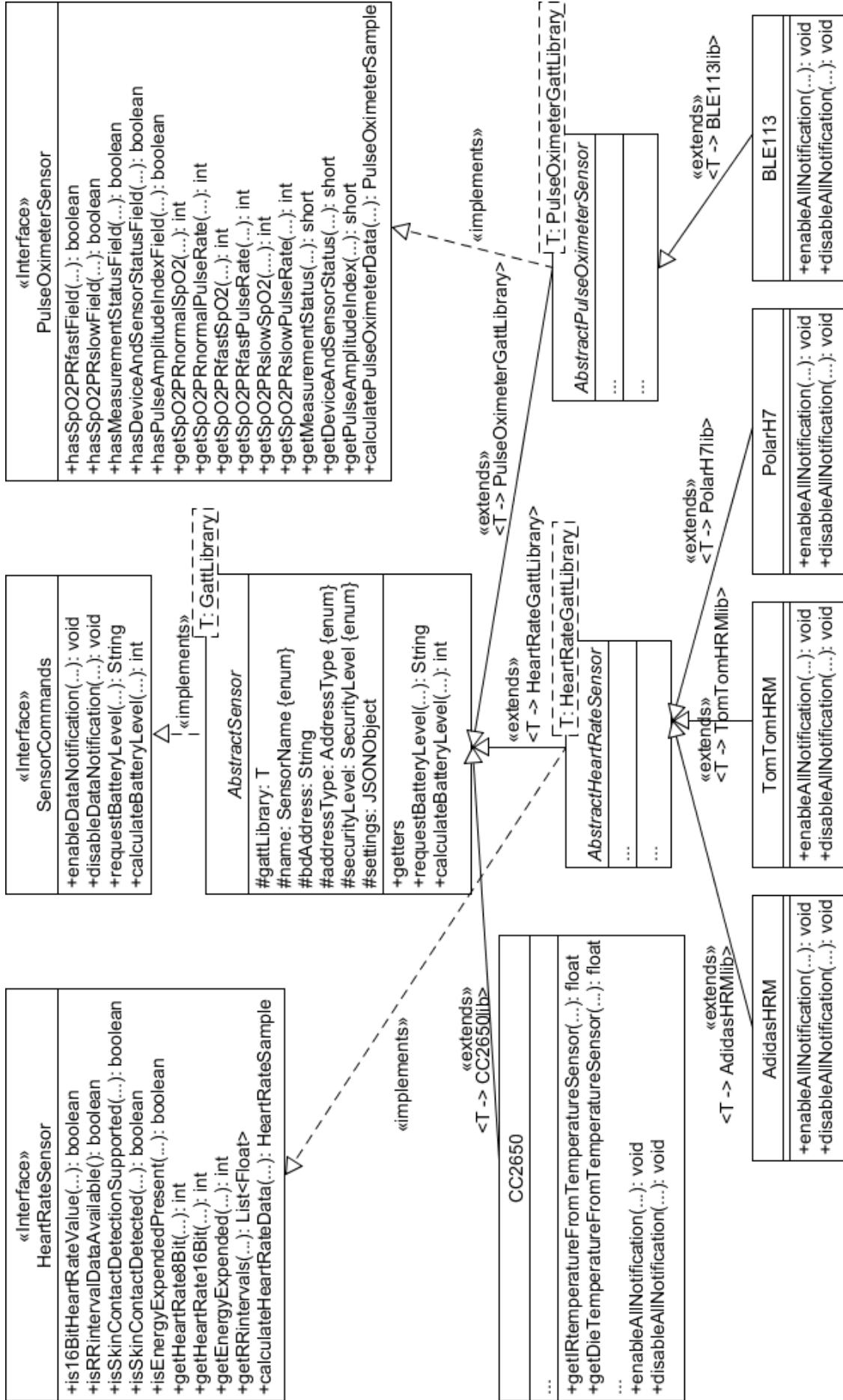


Abbildung 20: Klassendiagramm der Sensoren

6.2.4. SampleCollector

Die Klasse SampleCollector wird im Wesentlichen für die Interaktion mit der Datenbank und für das Hochladen der Daten benötigt. Die Daten vom BLE113 Entwicklungsboard als Pulsoximeter und vom CC2650 werden nur in der Datenbank abgespeichert, jedoch zu keinem externen Webserver übertragen. Die Programmabläufe sind bei diesen beiden Sensorwerten dementsprechend trivial. Komplizierter wird es bei den Herzfrequenzsamples, deren Verarbeitung in der Sensorplattform im Folgenden erläutert wird.

Abbildung 21 zeigt das Aktivitätsdiagramm für das Verarbeiten der Herzfrequenzdaten. Startzustand ist immer der schwarze Kreis, Endzustand der schwarze Kreis mit weißem Rahmen. Solange eine Aufzeichnung andauert, befindet sich der HeartRateSampleCollector in einer Schleife und arbeitet eine Warteschlange von Herzfrequenzdaten ab. Erkennbar ist die Schleife an den Verbindungen, die zum Startzustand zurückführen. Der Abbruch einer Aufzeichnung führt zum Abspeichern der verbleibenden Daten in die Datenbank. Gleicher ist der Fall, wenn die Warteschlange mehr als hundert Elemente aufgenommen hat. Das ist wichtig, damit bei einer Unterbrechung der Spannungsversorgung nicht zu viele Daten verloren gehen. Das Abspeichern von hundert Samples dauert zwischen einer und zwei Sekunden. In dieser Zeit werden weiterhin neue Herzfrequenz-Samples in die Queue aufgenommen, sodass es nötig ist, die hundert Samples in einen Zwischenspeicher zu schreiben. Ohne diesen Buffer würde die Sensorplattform bestimmte Samples gar nicht versuchen an den Webserver zu übertragen. Das Volllaufen der Warteschlange kann nur passieren, wenn das Übertragen zum Webserver im Durchschnitt länger als eine Sekunde pro Sample dauert. Das Aufzeichnen von (Herzfrequenz-) Samples geschieht dann schneller als das Hochladen. Der Grund dafür kann eine schlechte Signalqualität des Mobilfunks sein oder spontane Verbindungsabbrüche z. B. in Tunnels oder in einem Keller. Genauer analysiert werden die Zusammenhänge in Kapitel 7.3.3.

Die Samples werden zum Webserver übertragen, wenn dies energieeffizient möglich ist. Maßgebend für diese Entscheidung sind die Parameter RSSI, RSCP und E_c/I_o , die kontinuierlich erfasst und ausgewertet werden (siehe Kapitel 6.6). Ist die Übertragung erfolgreich, wird der Wert einer Boolean-Variable des Sample-Objekts zur Markierung auf *true* gesetzt. Danach wäre es nicht mehr erforderlich, die hochgeladenen Messdaten eines Samples in die Datenbank zu speichern. Zum Testen wird dies dennoch gemacht, um den relativen Anteil an nicht hochgeladenen Samples eines Aufzeichnungszeitraums zu bestimmen (siehe Kapitel 7.3.3).

Die SampleCollector sind während der Laufzeit einzigartig, das heißt per Singleton-Pattern und durch das Dependency-Injection-Framework Guice einmalig instanziert [Gita]. Mehrere SensorWrapper (Kapitel 6.2.5) können den selben SampleCollector verwenden. Damit ist gewährleistet, dass alle Samples eines Typs (z. B. Herzfrequenz) auf die gleiche Art und Weise in der Sensorplattform verarbeitet werden. Auch wäre es möglich, mehrere von verschiedenen Personen getragene Herzfrequenzsensoren mit einer einzigen Sensorplattform zu betreiben. Allerdings dürften sich die Personen nicht weiter als 10 Meter von der Sensorplattform entfernen (siehe Test zur BLE-Reichweite in Kapitel 7.3.4).

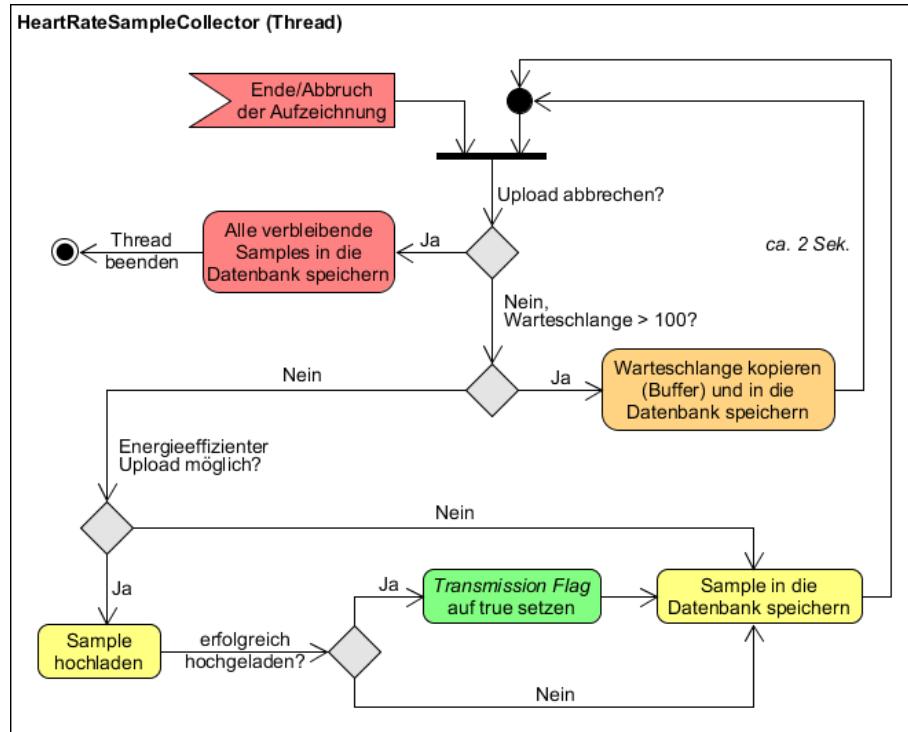


Abbildung 21: UML Aktivitätsdiagramm des HeartRateSampleCollectors

6.2.5. AbstractSensorWrapper

Die Klasse AbstractSensorWrapper vereint die Programmierobjekte Gatttool, Sensor und SampleCollector. Für die Bluetooth-spezifikationskonformen Sensoren stehen außerdem die Klassen HeartRateSensorWrapper und PulseOximeterSensorWrapper zur Verfügung, die jeweils für alle Sensoren eines Typs genutzt werden können (z. B. Herzfrequenzsenso-ren). Für die anderen Sensoren muss ein eigener SensorWrapper (z. B. CC2650SensorWrap-per) erstellt werden. Generics finden ebenfalls bei den Klassen der SensorWrapper Anwendung. Der SampleCollector ist das einzige Objekt, das in mehreren SensorWrappern vorkommen kann. Der SensorWrapper strukturiert die Software-Komponenten hinsichtlich des Datenflusses. An dieser Stelle wird das Beispiel eines HeartRateSensorWrapper vorgestellt.

Abbildung 22 zeigt die Verarbeitung eines Herzfrequenz-Samples in der Sensorplattform. Gezeigt ist das Sequenzdiagramm ab dem Erhalt einer Notification bis zum Abspeichern in der Datenbank. Das Programmierobjekt Gatttool liest kontinuierlich die Nachrichten vom InputStream des Gatttool-Prozesses (Input aus der Sicht der Java-Anwendung). Durch einen regulären Ausdruck wird jede Notification erkannt und durch Nutzung des Observer-Patterns an den HeartRateSensorWrapper weitergeleitet. Eine Notification besteht immer aus der Handle-Adresse und dem dazugehörigen Wert. Das Gatttool nutzt dafür eine hexadezimale Darstellung. Im Folgenden wird eine Notification vom Polar H7 gezeigt und ausgewertet. Zu beachten ist, dass für Werte, die mindestens 16 Bit lang sind, die Darstel-lung Least Significant Octet (LSO) zu Most Significant Octet (MSO) genutzt wird. Die Angabe der RR-Intervalle erfolgt in der Auflösung 1/1024 Sekunden.

- Beispiel 0x0013 16 44 9b 03 74 03
- Handle-Adresse: 0x0013
- Kontrollbyte Bitmaske: $16_{16} \equiv 00010110_2$
 - Bit 1 (Wert 0): Herzfrequenz als UINT8

- Bit 2 und 3 (Wert 11): Hautkontakt-Markierung ist darstellbar und es besteht Hautkontakt
 - Bit 4 (Wert 0): Der Wert für Energieverbrauch ist nicht vorhanden
 - Bit 5 (Wert 1): Mindestens ein RR-Intervall ist vorhanden (als `UINT16`)
 - Bit 6-8 (Wert 000): Reserviert (eventuelle Nutzung in der Zukunft)
- Herzfrequenz pro Minute: $44_{16} \equiv 68_{10}$
 - RR-Intervall 1: $9b\ 03_{16} \Rightarrow 039b_{16} \Rightarrow 923_{10}/1024 * 1000 = 901.3671875_{10}$ ms
 - RR-Intervall 2: $74\ 03_{16} \Rightarrow 0374_{16} \Rightarrow 884_{10}/1024 * 1000 = 863.28125_{10}$ ms

Der HeartRateSensorWrapper hat keine Kenntnis darüber, wie die hexadezimalen Daten in eine lesbaren Darstellung umgerechnet werden können. Wie ebenfalls in Abbildung 22 dargestellt, führt das Sensorobjekt die Berechnungen mit Methoden aus der Klasse `AbstractHeartRateSensor` aus und erstellt durch Hinzunahme einiger Metadaten ein `HeartRateSample`. Diese ist als Datenbank-Entity annotiert. Das heißt, das Framework Hibernate kann ein solches `HeartRateSample`-Objekt direkt in die Datenbank speichern und wieder laden [JBo]. Die dafür nötigen SQL-Befehle führt Hibernate implizit aus, diese müssen nicht vom Anwendungsentwickler geschrieben werden. Anschließend wird das Sample-Objekt in die Warteschlange des `HeartRateSampleCollectors` hinzugefügt.

In dem Sequenzdiagramm in Abbildung 22 ist auch der (vereinfachte) Ablauf des `HeartRateSampleCollectors` dargestellt. In dem Beispiel sendet dieser Daten vom Sensor Polar H7 an das TeLiPro-Webinterface. Über die Klasse `DBSession` erfolgt jeweils eine Interaktion mit der Datenbank: Transaktion beginnen, committen und wieder schließen.

Allgemein lässt sich festhalten, dass die Erhebung und die Weiterleitung der Sensordaten zwei getrennte Abläufe in der Sensorplattform darstellen. Durch die Verwendung mehrerer Threads werden beide Abläufe verknüpft und parallel ausgeführt. Durch die Parallelisierung ist sichergestellt, dass möglicherweise auftretende Latenzen (z. B. beim Hochladen von Samples) den jeweils anderen Ablauf nicht beeinflussen bzw. verzögern.

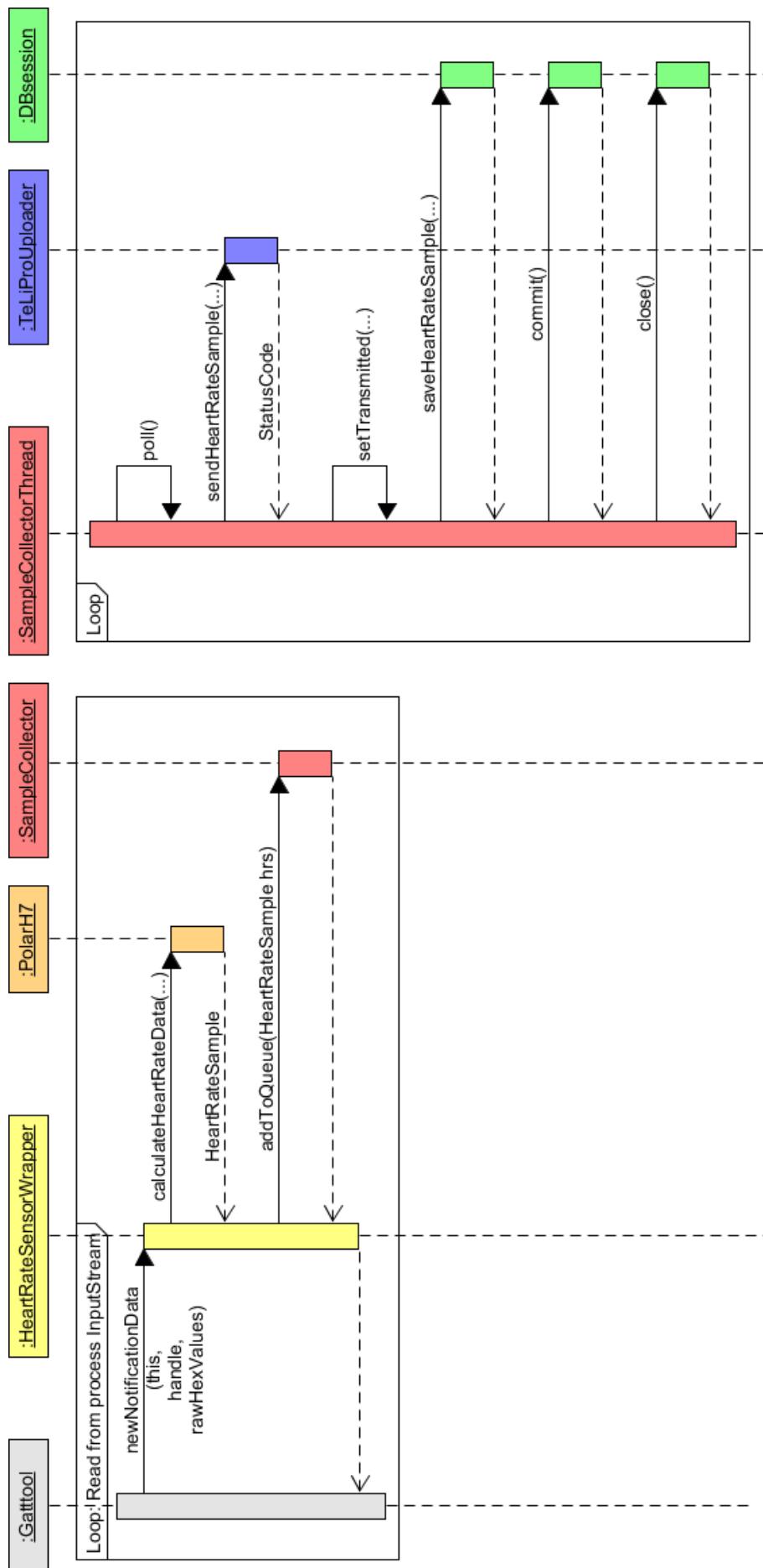


Abbildung 22: Sequenzdiagramm für die Verarbeitung von Herzfrequenz-Samples

6.3. Webanwendung der Sensorplattform

Wie bereits in Kapitel 5.2 (Mensch-Maschine-Interaktion) erwähnt, nutzt der Anwender eine Webanwendung, um die Sensorplattform zu konfigurieren und auszulesen. Zunächst wird das Backend und im Anschluss daran das Frontend beschrieben.

Die Grundlage für die Webanwendung ist die Bibliothek Jetty. Jetty bietet einen HTTP-Server und Servlet-Container, um sowohl statische als auch dynamische Inhalte bereitzustellen [Theb]. Der Webserver wird im Prozess der restlichen Anwendung der Sensorplattform ausgeführt. Das Dependency-Injection-Framework Guice eignet sich in Verbindung mit Jetty, um das Laden von Abhängigkeiten in Klassen und projektspezifischen Eigenschaften (properties) zu steuern. Auf Grund der engen Koppelung wird Guice zusammen mit der Webanwendung vorgestellt. Sobald beispielsweise eine passende HTTP-Anfrage an der URL <https://sensorplatform.fit.fraunhofer.de:8080/controller/start> empfangen wird, erzeugt Jetty eine Instanz der REST-Service-Klasse ControllerService. Alle Abhängigkeiten werden dem Konstruktor automatisch durch Guice übergeben. Der Datenaustausch zwischen Frontend und Backend erfolgt über diese Schnittstellen, deren HTTP-Methoden das JSON-Format verwenden.

Darüber hinaus bietet Guice einige Konfigurationsmöglichkeiten (z. B. Patterns) an, um der Implementierung eine bestimmte Struktur zuzuweisen. Um die Fehleranfälligkeit zu verringern, werden durch Guice alle Kernklassen, die den Programmablauf steuern oder interne Zustände haben, explizit als Singleton (Pattern) deklariert. Das bedeutet, zur Laufzeit erzeugt Guice ein Objekt jeder Klasse und verwaltet diese anschließend. Das Framework stellt das Objekt dann bei jeder Deklaration im weiteren Programmablauf bereit. Nachfolgend sind die Klassen aufgelistet, die nach diesem Konzept implementiert sind.

- Controller: Die Klasse die den allgemeinen Kontrollfluss der Sensorplattform Anwendung steuert.
- DBcontroller: Die Verbindung zur Datenbank. Die Klasse dient auch als Factory für Sessionobjekte für einzelne geregelte Interaktionen mit der Datenbank.
- HeartRateSampleCollector: Die Klasse für Interaktion von Herzfrequenzdaten mit der Datenbank und der Upload-Funktion.
- PulseOximeterSampleCollector: Speicherung der Pulseoximeterdaten in die Datenbank.
- CC2650SampleCollector: Speicherung der CC2650 Sensordaten in die Datenbank.
- SensorWrapperFactory: Keine direkte Kontrollklasse, auf Grund der Referenz zu den SampleCollector ist es jedoch sinnvoll, diese ebenfalls als Singleton zu deklarieren.

Der Jetty-Webserver ist zudem auch auf einem eingebetteten System mit begrenzten Hardware-Ressourcen effizient einsetzbar. Die sicherheitsrelevanten Aspekte des Webservers werden separat in Kapitel 6.10.1 erläutert. Im Folgenden wird noch das Frontend vorgestellt.

Die Webanwendung nutzt das CSS-Framework Bootstrap und bietet damit eine grafische Schnittstelle für die Verwendung aller grundlegenden Funktionen der Sensorplattform [Twi]. Abbildung 23 zeigt die Startseite der Webanwendung. Eine Beispielkonfiguration für eine Aufzeichnung mit den beiden Sensoren Polar H7 und CC2650 ist dargestellt. Zu sehen sind auch einige allgemeine Informationen, bspw. zu welchem externen Webinterface die Sensorplattform Messdaten hochlädt oder die Signalqualität der Mobilfunkverbindung. Ebenfalls erkennbar sind die einzelnen Tabs in der Webanwendung für die Mensch-Maschine-Interaktion aus Kapitel 5.2.

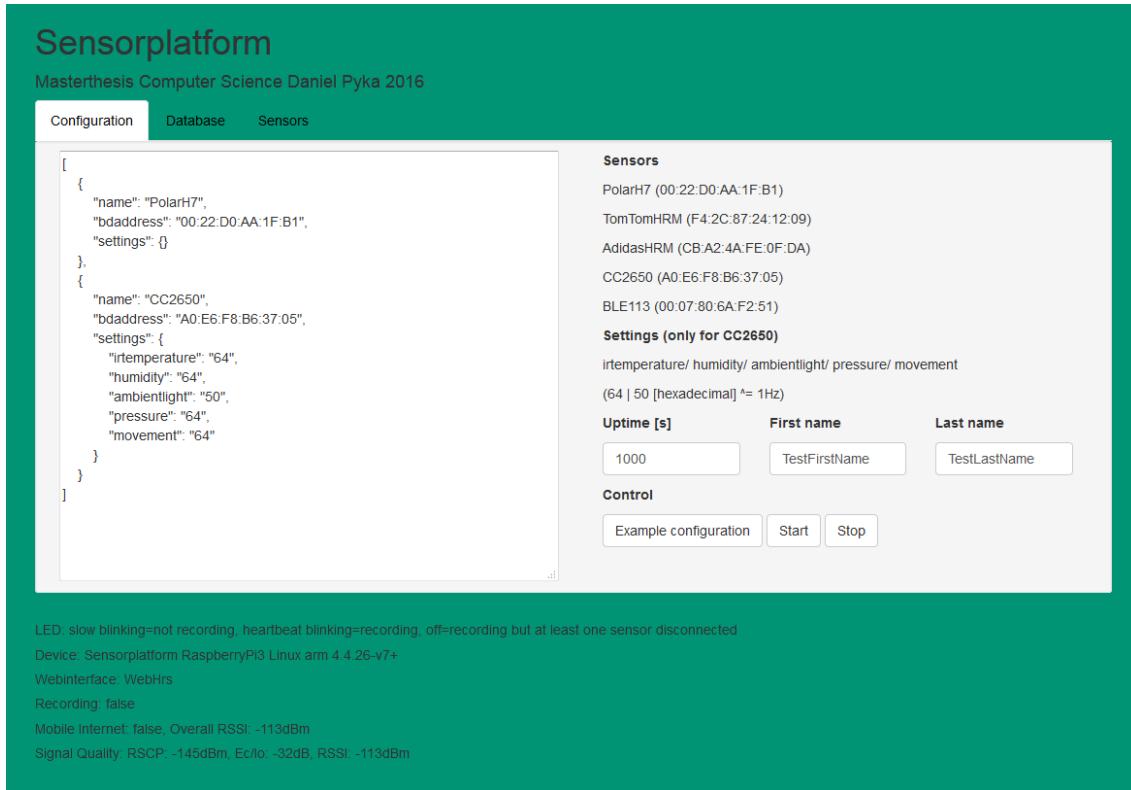


Abbildung 23: Webanwendung der Sensorplattform

6.4. Datenbank

Die Datenbank auf der Sensorplattform ist Hyper Structured Query Language Database (HSQLDB) Version 2.3.4, die unter anderem bei Open Office verwendet wird. HSQLDB ist eine vollständig in Java implementierte, relationale Datenbank und lässt sich mit wenig Aufwand als eingebettete Lösung für die Anwendung der Sensorplattform einsetzen. Die Tabellen der Sensorplattform sind einfach gehalten, sodass keine komplizierten Funktionen wie z. B. Joins erforderlich sind. Die Datenbank speichert die Messwerte und reichert diese mit einigen Metainformationen an. Die Größe (Anzahl der Zeilen) in den Tabellen ist durch die Abtastrate der Sensoren und die Aufzeichnungsdauer definiert, dazu mehr in Kapitel 7.3.2. Tabelle 6 zeigt die Datenfelder für Herzfrequenz- und Pulsoximetriesamples in der Datenbank.

Herzfrequenz	Pulsoximetrie
<ul style="list-style-type: none"> • ID (Primärschlüssel) • Vorname (Metainformation) • Nachname (Metainformation) • Bluetooth-Adresse (Metainformation) • Zeitstempel (Metainformation) • Herzfrequenz • Energieverbrauch • RR-Intervalle (String-Repräsentation einer ArrayList in Java, z. B. [978.5, 955.25, 994.0]) • Boolean-Markierung, ob ein Sample erfolgreich hochgeladen wurde. 	<ul style="list-style-type: none"> • ID (Primärschlüssel) • Vorname (Metainformation) • Nachname (Metainformation) • Bluetooth-Adresse (Metainformation) • Zeitstempel (Metainformation) • Pulsrate • SpO₂

Tabelle 6: Datenbankschemata für Herzfrequenz- und Pulsoximetriedaten

Es ist nicht beabsichtigt, alle erforderlichen Metadaten an dieser Stelle zu ermitteln. Diese Auswahl bezieht sich hauptsächlich auf die vom TeLiPro-Webinterface benötigten Daten (siehe 6.7), das sich aber auch noch in der Entwicklungsphase befindet. Die Daten sind demnach nur eine ungefähre Referenz für den späteren Einsatz.

Alle Daten von HSQLDB werden regelmäßig in einer *.script-Datei in Klartext auf der micro-SD-Karte gespeichert. Die Datenbank selbst kann nur über das Webinterface der Sensorplattform angesprochen werden (Abbildung 23 Tab Database), da eine direkte Verbindung weder möglich noch gewünscht ist. In der Webanwendung ist es für das medizinische Fachpersonal auch möglich, das Übertragen von Samples an den externen Webserver nachzuholen, wenn dies zuvor bspw. auf Grund von zu schlechter Signalqualität nicht möglich war. Wie bereits erwähnt wird das Framework Hibernate verwendet, um durch Programmierobjekte (Entity-Klassen) mit der Datenbank interagieren zu können.

6.5. HardwarePlatform

Die Anwendung der Sensorplattform greift durch Funktionen des Betriebssystems auf Hardwarekomponenten zu, die gegebenenfalls nicht auf jedem Board identisch sind. Das Interface umfasst Methoden für die Steuerung einer LED, zum Abfragen und Berechnen von Parametern der Mobilfunkverbindung und für die Suche nach Bluetooth-Geräten. Diese Funktionen sind im Interface HardwarePlatform definiert, das im UML-Klassendiagramm in Abbildung 24 dargestellt ist. Für jede Leiterplatte, die als Sensorplattform zum Einsatz kommt, gibt es eine Klasse, die dieses Interface implementiert. Für den Raspberry Pi ist die Klasse vollständig programmiert, für das Cubieboard 3 jedoch nur exemplarisch mit leeren Methoden hinzugefügt. Die Interaktion mit dem Mobilfunkmodul erfolgt durch die Klasse Huawei_E352S_5, bei der es sich um eine spezifische Implementierung für den Surfstick von Huawei handelt. Das Interface Runnable stammt aus der Java-API und liefert eine Methode, die beim Aufruf explizit in einem anderen Thread ausgeführt wird.

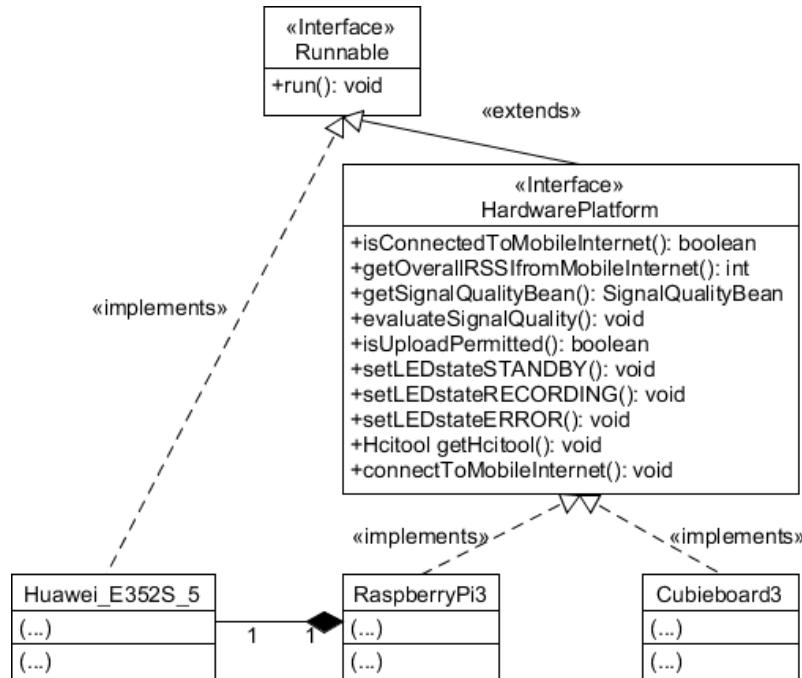


Abbildung 24: UML-Klassendiagramm der Schnittstelle zur Hardware

Die Auswahl eines bestimmten Maven-Profil beim Kompilieren der Anwendung bestimmt, welche konkrete Hardwareplattform genutzt wird (z. B. `mvn clean install -P raspberrypi3`). Die Profile beinhalten projektspezifische Einstellungen, die der Maven-Buildprozess in

die Properties-Dateien der Anwendung schreibt. Quelltext 2 zeigt diese Einstellung des Dependency-Injection-Frameworks Guice, um konkrete Klassen zum Interface Hardware-Platform zu binden (Zeile 5 und 10). Nebenbei werden diese Klassen auf gleiche Weise mit dem Singleton-Pattern aus Kapitel 6.3 deklariert. Falls kein Profil angegeben ist, kann die Sensorplattform nicht richtig funktionieren und der Prozess wird mit dem Fehlercode -1 beendet (Zeile 14).

```
1 switch (targetPlatform) {
2     case "raspberrypi3":
3         LOG.info("target platform is {}", targetPlatform);
4         bind(RaspberryPi3.class).in(Singleton.class);
5         bind(HardwarePlatform.class).to(RaspberryPi3.class);
6         break;
7     case "cubieboard3":
8         LOG.info("target platform is {}", targetPlatform);
9         bind(Cubieboard3.class).in(Singleton.class);
10        bind(HardwarePlatform.class).to(Cubieboard3.class);
11        break;
12    default:
13        LOG.error("unknown target platform {}", targetPlatform);
14        System.exit(-1);
15    break;
16 }
```

Quelltext 2: Guice-Konfiguration der Sensorplattform für eine konkrete Hardware

6.6. Schnittstelle zum Mobilfunkmodul

Die Mobilfunkverbindung ist mit dem Surfstick E352S-5 von Huawei (siehe Kapitel 5.1.1) umgesetzt, der über einen USB-Port an der Sensorplattform angeschlossen ist. Die Implementierung sowie Konfiguration des Treibers ist konkret für diesen Surfstick mit der Telekom als Provider ausgelegt. Die Sensorplattform hat an dieser Stelle eine enge Kopplung mit der Hardware, sodass Mehraufwand bei der Nutzung des Systems mit einem anderen Surfstick entsteht. Für dieses Modem gibt es den Linux-Treiber *huawei_cdc_ncm 1-1.4:1.1*, wodurch nur noch eine Einstellung für die Verwendung erforderlich ist. Damit der Kernel das Gerät als Modem und nicht als Massenspeicher initialisiert, benötigt das System eine *udev-Regel*. Diese udev-Regel, dargestellt in Quelltext 3, ist in der Datei `/etc/udev/rules.d/70-huawei_e352.rules` abgespeichert. Jedes mal wenn der Surfstick angeschlossen wird oder das System startet, lädt das Betriebssystem dann die richtige Konfiguration für das Mobilfunkmodul. Durch Anzeige der Nachrichten aus den Kernel-Meldungen *dmesg*, die mit dem Präfix *huawei_cdc_ncm 1-1.4:1.1* gekennzeichnet sind, lassen sich die korrekten Einstellungen nachvollziehen.

Quelltext 3: Udey-Regel für den Surfstick E352S-5

Der Treiber richtet unter anderem die drei seriellen Verbindungen /dev/ttyUSB0 bis /dev/ttyUSB2 ein. Der Surfstick nutzt für die Steuerung - wie die meisten Mobilfunkmodule - den AT-Befehlssatz, mit einigen herstellerspezifischen Erweiterungen. Anschließend kann einer von vielen Linux-Middleware-Treibern genutzt werden (z. B. wvdial), um den Surfstick anzusteuern.

Die Sensorplattform nutzt das Programm wvdial als Middleware-Treiber für die Steuerung des Mobilfunkmoduls. Es gibt viele weitere Linuxprogramme, die sich ebenfalls dafür eignen, bspw. Sakis3G und netctl. Im Kontext des Raspberry Pis findet wvdial häufig Anwendung und es existieren Beispielkonfigurationen aus verschiedenen anderen Projekten im Netz. Die generelle Funktionsweise ist bei allen Programmen gleich: das Point-to-Point Protokoll wird genutzt, um ein neues Netzwerkinterface (in der Regel ppp0) zu erstellen, über das der Netzwerkverkehr geleitet wird. Wvdial ist als Paket in der Paketverwaltung APT von Raspbian erhältlich und installiert automatisch das Paket für das Point-to-Point Protocol mit. Quelltext 4 zeigt die Konfigurationsdatei für wvdial. Zeile 2 setzt mit dem Befehl Init1 das Modem zurück (Init1) und Zeile 3 fragt die Signalstärke RSSI ab (Init2). Debug-Ausgaben für die serielle Schnittstelle aktiviert Zeile 4 (Init3) und der Verbindungsauftbau zum Mobilfunknetz der Telekom findet ab Zeile 5 statt (Init4 und nachfolgende Befehle).

```

1 [Dialer Defaults]
2 Init1 = ATZ
3 Init2 = AT+CSQ
4 Init3 = ATQ0 V1 E1 S0=0 &C1 &D2
5 Init4 = AT+CGDCONT=1,"IP","internet.t-mobile","0.0.0.0",0,0
6 Modem Type = Analog Modem
7 Modem = /dev/ttyUSB0
8 Baud = 9600
9 Phone = *99#
10 ISDN = 0
11 Username =
12 Password =
13 Stupid Mode = 1

```

Quelltext 4: Konfigurationsdatei für das Programm wvdial

Das Linux-Betriebssystem verwendet anschließend automatisch die Netzwerkschnittstelle ppp0 für das Routing und den Netzwerkverkehr. In der Anwendung sind keine weiteren Einstellungen mehr für die Internetverbindung notwendig, jedoch erfordern die Tests in Kapitel 7.3.3 weitere technische Informationen zur Signalqualität der Mobilfunkverbindung. Für den Surfstick E352S-5 gibt es vom Hersteller Huawei weder ein Datenblatt noch nennenswerte Dokumentation. Allerdings gibt es eine Vielzahl anderer Mobilfunkmodule von Huawei, deren Firmware auf gleiche oder sehr ähnliche Art und Weise funktionieren [Ibr11]. Die in dieser Arbeit verwendeten Befehle und Informationen basieren auf dem Datenblatt des Huawei MU736 HSPA+ M.2 Moduls und den Informationen des Forums M2MSupport.net [HUA13, M2M].

Der Surfstick liefert an der seriellen Schnittstelle automatisch den Wert von der Signalstärke (RSSI) der Mobilfunkverbindung. Ein Beispiel dafür ist in Quelltext 5 in Zeile 1 angegeben. Die serielle Schnittstelle /dev/ttyUSB2 empfängt diese Informationen. Für RSSI wird die Einheit Arbitrary Strength Unit (ASU) verwendet. Zum Beispiel lässt sich für eine 3G-Verbindung der Wert 7 ASU in -99 dBm umrechnen, der für eine sehr schlechte Signalstärke steht [Ibr11]. Die Formel dafür lautet $RSSI[dBm] = (2 * ASU) - 113$. Jede Änderung des Werts bewirkt die automatisch Ausgabe an der seriellen Schnittstelle durch die Firmware des Surfsticks.

```

1 ^RSSI: 23
2 ^DSFLOWRPT:00002280,00000000,00000000,000000000005C70,000000000005C54
   ,0010E000,0010E000

```

Quelltext 5: Debuginformationen des Surfsticks E352S-5

Im Fall einer aktiven Mobilfunkverbindung geschieht in regelmäßigen Abständen eine Textausgabe, dargestellt in Zeile 2 in Quelltext 5. Diese beinhaltet weitere hexadezimale Informationen über die Verbindung. Diese Werte lassen sich wie folgt interpretieren [Ibr11]:

- ^DSFLOWRPT: N1, N2, N3, N4, N5, N6, N7
- N1: Dauer der Verbindung in Sekunden.
- N2: Gemessene Upload Bandbreite.
- N3: Gemessene Download Bandbreite.
- N4: Anzahl von gesendeten Bytes.
- N5: Anzahl von empfangenen Bytes.
- N6: Ausgetauschte maximale Bandbreite im Upload.
- N7: Ausgetauschte maximale Bandbreite im Download.
- N6 und N7 sind davon abhängig, welche Übertragungstechnik gerade genutzt wird.

Der Surfstick sendet diese Daten regelmäßig an die serielle Schnittstelle `/dev/ttyUSB2`, anschließend werden diese von der Anwendung der Sensorplattform ausgelesen und in ein lesbares Format für die Log-Datei konvertiert. Der RSSI-Wert ist für die Bestimmung der Signalqualität jedoch noch nicht ausreichend, wodurch weitere Parameter benötigt werden. Die AT-Befehle, die dafür in Frage kommen, lauten AT+CSQ, AT^CSNR?\r und AT^SYSINFO\r. Der Befehl AT+CSQ\r, der auch in der Konfiguration für wvdial enthalten ist (siehe Quelltext 4), gibt Werte für RSSI und die Bitfehlerrate zurück. Die Bitfehlerrate wird jedoch vom Surfstick nicht unterstützt, erkennbar an dem Rückgabewert 99. Besser geeignet ist der Befehl AT^CSNR?\r, der Werte für RSSI, RSCP und E_c/I_o zurückgibt (siehe Kapitel 3.1.5). Die Parameter eignen sich für eine Bewertung der Signalqualität. Zusätzlich lässt sich durch Abfrage von AT^SYSINFO\r der Modus feststellen, in dem sich der Surfstick gerade befindet. Es stellt sich nämlich heraus, dass das Modem automatisch verschiedene 2G und 3G Übertragungstechniken mit der Funkstation aushandelt, abhängig davon ob gerade hohe Datenraten benötigt werden oder nicht und wie gut jeweils die Signalqualität ist.

Die vorgestellten AT-Befehle mit einem Zirkumflex als drittes Zeichen sind herstellerspezifisch. Für ein Mobilfunkmodul eines anderen Herstellers muss auch die Implementierung der Sensorplattform ergänzt werden. Grundsätzlich sind derartige Parameter von allen Modems abrufbar. Sierra Wireless ist beispielsweise ein anderer Hersteller für solche Geräte. Dieser stellt Dokumentation zur Verfügung, die die möglichen AT-Befehle erläutert [Sie13]. Die Parameter RSSI, RSCP und E_c/I_o lassen sich damit auch bei unterschiedlichen Mobilfunkmodulen von der Sensorplattform zur Bestimmung der Signalqualität nutzen.

6.7. Anbindung von externen Webanwendungen

Die Sensorplattform soll unter möglichst realen Bedingungen getestet werden, was sich auch auf die Verwendung eines externen Webservers zum Hochladen von Samples bezieht. Zum Testen der Sensorplattform stehen dafür zwei externe Webinterfaces zur Verfügung. Zukünftig kann die Sensorplattform für beliebig viele Webanwendungen erweitert werden.

TeLiPro ist das Portal des Deutschen Instituts für Telemedizin und Gesundheitsförderung (DITG), das unter anderem die Möglichkeit für das Übertragen und das Abspeichern von Herzfrequenzdaten über eine experimentelle REST-Schnittstelle bietet. Jeweils eine Herzfrequenzmessung inklusive einiger Metadaten wird über einen HTTP-Post-Request an den Server übermittelt. Das dafür genutzte JSON-Format orientiert sich an dem Apple Health Kit. Quelltext 6 zeigt ein Beispiel, wie diese Daten aussehen und wie sich einige Datenfelder von denen in der Datenbank der Sensorplattform unterscheiden. Das ist aber insofern unproblematisch, da unterschiedliche Webinterfaces an die Sensorplattform angebunden werden können, für die gegebenenfalls nur verschiedene Teilmengen der Daten erforderlich sind. Das Format kann für jedes Webinterface spezifisch definiert sein. In dem derzeitige JSON sind bspw. Werte für die RR-Intervalle noch nicht vorgesehen.

Bei der Sensorplattform werden zunächst nur Herzfrequenzdaten an den Webserver übertragen, die anschließend jedoch nicht weiter verarbeitet werden. In Zukunft ist es vorstellbar, dass die Daten für Algorithmen zur Diagnose oder individuellen Therapie von Krankheiten eingesetzt werden können.

```

1  {
2      "type": "sample",
3      "sampleType": "quantity",
4      "quantityType": "HeartRate",
5      "value": 65,
6      "unit": "bpm",
7      "startDate": "2016-10-16T12:57:10.550Z",
8      "endDate": "2016-10-16T12:57:10.550Z",
9      "metadata": {
10          "device": "11:22:33:AA:BB:CC"
11      }
12 }
```

Quelltext 6: JSON-Format der REST-Schnittstelle von TeLiPro

Wie bereits erwähnt, verwendet die Sensorplattform eine HTTPS-Verbindung für das Übertragen an den TeLiPro-Webserver. Lediglich ein einzelnes Sample wird pro Verbindung hochgeladen. Die daraus resultierende Menge an Nutzdaten (Payload) ist verhältnismäßig klein (einige Hundert Bytes). Entsprechend ist der relative Anteil von Headerbytes (durch TCP) hoch. Mehrere Samples könnten in einem Post-Request zusammengefasst werden, sodass das Verhältnis zwischen Headerbytes und Payload zu verbessern. Daraus würde sich eine geringere Datenrate ergeben. Eine Veränderung der Schnittstelle zu TeLiPro ist aber zunächst nicht vorgesehen, daher wird mit diesem Beispiel weiter gearbeitet.

Weil der TeLiPro-Server jedoch nur innerhalb des Fraunhofer-Netzwerks erreichbar ist, nicht aber über das Mobilfunknetz der Telekom, wird für die Sensorplattform ein zweiter externer Webserver implementiert. Dieser heißt im Folgenden WebHrs. WebHrs ist über einen Dynamic Domain Name System (DNS) Dienst erreichbar, nutzt aber zur Vereinfachung kein TLS/SSL (nicht unterstützt) und auch keine Methode zur Autorisierung. Der Webserver besitzt eine REST-Schnittstelle, die JSON-Daten in einem bestimmten Format empfängt. Das erforderliche Datenformat für die Herzfrequenz ist aus der TeLiPro-Vorlage übernommen. Für sämtliche Tests in Kapitel 7, die den Upload von Herzfrequenzsamples erfordern, wird WebHrs genutzt. Der Server speichert die Sensordaten in einer Log-Datei, damit das erfolgreiche Übertragen der Samples zu einem späteren Zeitpunkt nachvollzogen werden kann.

Abbildung 25 zeigt die Klassenbeziehung der Funktionalität zum Hochladen von Herzfrequenzdaten. Ein Uploader existiert ausschließlich im Kontext des HeartRateSampleCollectors (Komposition). Der HeartRateSampleCollector kann jedoch immer nur einen

Uploader besitzen. Jeweils eine konkrete Implementierung des Uploader-Interfaces nutzt ein externes Webinterface. Das Uploader-Interface definiert die drei abgebildeten Methoden, die für die Kommunikation mit einem Webserver ausreichend sind. Falls ein neues Webinterface angebunden werden soll, wird eine neue Klasse benötigt, die das Uploader-Interface ebenfalls implementiert. Auf diese Weise ist es auch möglich, Webinterfaces einzubinden, die andere Protokolle (TCP-basiert oder nicht-TCP-basierend) verwenden. Vorstellbar wäre die Nutzung von Real Time Streaming Protocol (RTSP), Message Queue Telemetry Transport (MQTT) oder Quic von Google.

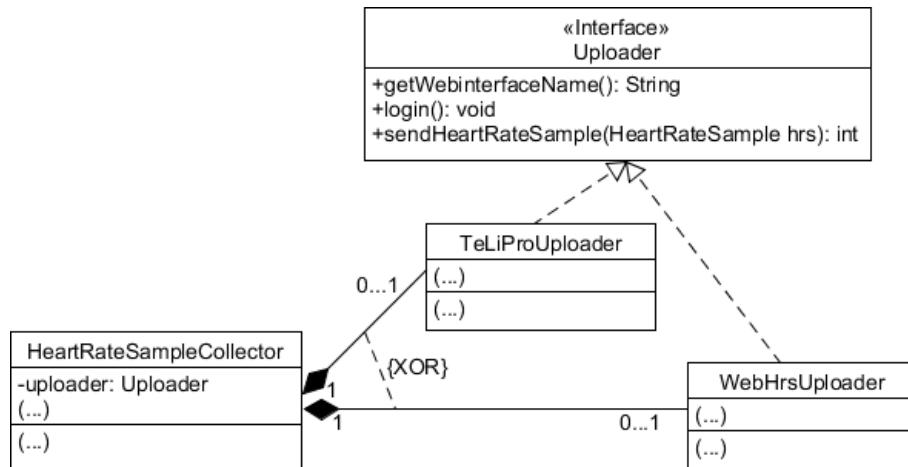


Abbildung 25: Klassenbeziehung der Upload-Funktionalität

Durch Maven-Profile lässt sich die Sensorplattform-Anwendung mit bestimmten Konfigurationen kompilieren. Genau wie bei der Hardwareplattform in Kapitel 6.5 steht jeweils ein Maven-Profil für TeLiPro und WebHrs zur Verfügung. Für eine neue externe Webanwendung wird entsprechend auch ein Profil angelegt. Auf Basis der Properties bindet das Guice eine konkrete Implementierung an das Uploader-Interface. Falls kein Profil ausgewählt wird, ist der Uploader als *null* definiert - ebenfalls eine gültige Konfiguration für den Fall, dass die Sensorplattform ausschließlich im Offline-Betrieb Daten aufzeichnen soll.

```

1  switch (webinterfaceName) {
2      case "TeLiPro":
3          LOG.info("webinterface is {}", webinterfaceName);
4          bind(TeLiProUploader.class).in(Singleton.class);
5          bind(Uploader.class).to(TeLiProUploader.class);
6      break;
7      case "WebHrs":
8          LOG.info("webinterface is {}", webinterfaceName);
9          bind(WebHrsUploader.class).in(Singleton.class);
10         bind(Uploader.class).to(WebHrsUploader.class);
11     break;
12     case "${webinterface.name}":
13         LOG.warn("No webinterface maven profile specified");
14         bind(Uploader.class).toProvider(Providers.of(null));
15     break;
16     default:
17         LOG.error("unknown webinterface {}", webinterfaceName);
18         bind(Uploader.class).toProvider(Providers.of(null));
19     break;
20 }
  
```

Quelltext 7: Konfiguration von Guice für unterschiedliche Uploader

Die Anwendung der Sensorplattform lässt sich dynamisch konfigurieren, sodass die Schnittstellen zur Hardware und zum externen Webserver austauschbar sind. Durch die Properties aus den Profilen von Maven bindet das Dependency-Injection Framework Guice konkrete Klassen an Interfaces.

6.8. Controller und SensorOverseer

Die Klasse Controller verwaltet den Programmablauf der Sensorplattform. Nach dem Start des Betriebssystems initialisiert sich auch automatisch die Anwendung der Sensorplattform (siehe Kapitel 6.9). Da keine Interaktion vom Patienten mit der Sensorplattform vorgesehen ist, muss die Sensorplattform darüber hinaus Ereignisse erkennen und Abläufe insbesondere in Fehlerfällen autonom ausführen.

Abbildung 26 zeigt das Aktivitätsdiagramm der Sensorplattform. Bei jedem Start initialisiert die Anwendung zunächst den Webserver und überprüft anschließend, ob eine vorherige Datenaufzeichnung unterbrochen wurde (z. B. durch Unterbrechung der Stromversorgung). Erkennbar ist dies, da die Einstellungen der letzten Aufzeichnung als Properties in der Datei `/home/administrator/Sensorplattform/recording.properties` abgelegt werden. Dazu gehören unter anderem die Sensorkonfiguration und der Endzeitpunkt der Aufzeichnung. Am Ende einer Aufzeichnung wird die Datei immer gelöscht. Ist keine Datei vorhanden, wurde auch keine Aufzeichnung unterbrochen und die Sensorplattform befindet sich im Bereitschaftsmodus. Der Standby-Modus ist der Endzustand in Abbildung 26. Ist die Datei nach dem Start vorhanden, wird zunächst der Endzeitpunkt ausgelesen. Nur wenn dieser noch in der Zukunft liegt, versucht die Sensorplattform, die Verbindungen zu den Sensoren wiederherzustellen und die Aufzeichnung fortzuführen. Andernfalls gelangt die Sensorplattform wieder in den Standby-Modus.

Die Sensorplattform kann den Endzustand jederzeit verlassen, in dem eine neue Aufzeichnung gestartet wird. Die Sensorplattform sammelt dann wieder solange Daten, bis der Endzeitpunkt überschritten ist oder der Benutzer die Aufzeichnung über die Webanwendung abbricht. Anschließend befindet sich das System wieder im Bereitschaftsmodus.

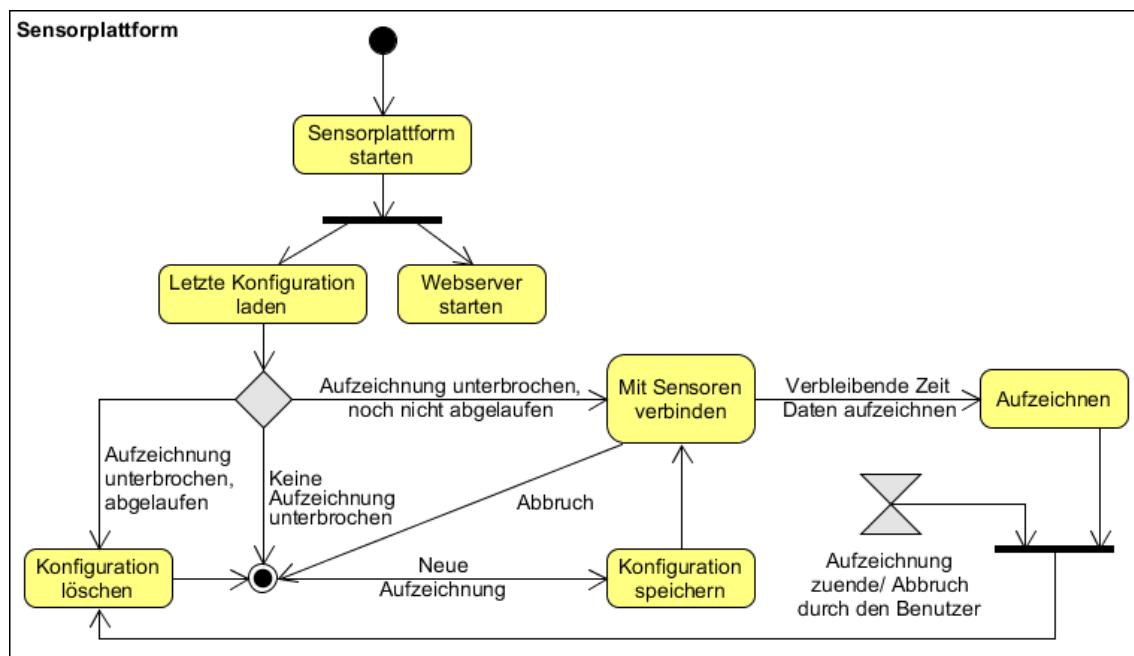


Abbildung 26: Aktivitätsdiagramm der Sensorplattform

Über REST-Schnittstellen kann der Controller in der Anwendung Befehle von der Webanwendung des medizinischen Fachpersonals erhalten. In Abbildung 27 ist der Start-

befehl für eine Aufzeichnung als Sequenzdiagramm dargestellt. In dem REST-Aufruf einer HTTP-Post-Methode sind alle Informationen zu den Sensoren und zur Aufzeichnung für den Controller enthalten. Mit Hilfe der SensorWrapperFactory erzeugt der Controller für alle Sensoren dieser Aufzeichnung die entsprechenden SensorWrapper. Die Methoden connectBlocking() der SensorWrapper stellt eine Bluetooth-Verbindung her, während enableLogging() die Notifications für Messwerte der Sensoren aktiviert. Beim Starten einer Aufzeichnung ist es erforderlich, dass alle Sensoren aktiv sind, sonst schlägt der Start fehl. Die Antwort des REST-Aufrufs enthält sowohl im Erfolgsfall, als auch im Fehlerfall eine entsprechende Benachrichtigung für den Benutzer an der Webanwendung. Wird eine unterbrochene Aufzeichnung nach dem Neustart der Sensorplattform fortgeführt, müssen nicht alle Sensoren wieder verfügbar sein. Dann versucht die Sensorplattform kontinuierlich, sich erneut mit den Sensoren zu verbinden.

Auf Grund der begrenzten Reichweite von Bluetooth Low Energy (BLE) ist es erforderlich, Verbindungsabbrüche zu den Sensoren und automatische Neuverbindungsversuche in der Anwendung der Sensorplattform zu berücksichtigen. Die Verbindung kann nicht vom Patienten wiederhergestellt werden, weil dieser keinen Zugriff auf die Webanwendung hat und den Verbindungsabbruch in den meisten Fällen gar nicht merken wird. Es sind auch keine Knöpfe an den Geräten vorhanden, mit denen dieser Verbindungsvorgang gesteuert werden könnte. Stattdessen geschieht dies automatisch mit der Klasse SensorOverseer. Der SensorOverseer überwacht alle SensorWrapper während der Datenaufzeichnung. Jedes mal wenn ein SensorWrapper eine Notification erhält, wird der Zeitpunkt in einer Variablen gespeichert. Es kann passieren, dass die Sensorplattform die Verbindung zum Sensor verliert, wenn bspw. zu lange kein Hautkontakt mehr besteht oder wenn die Sensoren außer Reichweite bewegt werden. Der Patient kann dies nicht an den Sensoren erkennen. Das Gatttool erhält davon auch keine Kenntnis, weil die Verbindung nicht regulär terminiert wird (es werden ausschließlich Notifications empfangen). Als Anhaltspunkt wird der Zeitpunkt der letzten Notification genutzt.

Der SensorOverseer überprüft jede Sekunde für alle SensorWrapper einer Aufzeichnung, ob die letzte Notification länger als 10 Sekunden zurückliegt. Ist dies der Fall, trennt das Gatttool die Verbindung explizit (um die internen Zustände von BlueZ zurückzusetzen), versucht anschließend kontinuierlich, eine neue Verbindung aufzubauen und die Notifications wieder zu aktivieren. Der Wert von 10 Sekunden ist über die Properties konfigurierbar und ist an die derzeitige Auswahl von Sensoren angepasst. Wenn ein Sensor nur ein mal pro Stunde einen Messwert durch Bluetooth überträgt (bspw. bei einem Blutdruckmessgerät), muss der Wert für den Overseer entsprechend höher gesetzt werden. Gegebenenfalls können als Ausbaustufe der Sensorplattform unterschiedliche Werte für einzelne Sensoren definiert werden. Mit dem SensorOverseer ist es für die Sensorplattform möglich, automatisch eine verlorene Verbindung zum Sensor zurückzusetzen und anschließend wiederherzustellen.

Diese Funktion kann auch für Sensoren verwendet werden, die nur einige Male am Tag eine Messung machen und die verbleibende Zeit ausgeschaltet sind, um Energie zu sparen. Beim Einschalten des Sensors könnte die Sensorplattform automatisch die Verbindung wiederherstellen und die Messwerte erfassen. Anschließend könnte der Patient den Sensor bewusst wieder ausschalten. Voraussetzung bei dieser Implementierung ist, dass der Sensor zu Beginn einer Aufzeichnung verfügbar ist, sonst wird der Start abgebrochen.

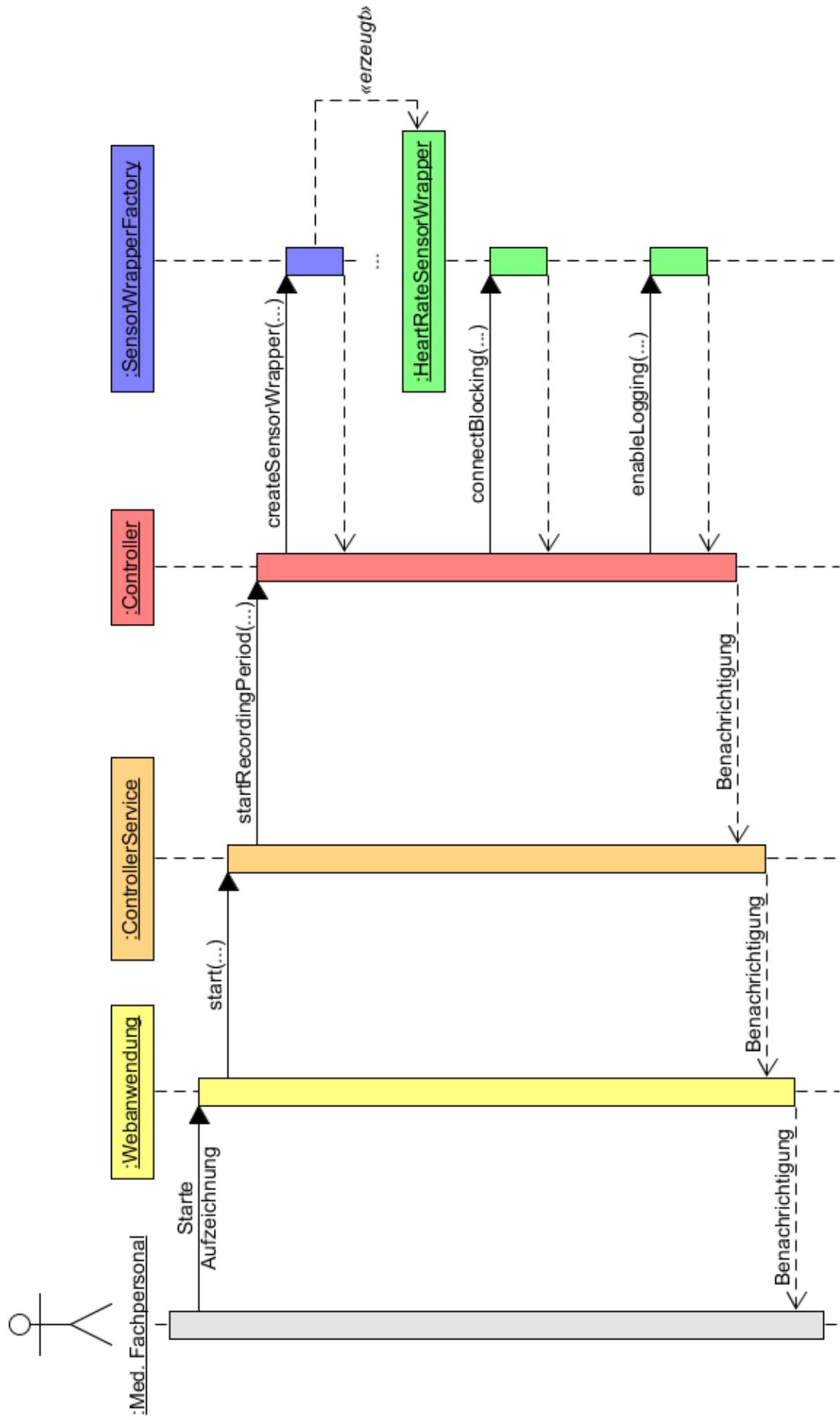


Abbildung 27: Sequenzdiagramm für den Start einer Aufzeichnung

Aus den im Interface HardwarePlatform (Kapitel 6.5) definierten Methoden geht hervor, dass das Board für die Sensorplattform mindestens eine LED besitzen muss, um dem Patienten den Status der Sensorplattform darzustellen. Beim Raspberry Pi 3 zeigt eine rote LED das Vorhandensein der Stromversorgung an, die aber nicht über eine Programmierschnittstelle für Entwickler steuerbar ist. Die gelbe LED ist für Anwendungen nutzbar. Der Controller der Sensorplattform (und der SensorOverseer) steuern die LED des Boards an.

Bei dem Raspberry Pi 3 stehen drei Dateien (trigger, delay_on und delay_off im Verzeichnis `/sys/class/leds/led0`) im Dateisystem zur Verfügung, die die Eingabe einiger Befehle und die Blinkdauer in Textform zu Blinkmustern umsetzen. Diese Schnittstelle ist bereits im Betriebssystem Raspbian integriert. Die Befehle heißen *timer*, *heartbeat* und *none*. Mittels FileInputStreams werden die Kommandos von der Anwendung in die Dateien geschrieben. Im Bereitschaftsmodus blinkt die gelbe LED regelmäßig eine Sekunde an und aus. Während der Datenaufzeichnung blinkt diese zwei mal kurz auf und ist anschließend für eine Sekunde aus, welches das Heartbeat-Muster darstellt. Geht die Verbindung zu mindestens einem Sensor verloren, blinkt die LED gar nicht und ist aus. In der Zeit versucht der SensorOverseer kontinuierlich die Verbindung wiederherzustellen. Gelingt dies im Laufe der Aufzeichnung, blinkt die LED erneut im Heartbeat-Muster. Nach dem Abschluss der Aufzeichnung befindet sich die Sensorplattform wieder Bereitschaftsmodus und die LED blinkt regelmäßig eine Sekunde an und aus. Dem Patienten signalisiert die LED während des Betriebs der Sensorplattform, ob das System korrekt alle Daten der Sensoren aufzeichnet. Falls die LED nicht mehr blinkt, könnte der Patient bspw. überprüfen, ob der Herzfrequenzsensor noch mit beiden Elektroden Hautkontakt hat. Eine Interaktion mit der Sensorplattform ist jedoch nicht vorgesehen.

Damit ist die Implementierung der Anwendung vorgestellt. Das Gatttool des Bluetooth-Treiber BlueZ ist die Schnittstelle zu den Sensoren. Die Anwendung verfügt über einen Webserver für die Mensch-Maschine-Interaktion. Die Sensorplattform lädt die Samples an eine Webanwendung hoch. Außerdem ist eine Datenbank integriert, die die Messwerte auch im internen Speicher der Sensorplattform persistiert. Durch die Nutzung von Dependency Injection können die Schnittstellen zur Hardware und zum Webserver konfiguriert und ausgetauscht werden. Eine LED zeigt den internen Zustand der Sensorplattform an.

6.9. Konfiguration des Betriebssystems

Die Implementierung der Anwendung deckt einen Großteil der Funktionalität bereits ab. Für den verbleibenden Teil sind einige Einstellungen am Betriebssystem erforderlich, die auch Aspekte der Usability, bspw. die Erreichbarkeit der Webanwendung, umfassen. Dieses Unterkapitel umfasst alle Konfigurationen des Betriebssystems Raspbian für die Sensorplattform. Zunächst wird der Systemstart des Geräts betrachtet.

6.9.1. Autostart mit systemd

Die Anwendung der Sensorplattform soll als Service beim Starten des Betriebssystems automatisch initialisiert werden. Raspbian benutzt wie die meisten Linux-Distributionen mittlerweile das Init-System *systemd*. Damit lassen sich verschiedene Prozesse in den definierten Runlevel parallel starten, wofür die Sensorplattform ein Shell-Skript im Verzeichnis `/etc/init.d/` gespeichert hat. Quellcode 8 zeigt den Inhalt des Skripts, der auf einer vom Raspbian Betriebssystem bereitgestellten Vorlage (Skeleton) basiert. Das Skript reagiert auf drei verschiedene Befehle als Eingabeparameter: start, stop und restart, womit die Ausführung der Anwendung gesteuert wird. Zusätzlich deaktivieren die Zeilen 14 und 27 den HDMI-Ausgang, um Strom zu sparen (dazu mehr in Kapitel 7.3). Außerdem nutzt die Sensorplattform für sämtliche Ausgaben der Anwendung eine Log-Datei `sensorplatform.txt` (Zeile 16 und 29), die zu einem späteren Zeitpunkt z. B. das Erkennen und

Zurückverfolgen von Exceptions ermöglicht.

```

1 #!/bin/sh
2 ##### BEGIN INIT INFO
3 # Provides:           Sensorplatform application
4 # Required-Start:
5 # Required-Stop:
6 # Default-Start:    2 3 4 5
7 # Default-Stop:     0 1 6
8 # Short-Description: Starts the sensorplatform application.
9 # Description:
10 ##### END INIT INFO
11
12 case "$1" in
13     start)
14         /opt/vc/bin/tvservice -o
15         cd /home/administrator/Sensorplatform
16         /lib/jvm/jdk1.8.0_101/bin/java -cp "bin/*" de.fhg.fit.biomas.
17             sensorplatform.main.Main >> sensorplatform.txt 2>&1 &
18         ;;
19     stop)
20         killall java
21         killall wvdial
22         killall pppd
23         ;;
24     restart)
25         killall java
26         killall wvdial
27         killall pppd
28         /opt/vc/bin/tvservice -o
29         cd /home/administrator/Sensorplatform
30         /lib/jvm/jdk1.8.0_101/bin/java -cp "bin/*" de.fhg.fit.biomas.
31             sensorplatform.main.Main >> sensorplatform.txt 2>&1 &
32         ;;
33 esac
34
35 exit 0

```

Quelltext 8: Systemd Startskript für die Anwendung der Sensorplattform

Quelltext 9 zeigt, wie das Skript anschließend ausführbar gemacht wird. Dieser Schritt ist obligatorisch, da die Anwendung im Startvorgang sonst nicht als Service berücksichtigt wird.

```

1 sudo chmod +x /etc/init.d/sensorplatform
2 sudo update-rc.d sensorplatform defaults

```

Quelltext 9: Shellskript ausführbar machen und zu systemd hinzufügen

Unter den Services der Standardkonfiguration von Raspbian wird ebenfalls ein SSH-Server als Hintergrunddienst gestartet. Die vordefinierte Konfiguration ermöglicht den Login von Benutzern über ein lokales Netzwerk. Dieser Zugang erleichtert das Testen der Anwendung, da die Log-Einträge der Anwendung direkt einsehbar sind. Für Wartungszwecke wäre es möglich, dass über den SSH-Zugang auch Softwareaktualisierungen vorgenommen werden. Eine Liste aller Start-Services lässt sich durch den Befehl `ls /etc/init.d/` ausgeben. Der SSH-Zugang ist nur von einem lokalen Netzwerk möglich, nicht aber über die Mobilfunkverbindung.

6.9.2. Hardware-Uhr

Eine weitere wichtige Systemeinstellung ist die Uhrzeit inklusive des Datums. Wie bereits in diesem Kapitel vorgestellt, verwendet die Anwendung für viele Funktionen die Uhrzeit. Beide Informationen werden auf dem Raspberry Pi durch das Network Time Protocol (NTP) gesetzt, falls der Internetzugang durch ein Netzwerk möglich ist. NTP findet mittlerweile bei fast allen Geräten Anwendung. Ist die Zeit einmal gesetzt, wird diese regelmäßig und beim Starten und Herunterfahren in eine Datei für den Kernel geschrieben, worum sich das in Raspbian vorinstallierte Softwarepaket `fake-hwclock` kümmert. Üblicherweise wird `fake-hwclock` von `systemd` als Hintergrundservice ausgeführt. Ist bei einem Start kein Netzwerk verfügbar, werden die zuletzt gespeicherten Angaben geladen. `Fake-hwclock` soll verhindern, dass die Systemzeit auf den 01.01.1970 zurückgesetzt wird, weil dies bei manchen Betriebssystemen zu Routinechecks (z. B. für das Dateisystem) führen würde, die unter Umständen sehr zeitintensiv sein können. Damit bietet das Raspbian Betriebssystem mit NTP und `fake-hwclock` eine annähernde Lösung für das Verwalten der Zeit, die für die Sensorplattform jedoch nicht ausreichend ist.

Bei dem flexiblen Einsatz der Sensorplattform ist eine Internetverbindung über Mobilfunk möglicherweise nicht immer gewährleistet, da Verbindungsabbrüche, bspw. bei Aufenthalt in einem Keller oder in einem Tunnel, auftreten können. Es wäre daher keine gute Lösung, wenn die Anwendung der Sensorplattform auf die mobile Internetverbindung warten müsste, um die Uhrzeit einzustellen. Außerdem benötigt die Initialisierung des Surfsticks signifikant mehr Zeit als der Systemstart des Raspberry Pis. Dies wäre für den Patienten wenig intuitiv und verlängert die Zeit nach dem Start, in dem die Sensorplattform noch nicht verwendbar wäre. Bei Android ist dies insofern besser gelöst, weil der Systemstart selbst schon deutlich länger dauert und dieser am Bildschirm zu erkennen ist. Kurz nach erscheinen der Android-Oberfläche ist in der Regel auch schon die Verbindung zum Mobilfunk hergestellt und die Uhr konfiguriert.

Anhand der Log-Datei der Anwendung ist erkennbar, dass beim Start zunächst die falsche Uhrzeit genutzt wird und diese kurz darauf vom NTP-Service (Startprozess durch `systemd`) aktualisiert wird. Unter Testbedingungen fällt dies nicht auf, da auch immer eine Kabelnetzwerkverbindung für den SSH-Zugang vorhanden ist und die Uhrzeit dann sehr schnell aktualisiert wird. Unter realen Bedingungen, durch die Internetverbindung über Mobilfunk, dauert dies wesentlich länger, weil die Initialisierung des Modems ca. eine Minute benötigt. Unter anderem in Kapitel 6.8 sind Funktionen beim Start der Sensorplattform beschrieben, für die die korrekte Uhrzeit im System obligatorisch sind, weswegen eine Hardware-Uhr notwendig ist. Dafür wird das Modul DS3231 verwendet, das auf die I²C-GPIO-Pins auf dem Board aufgesetzt wird. Das Modul ist in Abbildung 28 auf den GPIO-Pins dargestellt. Die Stromversorgung erfolgt entweder über den 3,3V Power Pin oder über eine kleine Batterie (Knopfzelle) auf der Unterseite und kann auf diese Weise auch unabhängig von der Eingangsspannung des Raspberry Pis betrieben werden. Zusätzlich verfügt die Hardware-Uhr über einen Temperatursensor, mit dessen Messungen eine höhere Präzision des Quarzoszillators erreicht wird [Max15]. Die Genauigkeit des DS3231 beträgt ±2 Minuten pro Jahr mit Berücksichtigung der Temperatur [Max15].

Viele Linux-Betriebssysteme verwenden für die Hardwarebeschreibungen und zum Laden von entsprechenden Treibern Device Trees und Overlays. Damit die Echtzeituhr funktionsfähig ist, müssen einige Einstellungen vorgenommen werden. Die Datei `/boot/config.txt` wird um die Zeile `dtoverlay=i2c-rtc,ds3231` ergänzt. Der entsprechende Treiber ist bereits in dem Raspbian Image vorhanden und wird durch diese Zeile explizit geladen, was sich nach dem nächsten Start durch den Befehl `dmesg | grep rtc` überprüfen lässt. Die korrekte Uhrzeit, die beim Systemstart vom NTP-Service gesetzt wird, muss einmalig in das Hardware-Modul mit dem Befehl `hwclock -w` geschrieben werden. Die Eingabe lässt sich



Abbildung 28: DS3231 Hardware-Uhr an den I²C GPIO-Pins des Raspberry Pi 3

durch `hwclock -r` oder durch `cat /proc/driver/rtc` verifizieren. Anschließend wird in dem systemd-Skript `/etc/init.d/sensorplatform` die Zeile `hwclock -s` eingetragen, damit vor jedem Start der Anwendung der Sensorplattform die Kernel-Zeit von der Echtzeituhr eingestellt wird. Dadurch ist gewährleistet, dass die Anwendung der Sensorplattform unmittelbar nach dem Start immer die korrekte Uhrzeit verwendet.

Das Paket `fake-hwclock` ist bei der Verwendung einer Hardware-Uhr überflüssig. Der Befehl `apt purge fake-hwclock` deinstalliert das Paket vom Betriebssystem, aber der NTP-Service bleibt bestehen und wird deaktiviert (`sudo update-rc.d -f ntp remove`). Um einen möglichen Drift der Uhrzeit zu einem späteren Zeitpunkt zu korrigieren, kann der NTP-Dienst wieder gestartet werden (`sudo update-rc.d ntp defaults`).

6.9.3. Benutzerverwaltung

Die meisten Linux Betriebssysteme verwenden mindestens zwei Benutzerkonten. Die Standardkonfiguration von Raspbian sieht einen voreingestellten Benutzer *pi* mit dem Passwort *raspberry* vor, der sich auch auf dem SSH-Server einloggen kann. Weil diese Daten allgemein bekannt sind, verwendet die Sensorplattform ausschließlich das Benutzerkonto *administrator* mit einem starken Passwort. Das Root-Benutzerkonto ist immer vorhanden, mit dem der SSH-Login jedoch nicht möglich ist. Prozesse, die Root-Rechte benötigen (z. B. die Anwendung der Sensorplattform und `bluetoothctl`), werden mit dem Befehl `sudo` gestartet, so wie es für die Nutzung von Debian empfohlen ist. Dies gewährleistet mehr Sicherheit, weil derartige Aufrufe in den Log-Dateien des Betriebssystems (zum Nachvollziehen zu einem späteren Zeitpunkt) gespeichert und auf diese Weise nur so wenige Prozesse wie nötig mit Root-Rechten gestartet werden. [Deb].

6.10. Sicherheit

Die Informationssicherheit und der Datenschutz sind bei der Sensorplattform von zentraler Bedeutung, da das Gerät sensible Gesundheitsdaten erhebt und verarbeitet. In dem folgenden Abschnitt sind die Sicherheitsfunktionen vorgestellt, um die Schnittstellen zur Sensorplattform abzusichern. Ergänzend dazu zeigen einfache Tests, wie sich die korrekte Anwendung dieser Funktionen überprüfen lässt. Sicherheitsmaßnahmen für die Hardware des Geräts sind im Rahmen dieser Masterarbeit nicht berücksichtigt, weil bspw. die micro-SD-Karte frei zugänglich ist und an einem anderen Computer ausgelesen werden kann. Alle Hardwarekomponenten bleiben unverändert.

Besonders in Bereichen des Internet-of-Things (IoT) ist die Innovationsgeschwindigkeit so hoch, dass die Sicherheit der Systeme in Verzug geraten kann, wie auch der weitreichenden Einfluss des Botnetzes Mirai gezeigt hat, das Sicherheitslücken in Geräten des IoT ausnutzt [Gre16]. Zudem ist es erforderlich, die Informationen der Sensorplattform

vor dem Abhören durch Dritte abzusichern, da es sich um sensible medizinische Daten handelt. Weil die Sensorplattform per Definition die meiste Zeit eine Netzwerkverbindung mit Internetzugang hat, bieten sich für potentielle Angreifer entsprechend lange Zeiträume, um das System zu kompromittieren. Im Folgenden werden zunächst die drahtlosen Schnittstellen zur Sensorplattform betrachtet: Bluetooth, der externe Webserver, der SSH-Zugang und die Mobilfunkverbindung) und letztlich auch das Betriebssystem der Sensorplattform. Als Grundlage gelten die Sicherheitsforderungen aus der Pressemitteilung des BSI bezüglich IoT-Sicherheit vom 25.10.2016 [Bun16]. Außerdem wurde am 16.11.2016 vom US-amerikanischen Department of Homeland Security (DHS) eine Security-Strategie veröffentlicht, die weitere Empfehlungen für die Sicherheit von Internet-of-Things-Geräten formuliert [U.S16]. Für die Webserver wird das in der Praxis etablierte Protokoll TLS/SSL verwendet.

6.10.1. Sicherheitsfunktionen der Webschnittstellen

Die Authentifizierung und die Verschlüsselung des Servers der Sensorplattform erfolgt durch ein generiertes Zertifikat. Für Webseiten ist dies die übliche Art und Weise, Verbindungen zu einem Webserver abzusichern. Mit dem Terminalprogramm *keytool*, das in der Installation des Java Development Kit (JDK) enthalten ist, lässt sich ein entsprechendes Zertifikat erstellen. Für den Testbetrieb ist ein nicht-verifiziertes Zertifikat ausreichend. Quelltext 10 zeigt die Parameter, die für die Erstellung des Zertifikats notwendig sind. Gegebenenfalls könnte für die Schlüsselgröße auch 4096 Bit gewählt werden.

```
1 $ keytool -genkey -alias Sensorplatform -keyalg RSA -keystore keystore.jks -keysize 2048
```

Quelltext 10: Erstellung des TLS-Zertifikats durch keytool

Für die Erstellung des TLS-Zertifikats fragt *keytool* schrittweise zusätzliche Eingaben ab, die wie folgt angegeben sind.

- Password: 123456
- First and last name: Daniel Pyka
- Organizational unit: Fraunhofer FIT
- Organization: Fraunhofer Gesellschaft
- City: Sankt Augustin
- State: Nordrhein-Westfalen
- Country code: DE
- Key password: 123456

Die resultierende Datei *keystore.jks* wird von Jetty mitsamt Passwörtern geladen und stellt nach der korrekten Konfiguration die Inhalte über eine HTTPS-Verbindung bereit. Beim ersten Aufruf der Webanwendung der Sensorplattform erscheint eine Meldung, dass der Herausgeber des Zertifikats nicht verifiziert werden kann. Das passiert deshalb, weil es nicht von einer offiziellen und anerkannten Zertifizierungsstelle stammt. Der Testbetrieb mit der Sensorplattform sieht eine Ausnahmeregel im Browser vor, woraus sich aber keine weiteren Nachteile ergeben. In Abbildung 29 (oben) sind allgemeine Informationen zur Sicherheit der Webseite dargestellt, die im Browser (in diesem Beispiel Mozilla Firefox) abrufbar sind. Unter anderem ist erwähnt, dass die Seite vor der Übertragung verschlüsselt wurde und das dafür genutzte Verfahren TLS/SSL ist. Abbildung 29 (unten) zeigt die Informationen zum TLS-Zertifikat an, die mit den Angaben aus dem Erstellungsprozess übereinstimmen.

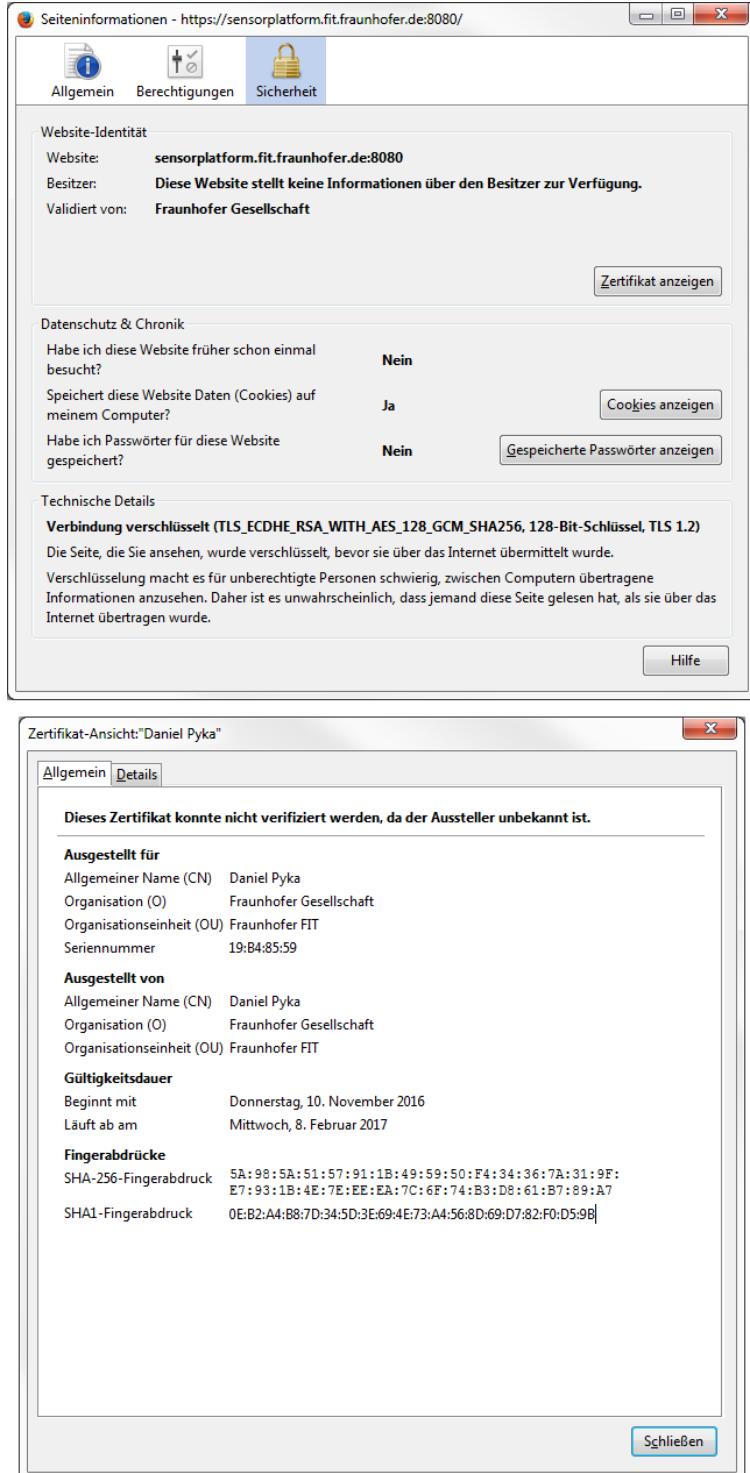


Abbildung 29: Informationen zur Sicherheit und zum TLS-Zertifikat

Die Authentifizierung und die verschlüsselte Verbindung der Webanwendung sind durch die Verwendung von TLS/SSL gewährleistet. Es fehlt nur noch die Autorisierung des Benutzers gegenüber dem Webserver, schließlich soll nur das medizinisches Fachpersonal Zugang zur Steuerung der Sensorplattform haben.

Für die Autorisierung gibt es Token-basierte Verfahren und HTTP-Basic-Authentication. Die Variante mit einem Token ist aufwendig zu implementieren und bietet für einen Demonstrator keine signifikanten Vorteile (bspw. die Limitierung der Anzahl von Login-Versuchen), weshalb die Sensorplattform HTTP-Basic-Authentication verwendet. Diese

Funktionalität stellt insbesondere hinsichtlich des Zeitaufwands der Implementierung und der resultierenden Sicherheit einen guten Kompromiss dar. Gültige Login-Daten (Benutzername und Passwort) sind durch Properties definiert, die der Jetty-Webserver lädt. Beim Aufruf der Webanwendung der Sensorplattform öffnet der Browser automatisch ein weiteres Fenster und fragt die Login-Daten vom Benutzer ab (siehe Abbildung 30). Auf die Startseite der Anwendung gelangt dieser nur nach Eingabe korrekter Login-Daten. Durch Abbrechen wird eine temporäre Seite mit der Fehlermeldung angezeigt. Der Nachteil dieser Implementierung ist, dass es unendlich viele Login-Möglichkeiten gibt. Ein potentieller Angreifer könnte das Passwort durch beliebig viele Versuche erraten.

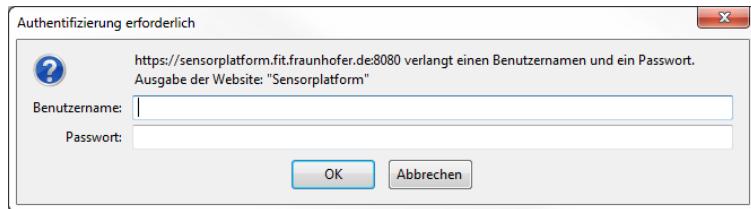


Abbildung 30: HTTP-Basic-Authentication Abfrage des Browsers

Die Authentifizierung des Webservers der Sensorplattform erfolgt durch TLS/SSL, die Autorisierung des Benutzers ist durch HTTP-Basic-Authentication umgesetzt und es verbleibt nur noch die Absicherung der externen Webserver. Das TeLiPro-Webinterface ist zwar nicht Bestandteil dieser Arbeit, zur Vervollständigung der Kommunikationskette werden diese Schnittstelle jedoch trotzdem vorgestellt.

Für die Authentifizierung und Verschlüsselung bei TeLiPro wird ebenfalls das Protokoll TLS/SSL mit einem Fraunhofer-Zertifikat genutzt. Die Autorisierung der Sensorplattform erfolgt durch Anmeldung mit einem zuvor festgelegten Benutzernamen und Passwort, für die ein Token-basiertes Verfahren zum Einsatz kommt. Die Sensorplattform nutzt für die Kommunikation zu der REST-Schnittstelle von TeLiPro den HTTP-Client der Apache HttpComponents [Thea]. Nach dem Login erhält die Sensorplattform ein Token mit einer bestimmten Gültigkeitsdauer, das bei den weiteren HTTPS-Aufrufen als Header-Eintrag mitgesendet wird.

Die Webanwendung WebHrs verwendet hingegen nicht das HTTPS-Protokoll, da dies der (kostenlose) DNS-Dienst nicht unterstützt. Außerdem erfolgt keine Autorisierung des Benutzers, bspw. durch Nutzung eines Token-basierten Systems wie bei TeLiPro, weil sonst Passwörter unverschlüsselt übertragen werden müssten. Die Tests in Kapitel 7, die das Hochladen von Herzfrequenzdaten beinhalten, werden mit dem Webinterface WebHrs durchgeführt. Absehbar ist, dass dieses Webinterface geringfügig performanter sein wird als TeLiPro. Zum Beispiel entfallen mehrere Paketumlaufzeiten (Round Trip Delays), die durch die Verwendung von TLS/SSL beim Aufbau der Verbindung auftreten würden.

6.10.2. Sicherheitsfunktionen von Bluetooth-Verbindungen

Nach der Absicherung der Schnittstellen zu den Webservern, behandelt dieses Kapitel die Sicherheitsfunktionen von Bluetooth-Verbindungen. Bei den bisher vorgestellten Sicherheitsfunktionen ist - bis auf die Eingabe von Logindaten bei HTTP-Basic-Authentication - keine Interaktion vom Benutzer erforderlich. Für die Sicherheit bei Bluetooth ändert sich dies jedoch: wie bereits im Grundlagenkapitel 3.1.4 erläutert, resultieren die Sicherheitsfunktionen aus dem Pairing- bzw. Bonding-Vorgang.

Die Herzfrequenzsensoren von Polar, TomTom und Adidas sowie der CC2650 bieten keine Möglichkeit für Bonding an. Daraus resultiert die niedrigste in Kapitel 3.1.4 angegebene LE Sicherheitsstufe, nämlich Stufe 1 im Modus 1. Um die Sensorplattform dennoch mit einer

sicheren Bluetooth-Übertragung testen zu können, kommt das BLE113 Entwicklungsboard zum Einsatz. Wie bereits in Kapitel 5.1.4 erwähnt, wird für das Board eine Firmware in der Programmiersprache BGScript geschrieben, die ein Pulsoximeter als Sensor simuliert. Der Quelltext 11 zeigt einen Ausschnitt der XML-Datei, die den GATT-Server definiert. Die Messwerte im GATT-Server sind hartkodiert, die der Mikrocontroller jede Sekunde in der Characteristic 2A5F (Zeile 9-13) mit dem Wert 000062003C₁₆ aktualisiert, um damit eine Notification auszulösen. Die Programm basiert auf dem Beispiel Whitelist Peripheral von Jeff Rowberg [Row13]. Die daraus kompilierte Firmware verwendet Just Works als Pairing-Methode und bietet darüber hinaus Möglichkeiten zur Verwaltung von Bonds (Sammlung von Schlüsseln eines Bonding-Vorgangs).

Die Firmware des BLE113 Entwicklungsboards ergänzt das Beispiel um die Minimalkonfiguration für den Service, der von der Bluetooth-Spezifikation für Pulsoximeter erforderlich ist [Blue]. In Zeile 11 ist das Datenfeld so markiert, dass mindestens LE Sicherheitsmodus 1 Stufe 2 (siehe Abschnitt 3.1.4 der Grundlagen) benötigt wird, um das Feld auszulesen. Der Name dieser Eigenschaft authenticated_read ist dabei missverständlich, da es sich um Verschlüsselung handelt. Die Characteristic wird, wie bei den Herzfrequenzsensoren, durch einen Software-Timer jede Sekunde aktualisiert, weswegen die Rate für Notifications auch auf ein Hertz festgelegt wird.

```

1  <!--1822: org.bluetooth.service.pulse_oximeter-->
2  <service uuid="1822" id="pulse_oximeter">
3    <description>Pulse Oximeter Service</description>
4
5  <!--2A5E: org.bluetooth.characteristic.plx_spot_check_measurement-->
6  <!--omitted-->
7
8  <!--2A5F: org.bluetooth.characteristic.plx_continuous_measurement-->
9  <characteristic uuid="2A5F" id="c_plx_continuous_measurement">
10   <description>Periodic Pulse Oximetry Measurements</description>
11   <properties read="true" authenticated_read="true" notify="true"/>
12   <value type="hex" length="5"/>
13 </characteristic>
14
15 <!--2A60: org.bluetooth.characteristic.plx_features-->
16 <characteristic uuid="2A60" id="c_plx_features">
17   <description>PLX Features Supported</description>
18   <properties read="true"/>
19   <value type="hex">0000</value>
20 </characteristic>
21
22 </service>
```

Quelltext 11: Deklaration des GATT-Servers eines Pulsoximeters

Anstatt Just Works Pairing nutzt diese Firmware das Verfahren Passkey-Entry. Das Entwicklungsboard zeigt den auf 6 Ziffern festgelegten PIN auf dem integrierten Display an, was die dargestellte Einstellung in Quelltext 12 umsetzt. Die Eins als erster Parameter definiert den MITM-Schutz, während der zweite die Länge des PINs auf 6 Zeichen festlegt. Für die Ausgabemöglichkeit des Sensors gibt es nur einen Bildschirm, auf dem der PIN bis zu 30 Sekunden lang angezeigt wird und der innerhalb dieser Zeit auf der Sensorplattform eingegeben werden muss.

```
1 call sm_set_parameters(1, 6, sm_io_capability_displayonly)
```

Quelltext 12: Einstellungen des Security-Managers von Bluetooth in BGScript

Sobald die Sensorplattform die Verbindung aufbaut, erzwingt die Sensorfirmware das Pairing. Das Gatttool selbst kann das Pairing nicht durchführen, wohl aber vorhandene Pairing-Schlüssel in den höheren Sicherheitsstufen nutzen. Das heißt, Pairing muss durchgeführt werden, bevor ein Sensor mit der Sensorplattform verbunden werden soll. Der Befehl sec-level stellt die Sicherheitsstufe für das Gatttool ein. Die folgenden Parameter als Sicherheitsstufen aus Kapitel 3.1.4 gibt es für dieses Programm:

- Low: Entspricht LE Sicherheitsmodus 1 Stufe 1.
- Medium: Entspricht LE Sicherheitsmodus 1 Stufe 2.
- High: Entspricht LE Sicherheitsmodus 1 Stufe 3 und 4.

Der Pairing-Vorgang lässt sich grundsätzlich auch mit der Webanwendung der Sensorplattform kombinieren. So könnte bspw. das medizinische Fachpersonal den abgelesenen PIN in die Webanwendung eintragen, der anschließend zum Bluetooth-Treiber weitergeleitet wird. Diese Funktion ist jedoch aufwendig zu implementieren und wird nur einmalig pro Sensor ausgeführt, weshalb in der Webanwendung der Sensorplattform darauf verzichtet wird.

Das Pairing wird mit dem Programm bluetoothctl von BlueZ durchgeführt. Darüber sind alle wichtigen Bluetooth-Funktionen für den Benutzer erreichbar. Das Pairing bei der Sensorplattform muss ein Benutzer z. B. über einen SSH-Zugang im Terminal durchführen. Der Befehl *agent KeyboardDisplay* im interaktiven Programm bluetoothctl lädt die entsprechende Beschreibung der Sensorplattform. Beim Verbindungsauftbau wird vom Sensor das Pairing initiiert und dessen PIN auf dem Display angezeigt. Der Benutzer tätigt die Eingabe des PINs in bluetoothctl. Entspricht die Eingabe des Pins dem angezeigten Wert auf dem Bildschirm des Sensors, ist das Pairing erfolgreich und alle weiteren Schlüssel werden erzeugt.

In der Datei `/var/lib/bluetooth/<sensorplatformBTaddress>/<sensorBTaddress>/info` lässt sich das Pairing verifizieren. Für das BLE113 Entwicklungsboard ist der Inhalt der Datei in Quelltext 13 dargestellt. Die Felder für die unterschiedlichen Schlüssel ab Zeile 10, sind nur nach erfolgreichem Pairing vorhanden. Die allgemeinen Informationen (Zeile 1-8) speichert BlueZ auch bei Verbindungen von bluetoothctl ohne die Verwendung von Sicherheitsfunktionen. Anhand der Werte bei den *Authenticated*-Feldern (Zeile 16, 21, 25 und 32) lässt sich außerdem LE Sicherheitsmodus 1 Stufe 2 von Stufe 3 bzw. 4 unterscheiden. Im Fall von Just Works-Pairing sind die Werte auf *false* bzw. *0* gesetzt.

```

1 [General]
2 Name=BLE113 Pulse Oximeter
3 AddressType=public
4 SupportedTechnologies=LE;
5 Trusted=false
6 Blocked=false
7 Services=00001800-0000-1000-8000-00805f9b34fb;0000180a
   -0000-1000-8000-00805f9b34fb;00001822-0000-1000-8000-00805f9b34fb;
8 Appearance=0x410c
9
10 [IdentityResolvingKey]
11 Key=EOBDE25048E2D2B74864B5ACF4782F34
12
13 [RemoteSignatureKey]
14 Key=ADFC807F0E807ED997D18634C10121CE
15 Counter=0
16 Authenticated=true
17
18 [LocalSignatureKey]
```

```

19 | Key=D4943318150CA340E090AB7CF302CDAC
20 | Counter=0
21 | Authenticated=true
22 |
23 | [LongTermKey]
24 | Key=C17B51C9F98678AD2CA5AF0D795170E4
25 | Authenticated=1
26 | EncSize=16
27 | EDIV=61046
28 | Rand=12701856190343943631
29 |
30 | [SlaveLongTermKey]
31 | Key=FB34185AD34FF98E5B56D2BC819CBD7B
32 | Authenticated=1
33 | EncSize=16
34 | EDIV=22528
35 | Rand=7712578126568108281

```

Quelltext 13: Gespeicherte Schlüssel der Sensorplattform durch Passkey-Entry-Pairing

Generell gilt als Design-Richtlinie, dass der Verbindungsauflauf zu anderen Bluetooth-Geräten immer durch die Sensorplattform erfolgt. Umgekehrt ist dies nicht möglich, da die Sichtbarkeit der Sensorplattform für andere Bluetooth-Geräte deaktiviert ist. Die Sensorplattform hat keine Kenntnisse über die Input-/Output-Möglichkeiten des Sensors. Deshalb wird - falls unterstützt - das Pairing von dem Sensor initiiert. Damit die Sensorplattform korrekt mit dem Pulsoximeter kommunizieren kann, ist vorheriges Pairing obligatorisch. Einige Parameter sind in der Anwendung hartkodiert, bspw. die zu verwendende Sicherheitsstufe *high* für das Gatttool des BLE113 (Verschlüsselung und Authentifizierung mit MITM-Schutz). Die Verbindung schlägt fehl, falls das Pairing zuvor nicht durchgeführt wurde. Als Ausbaustufe ist es vorstellbar, dass die Sensorplattform dynamisch mögliche Sicherheitsfunktionen für die Bluetooth-Verbindung zu einem Sensor detektiert und sich weitgehend selbst konfiguriert.

Um zu überprüfen, ob tatsächlich die Verschlüsselung für die Bluetooth-Verbindung des Pulsoximeters genutzt wird, ist das Programm btmon gebräuchlich. Es ist ebenfalls Teil der Installation von BlueZ und kann bspw. in einer separaten SSH-Session zum Debuggen der Verbindungen ausgeführt werden. Btmon stellt in Echtzeit Informationen über Bluetooth-Pakete dar. Quelltext 14 zeigt einen Ausschnitt beim Verbindungsauflauf zum BLE113 Entwicklungsboard, wenn eine Aufzeichnung von der Sensorplattform gestartet wird. Erkennbar ist in Zeile 1, dass eine verschlüsselte Verbindung initiiert wird. Auch wird der genutzte Long-Term-Key ausgegeben (Zeile 5). Die beiden Geräte sind demnach bereits gepaart. In Zeile 12 ist abschließend die verschlüsselte Verbindung mit AES im CCM-Modus bestätigt. Um die Verbindung noch weiter zu inspizieren, gibt es bspw. Bluetooth-Sniffer, die in Kombination mit der Software WireShark Bluetooth-Pakete aufzeichnen und darstellen.

Btmon, dessen Ausgabe in Quelltext 14 gezeigt ist, eignet sich unter anderem dafür, die AES-Verschlüsselung (mindestens LE Sicherheitsstufe 1 Modus 2) festzustellen. Ob der Sensor zuvor mit Just Works oder mit Passkey-Entry gepaart wurde, ist daraus jedoch nicht ersichtlich. Der durch Authentifizierung resultierende Modus 3 bzw. Modus 4 ist nur durch den Pairing-Vorgang bestimmt.

```

1 < HCI Command: LE Start Encryption (0x08|0x0019) plen 28      [hci0]
   68826.912511
2 Handle: 64
3 Random number: 0xb0460ee8658c15cf
4 Encrypted diversifier: 0xee76

```

```

5 Long term key: c17b51c9f98678ad2ca5af0d795170e4
6 > HCI Event: Command Status (0x0f) plen 4 [hci0]
   68826.913396
7 LE Start Encryption (0x08|0x0019) ncmd 1
8 Status: Success (0x00)
9 > HCI Event: Encryption Change (0x08) plen 4 [hci0]
   68827.249841
10 Status: Success (0x00)
11 Handle: 64
12 Encryption: Enabled with AES-CCM (0x01)

```

Quelltext 14: Initialisierung einer verschlüsselten Bluetooth-Verbindung

Zum Vergleich zeigt Quelltext 15 den Verbindungsauflauf bei dem Polar H7 Herzfrequenzsensor, der nur LE Sicherheitsstufe 1 Modus 1 unterstützt. In dem Textauszug sind keine Angaben bezüglich Verschlüsselung mit AES erkennbar. Es handelt sich lediglich um eine standardmäßige Verbindung, erkennbar an den Zeilen 1, 15, und 18.

```

1 < HCI Command: LE Create Connection (0x08|0x000d) plen 25 [hci0]
   18.958994
2 Scan interval: 60.000 msec (0x0060)
3 Scan window: 60.000 msec (0x0060)
4 Filter policy: White list is not used (0x00)
5 Peer address type: Public (0x00)
6 Peer address: 00:22:D0:AA:1F:B1 (Polar Electro Oy)
7 Own address type: Public (0x00)
8 Min connection interval: 50.00 msec (0x0028)
9 Max connection interval: 70.00 msec (0x0038)
10 Connection latency: 0x0000
11 Supervision timeout: 420 msec (0x002a)
12 Min connection length: 0.000 msec (0x0000)
13 Max connection length: 0.000 msec (0x0000)
14 > HCI Event: Command Status (0x0f) plen 4 [hci0]
   18.959776
15 LE Create Connection (0x08|0x000d) ncmd 1
16 Status: Success (0x00)
17 > HCI Event: LE Meta Event (0x3e) plen 19 [hci0]
   18.993515
18 LE Connection Complete (0x01)
19 Status: Success (0x00)

```

Quelltext 15: Initialisierung einer unverschlüsselten Bluetooth-Verbindung

In Quelltext 13 sind bereits die Schlüssel aufgelistet, die BlueZ auf der Sensorplattform speichert, die entsprechend auch auf dem Sensor vorhanden sind. Das BLE113 Entwicklungsboard nutzt einen für Bonds reservierten Speicherbereich im Electrically Erasable Programmable Read-Only Memory (EEPROM). Der höhere Speicherbedarf (und damit verbundene höhere Kosten bei der Produktion) sowie mangelnde Input-/Output-Möglichkeiten auf den Geräten sind mögliche Gründe, warum Sensoren wie bspw. der CC2650 kein Pairing unterstützen. Auf dem BLE113 Entwicklungsboard können maximal 8 Bonds gespeichert werden, bzw. das Board könnte mit maximal 8 Sensorplattformen gepaart werden. Daraus ergibt sich die Notwendigkeit, bestehende Bonds auf den Geräten auch wieder zu löschen. Der Benutzer könnte bspw. durch Drücken eines Knopfes alle Bonds löschen.

In Abbildung 31 ist ein Entwurf für eine sichere Sensorfirmware als Aktivitätsdiagramm vorgestellt. Nach dem Start ist der Sensor nur sichtbar, falls noch keine Bonds gespeichert sind. Ist mindestens ein Bond vorhanden, wird die Bluetooth-Sichtbarkeit ausgeschaltet

um Tracking zu erschweren. Mit einem separaten Knopf kann der Benutzer die Sichtbarkeit aktivieren, die für das Pairing des Sensors mit der Sensorplattform benötigt wird. Durch Drücken des zweiten Knopfes werden alle bisher gespeicherten Bonds gelöscht und die Bluetooth-Sichtbarkeit wieder aktiviert. Ein integriertes Display und zwei Knöpfe sind damit die minimalen Hardwareanforderungen an einen Sensor, um die größtmögliche Sicherheit für Bluetooth-Verbindungen zu gewährleisten. Zudem ist ein Bluetooth-Chip und -Stack mit Version 4.2 erforderlich. Wird dann das Pairing mit Passkey-Entry oder Numeric Comparison durchgeführt, ergibt sich für Bluetooth-Verbindungen die LE Sicherheitsstufe 1 Modus 4.

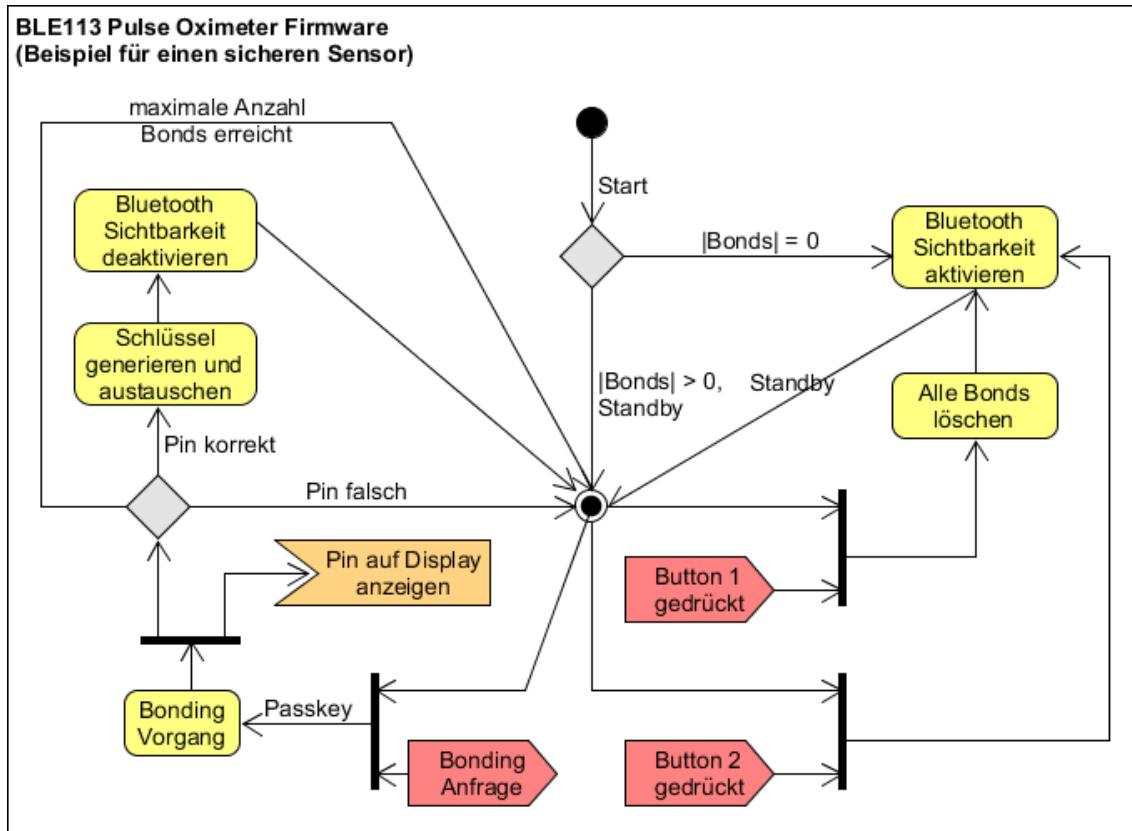


Abbildung 31: Aktivitätsdiagramm der sicheren BLE113 Firmware

Die Verwendung von Bonding-Informationen für Sicherheitsfunktionen macht es ebenfalls erforderlich, die Schlüssel auf beiden Geräten konsistent zu halten. Wird bspw. der Sensor mit einer neuen Firmware geflasht, gehen in der Regel auch die Bonds verloren, weil der nicht-flüchtige Speicher ebenfalls gelöscht wird. Bleibt die Bluetooth-Adresse und der Sensorname gleich, schlägt ein möglicher Verbindungsversuch der Sensorplattform fehl. Zunächst müssen die alten, nicht mehr gültigen Schlüssel auch auf der Sensorplattform gelöscht werden. Fällt die Verwaltung der Bonds in den Verantwortungsbereich des medizinischen Fachpersonals, ist der Umgang mit der Sensorplattform gegebenenfalls erschwert, weil mehr technische Hintergründe erforderlich sind. Nachfolgend sind Möglichkeiten vorgestellt, wie dieses Problem in der Praxis behandelt werden kann.

1. Bonding-Diskrepanz durch die Sensorplattform erkennen lassen und alte Informationen löschen. Den Pairing-Vorgang anschließend erneut durchführen.
2. Die Anzahl der Sensorplattformen begrenzen, zu denen ein einzelner Sensor verbinden kann. Das BLE113 Entwicklungsboard könnte mit maximal 8 Sensorplattformen kommunizieren.

3. Ausreichend Speicher in den Sensor verbauen, sodass Bonding-Informationen auf dem Sensor nur selten oder nie per Knopf gelöscht werden müssen.

Abschließend lässt sich festhalten, dass Bluetooth ab der Version 4.2 und durch Verwendung geeigneter Pairing-Methoden eine hohe Sicherheit für die Funkverbindung bietet. Für das Pairing notwendig ist jedoch eine geeignete Hardware- und Softwareausstattung der Sensoren. Diese können sowohl Einfluss auf den Preis der Geräte als auch negativen Einfluss auf die Akkulaufzeit nehmen, weil bspw. ein Display mehr Strom verbraucht. Es ist unwahrscheinlich, dass das Pairing für Bluetooth-Geräte von der Spezifikation zukünftig erzwungen wird, daher verfügen die Gerätehersteller bei der Konzeption der Hardware über viele Freiräume. Die daraus entstehenden Kompromisslösungen können sich nachteilig auf die Sicherheit der Bluetooth-Schnittstelle auswirken.

6.10.3. Sicherheitsaspekte des Betriebssystems

Die Firmware und auch das Betriebssystem sind für die Sicherheit eines medizinischen Geräts von wesentlicher Bedeutung. Ein Negativbeispiel stellt die Insulinpumpe OneTouch Ping von Animas dar, die über eine unverschlüsselte Funkverbindung ferngesteuert werden kann. Dadurch könnte ein nicht-autorisierter Nutzer die Insulindosis kontrollieren. Das Gerät wird jedoch nicht in Deutschland verkauft. Beim Linux-Betriebssystem haben Fehler in den Bibliotheken häufig weitreichenden Einfluss. In der Vergangenheit führten Bugs wie bspw. Heartbleed zu Sicherheitslücken auf einer Vielzahl von Systemen, die die Programmiererweiterung OpenSSL verwendeten [Bun14]. Auch bei anderer Software kann die Reichweite solcher Fehler ähnlich hoch sein. Nicht auszuschließen ist, dass solche Bugs tiefgreifende Sicherheitslücken im Linux-Betriebssystem einer Sensorplattform offenbaren, jedoch bisher nicht bekannt sind. Kritisch wird es erst, wenn diese Sicherheitslücken veröffentlicht werden, denn häufig werden für alte Geräte keine Aktualisierungen mehr angeboten, sodass diese Sicherheitslücken dann offen verweilen. [Hol16].

Für das Linux-Betriebssystem der Sensorplattform ist es daher ratsam, ein Paketmanagementsystem zu nutzen, über das insbesondere Sicherheits-Updates vorgenommen werden. Diese Updates sollten für einen definierten Nutzungszeitraum der Sensorplattform verfügbar sein [Bun16]. Auch auf dem 33. Chaos Communication Congress (33C3) wurden Forderungen geäußert, ein Mindesthaltbarkeitsdatum für Geräte des Internet-of-Things (IoT) einzurichten [Kre16], bis zu welchem Systeme Sicherheitsupdates erhalten [Kre16]. Diese könnten bei der Sensorplattform bspw. über den SSH-Zugang oder sogar direkt über die Webanwendung installiert werden. Die Anwendung der Sensorplattform ließe sich selbst als Linux-Paket verteilen. Bei dem Betriebssystem des Demonstrators entfällt ein Großteil der Wartungsarbeit, weil die Paketverwaltung von Raspbian direkt genutzt wird. Für die Nutzung eines individuell gefertigten Single-Board-Computers mit einer spezifischen Linux-Distribution ist es daher sinnvoll, ein Paketverwaltungssystem zu integrieren.

Außerdem bietet jedes Linux-Betriebssystem die Möglichkeit, entweder die gesamte Partition oder zumindest den Inhalt des Home-Verzeichnisses zu verschlüsseln, in dem auch die Datenbank der Sensorplattform gespeichert ist. Um bestmöglichen Datenschutz für den Patienten und Sicherheit für das System zu gewährleisten ist es ratsam, zukünftige eine der beiden Verschlüsselungsmethoden umzusetzen.

7. Tests

In diesem Kapitel werden die Tests von der Sensorplattform behandelt. Die Tests lassen sich in drei Kategorien einteilen: automatisch ausgeführte Unit-Tests, manuell durchführbare Test Cases und Tests zur Leistungsaufnahme. Die Modultests beziehen sich auf bestimmte Klassen der Implementierung, um deren korrekte Funktionsweise sicherzustellen. Mit den Test Cases kann überprüft werden, ob die Software alle Funktionen aus den Use Cases aus Kapitel 4.8 besitzt. Abschließend werden Tests zur Leistungsaufnahme des Systems durchgeführt, um eine Bewertung der Sensorplattform zu ermöglichen.

7.1. Modultests

Die Modultests überprüfen die einzelnen Komponenten der Software auf korrekte Funktionalität. Durch Verwendung der Programmiersprache Java wird für die Unit-Tests auf das Open-Source Framework *JUnit* zurückgegriffen [JUn]. In der Projektstruktur von Maven befinden sich die Tests immer in dem Unterverzeichnis `src/test/java` der einzelnen Projekte, z. B. `PolarH7/src/test/java/de/fhg/fit/biomos/sensorplatform/sensor/PolarH7Test.java`. Nach dem Kompilieren der Software durch Maven werden die Tests automatisch ausgeführt. Gegebenenfalls könnte die Sensorplattform derartige Tests auch nach jedem Start der Anwendung durchführen, um die Funktionsweise zu überprüfen oder eine mögliche Kompromittierung des Systems festzustellen. Aus zeitlichen Gründen wird jedoch für den Demonstrator darauf verzichtet. Die Modultests der Sensorplattform nutzen hauptsächlich Assertions, wobei es sich um eine generische Methode handelt, die vergleicht, ob der erwartete Soll-Wert dem Ist-Wert entspricht. Außerdem lassen sich damit Programmierobjekte vergleichen (bspw. deren Klassenzugehörigkeit bei Vererbung). Die Modultests umfassen den Controller (inklusive der SensorWrapperFactory), die Bluetooth-Schnittstelle Gatttool, die Datenbankschnittstelle sowie jeweils einen Test für den Herzfrequenzsensor Polar H7 und das BLE113 Entwicklungsboard. Im Folgenden wird zunächst der Modultest für das Gatttool vorgestellt.

7.1.1. Modultest des Gatttools

Der Modultest für das Gatttool überprüft das Aktivieren und Deaktivieren von Notifications, wobei sich die Testumgebung geringfügig von dem realen Anwendungsfall. Quelltext 16 zeigt einen Ausschnitt des Tests für den Sensor Polar H7. Die Annotation `@Test` in Zeile 1 bezeichnet eine Testmethode von JUnit. In der Anwendung wird das Objekt `streamToSensor` aus dem Stream des Gatttool-Prozesses erzeugt, weshalb der Test nur unter Linux mit einer BlueZ 5 Installation durchführbar ist. Um diese Referenz zu entfernen, kommt im Modultest ein virtueller `ByteArrayOutputStream` zum Einsatz (Zeile 3 und 4). Dies funktioniert, weil die Klasse `Java-BufferedWriter` ein Interface als Eingabeparameter im Konstruktor erwartet. In den Zeilen 6 und 10 aktiviert bzw. deaktiviert die Sensorplattform die Notifications. Der jeweils in den Stream geschriebene Steuerungsbefehl kann anschließend wieder ausgelesen und mit einer Assertion mit dem Soll-Text verglichen werden (Zeile 7 und 11). Die Handle-Adresse `0x0013` ist dabei spezifisch für den Sensor Polar H7. Die in Abhängigkeit vom Betriebssystem erzeugten Zeilenenden werden außerdem durch die `replace`-Methode entfernt, damit der Test auf allen Systemen die gleichen Ergebnisse liefert.

```
1  @Test
2  public void testStreamToSensor() {
3      ByteArrayOutputStream baos = new ByteArrayOutputStream();
4      BufferedWriter streamToSensor = new BufferedWriter(new
5          OutputStreamWriter(baos));
6      this.polarh7.enableDataNotification(streamToSensor, GatttoolImpl.
```

```

    CMD_CHAR_WRITE_CMD, GatttoolImpl.ENABLE_NOTIFICATION);
7   Assert.assertEquals("char-write-cmd 0x0013 01:00", new String(baos.
     toByteArray()).replace("\n", "").replace("\r", ""));
8   baos.reset();
9
10  this.polarh7.disableDataNotification(streamToSensor, GatttoolImpl.
     CMD_CHAR_WRITE_CMD, GatttoolImpl.DISABLE_NOTIFICATION);
11  Assert.assertEquals("char-write-cmd 0x0013 00:00", new String(baos.
     toByteArray()).replace("\n", "").replace("\r", ""));
12  baos.reset();
13 }

```

Quelltext 16: Testklasse für das Aktivieren und Deaktivieren von Bluetooth-Notifications

Für eine Ausbaustufe der Sensorplattform wäre es sinnvoll, virtuelle Sensoren zu unterstützen, wodurch insbesondere das Testen erleichtert wird, weil die Referenz auf konkrete Hardware, Treiber und Geräte entfällt. Die in Quelltext 16 vorgestellte Schnittstelle könnte bspw. in der Anwendung der Sensorplattform dynamisch konfiguriert werden, um zur Laufzeit zwischen Gatttools mit realen Bluetooth-Sensoren und virtuellen Sensoren mit Testdaten auszuwählen. Die regelmäßigen und asynchronen Notifications der Sensoren ließen sich durch die Verwendung von Multithreading simulieren, die den Betrieb der Sensorplattform in einer Testumgebung ermöglichen würden.

7.1.2. Modultest der Datenbank

Ein weiterer Modultest überprüft die Interaktion der Software mit der Datenbank, wofür zu Beginn des Tests eine temporäre Kopie der Datenbank mit leeren Tabellen erstellt wird. Der Test erzeugt zunächst einige Sample-Programmierobjekte unterschiedlicher Sensoren, die durch Hibernate gespeichert und anschließend wieder aus der Datenbank herausgeladen werden. Die Assertions testen in den Testmethoden die Wertzuweisungen der einzelnen Instanzvariablen der Samples. Dadurch lässt sich zum einen die korrekte Implementierung und Annotations in den Entity-Klassen und zum anderen die Korrektheit des Datenbank-Schemas überprüfen. Außerdem wird in den Tests die Anzahl der vorhandene Samples in den jeweiligen Tabellen verglichen. Die Voraussetzung dafür ist, dass der Test mit einer zuvor leeren Datenbank durchgeführt wird, weil auch IDs verglichen werden. Zum Schluss löscht der Test die Kopie der Datenbank.

7.1.3. Modultest der Sensoren Polar H7 und BLE113

Der Modultest für den Polar H7 Sensor testet die Handle-Adressen für den GATT-Server eines spezifischen Sensors und die Berechnungen der Samples für einzelne Werte aus den Bluetooth-Notifications. Dies umfasst die Auswertung von Bitmasken und die Konvertierung von Datentypen. Die Daten der Notifications erreichen die Anwendung als Datentyp String, deren Werte sich in einer hexadezimalen Darstellung befinden.

Die Vererbungshierarchie der Sensoren aus Kapitel 6.2 definiert unter anderem Methoden in abstrakten Klassen, die die Berechnungen für die Werte aus bestimmten Characteristics durchführen (z. B. AbstractHeartRateSensor). Da jeder Sensor eines gleichen Typs aus dieser abstrakten Klasse erbt, genügt der Test dieser Methoden in nur einer konkreten Implementierung (bspw. beim Polar H7). Der Test ist damit auch für alle neuen Sensoren gültig, die zukünftig in die Sensorplattform integriert werden und die aus einer der abstrakten Klassen erben (z. B. aus AbstractHeartRateSensor). Die Tests stellen sicher, dass das führende Kontrollbyte richtig interpretiert wird und anschließend aus den hexadezimalen Werten die korrekten Parameter berechnet werden (siehe das Berechnungsbeispiel in Kapitel 6.2). Weil der Quelltext des Tests umfangreich ist, befindet sich dieser in Anhang B.

Für das Pulsoximeter (BLE113 Firmware) gibt es ebenfalls einen Modultest, der ähnlich wie der Test für Herzfrequenzdaten konzipiert ist. Dieser gilt entsprechend auch für neue Sensoren, deren GATT-Server Bluetooth-spezifikationskonform sind und die in der Anwendung aus der Klasse AbstractPulseOximeterSensor erben. Der Modultest umfasst allerdings nur die Minimalkonfiguration der Pulsoximetrie-Characteristic (siehe Anhang C).

Es gibt außerdem einen Modultest für den Sensor CC2650, der ebenfalls die Berechnungen der Sensorwerte überprüft. Auf Grund der individuellen Characteristics des Sensors, lässt sich dieser Test nur für den CC2650 nutzen. Quelltext 17 zeigt die Testmethode für die Berechnungen des Bewegungssensors. Zeile 3 zeigt hexadezimale Daten als String, wie sie das Gatttool zurückgibt, wobei die führende Handle-Adresse nicht angegeben ist. Die Leerstellen werden entfernt und in die Methode calculateMovementSample(...) des Sensors eingegeben (Zeile 4). Die Assertions in den Zeilen 6 bis 15 überprüfen anschließend, ob die errechneten Werte korrekt sind. Abschließend soll die gleiche Berechnung mit einer falschen Handle-Adresse HANDLE_AMBIENTLIGHT_VALUE durchgeführt werden, bei der die Rückgabe dann *null* ist (Zeile 15 und 16). Durch die interne Überprüfung berechnet der Sensor tatsächlich nur Bewegungsdaten, wenn die Notification von der Handle-Adresse der Bewegungsdaten des GATT-Servers stammt.

```

1  @Test
2  public void testMovement() {
3      String data = "e8 00 39 00 41 ff c8 01 32 00 7c 0d 0a ff 8d 02 a8
4          ff";
5      CC2650MovementSample sample = this.cc2650.calculateMovementSample("2016-09-02T08:45:30.555Z", this.cc2650.gattLibrary.
6          HANDLE_MOVEMENT_VALUE,
7      data.replace(" ", ""));
8      Assert.assertEquals(Float.valueOf(1.7709924f), sample.getRotationX());
9      Assert.assertEquals(Float.valueOf(0.4351145f), sample.getRotationY());
10     Assert.assertEquals(Float.valueOf(-1.4580153f), sample.getRotationZ());
11     Assert.assertEquals(Float.valueOf(0.22265625f), sample.
12         getAccelerationX());
13     Assert.assertEquals(Float.valueOf(0.024414062f), sample.
14         getAccelerationY());
15     Assert.assertEquals(Float.valueOf(1.6855469f), sample.
16         getAccelerationZ());
17     Assert.assertEquals(Float.valueOf(-246.0f), sample.getMagnetismX());
18     Assert.assertEquals(Float.valueOf(653.0f), sample.getMagnetismY());
19     Assert.assertEquals(Float.valueOf(-88.0f), sample.getMagnetismZ());
20     Assert.assertEquals(null,
21         this.cc2650.calculateMovementSample("2016-09-02T08:45:30.555Z",
22             this.cc2650.gattLibrary.HANDLE_AMBIENTLIGHT_VALUE, data.replace(
23                 " ", "")));
24 }
```

Quelltext 17: Testmethode für die Sensorwerte der Bewegungsmessung

7.1.4. Modultest des Controllers und der SensorWrapperFactory

Die Klasse Controller steuert den gesamten Programmablauf der Sensorplattform und ist damit von zentraler Bedeutung. Dementsprechend wird der Test für diese Klasse ausführlicher vorgestellt. Der Controller-Test setzt voraus, dass dieser auf einem Linux-Betriebssystem mit BlueZ durchgeführt wird, weil dabei ein Gatttool-Prozess gestartet

wird. Außerdem muss der Polar H7 Sensor für die Bluetooth-Verbindung verfügbar sein. Der Test wird durch eine @Ignore-Annotation zunächst deaktiviert, damit Maven diesen nicht bei jeder Installation versucht auszuführen. Durch Entfernen der Markierung wird dieser im Bedarfsfall reaktiviert. Die Sensorplattform könnte diesen auch als Selbsttest beim Systemstart durchführen.

Zu Beginn des Controller-Tests findet die Initialisierung des Systems statt, dessen dazugehörige Methode setup() in Quelltext 18 angegeben ist. Die Annotation @BeforeClass gibt an, dass diese Methode einmalig vor dem JUnit-Test ausgeführt wird. Die Zeilen 4-21 legen die Properties der Sensorplattform fest. Das Framework für Dependency Injection Guice verwendet diese, um entsprechende Konfigurationen der Anwendung durchzuführen (Zeile 24-27). Dazu gehört unter anderem die Verwendung eines Raspberry Pi 3 als Hardwareplattform und WebHrs als Webserver zum Hochladen von Samples. Die Variablen injector und serverstarter aus Zeile 25 bis 27 sind außerhalb des Quelltexts deklariert und der injector wird in diesem Zusammenhang explizit verwendet, um instanzierte Objekte zu laden. In der Anwendung hingegen geschehen diese Aufrufe implizit. Außerdem wird in den Zeilen 30 bis 35 eine Sensorkonfiguration angegeben, wie sie im Anwendungsfall von dem Webinterface an die Sensorplattform übertragen wird. Diese beschreibt eine Aufzeichnung mit dem Sensor Polar H7.

```

1  @BeforeClass
2  public static void setup() {
3      // load standard properties for project setup
4      try {
5          properties.load(ClassLoader.getSystemResourceAsStream(
6              "SensorplatformApp.properties"));
7      } catch (IOException e) {
8          e.printStackTrace();
9      }
10
11     // add dynamic properties from maven profiles
12     properties.put("version", "0.1-SNAPSHOT");
13     properties.put("target.platform", "raspberrypi3");
14     properties.put("webapp.port", "8080");
15     properties.put("keystore.filename", "keystore.jks");
16     properties.put("keystore.password", "123456");
17     properties.put("webapp.username", "sensorplatform");
18     properties.put("webapp.password", "123456");
19     properties.put("http.useragent.boardname", "Junit test");
20     properties.put("webinterface.name", "WebHrs");
21     properties.put("webinterface.data.url",
22         "http://webhrs.ddns.de
23         :12345/hrs/upload");
24     properties.put("webinterface.timestamp.format",
25         "YYYY-MM-dd'T'HH:mm
26         :ss.SSS'Z'");
27
28     // setup injector for dependency injection
29     SensorplatformServletConfig sensorplatformServletConfig = new
30         SensorplatformServletConfig(properties);
31     serverstarter = new ServerStarter(properties,
32         sensorplatformServletConfig);
33     serverstarter.start();
34     injector = sensorplatformServletConfig.getCreatedInjector();
35
36     // example configuration
37     sensorConfiguration = new JSONArray();
38     JSONObject sensor1 = new JSONObject();
39     sensor1.put("name", "PolarH7");
40     sensor1.put("bdaddress", "00:22:D0:AA:1F:B1");
41     sensor1.put("settings", new JSONObject());
42     sensorConfiguration.put(sensor1);

```

36 }

Quelltext 18: Initialisierung des Systems für den Controller-Test

Im Controller-Test werden unter anderem die Bindungen durch Dependency Injection auf Basis der Properties überprüft. Die Testmethode testBindings() in Quelltext 19 überprüft, ob die Interfaces Uploader und HardwarePlatform an die konkreten Implementierungen WebHrsUploader bzw. RaspberryPi3 gebunden sind.

```

1  @Test
2  public void testBindings() {
3      Assert.assertTrue(injector.getInstance(Uploader.class) instanceof
4          WebHrsUploader);
5      Assert.assertTrue(injector.getInstance(HardwarePlatform.class)
6          instanceof RaspberryPi3);
7  }

```

Quelltext 19: Testmethode für die Konfiguration durch Properties

Anschließend ruft JUnit die in Quelltext 20 dargestellte Testmethode testSensorWrapperFactory() auf, die die Instanziierung von Sensorobjekten testet. Die Sensorkonfiguration stammt aus Quelltext 18 und umfasst den Sensor Polar H7. In Zeile 4 wird der dazugehörige AbstractSensorWrapper für eine Aufzeichnung des Testbenutzers TestFirstname TestLastName erzeugt. Das interne Sensor- und Gatttoolobjekt wird jeweils in Zeile 5 und 6 für eine bessere Übersicht in lokale Variablen gespeichert. Die Zeilen 8-15 testen die korrekte Zuweisung aller internen Zustände des Sensors. Diese Testmethode stellt damit sicher, dass die Sensorkonfiguration als JSON-Array (Textform) korrekt zu Programmierobjekten umgewandelt wird.

```

1  @Test
2  public void testSensorWrapperFactory() {
3      SensorWrapperFactory swFactory = injector.getInstance(
4          SensorWrapperFactory.class);
5      List<AbstractSensorWrapper<?>> aswList = swFactory.
6          createSensorWrapper(sensorConfiguration, "TestFirstName", "TestLastName");
7      AbstractSensor<?> sensor = aswList.get(0).getSensor();
8      Gatttool gatttool = aswList.get(0).getGatttool();
9
10     Assert.assertTrue(sensor instanceof PolarH7);
11     Assert.assertEquals(SensorName.PolarH7, sensor.getName());
12     Assert.assertEquals("00:22:00:AA:1F:B1", sensor.getBDaddress());
13     Assert.assertEquals(AddressType.PUBLIC, sensor.getAddressType());
14     Assert.assertEquals(SecurityLevel.LOW, sensor.getSecurityLevel());
15     Assert.assertEquals("[{}]", sensor.getSettings().toString());
16
17     Assert.assertEquals(Gatttool.State.DISCONNECTED, gatttool.
18         getInternalState());
19  }

```

Quelltext 20: Testmethode für die Instanziierung des Sensors Polar H7

Die vorangegangenen Testmethoden beziehen sich auf die Funktionen, die die Sensorplattform vor dem Start einer Aufzeichnung durchführt. Quelltext 21 zeigt die JUnit-Methode für das Starten und Beenden einer Aufzeichnung. Zeile 4 stellt zunächst sicher, dass keine Aufzeichnung mit der Sensorplattform aktiv ist und Zeile 5 startet eine Aufzeichnung für 10 Sekunden (10000 Millisekunden) mit dem Testbenutzer TestFirstName TestLastName.

Der letzte Parameter *true* gibt an, dass es sich um eine neue, nicht unterbrochene Aufzeichnung handelt. Die Methode startRecordingPeriod() führt auch die Instanziierung der AbstractSensorWrapper durch die SensorWrapperFactory aus (siehe Quelltext 20). Anschließend wird überprüft, ob die Aufzeichnung aktiv ist und ob die Datei recording.properties existiert, in der die Sensorkonfiguration abgespeichert ist. Diese Datei wird von der Anwendung gelesen, falls eine Aufzeichnung unterbrochen wurde, sodass die Aufzeichnung mit der alten Sensorkonfiguration wiederhergestellt werden kann. Nach zweisekündigem Warten unterbricht der Test die zuvor gestartete Aufzeichnung, wobei durch explizites Abbrechen auch die Datei gelöscht wird. Um dem System anschließend Reaktionszeit für das Beenden der Aufzeichnung zu gewähren, wartet der Test erneut zwei Sekunden. Anschließend wird überprüft, ob die Sensorplattform tatsächlich wieder im Bereitschaftsmodus ist und ob die Datei gelöscht wurde.

```

1  @Test
2  public void testControllerWorkflow() {
3      Controller controller = injector.getInstance(Controller.class);
4      Assert.assertFalse(controller.isRecording());
5      controller.startRecordingPeriod(10000, "TestFirstName", "TestLastName", sensorConfiguration, true);
6      Assert.assertTrue(controller.isRecording());
7      Assert.assertTrue(new File("recording.properties").exists());
8      try {
9          Thread.sleep(2000);
10     } catch (InterruptedException e) {
11         e.printStackTrace();
12     }
13     controller.interruptController();
14     try {
15         Thread.sleep(2000);
16     } catch (InterruptedException e) {
17         e.printStackTrace();
18     }
19     Assert.assertFalse(controller.isRecording());
20     Assert.assertFalse(new File("recording.properties").exists());
21 }
```

Quelltext 21: Testmethode für das Starten und Beenden einer Aufzeichnung

Die Modultests beziehen sich auf einige grundlegende Funktionen, um die Sensorplattform zu initialisieren und mit dieser Aufzeichnungen zu starten. Der Nachteil bei den Modultests ist, dass eine komplexe Testumgebung existieren muss. Der Controller-Test kann nur auf Linux-Systemen mit BlueZ ausgeführt werden und es muss ein physischer Sensor vorhanden sein. Andere Funktionen, bspw. das Hochladen von Herzfrequenzdaten zum Webserver WebHrs sind noch nicht durch Modultests erfasst. Der Test könnte durch JUnit erfolgreich ausgeführt werden, auch wenn es keine Internetverbindung zum WebHrs-Server gibt. Das Hochladen von Samples wird in Kapitel 7.3 daher noch genauer betrachtet. Zunächst behandelt das Kapitel 7.2 jedoch die Test Cases, mit denen alle Anwendungsfälle aus Kapitel 4.8 getestet werden.

7.2. Testfälle (Test Cases)

Durch die Testfälle kann ein Tester feststellen, ob die Funktionalität der Software gemäß der zuvor definierten Anforderungen gegeben ist. Ein Test Case erzeugt in wenigen Schritten ein reproduzierbares Szenario für die Sensorplattform, in dem ein bestimmtes Verhalten der Anwendung erwartet wird. Die Test Cases der Sensorplattform beinhalten die folgenden Informationen:

- Die allgemeinen Eingabedaten für Autorisierungen und Angaben zu den Aufzeichnungen.
- Eine Beschreibung zu der Funktionalität, die getestet wird.
- Die Information, ob es sich um einen Positiv- oder Negativtest handelt.
- Referenz zum Anwendungsfall aus Kapitel 4.8, auf den sich der Testfall bezieht.
- Mögliche Vorbedingungen, die der Tester schaffen muss.
- Eine Anweisung, wie der Test durchzuführen ist.
- Das erwartete Ergebnis, bzw. die erwartete Funktion der Software.

Die Testfälle sehen unter anderem die Interaktion mit den Sensoren und dem Webserver der Sensorplattform vor. Auf Grund der fehlenden Testumgebung können die Test Cases nicht automatisiert als Modultests durchgeführt werden. Außerdem besitzt die Sensorplattform Laufzeit-übergreifendes Verhalten, das sich nur schwer mit automatisierten Tests nachbilden lässt. Zum Beispiel versucht das Gerät nach dem Start des Betriebssystems und der Anwendung eine vorherige Aufzeichnung fortzuführen, falls eine Aufzeichnung unterbrochen wurde. Für die Durchführung der Test Cases ist deshalb eine Testperson erforderlich.

Die Test Cases befinden sich in Anhang A. Tabelle 7 zeigt, welche Testfälle sich auf die in Kapitel 4.8 genannten Anwendungsfälle beziehen. Ein schwarzer Punkt bedeutet, dass der Test Case für einen spezifischen Use Case konzipiert ist. Ein blauer Punkt zeigt an, dass der Test Case auch einen Teil von anderen Use Cases testet. Gegebenenfalls ist dieser Test aber nicht vollständig, weil z. B. nicht alle möglichen Kombinationen von Sensoren in der Sensorkonfiguration getestet werden können. Schon allein durch die fünf Sensoren, die initial in das System integriert sind, würden sich $5! = 120$ verschiedene Möglichkeiten für die Sensorkonfiguration ergeben. Die Test Cases überprüfen deshalb stichprobenhaft eine Auswahl von Konfigurationen. Testfälle ohne Referenz zu einem Use Case beziehen sich auf die allgemeine Konnektivität der Sensorplattform, die nicht explizit als Use Case formuliert ist. Dies umfasst z. B. den SSH-Zugang oder die Unerreichbarkeit der Webanwendung über das Mobilfunknetz als Sicherheitsfunktion. In der entsprechenden Zeile kommt dann kein Punkt vor.

Durch Tabelle 7 wird ebenfalls deutlich, dass es in jeder Spalte, für jeden Anwendungsfall mindestens einen Testfall gibt. Wenn viele Test Cases Teilstücke eines Use Cases testen, wird dies auch als ausreichend bewertet. Viele blaue Punkte in der Tabelle sind dann gleichbedeutend mit einem schwarzen Punkt. Mit den Test Cases kann somit festgestellt werden, dass die Anwendung der Sensorplattform alle Anforderungen aus den Use Cases erfüllt.

Sowohl die Modultests als auch die Testfälle beziehen sich auf die Funktionalität der Software. Im Folgenden werden weitere Tests mit der Sensorplattform durchgeführt, um die Leistungsfähigkeit der Implementierung auf der ausgewählten Hardware anhand bestimmter Parameter zu bewerten.

Test Cases	Use Cases																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
TEST-1.1	●																
TEST-2.1		●															
TEST-2.2		●															
TEST-2.3		●															
TEST-3.1	●																
TEST-3.2																	
TEST-3.3																	
TEST-4.1																	
TEST-4.2																	
TEST-4.3																	
TEST-5.1	●																
TEST-5.2			●	●					●								
TEST-5.3			●	●			●		●								
TEST-5.4			●	●				●		●							
TEST-5.5			●	●					●			●					
TEST-5.6			●	●					●		●						
TEST-5.7			●	●					●				●				
TEST-5.8			●	●					●								
TEST-5.9			●	●					●								
TEST-5.10			●	●					●								●
TEST-5.11			●	●	●				●								
TEST-5.12					●												
TEST-5.13					●												
TEST-5.14			●	●					●	●							
TEST-5.15			●	●					●								●
TEST-6.1													●				
TEST-6.2													●				
TEST-6.3														●			
TEST-6.4														●			
TEST-6.5														●			
TEST-7.1			●	●					●								
TEST-7.2			●	●					●								
TEST-8.1						●											
Gesamt	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Tabelle 7: Testbarkeit der Use Cases durch Test Cases

7.3. Leistungsaufnahme der Sensorplattform

Die Tests zur Leistungsaufnahme der Sensorplattform umfassen die Parameter Stromverbrauch, Prozessor- und Hauptspeicherauslastung und Signalqualität der Mobilfunkverbindung. Außerdem wird die Reichweite der Bluetooth Low Energy (BLE) Verbindung überprüft und es wird zudem festgestellt, wie hoch die Datenmengen durch das Abspeichern von Samples in der Datenbank ausfallen. Diese Parameter erlauben anschließend eine Bewertung der Implementierung auf der ausgewählten Hardware. Im Folgenden wird zunächst der Stromverbrauch der Sensorplattform gemessen.

7.3.1. Stromverbrauch

Der Stromverbrauch ist von signifikanter Bedeutung, um die Laufzeit von mobilen Geräten zu maximieren. Die Sensorplattform soll mit einer Akkuladung (3000 mAh) eine Betriebszeit von 16 Stunden erreichen (siehe Anforderungsanalyse in Kapitel 4). Es wird zunächst der Grundstromverbrauch des Raspberry Pi 3 mit jeweils verschiedenen Anschlüssen (z. B. für Netzwerk) festgestellt. Tabelle 8 zeigt in der zweiten Spalte die unterschiedlichen Konfigurationsmöglichkeiten. Die Messung mit der ID 1 bezeichnet die Basiskonfiguration der Sensorplattform und die Messungen 2 bis 7 beinhalten jeweils eine Ergänzung zu der Basiskonfiguration. Der daraus resultierende Stromverbrauch ist in der vierten Spalte dargestellt, wobei es sich um Durchschnittswerte handelt, die innerhalb einer Stunde im Abstand von 5 Minuten gemessen werden (20 Messungen). Dieser Stromverbrauch im Bereitschaftsmodus (in der Spalte mit der Markierung Ø dargestellt) ist jedoch konstant und unterliegt bei dieser Messgenauigkeit keinen Schwankungen. Wird davon der Verbrauch der Basiskonfiguration subtrahiert, ergibt sich der Stromverbrauch einer einzelnen Veränderung am System in Spalte drei. Die Messgenauigkeit des Multimeters ist auf zwei Dezimalstellen genau und dessen Ausregelzeit ist durch den längeren Messzeitraum berücksichtigt. Genauere Ergebnisse wären nur mit Messungen eines Oszilloskops und einer speziellen Schaltung am Eingang des Raspberry Pi 3 möglich. Für die Messung 6 kann der HDMI-Ausgang bspw. bei jedem Programmstart mit dem Befehl `/opt/vc/bin/tvservice -o` deaktiviert werden. Der Audio-Ausgang für Messung 7 wird deaktiviert, indem das Laden des Treibers verhindert wird. Dies geschieht in der Datei `/etc/modprobe.d/audio-blacklist.conf` mit einer Zeile `blacklist snd_bcm2835` und anschließendem Neustart der Sensorplattform.

Aus der Tabelle 8 geht hervor, dass die Verwendung von zusätzlichen (Netzwerk-) Anschlüssen zu einem höheren Stromverbrauch führen. Das Deaktivieren des HDMI-Ausgangs ist zunächst die einzige Möglichkeit, den Stromverbrauch zu reduzieren. Aus diesem Teilergebnis geht hervor, dass der Stromverbrauch der Sensorplattform im Bereitschaftsmodus mindestens 250 mA (1,25 Watt) beträgt (Tabelle 8 Messung 8). Der HDMI-Ausgang wird bei jedem Start deaktiviert, während der Surfstick für die notwendige Mobilfunkverbindung immer angeschlossen ist. Mit der Akkukapazität von 3000 mAh errechnet sich die Laufzeit im Standby-Modus wie folgt: $3000mA \cdot h : 250mA = 12h$. Auf Grund des verhältnismäßig hohen Stromverbrauchs des Raspberry Pi 3 und des Surfsticks, kann die Sensorplattform sogar im Bereitschaftsmodus die Laufzeit von 16 Stunden nicht erreichen.

Die bisherigen Messungen beziehen sich ausschließlich auf den Standby-Betrieb der Sensorplattform. Inwiefern die Datenaufzeichnung zu einem höheren Stromverbrauch des Systems führt, soll an dieser Stelle überprüft werden, wofür zunächst einige Einstellungen notwendig sind.

Bei allen weiteren Messungen ist die Sensorplattform sowohl mit dem Mobilfunknetz, als auch per Kabel mit einem lokalen Netzwerk verbunden. Dies ist erforderlich, da sonst der Webserver der Sensorplattform nicht erreichbar ist und keine Aufzeichnungen gestartet werden können. Auf Grund der Kabelnetzwerkverbindung ist der in den Tabellen angege-

ID	Konfiguration	Stromverbrauch [mAh]	
		ID _n -ID ₁	∅
1	Basiskonfiguration: • Raspbian Lite 4.413-v7+ • Keine Peripheriegeräte • Keine Netzwerkverbindungen	0	220
2	Anwendung der Sensorplattform (Bereitschaftsmodus)	0	220
3	Netzwerk (Kabel)	20	240
4	Netzwerk (WLAN)	20	240
5	Mobilfunk (Surfstick)	40	260
6	HDMI Ausgang deaktiviert	-10	210
7	Audio-Ausgang deaktiviert	<10	220
8	Standardkonfiguration (Messungen 1 + 2 + 5 + 6)	30	250
9	Testkonfiguration (Messungen 1 + 2 + 3 + 5 + 6)	50	270

Tabelle 8: Stromverbrauch der Sensorplattform im Bereitschaftsmodus

bene Strombedarf somit immer 20 Milliampere höher als es unter realen Bedingungen der Fall wäre. Der Stromverbrauch der Sensorplattform im Bereitschaftsmodus beträgt dann nicht 250 mA (Tabelle 8 Messung 8), sondern 270 mA (Tabelle 8 Messung 9). Die Aufzeichnungsdauer beträgt jeweils 5 Minuten, damit die Messungen nicht durch kurzfristige Schwankungen verfälscht werden.

Das Routing für Netzwerkpakete ist standardmäßig so konfiguriert, dass die Kabelnetzwerkverbindung bevorzugt genutzt wird. Durch unterschiedliche Latenzen lässt sich die Verwendung der beiden Netzwerkschnittstellen unterscheiden. Bei 3G-Netzen ist die Verzögerung höher als bei der Kabelverbindung, was in Kapitel 7.3.3 getestet wird. Das Hochladen von Samples muss für die nachfolgenden Tests jedoch über die Mobilfunkverbindung erfolgen. Die Einstellung dafür befindet sich in der Datei `/etc/ppp/peers/wvdial`. In dieser werden zwei Einträge `defaultroute` und `replacedefaultroute` ergänzt und anschließend die Mobilfunkverbindung erneuert. Eingehender Datenverkehr durch Zugriff auf die Webanwendung der Sensorplattform und der SSH-Zugang sind damit weiterhin über die Interfaces `eth0` (Ethernet) und `wlan0` (WLAN) möglich. Sämtliche ausgehende Daten werden aber - falls möglich - über die Mobilfunkverbindung gesendet. Die Sensorplattform speichert alle Sensordaten immer in der Datenbank, unabhängig davon, ob das Hochladen erfolgreich war. Dies ist nicht unbedingt notwendig, bietet jedoch weitere Informationen für nachfolgende Tests, weil bspw. die Anzahl der nicht-übertragenen Samples in Bezug zu der gesamten Anzahl von Samples gestellt werden kann.

Eine Aufzeichnung der Sensorplattform lässt sich in die drei Abschnitte T_{Start} , $T_{\text{Aufzeichnung}}$ und T_{Ende} unterteilen. In T_{Start} erzeugt die Sensorplattform die Gatttool-Prozesse und verbindet sich über Bluetooth Low Energy (BLE) mit den Sensoren. Wenn alle Sensoren beim Start verfügbar sind, ist diese Phase in ungefähr zwei Sekunden abgeschlossen. In $T_{\text{Aufzeichnung}}$ werden Bluetooth-Notifications empfangen, Samples in die Datenbank gespeichert und zu einem Webserver übertragen. In T_{Ende} trennt die Sensorplattform die Verbindungen und beendet die Prozesse, wofür ungefähr eine Sekunde benötigt wird. So wohl in T_{Start} als auch in T_{Ende} verbraucht die Sensorplattform geringfügig mehr Strom als in $T_{\text{Aufzeichnung}}$. Weil $T_{\text{Aufzeichnung}}$ in der Praxis und auch bei den Tests signifikant größer ist als die anderen beiden Zeiträume, wird nur $T_{\text{Aufzeichnung}}$ bei den nachfolgen-

den Messungen berücksichtigt. Die Tests werden jeweils 5 Minuten lang ausgeführt und Mittelwerte für einzelne Parameter berechnet.

Tabelle 9 stellt den Stromverbrauch während der Datenaufzeichnung bei bestimmten Sensorkonfigurationen dar. Aus den Messungen 1, 3 und 7 geht hervor, dass die Datenaufzeichnung eines Sensors mit einer Abtastrate von einem Hertz den Stromverbrauch im Rahmen der Messgenauigkeit nicht beeinflusst. Werden zusätzlich Herzfrequenzdaten zum Webserver hochgeladen (Messung 2), ist der Stromverbrauch zwischen 30 und 210 Milliampere höher, der auf die Leistungsaufnahme des Mobilfunkmodul zurückzuführen ist. Welche Parameter diesen teilweise sehr hohen Stromverbrauch verursachen, wird in Kapitel 7.3.3 erläutert. Der Surfstick registriert sich unter Linux mit einem maximalen Stromverbrauch für USB-2.0-Ports von 500 Milliampere (Ausgabe des Befehls `lsusb -v` im Terminal), womit diese Werte plausibel werden. Zunächst gibt es eine Auffälligkeit, die die Messungen 4 und 5 in Tabelle 9 zeigen. Der Stromverbrauch ist signifikant höher, wenn die Abtastrate für den Sensor CC2650 für alle fünf Messungen auf 3,3 Hz erhöht wird. Diese Sensorwerte werden jedoch zu keinem Webserver hochgeladen. Der Stromverbrauch muss demnach entweder durch die Java-Anwendung der Sensorplattform oder durch das Gatttool zustande kommen. Weitere Tests in Tabelle 10 schränken dieses Problem auf die Java-Anwendung ein, da die Prozessorauslastung bei dieser Sensorkonfiguration dann bei ungefähr 25% liegt (Tabelle 10 Messung 5). Die Ursache für diese hohe Auslastung ist die Datenbank bzw. die Häufigkeit von Schreibzugriffen auf die micro-SD-Karte.

ID	Sensorkonfiguration	Rate [Hz]	Upload	Stromverbrauch [mAh]	
				ID _n -ID ₁	∅
1	1x Herzfrequenzsensor	1	✗	<10	270
2	1x Herzfrequenzsensor	1	✓	30-210	300-480
3	CC2650, ein beliebiger Messwert	1	✗	<10	270
4	CC2650, 5 Messwerte	1	✗	<10	270
5	CC2650, 5 Messwerte (<i>Write Delay 500 ms</i>)	3,3	✗	50	320
6	CC2650, 5 Messwerte (<i>Write Delay 5000 ms</i>)	3,3	✗	<10	270
7	BLE113	1	✗	<10	270
8	Stresstest: 3x Herzfrequenzsensor CC2650, 5 Messwerte BLE113	1 bzw. 3,3	✗	10	280
9	Stresstest: 3x Herzfrequenzsensor CC2650, 5 Messwerte BLE113	1 bzw. 3,3	✓	50-220	320-480

Tabelle 9: Stromverbrauch der Sensorplattform während einer Aufzeichnung

Write Delay ist der Parameter, wie häufig HSQLDB den Inhalt der Datenbank auf die micro-SD-Karte abspeichert. Dieser ist standardmäßig auf 500 Millisekunden festgelegt. Werden alle 300 Millisekunden 5 verschiedene Messwerte abgespeichert, erzeugt dies mit der Standardeinstellung viel Last in der Java-Anwendung. Das Problem lässt sich lösen, indem der Wert auf bspw. 5000 Millisekunden erhöht wird, woraufhin sich der Stromverbrauch wieder normalisiert (Messung 6). Der Nachteil ist, dass die Daten von einigen Sekunden unter Umständen verloren gehen, falls die Sensorplattform abrupt beendet wird (bspw. bei Unterbrechung der Stromversorgung). In Relation zu der Aufzeichnungszeit von

mehreren Stunden ist ein derartiger Datenverlust jedoch nur von geringer Bedeutung. Zusätzliche Tests wären nötig, ob dieses Verhalten von HSQLDB auch bei micro-SD-Karten anderer Hersteller oder bei integriertem Flash-Speicher auftritt. Das Format der Partition (ext4 bei Raspbian) und die Blockgröße könnten ebenfalls Einfluss auf dieses Verhalten haben.

Der Stresstest der Messung 8 in Tabelle 9 zeigt, dass der Stromverbrauch auch durch die Verwendung vieler Sensoren nicht wesentlich höher ausfällt, als es im Bereitschaftsmodus ohne aktive Aufzeichnung der Fall ist. Da die Sensorplattform nur Bluetooth-Notifications empfängt aber selbst keine Informationen zu den Sensoren überträgt, ist dies nachvollziehbar. Aktive Verbindungen durch Bluetooth Low Energy (BLE) wirken sich deshalb im Rahmen der Messgenauigkeit nur marginal negativ auf die Laufzeit der Sensorplattform aus. Die in dieser Arbeit verwendeten Sensoren besitzen dazu verhältnismäßig hohe Abtastraten bis 3,3 Hz. Es gibt viele andere Sensoren für Vitalparameter die zukünftig mit der Sensorplattform kommunizieren könnten, die niedrigere Abtastraten haben (z. B. ein Blutdruckmessgerät).

Auf Grund des stark schwankenden Stromverbrauchs des Mobilfunkmodems lässt sich noch keine Aussage darüber treffen, wie lange die Sensorplattform tatsächlich eine Aufzeichnung durchführen kann. Bevor dazu weiter ins Detail gegangen wird, ist es zunächst empfehlenswert, Prozessor- und Hauptspeicherauslastung während der Datenaufzeichnung zu betrachten.

7.3.2. Prozessor- und Hauptspeicherauslastung

In Tabelle 10 ist die Prozessorauslastung während der Datenaufzeichnung jeweils von der Anwendung der Sensorplattform und von allen Gatttool-Prozessen dargestellt (falls mehrere Sensoren verwendet werden). Die Daten sind mit dem Befehl *top* im Terminal einsehbar und nach Prozess-ID und prozentualer Auslastung sortiert. Die CPU-Auslastung von den Prozessen der Sensorplattform sind auch im aktiven Betrieb niedrig und decken sich mit den Messungen des Stromverbrauchs. In Messung 5 und 6 wird auch noch einmal der Unterschied durch den Parameter *Write Delay* deutlich. Die Datenbank, die in der Java-Anwendung integriert ist, erzeugt dann eine hohe Prozessorauslastung. In den restlichen Fällen befindet sich der Prozessor während einer Aufzeichnung die meiste Zeit im Leerlauf. Das deutet darauf hin, dass die Leistung der CPU für die Sensorplattform überdimensioniert ist.

In der Datei `/proc/<pid>/status` ist unter anderem sichtbar, wie viel Speicher der jeweilige Prozess belegt. Die Gesamtgröße des Prozesses bestimmt der Wert `VmSize`. Der Parameter Virtual Memory Resident Set Size (`VmRSS`) bezeichnet den Speicher, der nicht aus dem RAM verschoben werden kann. Angaben über die Verwendung des Hauptspeichers gibt ebenfalls der Befehl `top` aus. Der Raspberry Pi 3 verfügt über 1024 MB Hauptspeicher, von dem nur ein kleiner Teil für andere Dienste (z. B. 16 MB als Grafikspeicher) reserviert wird. Von den verfügbaren ca. 996 MB benötigt das Betriebssystem ungefähr 250 MB. Die Anwendung der Sensorplattform belegt sowohl im Bereitschaftsmodus als auch beim Aufzeichnen ca. 98 MB Hauptspeicher (`VmRSS`) und dessen Gesamtgröße beträgt ungefähr 387 Megabyte (`VmSize`). Das erscheint zunächst als umfangreich, allerdings ist dies für Java-Anwendungen mit integriertem Webserver und Datenbank nicht unüblich. Die Größe eines Gatttools beträgt 3,4 MB (`VmRSS`) bzw. 6,2 MB (`VmSize`). Eine Reduzierung des Hauptspeichers auf z. B. 512 MB wäre daher für die Sensorplattform möglich.

ID	Sensorkonfiguration	Rate [Hz]	Upload	CPU-Last [%]	
				Java	Gatttool(s)
1	1x Herzfrequenzsensor	1	X	3	0,3
2	1x Herzfrequenzsensor	1	✓	3	0,3
3	CC2650, ein beliebiger Messwert	1	X	3	0,3
4	CC2650, 5 Messwerte	1	X	8	0,3
5	CC2650, 5 Messwerte (<i>Write Delay 500 ms</i>)	3,3	X	25	0,7
6	CC2650, 5 Messwerte (<i>Write Delay 5000 ms</i>)	3,3	X	3	0,7
7	BLE113	1	X	3	0,3
8	Stresstest: 3x Herzfrequenzsensor CC2650, 5 Messwerte BLE113	1 bzw. 3,3	X	15	2
9	Stresstest: 3x Herzfrequenzsensor CC2650, 5 Messwerte BLE113	1 bzw. 3,3	✓	15	2

Tabelle 10: Prozessorauslastung während einer Aufzeichnung

Als Exkurs wird noch die Größe der Datenbank auf der micro-SD-Karte betrachtet. Diese hängt im Wesentlichen von vier Faktoren ab:

- Anzahl der Sensoren: verschiedene Krankheiten erfordern unterschiedliche Sensoren.
- Sensorart und deren Abtastraten: zum Beispiel misst ein Beschleunigungssensor wesentlich häufiger als ein Blutdruckmessgerät. Beim CC2650 kann außerdem die Abtastrate eingestellt werden.
- Datenbankschema: zusätzlich zu den Sensordaten werden verschiedene Metainformationen gespeichert. Die Auswahl von Metadaten ist in der Regel anwendungsspezifisch und kann beliebig erweitert werden. Die Sensorplattform speichert bspw. den Vor- und Nachnamen eines Patienten, die Bluetooth-Adresse des Sensors und einige weitere Informationen.
- Hochladefunktion zu einem externen Webserver: grundsätzlich soll die Sensorplattform nur lokal Daten abspeichern, wenn das Übertragen zum Webserver nicht möglich ist. Nur für den Testbetrieb speichert die Sensorplattform immer alle Daten in der Datenbank.

Diese vier Faktoren ermöglichen eine hohe Variabilität der Datenmengen, daher ist es kaum möglich eine Aussage darüber zu treffen, wie viel Massenspeicher die Sensorplattform mindestens benötigt. Mit allen fünf Sensoren (Messung 8 in Tabelle 9 bzw. 10) und höchsten Abtastraten im Zeitraum von ca. 10 Stunden ist die Dateigröße der Datenbank kleiner als 100 MB.

Da der Prozessor mehr Rechenleistung liefern kann als von der Sensorplattform gebraucht wird, gilt es zu überprüfen, ob das Heruntertakten der CPU den Stromverbrauch reduzieren kann. Tabelle 11 zeigt die Taktfrequenzen des Prozessors bei Standardeinstellungen (Messung 1), den daraus resultierenden Stromverbrauch im Bereitschaftsmodus (mit der Testkonfiguration) sowie die Startzeit der Java-Anwendung. Der Prozessor wird dynamisch

zwischen diesen beiden Grenzen getaktet. Im Standby-Betrieb hat der Prozessor eine Taktfrequenz von 600 MHz, die unter Last bis 1200 MHz gesteigert wird. Dies lässt sich mit dem Befehl `cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq` verifizieren. In der Datei `/boot/config.txt` können derartige Systemeinstellungen mit entsprechenden Befehlen verändert werden. Die Dokumentation zu den Kommandos ist auf der Webseite der Raspberry Pi Foundation veröffentlicht [Rasa]. Dazu wird einmal die Minimaltakt auf 200 MHz gesetzt (der Maximaltakt bleibt unverändert) und eine konstante Frequenz von 300 MHz getestet (Tabelle 11 Messungen 2 und 3). Aus Versuch 2 ergibt sich, dass die Taktfrequenz der Sensorplattform im Bereitschaftsmodus die meiste Zeit 200 MHz beträgt, der Stromverbrauch und die Startzeit jedoch gleich bleiben. Bei Versuch 3 in Tabelle 11 ist die Startzeit der Sensorplattform signifikant höher, weil der Prozessor beim Systemstart nicht mehr dynamisch höher takten kann. Das Heruntertakten des Prozessors ist beim Raspberry Pi 3 somit als Maßnahme zum Senken des Stromverbrauchs ungeeignet. Erklären lassen sich diese Ergebnisse so, dass ARM-Prozessoren bereits sehr energieeffizient für den Leerlauf konzipiert sind. Gegebenenfalls würde sich durch Verringerung der Eingangsspannung am Prozessor ein messbarer Unterschied ergeben [Rasa]. Dann müsste die CPU jedoch immer mit niedrigeren Frequenzen takten, damit das System nicht instabil wird. Dies führt wiederum zu inakzeptabel langen Startzeiten. Alternativ lässt sich auch die Grafikeinheit weiter herunter takten (da diese sowieso ungenutzt ist). Dies beeinflusst allerdings auch den L2-Cache des Prozessors und würde das gesamte System zusätzlich verlangsamen [Rasa].

ID	Taktfrequenz der CPU [MHz]		Strom [mAh]	Startzeit [s]
	Min.	Max.		
1	600	1200	270	17
2	200	1200	270	17
3	300	300	270	53

Tabelle 11: Untertakten der CPU des Raspberry Pi 3

Zusammenfassend lassen sich bereits einige Aussagen über die Leistungsaufnahme der Sensorplattform formulieren. Die Bluetooth Low Energy Verbindungen sind tatsächlich energieeffizient und haben im Rahmen der vorgestellten Messgenauigkeit keinen zusätzlichen Einfluss auf den Stromverbrauch der Sensorplattform. Ohne die Übertragung von Samples zu einem externen Webserver ergibt sich daher eine Standby- und Aufzeichnungszeit von ungefähr 12 Stunden. Bei der Datenbank wird eine Standardeinstellung als kritisch erkannt und verbessert: die Datenpersistierung auf den Massenspeicher erfolgt nur noch alle fünf Sekunden anstatt alle 500 Millisekunden. Der Prozessor und auch die Größe des Hauptspeichers sind für den Anwendungsfall der Sensorplattform überdimensioniert. Dafür wären die Vorgängermodelle des Raspberry Pi ebenfalls ausreichend und nebenbei auch stromsparender, die allerdings keinen integrierten WLAN- bzw. Bluetooth-Chip [Pre16] besitzen. Weil die Laufzeit der Sensorplattform durch die Hochladefunktion stark schwanken kann, wird der Surfstick und die Mobilfunkverbindung in dem Folgenden Abschnitt noch genauer betrachtet.

7.3.3. Signalqualität der Mobilfunkverbindung

Unter bestimmten Umständen ist der Stromverbrauch des Surfsticks hoch, weil dieser mit entsprechend hoher Sendeleistung Daten zum externen Webserver überträgt. Das Ziel in diesem Abschnitt ist es, Parameter zu bestimmen, mit denen die Signalqualität der Mobilfunkverbindung ausgewertet werden kann. Die Sensorplattform soll mit diesen Werten die Hochladefunktion regulieren. Diese Parameter lassen sich dynamisch konfigurieren und bilden einen Kompromiss zwischen der Dienstgüte (Quality of Service) der Sensorplattform

und der Energieeffizienz.

Die Signalqualität der Mobilfunkverbindung wird an drei verschiedenen Standorten (jeweils in einem Gebäude) ermittelt: in Siegburg, in Sankt Augustin an der Hochschule Bonn-Rhein-Sieg (HBRs) und in Sankt Augustin am Fraunhofer FIT. Durch die Nutzung von Smartphones ist bereits bekannt, dass die Mobilfunkverbindung am Standort des Instituts besonders schlecht ist. Bei den Aufzeichnungen mit der Sensorplattform (mit dem Übertragen von Herzfrequenzdaten an einen externen Webserver) an den verschiedenen Orten, können signifikante Unterschiede beim Stromverbrauch der Sensorplattform festgestellt werden. In der ersten Iteration liest die Anwendung den RSSI-Wert von der seriellen Schnittstelle des Modems aus, weil dieser automatisch bei jeder Veränderung ausgegeben wird. Dabei lässt sich erkennen, dass dieser Parameter am Fraunhofer FIT höher (besser) ist, als an den anderen beiden Standorten. Dies ist zunächst ein Widerspruch. Tatsächlich ist es so, dass sich der Wert RSSI nicht dafür eignet, um die Signalqualität zu bestimmen (siehe das Beispiel Pilot Pollution aus Kapitel 3.1.5).

Zusätzlich zum Parameter RSSI fragt die Sensorplattform die Werte für RSCP und E_c/I_o vom Modem ab (siehe Kapitel 6.6). Alle drei Parameter in Kombination eignen sich für eine Bewertung der Signalqualität. Außerdem ermittelt die Anwendung noch den Modus und Submodus, in dem sich das Mobilfunkmodul gerade befindet (siehe Tabelle 12). Alle Parameter werden in die Log-Datei gespeichert und ebenfalls in der Webanwendung dargestellt. Um daher die Daten in Echtzeit zu beobachten, ist eine Netzwerkverbindung und ein zweiter Rechner erforderlich (ein SSH-Zugang oder Aufruf der Webanwendung). Aus diesem Grund werden für diese Messungen nur drei Standorte betrachtet. Die Messung des Stromverbrauchs am Eingang des Raspberry Pi 3 ist keinem genauen Zeitpunkt zuzuordnen, sondern müssen abgelesen werden. Dies kann zu zeitlicher Ungenauigkeit führen. Das ist jedoch insofern unkritisch, da sich die Testumgebung (und damit der Stromverbrauch des Modems) nicht sprunghaft verändert.

Tabelle 12 zeigt die Werte der Signalqualität und des Stromverbrauchs an den zuvor genannten Standorten. Es handelt sich dabei um die Durchschnittswerte bei einer Aufzeichnungsdauer von einer Stunde mit einem Herzfrequenzsensor, dessen Daten an den WebHrs-Server hochgeladen werden. Ein HTTP-Post-Aufruf (inklusive Header) der Sensorplattform zum Hochladen eines Samples, ist nur einige Hundert Bytes groß. Diese Tests werden mehrfach wiederholt und davon Durchschnittswerte ermittelt. Bei Messung 1 ist zu erkennen, dass die Parameter der Signalqualität im Institut die schlechtesten möglichen Zahlenwerte annehmen [Ibr11]. Insbesondere das Signal-zu-Interferenz-Verhältnis ist mit -32 dBm besonders schlecht. Theoretisch sollte damit keine Verbindung mehr zustande kommen, tatsächlich ist eine Mobilfunkverbindung mit einer sehr geringen Datenrate möglich. Bei den Verbindungen von Smartphones lässt sich der gleiche Effekt feststellen. Der Strombedarf des Modems beim Hochladen ist dementsprechend hoch. Der Stromverbrauch in Tabelle 12 bezieht sich immer auf das gesamte System und zeigt jeweils den Durchschnittswert für eine Stunde. Während die einzelnen Messwerte im Standby-Betrieb konstant sind, fluktuiert der Stromverbrauch bei dem Übertragen von Samples an einen Webserver. Der Grund dafür könnten unterschiedlich lange Latenzen einzelner Übertragungen sein. Von den bspw. 480 mA (Messung 1) benötigt das Board und der Surfstick im Bereitschaftsmodus bereits 270 mA (wie in den Tests zuvor). An den beiden anderen Standorten sind die Werte für RSCP und E_c/I_o besser und der Strombedarf von nur 350 mA beim Hochladen ist geringer (Messung 2 und 3). Damit ist der Zusammenhang zwischen Signalqualität und Stromverbrauch des Surfsticks nachgewiesen.

Es ist ebenfalls in Tabelle 12 erkennbar, dass sich der Surfstick während des Betriebs in unterschiedlichen Modi und Submodi befindet. Nach welchen Kriterien die Firmware zwischen den Modi wechselt, ist nicht ersichtlich, da dafür entweder die Dokumentation

ID	Ort	RSCP [dBm]	E_c/I_o [dB]	RSSI [dBm]	Modus	Submodus	Stromverbrauch [mAh]
1	FIT	-145	-32	-113	GSM/GPRS	EDGE	480
2	HBRS	85	-5	-80	WCDMA	HSPA+	350
3	Siegburg	-88	-6	-82	WCDMA	HSPA+	350

Tabelle 12: Signalqualität der Mobilfunkverbindung und Stromverbrauch der Sensorplattform

des Herstellers oder Einblick in den Quelltext der Firmware nötig wäre. Es ist jedoch zu erwarten, dass die Signalqualität und die benötigte Datenrate zu einem Zeitpunkt diese Entscheidung beeinflussen, was sich auch bei Smartphones beobachten lässt. Der hohe Stromverbrauch lässt sich so deuten, dass die Gewährleistung des Mobilfunkdienstes beim Surfstick die höchste Priorität besitzt, auch wenn der Stromverbrauch dadurch steigt. Da Surfsticks häufig an Computern mit einem größeren Akku (z. B. Notebook) oder einer stationären Stromversorgung verwendet werden, ist dies in Bezug auf die Nutzungsdauer weniger kritisch. Für die Verwendung mit der Sensorplattform im Akkubetrieb ergeben sich jedoch deutlich kürzere Laufzeiten. Nachfolgend wird eine Softwarelösung vorgestellt, um den Stromverbrauch des Mobilfunkmoduls unter bestimmten Bedingungen zu reduzieren.

Die Sensorplattform soll so konfiguriert sein, dass wenn es nicht möglich ist, energieeffizient Samples zu übertragen, diese Funktion vorübergehend eingestellt wird. Das medizinische Fachpersonal kann zu einem späteren Zeitpunkt das Übertragen über die Webanwendung nachholen. Verbessert sich die Signalqualität während einer Aufzeichnung wieder, bspw. wenn der Patient in Bewegung ist, wird das Hochladen fortgeführt.

Näherungsweise lassen sich aber Grenzwerte verwenden, die auch bei der Planung von Mobilfunknetzwerken berücksichtigt werden. Die Sensorplattform verwendet die Grenzwerte, die ein Einwählen in das Funknetzwerk gerade noch ermöglichen. Das heißt der Wert für E_c/I_o muss größer als -9 dB sein und die Werte für RSCP und RSSI müssen größer als -114 dBm bzw. -106 dBm sein. Unterschreitet einer dieser Werte den Grenzwert, wird das Hochladen von Samples zum Webserver verhindert. Sind alle Werte wieder unterschritten, wird das Hochladen fortgeführt. Die drei Parameter lassen sich über eine Properties-Datei verändern, sodass ein Kompromiss zwischen Quality of Service und Energieeffizienz gewählt werden kann. [Ele07].

Bei kritischen Werten für Vitalparameter (z. B. bei der Herzfrequenz) erfolgt bei dieser Implementierung keine Benachrichtigung mehr an einen Webserver. Als Ausbaustufe der Anwendung wäre es daher sinnvoll, wenn die Sensorplattform das Überschreiten von Grenzwerten bestimmter Vitalparameter überprüft und im Bedarfsfall die Daten trotzdem hochlädt, auch wenn dies nicht energieeffizient möglich ist. Ein Teil der Analyse würde dann schon auf der Sensorplattform stattfinden, nicht erst in einer Webinfrastruktur.

Durch die begrenzte Messgenauigkeit der micro-USB-Steckverbindungen und durch die fehlende Kontrolle über das Modem ist die Qualität der Messungen eingeschränkt. Das Messgerät zeigt nur jeweils einen Wert für Stromstärke und Spannung zwei Sekunden lang an. Durch das Ablesen gibt es keinen exakten Zeitpunkt, zu dem ein Messwert gehört. Der Stromverbrauch einzelner HTTP-Aufrufe ließe sich nur mit einem Oszilloskop und einer Schaltung am Eingang des Raspberry Pi messen. Außerdem müsste das Mobilfunkmodul in allen Testgebieten den gleichen Modus und Submodus verwenden, um mögliche Unterschiede bei 2G- und 3G-Funknetzen auszuschließen.

Aus Tabelle 12 lässt sich auch entnehmen, dass die maximale Laufzeit der Sensorplattform ortsabhängig ist. Findet keine Regulierung der Hochladefunktion durch die Parameter RSCP, RSSI und E_c/I_o statt, ergeben sich die folgenden Laufzeiten an den drei Orten:

am Fraunhofer FIT $3000mAh : 480mA = 6,25h$, in Siegburg und an der HBRS dagegen $3000mAh : 480mA = 8,57h$.

In der Anwendung der Sensorplattform ist eine Softwarelösung vorgestellt, um den Stromverbrauch des Geräts und damit einhergehend die Dienstgüte (Quality of Service) zu verringern. Die drei vorgestellten Parameter lassen sich bei allen Mobilfunkmodulen für eine Regulierung nutzen (auch bei LTE). Sinnvoller wäre es jedoch ein Modem in die Sensorplattform einzubauen, das standardmäßig in der Sendeleistung und in der Datenrate begrenzt ist. Eine mögliche Hardwarelösung wird im Ausblick in Kapitel 9 vorgestellt. Dieses Modul nutzt den LTE-Standard M1 für Anwendungen des Internet-of-Things (IoT), der eine Datenrate von 1 Mbps im Up- und Download sicherstellt und dessen Sendeleistung auf 20 bzw. 23 dBm limitiert ist [Tir16]. Nachfolgend wird ein weiterer Test vorgestellt, mit denen das Übertragen an einen Webserver getestet wird. Dadurch wird insbesondere überprüft, ob die Implementierung der Warteschlange korrekt funktioniert.

Es werden fünf Aufzeichnung mit einem Herzfrequenzsensor für jeweils 10 Minuten gestartet. Die Grenzwerte sind wie folgt festgelegt: $E_c/I_o = -32dB$, RSCP = $-145dBm$ und RSSI = $-113dBm$ festgelegt. Das heißt, die Daten werden immer hochgeladen, auch bei der schlechtesten möglichen Signalqualität. Ein Hilfsprogramm sucht mit regulären Ausdrücken in der Log-Datei nach den Werten für die Latenz beim Übertragen einzelner Herzfrequenzsamples und die Größe der Warteschlange. Dieses befindet sich ebenfalls auf der CD im Verzeichnis **Resources/LogFileExtractor**. Um die Dauer eines Uploads zu bestimmen, liegt jeweils ein Messpunkt unmittelbar vor und nach der Hochladefunktion. Im Folgenden sind die Angaben zu der Aufzeichnung mit einem Herzfrequenzsensor dargestellt. Zur Übersicht ist die Dauer auf eine Minute begrenzt. Diese Daten sind vom Standort Fraunhofer FIT bei der schlechtesten Signalqualität ermittelt worden.

- Upload-Latenz (ms): 1503, 303, 321, 434, 296, 2807, 2087, 684, 2067, 846, 2267, 190, 185, 167, 208, 166, 166, 185, 266, 187, 165, 166, 188, 245, 3427, 444, 767, 190, 846, 1708, 846, 1548, 207, 1488, 2047, 927, 1941, 1158, 1885, 1331, 2328, 1247, 846, 209, 1326, 2706, 2292, 3225, 1926
- Warteschlange (Anzahl): 1, 1, 1, 1, 1, 1, 2, 3, 3, 4, 4, 5, 5, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 2, 1, 1, 2, 1, 2, 3, 3, 4, 4, 5, 5, 6, 7, 7, 6, 6, 8, 9, 11
- Durchschnittliche Latenz: 1080 ms
- Nicht hochgeladene Samples: 11 (18,33%)

Zu erkennen ist: sobald einzelne Uploads länger als eine Sekunde dauern, erhöht sich die Anzahl der Elemente in der Warteschlange. Die Warteschlange kann auch wieder abgearbeitet werden, wenn die Übertragungen in weniger als einer Sekunde durchgeführt werden. Beim Abschluss einer Aufzeichnung werden alle verbleibende Samples (11) von der Queue in die Datenbank gespeichert.

Tabelle 13 zeigt die Minimal-, Maximal- und Durchschnittswerte der Latenz sowie der prozentuale Anteil an Samples, die nicht hochgeladen worden sind. Dieser Test zeigt, dass unter schlechtesten Bedingungen bis zu 60% der Samples nicht hochgeladen werden (ID 1), weil hohe Latenzspitzen bis fast 27 Sekunden auftreten können. Dies kann die Echtzeitfähigkeit des Systems verschlechtern. Bei durchschnittlicher Signalqualität in Siegburg und an der HBRS werden alle Daten hochgeladen (ID 2 und 3).

Auch dafür lässt sich eine weitere Ausbaustufe der Anwendung definieren, die aus zeitlichen Gründen jedoch nicht weiter verfolgt werden kann. Es könnte ein Timeout für die HTTP-Verbindungen zum WebHrs-Server auf bspw. den Wert 10 Sekunden gesetzt werden, um hohe Latenzspitzen zu verhindern. Alternativ könnten mehrere Samples parallel

in separaten Threads hochgeladen werden, wodurch die Datenrate nicht mehr fest, sondern variabel wäre. Beide Maßnahmen könnten dazu führen, dass insgesamt mehr Samples hochgeladen werden, aber auch wiederum Einfluss auf den Stromverbrauch des Surfsticks haben.

ID	Ort	Latenz [ms]			Samples [%]
		Min.	Max.	Ø	
1	FIT	165	26748	ca. 600	0-60
2	HBRS	71	3643	ca. 100	0
3	Siegburg	63	2587	ca. 100	0

Tabelle 13: Latenzen beim Übertragen von Herzfrequenzdaten an WebHrs

In diesem Abschnitt sind drei Parameter zur Bestimmung der Signalqualität einer Mobilfunkverbindung erläutert worden, die bei allen 3G- und 4G-Techniken abrufbar sind. Die Softwarelösung zur Begrenzung des Stromverbrauchs des Modems gehen mit einer Verringerung der Dienstgüte (Quality of Service) einher und haben insgesamt nur geringen Einfluss. Es ist daher naheliegend, zuerst Verbesserungen an der Hardware vorzunehmen und die Softwarelösungen zur Feineinstellung zu verwenden. In Kapitel 8 wird dafür unter anderem ein neuer energieeffizienter LTE-Chip vorgestellt, der zukünftig als Modul für den Raspberry Pi 3 verkauft werden soll. Die Tests der Sensorplattform sind damit abgeschlossen. Abschließend wird noch die Reichweite der Bluetooth-Verbindungen der Sensoren getestet.

7.3.4. Reichweite der Bluetooth-Geräte

In diesem Abschnitt wird die Reichweite der Bluetooth Low Energy-Verbindung zwischen Sensor und Sensorplattform ermittelt. Die Reichweite ist hauptsächlich durch drei Parameter bestimmt: die Leistung des Funksenders auf den Geräten, die Empfindlichkeit der Empfänger-Hardware (insbesondere der Antenne) und mögliche Hindernisse in der Umgebung zwischen zwei Geräten. Die Sendeleistung ist durch die Bluetooth Low Energy Spezifikation auf höchstens 10 mW begrenzt, allerdings gibt es bei den in dieser Arbeit genutzten Sensoren keine Herstellerangaben dazu. Im Test gibt es keine Hindernisse zwischen den Geräten und auch keine Störsignale durch andere Bluetooth-Geräte. Eine Antenne für Bluetooth bzw. WLAN ist auf dem Raspberry Pi 3 nicht vorgesehen. Der Test für die Reichweite der Bluetooth-Sensoren lässt sich mit einer BlueZ-Installation ohne die Software der Sensorplattform durchführen.

Zunächst wird das Programm Gatttool gestartet und eine Verbindung zu einem Sensor aufgebaut. Anschließend werden die Notifications aktiviert (mit einer Abtastrate von einer Messung pro Sekunde). Die Sensorplattform bleibt stationär, der Sensor wird danach kontinuierlich von der Sensorplattform weg bewegt. Ab einer bestimmten Distanz empfängt die Sensorplattform keine Nachrichten mehr. Daraus lässt sich abschätzen, dass die Reichweite der Bluetooth Low Energy-Verbindung mit den vorhandenen Sensoren ungefähr 9 Meter beträgt.

Theoretisch wäre es bei bestimmten Aktivitäten möglich, eine Sensorplattform mit mehreren Patienten zu betreiben (z. B. eine Gruppe von Joggern). Für eine kontinuierliche Datenaufzeichnung müssten alle Personen jedoch sehr nah beisammen sein, weshalb es sinnvoller ist, eine Sensorplattform pro Person zu benutzen.

8. Diskussion und Fazit

In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst und diskutiert. Die Sensorplattform stellt ein Gesundheitsüberwachungssystem dar, das zunächst mit fünf Sensoren kommunizieren kann. Die Anwendung lässt sich für neue Sensoren erweitern und diese können in unterschiedlichen Kombinationen genutzt werden. Grundlage dafür sind offene Schnittstellen wie bspw. Bluetooth Low Energy (BLE). Die Sensorplattform ist aus kommerziellen Hardwarekomponenten zusammengesetzt.

Die Anforderungsanalyse definiert unter anderem Use Cases für die Anwendung der Sensorplattform, deren Umsetzung ein wesentlicher Bestandteil dieser Arbeit ist. Alle Funktionen und Anforderungen sind in der Implementierung berücksichtigt, welche sich durch die Test Cases im Anhang überprüfen lassen. Für die Sensorplattform gibt es keine Testumgebung, was heißt, dass Referenzen zu den Hardwaresensoren, zum Betriebssystem und zum Bluetooth-Stack in der Software immer vorhanden sind. Die Software lässt sich bspw. nicht auf einem Windows-PC ausführen und für die Aufzeichnungen müssen reale Sensoren vorhanden sein. Eine separate Testumgebung würde die zukünftige Entwicklung am System erleichtern.

Die Anwendung der Sensorplattform kommuniziert mit dem offiziellen Bluetooth-Stack BlueZ 5 für Linux-Betriebssysteme. Die GATT-Profile von Bluetooth Low Energy (BLE) bieten eine gute Grundlage auf der Anwendungsebene des Softwarestacks, um Sensordaten zwischen mehreren Geräten auszutauschen. Für die Kommunikation zu den GATT-Servern der Sensoren verwendet die Sensorplattform das Programm Gatttool, das Teil der Installation von BlueZ 5 ist. Die notwendige Dokumentation für Services und Characteristics einiger Vitalparameter ist von der Bluetooth SIG veröffentlicht, sodass diese Daten vom Sensor ausgelesen werden können. Es sind keine Kenntnisse über die zugrunde liegende Hardware erforderlich und auch die einzelnen Schichten und Protokolle der Bluetooth-Verbindung sind von der Anwendung der Sensorplattform abgeschottet. Schwieriger ist es bei Sensoren, deren Services und Characteristics geräte- oder herstellerspezifisch sind. Ohne die Dokumentation des Herstellers ist es fast unmöglich, die Daten korrekt zu interpretieren. Der Sensor CC2650 ist ebenfalls in die Sensorplattform integriert, weil die Dokumentation von Texas Instruments öffentlich zugänglich ist. Manche Hersteller werben explizit damit, dass ihre Geräte auch in andere Systeme integriert werden können, z. B. das Pulsoximeter Nonin 3230 [Nona]. Ob sich die Gerätehersteller zukünftig an die GATT-Profile der Bluetooth-Spezifikation halten oder ihre eigenen Profile definieren, lässt sich nicht vorhersagen. Diese Bluetooth-Schnittstelle ist für die Sensorplattform von grundlegender Bedeutung. Der GATT-Server des BLE113 Entwicklungsboards kann beliebig konfiguriert werden, sodass sich das Board gut dafür eignet, neue Sensoren in die Sensorplattform zu integrieren.

Die Anwendung steuert die Sensoren generisch durch die Services und Characteristics des GATT-Servers. Dies ermöglicht, dass neue Sensoren einfach in die Sensorplattform integriert werden können. Grundsätzlich gelingt dies in der Software gut, wie bspw. die Vererbungshierarchie für Herzfrequenzsensoren in der Anwendung verdeutlicht. Allerdings verwendet das Gatttool für einige Funktionen (z. B. für Notifications) nur Handle-Adressen und keine UUIDs aus der Bluetooth-Spezifikation. Die Handle-Adressen sind wiederum gerätespezifisch und müssen in Form einer GattLibrary in der Sensorplattform hinterlegt sein. Eine aufwendigere Implementierung wäre aber auch mit dem Gatttool in der Lage, den GATT-Server des Sensors selbstständig zu durchsuchen und entsprechende Verknüpfungen von Handles und UUIDs herzustellen.

Die Anwendung basiert auf einer Linux-Distribution und ist in der Programmiersprache Java implementiert, was beides Vorgaben des Fraunhofer FIT sind. Der größte Vorteil von Java, die Portierbarkeit zwischen unterschiedlichen Betriebssystemen (Linux, Win-

dows oder Mac), ist demnach ungenutzt. Für die Interaktion mit dem Bluetooth-Stack BlueZ 5 ist Java im Zeitrahmen der Masterarbeit weniger gut geeignet. Der von den Entwicklern von BlueZ bevorzugte Ansatz für die Bluetooth-Kommunikation ist das Framework für Interprozesskommunikation DBus [Jan16]. Diese Schnittstelle war zu Beginn der Arbeit noch in einem experimentellen Status, wodurch es unklar war, ob sich diese Grundlage für die Sensorplattform eignet [Hed16]. Die einzige Bibliothek für DBus in Java wird seit 2009 nicht weiterentwickelt [freb]. Ein anderes Projekt, TinyB, stellt für die BlueZ-DBus-Objekte eine Schnittstelle für Java bereit [Gitb], das sich aber auch noch in einer frühen Entwicklungsphase befindet. Die DBus-Bindings (Java Native Interface) für die Sensorplattform zu implementieren, ist im Rahmen dieser Arbeit zu umfangreich. Daher verwendet die Sensorplattform eine individuelle Softwarelösung mit Interprozesskommunikation zu den Gatttool-Instanzen. Ein Framework für den standardisierten und einheitlichen Zugriff der Software auf Hardwaresensoren ist jedoch erstrebenswert, weil dadurch die Strukturen der Softwarekomponenten und deren Schnittstellen klar ersichtlich sind. Dies gilt nicht nur für die Sensorplattform, sondern zukünftige für viele weitere Anwendungen des Internet-of-Things (IoT) und des Web-of-Things (WoT).

Mit der Generic Sensor API von Web-Bluetooth gibt es eine solche Interaktionsmöglichkeit für Bluetooth Low Energy Sensoren, allerdings befindet sich dieses Framework ebenfalls noch in der Entwicklungsphase [Worb, Wora]. Das Framework kann beispielsweise durch einen NodeJS-Server für Webanwendungen bereitgestellt werden. Es ist absehbar, dass diese Javascript-API auf Grund ihrer Qualität und des weitreichenden Einflusses des W3C in Zukunft Verwendung findet. Diese Stufe der Hardwareabstraktion lässt sich zum Zeitpunkt dieser Arbeit daher noch als *Bleeding-Edge*-Technologie bezeichnen. Falls z. B. in ein bis zwei Jahren erneut eine Anwendung für die Sensorplattform implementiert wird, kann es sein, dass sich die Softwarearchitektur deutlich von der aktuellen Lösung unterscheidet, sofern die Vorgaben für Linux und Java entfallen. Gegebenenfalls könnte für die Interaktion mit der Hardware auch eine Middleware genutzt werden. Die Schnittstelle zu den Sensoren ist dann nicht mehr Teil der Anwendung der Sensorplattform. Die Middleware könnte für die Anwendungsebene Schnittstellen bieten, um unterschiedliche Sensoren einheitlich anzusteuern. Weitere Funkübertragungstechniken sind Ant bzw. Ant+ und Zigbee, die ebenfalls sehr energieeffizient sind.

Die Erweiterbarkeit der Sensorplattform ist bei der Auswahl der Hardwarekomponenten ein wesentliches Kriterium. Über die GPIO-Pins oder über die verbleibenden USB-2.0-Ports des Raspberry Pi ist es möglich, Module für zusätzliche Funkschnittstellen der Sensorplattform zu ergänzen. Lediglich die I²C-Pins auf dem Board sind durch die Hardwareuhr belegt.

Das Konzept der Sensorplattform sieht ausschließlich die Erfassung von physischen Vitalparametern vor. Das Monarca-Projekt zeigt jedoch, dass es Krankheiten gibt, für die die Erfassung von psychischen Parametern ebenfalls von Relevanz sind. Zum Beispiel kann die Häufigkeit der Nutzung von Telefon und SMS ein Indikator für eine manische Phase sein. Bei Monarca ist die Funktionalität der Sensorplattform auf dem Smartphone des Patienten in Form einer App integriert. Für den Gebrauch eines Smartphones mit der Sensorplattform könnte eine App installiert werden, die einen GATT-Server auf dem Smartphone bereitstellt. Die Sensorplattform kann sich anschließend auf gleiche Weise mit dem Smartphone wie mit den Sensoren verbinden. Im GATT-Profil des Smartphones können die Daten über die Nutzung von Telefon und SMS angegeben werden, die die Sensorplattform sammelt. Allerdings ist die Verwendung eines Smartphones aus Sicht des Datenschutzes und der Informationssicherheit bedenklich, weil es schwierig ist festzustellen, ob bestimmte Software das Gesamtsystem kompromittieren könnte.

Sämtliche Funk- und Netzwerkschnittstellen zur Anwendung der Sensorplattform nutzen

gängige Sicherheitsfunktionen, um die Verbindungen abzusichern. Die Webanwendung der Sensorplattform verwendet TLS/SSL zur Authentifizierung (MITM-Schutz) gegenüber dem Client und zur Verschlüsselung der Verbindung. Außerdem ist die Anmeldung (Autorsierung) des Benutzers durch HTTP-Basic-Authentication an der Webanwendung umgesetzt. Das externe Webinterface TeLiPro nutzt ebenfalls TLS/SSL und ein Token-basiertes System zur Autorisierung eines Benutzers. Dies Verfahren sind gängige Praxis bei Webservern und bieten ein hohes Maß an Sicherheit. Für die Verwendung von Verschlüsselung und Authentifizierung bei der Bluetooth-Verbindung sind auf beiden Geräten gegebenenfalls weitere Eingabe-/ Ausgabeperipherien notwendig. Außerdem muss der Bluetooth-Stack auf dem Sensor den Bluetooth-Security-Manager unterstützen, ansonsten sind keine Sicherheitsfunktionen möglich. Die Sensorplattform unterstützt sichere Bluetooth-Verbindungen, wofür das Pairing zuvor im Terminal durchgeführt werden muss. Anhand der sicheren Pulsoximeter-Firmware für das BLE113 Entwicklungsboard ist ein Sensor realisiert, der Verschlüsselung und Authentifizierung nutzt. Die Firmware ermöglicht außerdem die Verwaltung von Bonds auf dem Gerät. Das BLE113 Board verwendet jedoch nur Bluetooth 4.0. Bluetooth-Chips der Version 4.2 oder höher bieten durch die Verwendung besserer Algorithmen die höchste Sicherheit für die Pairing-Verfahren.

Ein wesentlicher Kritikpunkt ist die Energieeffizienz der Sensorplattform. Der Stromverbrauch des Raspberry Pis 3 liegt selbst im Bereitschaftsmodus bei ca. 250 mA und unter Last steigt dieser Wert noch weiter an. Die Anwendung lastet jedoch den Prozessor und auch den Hauptspeicher kaum aus, sodass die Leistung der Hardware nicht ansatzweise ausgenutzt wird. Benchmarks haben gezeigt, dass die Rechenleistung und der Stromverbrauch bei neueren Modellen des Raspberry Pi stetig gestiegen ist [Pre16]. Weil die alten Versionen keinen WLAN- und Bluetooth-Chip integriert haben, eignen sich diese aber nicht automatisch besser als Hardware für eine Sensorplattform. Es wäre daher sinnvoll, die Sensorplattform auf ein anderes Board zu portieren und erneut zu testen. Durch die Verwendung von Linux als Betriebssystem, BlueZ und Java ist die Portierbarkeit auf andere Linux-Systeme mit wenig Aufwand verbunden. Die Hardwareschnittstellen in der Implementierung sind durch die Verwendung von Dependency Injection modular und austauschbar.

Die Funkübertragung mit Bluetooth Low Energy (BLE) geschieht tatsächlich sehr energieeffizient. Auch bei vielen Sensoren und hohen Abtastraten bleibt der Stromverbrauch der Sensorplattform im Vergleich zum Bereitschaftsmodus unverändert. Auch die Anwendung, die die Daten empfängt und abspeichert erhöht dann nur geringfügig die Prozessorauslastung. Allerdings haben die Tests gezeigt, dass die Datenbank nicht alle 500 ms auf die micro-SD-Karte persistiert werden darf, weil sonst die Prozessorauslastung und der Stromverbrauch signifikant ansteigen. Stattdessen wird der Intervall auf 5 Sekunden festgelegt, um dem entgegenzuwirken. Die in dieser Arbeit verwendeten Sensoren besitzen verhältnismäßig hohe Abtastraten. Mit anderen Sensoren, z. B. mit einem Blutdruckmessgerät werden nur einige Male am Tag Messungen durchgeführt.

Problematisch ist der Stromverbrauch des Mobilfunkmoduls, wenn Samples zu einem externen Webserver hochgeladen werden. Bei schlechter Signalqualität der Mobilfunkverbindung kann das sekündliche Übertragen von Herzfrequenzdaten zu einem Stromverbrauch von 480 mA führen. Dadurch lässt sich mit dem Sensorplattform nur noch eine Laufzeit von ungefähr 6 Stunden erzielen. Die Anwendung wertet daher bestimmte Parameter zur Signalqualität aus und stellt gegebenenfalls die Hochladefunktion ein, wenn diese zu viel Sendeleistung erfordern würde. Auch kritische Messdaten werden damit nicht zum Webserver übertragen, was eine Reaktion in Echtzeit nicht mehr möglich macht. Eine Ausbaustufe ist dafür vorstellbar, bei der die Sensorplattform die Messdaten auf Überschreitung von Grenzwerten überprüfen könnte und diese gegebenenfalls trotzdem hochlädt, auch wenn

dies nicht energieeffizient durchgeführt werden kann.

Im Allgemeinen ist ein Surfstick, der die 500 mA Obergrenze der Spezifikation für USB-2.0-Ports ausnutzen kann, für ein mobiles Gesundheitsüberwachungssystem ungeeignet. Es wäre besser ein Modul zu nutzen, das hardwareseitig in der Sendeleistung begrenzt ist. Die Entwicklung von LTE sieht unter anderem einige Standards speziell für Anwendungen des Internet-of-Things (IoT) vor, die nur eine geringe Datenrate ermöglichen, dafür aber besonders energieeffizient sind [Tir16]. Für den Raspberry Pi soll es zukünftig ein Funkmodul geben, das die LTE-Kategorie M1 verwendet [Mö16]. Die Sendeleistung ist damit auf 20 bzw. 23 dBm begrenzt und es sind nur Datenraten von 1 Mbps im Uplink und Downlink möglich [Tir16], welche jedoch für die Sensorplattform ausreichend sind.

Zusammenfassend lässt sich festhalten, dass die Software alle notwendigen Funktionen und Anforderungen bis auf die geforderte Akkulaufzeit erfüllt, die in der Anforderungsanalyse in Kapitel 4 definiert sind. Die ausgewählte Hardware hat durch das Vorhandensein einer aktuellen Linux-Distribution und durch die Treiberunterstützung die Softwareentwicklung gut unterstützt. Allerdings ist der Energieverbrauch der Komponenten so hoch, dass die erforderliche Laufzeit von ungefähr 16 Stunden nicht erreicht werden kann. Gegebenenfalls wäre für die Sensorplattform die Entwicklung einer individuellen Leiterplatte sinnvoll, um diese kompakter und stromsparender zu konzipieren, wodurch das System dann aber signifikant teurer wird. Da die Implementierung darauf ausgelegt ist, auch auf anderen Einplatinenrechner mit Linux zu funktionieren, kann die Software einfach portiert werden.

9. Ausblick

Das Internet-of-Things (IoT) wird zukünftig weiter an Bedeutung gewinnen, indem immer neue Anwendungsbereiche erschlossen werden. Die Medizintechnik stellt ein Beispiel für einen solchen Anwendungsbereich dar. Mobile Gesundheitsüberwachungssysteme stellen eine Möglichkeit dar, um zukünftig den steigenden Kosten im Gesundheitswesen entgegenzuwirken und trotzdem eine hohe Qualität der gesundheitlichen Versorgung zu gewährleisten. Auch der Umgang mit Krankheiten kann sich gegebenenfalls durch diese Systeme verändern. Der Schwerpunkt verschiebt sich von der reaktiven Behandlung von Krankheitsbildern zu einer proaktiven Prävention und Früherkennung von Krankheiten [MOJ06]. Es ist daher wahrscheinlich, dass derartige Systeme einen festen Platz im Gesundheitswesen einnehmen werden.

Aus technischer Sicht ist die Umsetzung eines mobilen Gesundheitsüberwachungssystems heute bereits möglich. Sowohl die Hardware als auch die Funkübertragungstechniken entwickeln sich zudem kontinuierlich weiter, sodass immer bessere Systeme konzipiert werden können. Schon während der Bearbeitungszeit dieser Arbeit hat es z. B. zwei Neuerungen gegeben, die weitere Anwendungen für die Sensorplattform ermöglichen. Bluetooth 5.0 wurde im Dezember 2016 veröffentlicht und definiert unter anderem neue GATT-Characteristics. Ebenfalls im Dezember wurde das europäische Satellitennavigationssystem Galileo freigegeben. Verwendet die Sensorplattform ein entsprechendes Modul, kann die Position des Patienten mit dem Gerät noch genauer bestimmt werden, als es bisher bei der zivilen Nutzung von GPS der Fall ist. Die Computer-gestützten Verfahren für die Therapie und Diagnose von Krankheiten gibt es heutzutage noch nicht. Diese müssen jedoch eine besonders hohe Qualität haben, um den Arzt, der auch eine Bezugs- bzw. Vertrauensperson ist, zu ersetzen. Ähnlich ist es bei dem autonomen Fahren oder Fliegen.

Neben den technischen Aspekten gibt es auch noch weitere Anforderungen, die für den Einsatz mobiler Gesundheitsüberwachungssysteme im Alltag berücksichtigt werden müssen. Dazu zählt unter anderem die Annahme des technischen Systems durch den Patienten und auch des medizinischen Personals. Es hat sich gezeigt, dass ältere Menschen ein solches System verwenden wollen, sofern es ihnen Unabhängigkeit im Alltag gewährleistet [Var07]. Aber auch Schulungen für den Umgang mit den neuen Geräten sind erforderlich, damit diese korrekt eingesetzt werden.

Das Konzept der Sensorplattform sieht ein mobiles Gesundheitsüberwachungssystem vor, das mit unterschiedlichen Sensoren kombiniert werden kann, die verschiedene Funkübertragungstechniken nutzen können. Für die einheitliche Ansteuerung einer Vielzahl von Geräten fehlt es zur Zeit noch an Bibliotheken und Frameworks, um eine schnelle und effektive Produktentwicklung zu ermöglichen. Softwareentwickler schaffen individuelle Lösungen, die gegebenenfalls nicht kompatibel zueinander sind. Bestimmte Kategorien von Sensoren gilt es generisch anzusteuern, so wie es bspw. durch das GATT-Profil bei Bluetooth Low Energy (BLE) möglich ist. Für die Interoperabilität der Systeme müssen offene Schnittstellen und Standards auf verschiedenen Ebenen des Software-Stacks geschaffen werden, wodurch letztlich auch die Senkung von Kosten bei der Softwareentwicklung möglich wäre.

Diese Arbeit liefert eine Vielzahl von Ergebnissen zur Hardware, Software und Informatiionssicherheit, die für die Konzeption eines mobilen Gesundheitsüberwachungssystems für zukünftige Projekte am Fraunhofer FIT berücksichtigt werden können. Außerdem lassen sich einige der Ergebnisse, z. B. die Absicherung von Bluetooth-Verbindungen auch für andere Anwendungen des Internet-of-Things (IoT) wiederverwerten.

Literaturverzeichnis

- [adi] adidas Int. Trading B.V. micoach. Webseite. Online erhältlich unter <http://www.adidas.de/micoach>; abgerufen am 04. Januar 2017.
- [And] Android. Bluetooth. Webseite. Online erhältlich unter <https://source.android.com/devices/bluetooth.html>; abgerufen am 16. Januar 2017.
- [Apa] Apache Maven Project. Welcome to Apache Maven. Website. Online erhältlich unter <https://maven.apache.org/>; abgerufen am 15. Januar 2017.
- [Blua] BlueRadios, Inc. Bluetooth Low Energy - Technical Facts. Webseite. Online erhältlich unter http://www.blueradios.com/Bluetooth_Low_Energy_Factsheet.pdf; abgerufen am 20. Dezember 2016.
- [Blub] Bluetooth SIG Inc. Bluetooth Core Specification. Webseite. Online erhältlich unter <https://www.bluetooth.com/specifications/bluetooth-core-specification>; abgerufen am 28. Oktober 2016.
- [Bluc] Bluetooth SIG Inc. Generic attributes (gatt) and the generic attribute profile. Webseite. Online erhältlich unter <https://www.bluetooth.com/specifications/generic-attributes-overview>; abgerufen am 28. Oktober 2016.
- [Blud] Bluetooth SIG Inc. Viewer Heart Rate Service. Webseite. Online erhältlich unter https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml; abgerufen am 16. Januar 2017.
- [Blue] Bluetooth SIG Inc. Viewer Pulse Oximeter Service. Webseite. Online erhältlich unter https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.pulse_oximeter.xml; abgerufen am 16. Januar 2017.
- [Blu15] Bluetooth SIG Inc. The Wheels on the Bike are Bluetooth Smart. Bluetooth Smart. Webseite, September 2015. Online erhältlich unter <http://blog.bluetooth.com/the-wheels-on-the-bike-are-bluetooth-smart-bluetooth-smart%2Dbluetooth-smart/>; abgerufen am 24. November 2016.
- [Blu16a] Bluetooth SIG Inc. Bluetooth 5 Now Available. Webseite, Dezember 2016. Online erhältlich unter <https://www.bluetooth.com/news/pressreleases/2016/12/07/bluetooth-5-now-available>; abgerufen am 10. Dezember 2016.
- [Blu16b] Bluetooth SIG Inc. Bluetooth 5 quadruples range, doubles speed, increases data broadcasting capacity by 800. Webseite, Juni 2016. Online erhältlich unter <https://www.bluetooth.com/news/pressreleases/2016/06/16/%2Dbluetooth5%2Dquadruples%2Drangedoubles%2Dspeedincreases%2Ddata%2Dbroadcasting%2Dcapacity%2Dby%2D800>; abgerufen am 28. Oktober 2016.
- [Blu16c] Bluetooth SIG Inc. Bluetooth Low Energy Security. Webseite, 2016. Online erhältlich unter <https://www.bluetooth.com/~/media/files/specification/bluetooth-low-energy-security.ashx?la=en&usg=AFQjCNFAPWPrq31H3x-vEiPBSy-tYNx79Q&cad=rja>; abgerufen am 02. November 2016.

- [Bun06] Bundeszentrale für politische Bildung. Zur demografischen Lage der Nation. Webseite, Mai 2006. Online erhältlich unter <http://www.bpb.de/politik/innenpolitik/demografischer-wandel/70883/demografische-lage-der-nation>; abgerufen am 22. Dezember 2016.
- [Bun11] Bundesamt für Sicherheit in der Informationstechnik (BSI). G 5.159 Erstellung von Bewegungsprofilen unter Bluetooth. Webseite, 2011. Online erhältlich unter https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/g/g05/g05159.html;jsessionid=1634B65ACC0B1D934853DB638DDF1B1C.2_cid368?nn=6604950; abgerufen am 16. November 2016.
- [Bun14] Bundesamt für Sicherheit in der Informationstechnik. BSI stuft Heartbleed Bug als kritisch ein. Webseite, April 2014. Online erhältlich unter https://www.bsi.bund.de/DE/Presse/Pressemitteilungen/Presse2014/Heartbleed_11042014.html; abgerufen am 24. Januar 2017.
- [Bun16] Bundesamt für Sicherheit in der Informationstechnik (BSI). Cyber-Angriffe durch IoT-Botnetze: BSI fordert Hersteller zu mehr Sicherheitsmaßnahmen auf. Webseite, Oktober 2016. Online erhältlich unter https://www.bsi.bund.de/DE/Presse/Pressemitteilungen/Presse2016/Cyber-Angriffe_durch_IoT-Botnetze_25102016.html; abgerufen am 23. November 2016.
- [Deb] Debian. Debian Wiki sudo. Webseite. Online erhältlich unter <https://wiki.debian.org/sudo>; abgerufen am 01. Januar 2017.
- [Deu] Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Bis auf den Meter genau: Navigation mit Galileo. Webseite. Online erhältlich unter http://www.dlr.de/next/desktopdefault.aspx/tabcid-6804/11164_read-25462/; abgerufen am 06. Januar 2017.
- [Dig] Digi-Key Electronics. BLE113 Development Kit. Webseite. Online erhältlich unter <https://www.polar.com/de/produkte/accessoires/H7>; abgerufen am 28. Dezember 2016.
- [Dyna] Dynastream Innovations Inc. About us. Webseite. Online erhältlich unter <https://www.thisisant.com/company>; abgerufen am 28. Oktober 2016.
- [Dynb] Dynastream Innovations Inc. ANT / ANT+ Defined. Webseite. Online erhältlich unter <https://www.thisisant.com/developer/ant-plus/ant-antplus-defined>; abgerufen am 28. Oktober 2016.
- [Dync] Dynastream Innovations Inc. ANT+ Device Profiles. Webseite. Online erhältlich unter <https://www.thisisant.com/developer/ant-plus/device-profiles>; abgerufen am 28. Oktober 2016.
- [Ele07] Electronic Communications Committee (ECC). UMTS Coverage Measurements. Webseite, Mai 2007. Online erhältlich unter <http://www.erodocdb.dk/docs/doc98/official/pdf/ECCRep103.pdf>; abgerufen am 27. Dezember 2016.
- [Fraa] Fraunhofer Gesellschaft. MAS. Webseite. Online erhältlich unter <https://www.fit.fraunhofer.de/de/fb/life/projects/mas.html>; abgerufen am 13. Dezember 2016.

- [Frab] Fraunhofer Gesellschaft. POLYCARE. Webseite. Online erhältlich unter <https://www.fit.fraunhofer.de/de/fb/ucc/projects/polycare.html>; abgerufen am 13. Dezember 2016.
- [frea] freedesktop.org. D-Bus users. Webseite. Online erhältlich unter <https://www.freedesktop.org/wiki/Software/DbusProjects/>; abgerufen am 16. Januar 2017.
- [freb] freedesktop.org. DBusBindings. Webseite. Online erhältlich unter <https://www.freedesktop.org/wiki/Software/DBusBindings/>; abgerufen am 29. Dezember 2016.
- [Gita] Github. Google Guice. Webseite. Online erhältlich unter <https://github.com/google/guice>; abgerufen am 25. Januar 2017.
- [Gitb] Github. Intel IoT Developer Kit TinyB. Webseite. Online erhältlich unter <https://github.com/intel-iot-devkit/tinyb>; abgerufen am 15. Januar 2017.
- [Gre16] F. Greis. Dyn bestätigt Angriff von zig Millionen IP-Adressen. Webseite, Oktober 2016. Online erhältlich unter <http://www.golem.de/news/mirai-botnetz-dyndns-bestaeigt-angriff-von-zig-millionen-ip%2Dadressen-1610-123981.html>; abgerufen am 13. Januar 2017.
- [Hed12] J. Hedberg. Releases of Bluez 5.0. Webseite, Dezember 2012. Online erhältlich unter <http://www.bluez.org/release-of-bluez-5-0/>; abgerufen am 02. November 2016.
- [Hed16] J. Hedberg. Release of BlueZ 5.42. Webseite, September 2016. Online erhältlich unter <http://www.bluez.org/release-of-bluez-5-42/>; abgerufen am 15. Januar 2017.
- [Hol16] M. Holland. Medizintechnik: US-Hersteller warnt vor Hackerangriffen auf Insulinpumpen. Webseite, Oktober 2016. Online erhältlich unter <https://www.heise.de/newsticker/meldung/Medizintechnik-US-Hersteller-warnt-vor-Hackerangriffen-auf%2DInsulinpumpen-3341645.html>; abgerufen am 24. Januar 2017.
- [HUA13] HUAWEI Technologies Co., Ltd. AT Command Interface Specification. Webseite, Dezember 2013. Online erhältlich unter download-c.huawei.com/download/downloadCenter?downloadId=17948; abgerufen am 18. November 2016.
- [Ibr11] A. Ibrahim. 3G Modem Internet Dialer. Webseite, Juli 2011. Online erhältlich unter http://www.codeproject.com/KB/IP/3G_Modem_Internet_Dialer.aspx?display=Print; abgerufen am 18. November 2016.
- [Jan16] S. Janc. Bluetooth on modern Linux. Webseite, April 2016. Online erhältlich unter http://events.linuxfoundation.org/sites/events/files/slides/Bluetooth%20on%20Modern%20Linux_0.pdf; abgerufen am 28. Oktober 2016.
- [Jaw] Jawbone. Jawbone Fitness Tracker. Webseite. Online erhältlich unter <https://jawbone.com/up/trackers>; abgerufen am 21. Dezember 2016.
- [JBo] JBoss (Red Hat). Hibernate. Website. Online erhältlich unter <http://hibernate.org/>; abgerufen am 24. Januar 2017.

- [JUn] JUnit. JUnit Frequently Asked Questions. Webseite. Online erhältlich unter http://junit.org/junit4/faq.html#overview_1; abgerufen am 06. Dezember 2016.
- [KFR14] Z. Khalid, N. Fisal, and M. Rozaini. A Survey of Middleware for Sensor and Network Virtualization. *Sensors*, 14, Dezember 2014. (12).
- [Kon] Koninklijke Philips N.V. Blutdruckerfassung mit App-Anbindung. Webseite. Online erhältlich unter <http://www.philips.de/c-m-hs/health-programme/handgelenk-blutdruckmessgeraet>; abgerufen am 20. Dezember 2016.
- [Kre16] S. Krempl. 33C3: Hacker rufen nach Mindesthaltbarkeitsdatum für vernetzte Geräte. Webseite, Dezember 2016. Online erhältlich unter <https://www.heise.de/newsticker/meldung/33C3-Hacker-rufen-nach-Mindesthaltbarkeitsdatum-fuer-vernetzte%2DGeraete-3583224.html>; abgerufen am 02. Januar 2017.
- [Lü01] C.-F. Lüders. *Mobilfunksysteme*. Kamprath-Reihe. Vogel Buchverlag, Würzburg, 2001. Seite 247-248 Abbildung 6.24.
- [Mö16] A. Möcker. Raspberry Pi: Bezahlbares LTE-Funkmodul im Anmarsch. Webseite, Dezember 2016. Online erhältlich unter <https://www.heise.de/newsticker/meldung/Raspberry-Pi-Bezahlbares-LTE-Funkmodul-im-Anmarsch-3530126.html>; abgerufen am 02. Dezember 2016.
- [M2M] M2MSupport.net. M2M Support AT Command SYSINFO. Webseite. Online erhältlich unter <http://m2msupport.net/m2msupport/atsysinfo-get-the-system-mode/>; abgerufen am 14. Dezember 2016.
- [Max15] Maxim Integrated Products, Inc. DS3231 Datenblatt. Webseite, 2015. Online erhältlich unter <https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>; abgerufen am 18. November 2016.
- [Meg] Mega Electronics Ltd. Faros technical details. Webseite. Online erhältlich unter <http://www.megaemg.com/products/faros/?GTTabs=1>; abgerufen am 20. Dezember 2016.
- [MM12] K. Malhi and S. C. Mukhopadhyay. A Zigbee-Based Wearable Physiological Parameters Monitoring System. *IEEE Sensors Journal*, 12, März 2012. Issue 3.
- [MOJ06] A. Milenkovic, C. Otto, and E. Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications*, 29, August 2006. Issue 13-14.
- [Nat96] National Institute of Standards and Technology (NIST). Algorithm Validation Lists. Webseite, Januar 1996. Online erhältlich unter <http://csrc.nist.gov/groups/STM/cavp/validation.html>; abgerufen am 15. Januar 2017.
- [Nat01] National Institute of Standards and Technology (NIST). Security Requirements for Cryptographic Moduls (FIPS 140-2). Webseite, Mai 2001. Online erhältlich unter <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf>; abgerufen am 15. Januar 2017.

- [Nat12] National Institute of Standards and Technology (NIST). Guide to Bluetooth Security. Webseite, Juni 2012. Online erhältlich unter http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=911133; abgerufen am 16. November 2016.
- [Nona] Nonin Medical Inc. Introducing Nonin 3230 Bluetooth Smart Pulse Oximeter. Webseite. Online erhältlich unter http://www.nonin.com/OEMSolutions/Nonin_3230_Bluetooth_SMART; abgerufen am 20. Dezember 2016.
- [Nonb] Nonin Medical Inc. Nonin Media Kits and Contact. Webseite. Online erhältlich unter <http://www.nonin.com/presskits>; abgerufen am 20. Dezember 2016.
- [PG10] A. Pantelopoulos and N. G.Bourbakis. A Survey on Wearable Sensor-Based Systems for Health Monitoring and Prognosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 2010.
- [PMGM11] A. Puiatti, S. Mudda, S. Giordano, and O. Mayora. Smartphone-Centred Wearable Sensors Network for Monitoring Patients with Bipolar Disorder. *Conf Proc IEEE Eng Med Biol Soc.*, 2011.
- [Pola] Polar Electro. Herzfrequenz-Messgeräte und GPS-Sportuhren. Webseite. Online erhältlich unter <https://www.polar.com/de/produkte>; abgerufen am 04. Januar 2017.
- [Polb] Polar Electro 2017. H7 Herzfrequenz-Sensor. Webseite. Online erhältlich unter <https://www.polar.com/de/produkte/accessoires/H7>; abgerufen am 08. Januar 2017.
- [Polc] Pollin Electronic GmbH. Cubieboard 3 (Cubietruck) Kit. Webseite. Online erhältlich unter http://www.pollin.de/shop/dt/MjY3NzkyOTk-/Bauelemente_Bauteile/Entwicklerboards/Cubie_Board/Cubieboard_3_Cubietruck_Kit_A20_2_GB_8_GB_WLAN_BT_SATA.html; abgerufen am 29. Dezember 2016.
- [Pre16] Premier Farnell Ltd. A Comprehensive Raspberry Pi 3 Benchmark. Webseite, Februar 2016. Online erhältlich unter <https://www.element14.com/community/community/raspberry-pi/blog/2016/02/29/the-most-comprehensive-raspberry-pi-comparison-benchmark-ever>; abgerufen am 16. Januar 2017.
- [Qua06] Qualcomm ESG Engineering Services Group. WCDMA Network Planning and Optimization. Webseite, Mai 2006. Online erhältlich unter <https://www.qualcomm.com/media/documents/files/wcdma-network-planning-and-optimization.pdf>; abgerufen am 08. Januar 2017.
- [Rao16] L. Rao. Jawbone To Release A New Health-Focused Wearable. Webseite, Mai 2016. Online erhältlich unter <http://fortune.com/2016/05/31/jawbone-new-wearable/>; abgerufen am 20. Dezember 2016.
- [Rasa] Raspberry Pi Foundation. config.txt. Webseite. Online erhältlich unter <https://www.raspberrypi.org/documentation/configuration/config-txt.md>; abgerufen am 16. Januar 2017.

- [Rasb] Raspberry Pi Foundation. Raspberry Pi 3 Model B. Webseite. Online erhältlich unter <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>; abgerufen am 29. Dezember 2016.
- [Ras16] Raspberry Pi Foundation. Schematics Raspberry Pi 3 Model B. Webseite, 2016. Online erhältlich unter <https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/RPI-3B-V1-2-SCHEMATIC-REDUCED.pdf>; abgerufen am 01. Dezember 2016.
- [Rat93] Rat der europäischen Gemeinschaft. Richtlinie 93/42/EWG Des Rates vom 14. Juni 1993 über Medizinprodukte. Webseite, Juni 1993. Online erhältlich unter <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:1993L0042:20071011:de:PDF>; abgerufen am 19. Dezember 2016.
- [Rat14] Rat der europäischen Gemeinschaft. Richtlinie 2014/30/EU des europäischen Parlaments und des Rates vom 26. Februar 2014 zur Harmonisierung der Rechtsvorschriften der Mitgliedstaaten über die elektromagnetische Verträglichkeit (Neufassung). Webseite, Februar 2014. Online erhältlich unter eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32014L0030&from=DE; abgerufen am 19. Dezember 2016.
- [Row13] J. Rowberg. [BGScript]: whitelist_peripheral - BLE peripheral with pairing/bonding and whitelisting. Webseite, 2013. Online erhältlich unter <https://bluegiga.zendesk.com/entries/36943196--BGScript%2Dwhitelist%2Dperipheral%2DBLE%2Dperipheral%2Dwith%2Dpairing%2Dbonding%2Dand%2Dwhitelisting>; abgerufen am 19. November 2016.
- [RS] RS Components GmbH. Texas Instruments CC2650STK, Evaluation Kit, Wireless. Webseite. Online erhältlich unter <http://de.rs-online.com/web/p/entwicklungskits-hf/8735892/>; abgerufen am 28. Dezember 2016.
- [Sie13] Sierra Wireless. Supported AT Command Reference. Webseite, 2013. Online erhältlich unter https://www.sierrawireless.com/resources/documents/support/2130617_supported_at_command_reference-v2.4.pdf; abgerufen am 14. Dezember 2016.
- [Sil] Silicon Laboratories. Bluegiga Bluetooth Smart Software Stack. Webseite. Online erhältlich unter <https://www.silabs.com/products/wireless/bluetooth/bluetooth-smart-modules/Pages/bluegiga-bluetooth-smart-software-stack.aspx>; abgerufen am 16. Dezember 2016.
- [Tex] Texas Instruments Incorporated. SimpleLink Bluetooth low energy/Multi-standard SensorTag. Webseite. Online erhältlich unter <http://www.ti.com/tool/cc2650stk>; abgerufen am 29. Dezember 2016.
- [Tex16] Texas Instruments Incorporated. CC2650 SensorTag User's Guide. Webseite, 2016. Online erhältlich unter http://processors.wiki.ti.com/index.php/CC2650_SensorTag_User's_Guide; abgerufen am 29. Dezember 2016.
- [Thea] The Apache Software Foundation. Apache HttpComponents. Website. Online erhältlich unter <https://hc.apache.org/>; abgerufen am 24. Januar 2017.
- [Theb] The Eclipse Foundation. jetty. Webseite. Online erhältlich unter <https://eclipse.org/jetty/>; abgerufen am 24. Januar 2017.

- [Tir16] T. Tirronen. Cellular IoT alphabet soup. Webseite, Februar 2016. Online erhältlich unter <https://www.ericsson.com/research-blog/internet-of-things/cellular-iot-alphabet-soup/>; abgerufen am 16. Januar 2017.
- [TLL⁺11] D. M. Taub, S. B. Leeb, E. C. Lupton, R. T. Hinman, J. Zeisel, and S. Blackler. The Escort System: A Safety Monitor for People Living with Alzheimer's Disease. *IEEE Pervasive Computing*, 10, April 2011. Issue 2.
- [Tom] TomTom International BV. Fitness Trackers. Webseite. Online erhältlich unter https://www.tomtom.com/de_de/sports/fitness-trackers/; abgerufen am 04. Januar 2017.
- [Tra] Tragant Handels- und Beteiligungs GmbH. Delock Micro USB Adapter mit LED Anzeige für Volt und Ampere. Webseite. Online erhältlich unter http://www.delock.de/produkte/G_65656/merkmale.html; abgerufen am 06. Dezember 2016.
- [Tre] Treats for Geeks. PixiePro. Webseite. Online erhältlich unter <https://store.treats4geeks.com/index.php/pixiepro-27.html>; abgerufen am 29. Dezember 2016.
- [Twi] Twitter Inc. Bootstrap. Website. Online erhältlich unter <http://getbootstrap.com/>; abgerufen am 24. Januar 2017.
- [U.S16] U.S. Department of Homeland Security (DHS). Strategic Principles for Securing the Internet of Things (IoT). Webseite, November 2016. Online erhältlich unter https://www.dhs.gov/sites/default/files/publications/Strategic_Principles_for_Securing_the_Internet_of_Things-2016-1115-FINAL....pdf; abgerufen am 23. November 2016.
- [Vara] Variscite. DART-MX6 CPU: NXP/Freescale iMX6. Webseite. Online erhältlich unter <http://www.variscite.com/products/system-on-module-som/cortex-a9/dart-mx6-cpu-freescale-imx6>; abgerufen am 29. Dezember 2016.
- [Varb] Variscite. VAR-SOM-MX6 CPU: NXP/Freescale iMX6. Webseite. Online erhältlich unter <http://www.variscite.com/products/system-on-module-som/cortex-a9/var-som-mx6-cpu-freescale-imx6>; abgerufen am 29. Dezember 2016.
- [Var07] U. Varshney. Pervasive Healthcare and Wireless Health Monitoring. *Mobile Networks and Applications*, 12, März 2007. Issue 2-3.
- [Ver16] N. Versel. Philips launches line of clinically validated consumer health devices. Webseite, August 2016. Online erhältlich unter <http://medcitynews.com/2016/08/philips-consumer-health-devices/>; abgerufen am 20. Dezember 2016.
- [Wora] World Wide Web Consortium (W3C). Generic Sensor API. Webseite. Online erhältlich unter <https://w3c.github.io/sensors/>; abgerufen am 15. Januar 2017.
- [Worb] World Wide Web Consortium (W3C). Web Bluetooth. Webseite. Online erhältlich unter <https://webbluetoothcg.github.io/web-bluetooth/>; abgerufen am 15. Januar 2017.

A. Test Cases

Im Folgenden werden Test Cases für die Sensorplattform basierend auf einem Raspberry Pi 3 vorgestellt. Für den vollständigen Test sind unter anderem SSH-Zugang und Kenntnisse über IP-Adressen in verschiedenen Netzwerken erforderlich, die dem medizinischen Fachpersonal jedoch in der Praxis nicht zur Verfügung stehen. Die Test Cases sind so konzipiert, dass sie zu einem späteren Zeitpunkt, beispielsweise nach Änderungen oder Erweiterungen, wiederholt durchgeführt werden können. Zu jedem Test Case ist angegeben, welcher Anwendungsfall aus Kapitel 4.8 getestet wird. Die Test Cases bestehen aus Positivtests, um die korrekte Funktionsweise der Software zu Überprüfen und aus einigen Negativtests, um die Robustheit des Systems zu gewährleisten.

Voraussetzung:

- Eine Sensorplattform mit installierter Software auf einer micro-SD-Karte
- Huawei E352S-5 Telekom Surfstick an der Sensorplattform angeschlossen
- PC und Browser
- Gerätename bzw. IP-Adressen der Sensorplattform im Netzwerk

Login-Daten der Webanwendung:

- URL: <https://sensorplatform.fit.fraunhofer.de:8080> bzw. IP-Adresse
- Login-Name: sensorplatform
- Passwort: 123456
- SSH LoginName: administrator
- SSH Passwort: <bei der Installation festgelegt>

1. Start der Sensorplattform

Zu Beginn wird der Startvorgang und die Erreichbarkeit der Sensorplattform getestet.

TEST-1.1 Start der Sensorplattform ohne Ereignisse

Positivtest: Use Case 1

Vorbedingung: -

Durchführung: Die Sensorplattform wird entweder an ein Netzteil oder den Akku angeschlossen und startet damit automatisch das Betriebssystem und die Anwendung. Außerdem ist sie über Kabel mit einem Netzwerk verbunden. Nach einigen Sekunden wird die Webanwendung der Sensorplattform mit einem Browser aufgerufen.

Erwartetes Ergebnis	Trifft zu	Bemerkung
<p>Der Browser meldet eine potenziell unsichere Webseite, da ein TLS/SSL Zertifikat genutzt wird, das nicht von einer anerkannten Dienststelle erzeugt wurde.</p> <p>Der Tester kann eine Ausnahmeregel für diese Seite hinzufügen und diese anschließend aufrufen. Außerdem blinkt die gelbe LED regelmäßig eine Sekunde an/aus.</p>		

2. Zugang zur Webanwendung der Sensorplattform

Die Sensorplattform ist ein sogenanntes HEADLESS-System, das heißt an diesem Gerät ist kein Monitor, keine Tastatur oder Maus angeschlossen. Alle Einstellungen und Konfigurationen werden über die Webanwendung der Sensorplattform mit Hilfe eines Webbrowsers

durchgeführt. Dies stellt besondere Ansprüche an die Sicherheit dar, weil darauf nur spezielles Personal Zugriff haben soll.

TEST-2.1 Anmeldung an der Webanwendung (falsche Login-Daten)

Negativtest: Use Case 2

Vorbedingung: TEST-1.1

Durchführung: Der Tester ruft die Webanwendung der Sensorplattform über die URL oder IP-Adresse auf. Der Tester gibt falsche Login-Daten ein: Benutzername=123, Passwort=123.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Beim Aufruf der Seite erscheint zunächst ein neues Fenster für die Autorisierung. Die „Ausgabe der Webseite“ lautet „Sensorplatform“. Das Fenster erscheint nach Eingabe der Login-Daten erneut und ermöglicht weitere Login-Versuche. Die Webanwendung der Sensorplattform wird jedoch nicht geladen.		

TEST-2.2 Anmeldung an der Webanwendung (Abbruch)

Negativtest: Use Case 2

Vorbedingung: TEST-1.1

Durchführung: Der Tester ruft die Webanwendung der Sensorplattform über die URL oder IP-Adresse auf. Der Tester klickt auf „abbrechen“ oder auf das Kreuz oben rechts, um das Fenster zu schließen.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Beim Aufruf der Seite erscheint zunächst ein neues Fenster für die Autorisierung. Die „Ausgabe der Webseite“ lautet „Sensorplatform“. Die Webseite gibt nach Eingabe der Login-Daten den Fehler „HTTP ERROR 401 unauthorized“ aus. Außerdem wird die Version des Jetty Webservers angezeigt.		

TEST-2.3 Anmeldung an der Webanwendung (korrekte Login-Daten)

Positivtest: Use Case 2

Vorbedingung: TEST-1.1

Durchführung: Der Tester ruft die Webanwendung der Sensorplattform über die URL oder IP-Adresse auf. Der Tester gibt korrekte Login-Daten ein.

Erwartetes Ergebnis	Trifft zu	Bemerkung
<p>Beim Aufruf der Seite erscheint zunächst ein neues Fenster für die Autorisierung. Die „Ausgabe der Webseite“ lautet „Sensorplatform“. Die Webseite wird nach Eingabe der Login-Daten geladen. Die Anmeldung ist so lange gültig, bis der Browser neu gestartet wird oder der Zwischenspeicher (Cache) des Browsers gelöscht wird.</p> <p>Es werden zudem folgende Informationen angezeigt: Hardware der Sensorplattform, externes Webinterface, Aufzeichnungsstatus und Informationen zu Signalqualität der Mobilfunkverbindung.</p>		

3. Aufruf der Webanwendung

Aus Sicherheitsgründen soll die Webanwendung der Sensorplattform ausschließlich in lokalen Netzwerken verfügbar sein. Der Zugriff auf die Webanwendung über das Mobilfunknetzwerk (ppp0) ist nicht möglich.

TEST-3.1 Aufruf der Webanwendung über das Kabel-Netzwerk

Positivtest: Use Case 1

Vorbedingung: TEST-1.1. Die Sensorplattform ist per Kabel mit einem Netzwerk verbunden.

Durchführung: Aufruf der Webanwendung im Browser.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Webanwendung der Sensorplattform ist mit dem Browser erreichbar.		

TEST-3.2 Aufruf der Webanwendung über WLAN

Positivtest: Sonstige

Vorbedingung: TEST-1.1. Die Sensorplattform ist mit einem WLAN verbunden.

Durchführung: Aufruf der Webanwendung im Browser.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Webanwendung der Sensorplattform ist mit dem Browser erreichbar.		

TEST-3.3 Aufruf der Webanwendung über das Mobilfunk-Netzwerk

Negativtest: Sonstige

Vorbedingung: TEST-1.1. Die Sensorplattform ist per Surfstick mit Mobilfunk (Internet) verbunden.

Durchführung: Aufruf der Webanwendung im Browser.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Webanwendung der Sensorplattform ist nicht mit dem Browser erreichbar.		

4. SSH Zugang zur Sensorplattform

Als Möglichkeit der (Fern-) Wartung soll die Sensorplattform (zum Beispiel für Wartungstechniker) auch über SSH verfügbar sein. Aus Sicherheitsgründen kann der SSH-Zugriff aber nur von internen Netzen (Kabel, WLAN) erfolgen, nicht aber über mobiles Internet, falls die Sensorplattform auf diese Weise verbunden ist.

TEST-4.1 SSH Zugang über das Kabel-Netzwerk

Positivtest: Sonstige

Vorbedingung: TEST-1.1. Die Sensorplattform ist per Kabel mit einem Netzwerk verbunden.

Durchführung: Der Tester nutzt einen SSH-Client und entsprechende Login-Daten um eine Verbindung zur Sensorplattform aufzubauen und anschließend mit dem Befehl „exit“ wieder zu schließen.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Verbindung wird erfolgreich aufgebaut und wieder getrennt.		

TEST-4.2 SSH Zugang über WLAN

Positivtest: Sonstige

Vorbedingung: TEST-1.1. Die Sensorplattform ist mit einem WLAN verbunden.

Durchführung: Der Tester nutzt einen SSH-Client und Login-Daten um eine Verbindung zur Sensorplattform aufzubauen und anschließend mit dem Befehl „exit“ wieder schließen.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Verbindung wird erfolgreich aufgebaut und wieder getrennt.		

TEST-4.3 SSH Zugang über das Mobilfunk-Netzwerk

Negativtest: Sonstige

Vorbedingung: TEST-1.1. Die Sensorplattform ist per Surfstick mit Mobilfunk (Internet) verbunden.

Durchführung: Der Tester versucht mit einem SSH-Client und den Login-Daten eine Verbindung zur Sensorplattform aufzubauen.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Es wird keine Verbindung aufgebaut.		

5. Verschiedene Nutzungsszenarien

Die Sensorplattform reagiert auf verschiedene Ereignisse autonom, da der Patient so wenig Interaktion wie möglich mit dem System haben soll. Mit den folgenden Tests soll der korrekte Funktionsablauf der Sensorplattform überprüfbar sein. Das Board startet bei angeschlossener Stromversorgung automatisch das Betriebssystem.

Als allgemeine Vorbedingung ist die Sensorplattform eingeschaltet und per Kabel mit einem Netzwerk und per Surfstick mit dem Mobilfunknetz verbunden. Außerdem ist die Webanwendung der Sensorplattform im Browser aufgerufen.

TEST-5.1 Start ohne Unterbrechung einer vorherigen Aufzeichnung

Positivtest: Use Case 1

Vorbedingung: -

Durchführung: Falls bereits eine Stromversorgung der Sensorplattform angeschlossen ist, wird diese unterbrochen und anschließend wiederhergestellt. Der Tester wartet ungefähr 15 Sekunden und ruft danach die Webanwendung der Sensorplattform auf.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform startet den Webserver. Es werden keine Sensordaten aufgezeichnet.		

TEST-5.2 Aufzeichnung von beliebigen Sensordaten

Positivtest: Use Case 4

Betrifft auch: Use Cases 3 und 9

Vorbedingung: -

Durchführung: Starten einer Aufzeichnung mit einem oder mehreren beliebigen Sensoren, einem Zeitintervall von 30 Sekunden und einem Patientennamen. Dafür gibt der Tester die entsprechenden Daten (zum Beispiel wie in Abbildung 23 in Kapitel 6.3) in der Webanwendung der Sensorplattform ein. Die Sensoren sind während der gesamten Zeit verfügbar.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Es erscheint ein Popup mit dem Text „A new recording period was started successfully!“. Die Sensorplattform zeichnet für 30 Sekunden Samples auf und kehrt anschließend wieder in den Bereitschaftsmodus zurück (und löscht die alte Konfiguration). Diese Statusinformation („recording: true/false“) ist an der Webanwendung ablesbar und ändert sich entsprechend. Außerdem blinkt die LED während der Aufzeichnung regelmäßig zweimal kurz auf (<i>Heartbeat</i>).		

TEST-5.3 Abbruch einer beliebigen Aufzeichnung

Positivtest: Use Case 6

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Es wird eine beliebige Aufzeichnung mit einer Dauer von 600 Sekunden gestartet. Sobald die Webanwendung den Status der Sensorplattform zu „recording:true“ ändert, wird der Stop-Button in der Webanwendung gedrückt.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform stoppt die aktuelle Aufzeichnung und ist wieder im Bereitschaftsmodus („recording:false“). Die LED blinkt regelmäßig je eine Sekunde an/aus.		

TEST-5.4 Abbruch vor Beginn einer Aufzeichnung

Positivtest: Use Case 10

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Starten einer Aufzeichnung mit einem oder mehreren beliebigen Sensoren und einer Dauer von 30 Sekunden. Mindestens ein ausgewählter Sensor ist nicht verfügbar und kann somit nicht mit der Sensorplattform verbunden werden. Um einen nicht verfügbaren Sensor zu simulieren, kann bspw. eine zufällige Bluetooth-Adresse gewählt werden.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform versucht sich fünf Sekunden lang mit den Sensoren zu verbinden. Der Versuch für mindestens ein Sensor scheitert und die Initialisierung der Datenaufzeichnung wird abgebrochen. Die Sensorplattform ist wieder im Bereitschaftsmodus. Es erscheint ein Popup mit dem Text „A sensor is not available: <Sensorname> <Bluetooth-Adresse>“. Es findet keine Datenaufzeichnung statt. Gegebenenfalls kann für 3 Sekunden der Status „recording:true“ angezeigt werden, da dieser nur alle drei Sekunden aktualisiert wird. Die LED blinkt wieder regelmäßig je eine Sekunde an/aus.		

TEST-5.5 Start nach Unterbrechung einer Aufzeichnung (abgelaufen)

Positivtest: Use Case 12

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Es wird eine beliebige Aufzeichnung mit einer kurzen Dauer (30 Sekunden) gestartet. Die Stromversorgung der Sensorplattform wird unterbrochen, aber erst nach mehr als 30 Sekunden wiederhergestellt. Anschließend wird die Webanwendung der Sensorplattform über einen Browser erneut aufgerufen.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform startet den Webserver. Die letzte Aufzeichnung wird nicht fortgeführt, weil der Endzeitpunkt in der Vergangenheit liegt. Der Status lässt sich in der Webanwendung ablesen („recording:false“). Die LED blinkt regelmäßig eine Sekunde an/aus.		

TEST-5.6 Start nach Unterbrechung einer Aufzeichnung (nicht abgelaufen)

Positivtest: Use Case 11

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Es wird eine beliebige Aufzeichnung mit einer Dauer von 300 Sekunden gestartet und der Tester merkt sich den Startzeitpunkt. Die Stromversorgung der Sensorplattform wird für wenige Sekunden unterbrochen und anschließend wiederhergestellt. Danach wird die Webanwendung der Sensorplattform über einen Browser erneut aufgerufen. Die genutzten Sensoren sind im gesamten Zeitraum verfügbar.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform startet den Webserver, verbindet sich mit den Sensoren und führt die Messung für die verbleibende Zeit fort. Der Status des Systems lässt sich in der Webanwendung ablesen („recording: true“). Die LED blinkt regelmäßig zweimal kurz auf (<i>Heartbeat</i>). Nach Ablauf der 5 Minuten geht die Sensorplattform wieder in den Bereitschaftsmodus zurück („recording:false“). Die LED blinkt dann wieder regelmäßig eine Sekunde an/aus.		

TEST-5.7 Aufzeichnung von Herzfrequenzdaten mit Unterbrechung

Positivtest: Use Case 13

Betrifft auch: Use Case 3, 4 und 9

Vorbedingung: -

Durchführung: Es wird eine Aufzeichnung mit einem Herzfrequenz-Sensor (Polar, Adidas, oder TomTom) für 600 Sekunden gestartet. Der Sensor ist zu Beginn verfügbar, danach ca. zwei Minuten ohne Hautkontakt. Bereits nach ungefähr 15 Sekunden geht der Sensor in den Energiesparmodus und sendet keine Daten mehr. Nach Ablauf der zwei Minuten stellt der Tester den Hautkontakt mit den Sensoren wieder her.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Es erscheint ein Popup mit dem Text „A new recording period was started successfully!“. Die Sensorplattform erkennt nach einigen Sekunden, dass der Sensor keine Daten mehr sendet und versucht kontinuierlich die Verbindung wiederherzustellen. Die LED blinkt nicht, sondern ist während der Unterbrechungsphase aus. Nachdem der Sensor wieder Hautkontakt hat, verbindet sich die Sensorplattform erneut und zeichnet für die verbleibende Zeit Daten auf. In der Webanwendung können die Herzfrequenzdaten angezeigt werden. Anhand der einzelnen Zeitstempel lässt sich die Verfügbarkeit und die Abwesenheit des Sensors über die Zeit nachvollziehen.		

TEST-5.8 Fortführen einer Aufzeichnung mit nicht aktivem Sensor

Positivtest: Sonstige

Betrifft auch: Use Case 3, 4 und 9

Vorbedingung: -

Durchführung: Es wird eine Aufzeichnung mit einem Herzfrequenzsensor für 10 Minuten gestartet. Die Stromversorgung der Sensorplattform wird für ca. 30 Sekunden unterbrochen und anschließend wiederhergestellt. In der Zwischenzeit ist der Sensor nicht mehr verfügbar (Energiesparmodus). Ab einem beliebigen Zeitpunkt vor dem Ende der Aufzeichnung stellt der Tester den Hautkontakt mit dem Sensor wieder her.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform versucht, die Aufzeichnung bis zum Endzeitpunkt fortzuführen, auch wenn der Sensor nicht verfügbar ist. Das Gerät versucht sich kontinuierlich für die verbleibende Aufzeichnungszeit mit dem Sensor zu verbinden („recording:true“). Solange keine Verbindung mit dem Sensor hergestellt ist, blinkt die LED nicht auf. Sobald der Sensor wieder Hautkontakt hat, werden die Messungen fortgeführt und die LED blinkt wieder im <i>Heartbeat</i> -Muster.		

TEST-5.9 Abschluss einer Aufzeichnung mit einem nicht mehr aktiven Sensor

Positivtest: Sonstige

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Es wird eine Aufzeichnung für 5 Minuten mit einem beliebigen Herzfrequenzsensor gestartet. Der Sensor ist nur zu Beginn verfügbar, danach bis zum Ende der Aufzeichnung nicht mehr.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform beendet die Aufzeichnung nach Ablauf der Zeit und versucht danach nicht mehr, sich mit dem Sensor zu verbinden („recording:false“). Die LED blinkt von da an wieder regelmäßig eine Sekunde an/aus.		

TEST-5.10 Neue Aufzeichnung während eine andere Aufzeichnung aktiv ist

Positivtest: Use Case 17

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Eine beliebige Aufzeichnung wird gestartet. Anschließend versucht der Tester eine weitere beliebige Aufzeichnung zu starten, während die erste noch aktiv ist.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die erste Aufzeichnung wird gestartet, erkennbar an dem Status („recording:true“) und der LED (<i>Heartbeat</i>). Die zweite Aufzeichnung wird nicht gestartet und beim zweiten Startversuch erscheint ein Popup mit dem Text „A recording period is already running!“.		

TEST-5.11 Leere Sensorkonfiguration

Positivtest: Use Case 5

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Eine beliebige Aufzeichnung soll gestartet werden. Die Sensorkonfiguration hat den Text „[]“ (ein leeres JSON-Array). Es werden beliebige Eingaben für die Namen und Dauer der Aufzeichnung gewählt.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Es erscheint ein Popup mit dem Text „There are no sensors in the configuration!“.		

TEST-5.12 Fehlende Startangabe der Sensorkonfiguration

Positivtest: Use Case 5

Vorbedingung: -

Durchführung: Eine beliebige Aufzeichnung soll gestartet werden. Das Feld für die Sensorkonfiguration ist leer.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Es erscheint die Fehlermeldung „Bad sensor configuration JSON syntax!“. Keine Aufzeichnung wird gestartet.		

TEST-5.13 Fehlende Startangabe der Messdauer

Positivtest: Use Case 5

Vorbedingung: -

Durchführung: Eine beliebige Aufzeichnung soll gestartet werden. Ein Wert für die Dauer der Aufzeichnung wird nicht angegeben.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Es erscheint die Fehlermeldung „No uptime specified!“. Keine Aufzeichnung wird gestartet.		

TEST-5.14 Aufzeichnung über mehrere Stunden

Positivtest: Use Case 8

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Starten einer Aufzeichnung mit mindestens einem Herzfrequenzsensor und beliebig vielen weiteren Sensoren, einem Zeitintervall von 5 Stunden und einem Patientennamen. Die Sensoren sind gegebenenfalls zwischendurch nicht mehr verfügbar (beispielsweise kein Hautkontakt), sodass Verbindungsabbrüche auftreten können. Der Tester trägt das System am Körper.

Erwartetes Ergebnis	Trifft zu	Bemerkung
<p>Es erscheint ein Popup mit dem Text „A new recording period was started successfully!“. Die Sensorplattform zeichnet für 5 Stunden Samples auf und kehrt anschließend wieder in den Bereitschaftsmodus zurück (und löscht die alte Konfiguration). Diese Statusinformation („recording: true/false“) ist an der Webanwendung ablesbar und ändert sich entsprechend. Außerdem blinkt die LED während der Aufzeichnung regelmäßig zweimal kurz auf (<i>Heartbeat</i>).</p> <p>Der Test wird als erfolgreich gewertet, falls die Sensorplattform die Sensordaten aufzeichnet, gegebenenfalls Verbindungen zu Sensoren erneuert und die Samples hochlädt. Es ist in den 5 Stunden keine Interaktion des Testers mit der Sensorplattform notwendig.</p>		

TEST-5.15 Aufzeichnung mit einer simulierten Mobilfunkstörung

Positivtest: Use Case 15

Betrifft auch: Use Cases 3, 4 und 9

Vorbedingung: -

Durchführung: Starten einer Aufzeichnung mit einem Herzfrequenzsensor, einem Zeitintervall von 300 Sekunden und einem Patientennamen. Der Sensor hat während des Tests kontinuierlichen Hautkontakt. Der Tester simuliert nach 180 Sekunden eine Störung des Mobilfunksignals durch das kurzzeitige Trennen des Surfsticks von der Sensorplattform. Nach Ablauf der 300 Sekunden öffnet der Tester den Tab „Database“.

Erwartetes Ergebnis	Trifft zu	Bemerkung
<p>Es erscheint ein Popup mit dem Text „A new recording period was started successfully!“. Die Sensorplattform zeichnet für 5 Minuten Samples auf und kehrt anschließend wieder in den Bereitschaftsmodus zurück (und löscht die alte Konfiguration). Diese Statusinformation („recording: true/false“) ist an der Webanwendung ablesbar und ändert sich entsprechend. Außerdem blinkt die LED während der Aufzeichnung regelmäßig zweimal kurz auf (<i>Heartbeat</i>).</p> <p>Der Tab „Database“ zeigt die gesamte Anzahl Herzfrequenzsamples und die Anzahl nicht-hochgeladener Samples. Es gibt insgesamt ungefähr 900 Herzfrequenzsamples. Die Anzahl nicht-hochgeladener Samples ist von 0 verschieden.</p>		

6. Datenbank

Über die Webanwendung der Sensorplattform lassen sich die in der Datenbank gespeicherten Samples anzeigen und weitere Aktionen mit diesen durchführen. Für die Tests gilt, falls keine Samples in der Datenbank vorhanden sind, können durch entsprechende Aufzeichnungen Samples erzeugt werden und anschließend mit den Tests fortgefahrene werden.

Als allgemeine Vorbedingung ist die Sensorplattform eingeschaltet, per Kabel mit einem Netzwerk und per Surfstick mit dem Mobilfunknetz verbunden. Außerdem ist die Webanwendung der Sensorplattform im Browser aufgerufen.

TEST-6.1 Anzahl der Herzfrequenz-Samples anzeigen

Positivtest: Use Case 14

Vorbedingung: -

Durchführung: Aufruf des Tabs „Database“.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Rechts wird die gesamte Anzahl an Herzfrequenz-Samples und die Anzahl der Herzfrequenz-Samples, die nicht hochgeladen wurden, angezeigt.		

TEST-6.2 Samples anzeigen

Positivtest: Use Case 14

Vorbedingung: TEST-6.1

Durchführung: Klicks auf die Buttons „Load all“ bei „Heart Rate Samples“ und bei „Pulse Oximeter Samples“. Klicks auf die Buttons „Temperature“, „Humidity“, „Pressure“, „Ambientlight“ und „Movement“.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Alle Samples des jeweiligen Typs werden im JSON-Format links in der Webanwendung angezeigt.		

TEST-6.3 Herzfrequenz-Samples manuell hochladen

Positivtest: Use Case 16

Vorbedingung: TEST-6.1

Durchführung: Aufruf des Tabs „Database“. Klick auf den Button „Upload samples“.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Anzahl aller Samples und die Anzahl der nicht-hochgeladenen Samples wird dargestellt. Der Status wird auf „recording:true“ gesetzt, während die Samples hochgeladen werden. Nach einigen Sekunden wird die Zahl der nicht-hochgeladenen Samples auf 0 aktualisiert und „recording:false“ gesetzt. Es erscheint ein Popup mit dem Text „All not-transmitted heart rate samples uploaded“.		

TEST-6.4 Herzfrequenz-Samples löschen

Positivtest: Use Case 14

Vorbedingung: TEST-6.1

Durchführung: Aufruf des Tabs „Database“. Klick auf Button „Delete all“ in der Kategorie „Heart Rate Samples“.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Alle Herzfrequenz-Samples werden aus der Datenbank gelöscht und die Zahlenangaben in der Webanwendung aktualisiert.		

TEST-6.5 Pulse-Oximeter-Samples löschen

Positivtest: Use Case 14

Vorbedingung: TEST-6.1

Durchführung: Aufruf des Tabs „Database“. Klick auf Button „Delete all“ in der Kategorie „Pulse Oximeter Samples“.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Alle Pulsoximeter-Samples werden aus der Datenbank gelöscht und die Zahlen aktualisiert.		

7. Texas Instruments SensorTag CC2650

Im Gegensatz zu den Herzfrequenzsensoren ist die Abtastrate des CC2650 nicht auf 1 Hz festgelegt, sondern lässt sich verändern (zwischen 0,4 Hz und 3,3 Hz). Dieser kann ein bis fünf Parameter aufzeichnen und es können einzelne Messungen deaktiviert werden (siehe Kapitel 5.1.3). Diese Konfiguration wird ebenfalls über die Webanwendung vorgenommen und mit folgenden Tests auf korrekte Funktionsweise überprüft.

Als allgemeine Vorbedingung ist die Sensorplattform eingeschaltet, per Kabel mit einem Netzwerk und per Surfstick mit dem Mobilfunknetz verbunden ist. Außerdem ist die Webanwendung der Sensorplattform im Browser aufgerufen.

TEST-7.1 Aufzeichnung von CC2650-Daten mit der Beispielkonfiguration

Positivtest: Use Case 4

Betrifft auch: Use Cases 3 und 9

Vorbedingung: -

Durchführung: Es wird eine Aufzeichnung mit beliebiger Zeit mit dem CC2650 Sensor gestartet. Die „settings“ werden aus der Beispielkonfiguration („example configuration“) für alle fünf Messgrößen übernommen. Die Abstände zwischen zwei Messungen eines Typs sollen damit eine Sekunde betragen.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform erhält einmal pro Sekunde Daten vom Sensor für jede der fünf zu erfassenden Messgrößen. Der Status ist während der Zeit als <i>recording:true</i> sichtbar und über die Ausgabe von Samples und deren Zeitstempel aus der Datenbank lässt sie die Abtastrate überprüfen.		

TEST-7.2 Aufzeichnung von CC2650-Daten

Positivtest: Use Case 4

Betrifft auch: Use Cases 3 und 9

Vorbedingung: -

Durchführung: Es wird eine Aufzeichnung mit beliebiger Zeit mit dem CC2650 Sensor gestartet. Die „settings“ werden wie folgt definiert: „temperature:1E“, alle vier anderen Parameter haben den Wert 0A. Die Abstände zwischen zwei Messungen sollen damit 300 Millisekunden betragen.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Die Sensorplattform erhält ungefähr alle 300ms Daten vom Sensor für jede der fünf zu erfassenden Messgrößen. Der Status ist während der Zeit als „recording:true“ sichtbar und über die Ausgabe von Samples und deren Zeitstempel aus der Datenbank lässt sie die Abtastrate überprüfen.		

8. Bluetooth Low Energy Geräte finden

In der Webanwendung sind bereits einige Geräteinformationen (Name und Bluetooth-Adresse) vorgegeben, mit denen die Sensorplattform entwickelt und getestet wurde. Um neue Geräte des gleichen Typs mit der Sensorplattform zu verbinden, ist es erforderlich, die Bluetooth-Adressen herauszufinden.

TEST-8.1 Scan nach Bluetooth Low Energy Geräten

Positivtest: Use Case 7

Vorbedingung: -

Durchführung: Zum Tab „Sensors“ wechseln. Der Tester wählt eine „Scan duration“ von 5 Sekunden und drückt auf den „Scan“ Knopf. Nach Ablauf der 5 Sekunden gibt die Webanwendung eine Rückmeldung über alle in der Nähe gefundenen Geräte aus.

Erwartetes Ergebnis	Trifft zu	Bemerkung
Der Name und die Bluetooth-Adresse für jeden gefundenen Sensor wird im JSON-Format angezeigt.		

B. Modultest Polar H7

```
1 package de.fhg.fit.biomos.sensorplatform.sensor;
2
3 import org.json.JSONObject;
4 import org.junit.Assert;
5 import org.junit.BeforeClass;
6 import org.junit.Test;
7
8 import de.fhg.fit.biomos.sensorplatform.gatt.PolarH7lib;
9 import de.fhg.fit.biomos.sensorplatform.sample.HeartRateSample;
10 import de.fhg.fit.biomos.sensorplatform.util.AddressType;
11 import de.fhg.fit.biomos.sensorplatform.util.SecurityLevel;
12 import de.fhg.fit.biomos.sensorplatform.util.SensorName;
13
14 /**
15 * This test ensures correct calculation of heart rate measurement data
16 * of the class AbstractHeartRateSensor. Only one concrete
17 * implementation is tested
18 * (PolarH7). This is sufficient, because all heart rate sensors inherit
19 * from AbstractHeartRateSensor.
20 *
21 * @see <a href=
22 *      "https://developer.bluetooth.org/gatt/characteristics/Pages/
23 *      CharacteristicViewer.aspx?u=org.bluetooth.characteristic.
24 *      heart_rate_measurement.xml">
25 *      Bluetooth Heart Rate Measurement</a>
26 *
27 * @author Daniel Pyka
28 */
29
30 public class PolarH7Test {
31
32     // control byte CC: 00 to 1F
33     // heart rate value HH/ HH HH: uint8/ uint16
34     // energy expended EE: uint16
35     // rr intervals (RR RR)*: uint16
36     // for 16 bit values gatttool display order is
37     // most significant octet (MSO) least significant octet (LSO)
38     // a number "ABCD" (hex) is displayed as "CD AB" (hex)
39
40     // CC HH EE EE RR RR RR RR
41     private static final String heartrateData1 = "1E 44 64 00 9b 03 74
42         03";
43     // CC HH EE EE
44     private static final String heartrateData2 = "0E 44 64 00 9b";
45     // CC HH RR RR RR RR
46     private static final String heartrateData3 = "16 44 9b 03 74 03";
47     // CC HH
48     private static final String heartrateData4 = "06 44";
49     // CC HH HH EE EE RR RR RR RR
50     private static final String heartrateData5 = "1F 44 00 64 00 9b 03
51         74 03";
52     // CC HH HH EE EE
53     private static final String heartrateData6 = "0F 44 00 64 00";
54     // CC HH HH RR RR RR RR
55     private static final String heartrateData7 = "17 44 00 9b 03 74 03"
56         ;
57     // CC HH HH
58     private static final String heartrateData8 = "07 44 00";
59
60     private static PolarH7 polarh7;
```

```
54     private static HeartRateSample hrs1;
55     private static HeartRateSample hrs2;
56     private static HeartRateSample hrs3;
57     private static HeartRateSample hrs4;
58     private static HeartRateSample hrs5;
59     private static HeartRateSample hrs6;
60     private static HeartRateSample hrs7;
61     private static HeartRateSample hrs8;
62
63     @BeforeClass
64     public static void setup() {
65         polarh7 = new PolarH7(SensorName.PolarH7, PolarH7lib.
66             DEFAULT_BDADDRESS, new JSONObject());
67         hrs1 = polarh7.calculateHeartRateData("2016-09-02T08:45:30.555Z
68             ", polarh7.gattLibrary.getHandleHeartRateMeasurement(),
69             heartrateData1);
70         hrs2 = polarh7.calculateHeartRateData("2016-09-01T23:12:55.378Z
71             ", polarh7.gattLibrary.getHandleHeartRateMeasurement(),
72             heartrateData2);
73         hrs3 = polarh7.calculateHeartRateData("2016-08-31T15:59:00.004Z
74             ", polarh7.gattLibrary.getHandleHeartRateMeasurement(),
75             heartrateData3);
76         hrs4 = polarh7.calculateHeartRateData("2016-08-30T01:01:01.010Z
77             ", polarh7.gattLibrary.getHandleHeartRateMeasurement(),
78             heartrateData4);
79         hrs5 = polarh7.calculateHeartRateData("2016-08-29T20:20:23.299Z
80             ", polarh7.gattLibrary.getHandleHeartRateMeasurement(),
81             heartrateData5);
82         hrs6 = polarh7.calculateHeartRateData("2016-08-28T11:30:40.990Z
83             ", polarh7.gattLibrary.getHandleHeartRateMeasurement(),
84             heartrateData6);
85         hrs7 = polarh7.calculateHeartRateData("2016-08-27T16:10:57.083Z
86             ", polarh7.gattLibrary.getHandleHeartRateMeasurement(),
87             heartrateData7);
88         hrs8 = polarh7.calculateHeartRateData("2016-08-26T22:20:16.224Z
89             ", polarh7.gattLibrary.getHandleHeartRateMeasurement(),
90             heartrateData8);
91     }
92
93     @Test
94     public void testSensorParameters() {
95         Assert.assertEquals(AddressType.PUBLIC, PolarH7Test.polarh7.
96             getAddressType());
97         Assert.assertEquals(SecurityLevel.LOW, PolarH7Test.polarh7.
98             getSecurityLevel());
99     }
100
101    @Test
102    public void testHandles() {
103        Assert.assertEquals("0x0003", PolarH7Test.polarh7.gattLibrary.
104            getHandleDeviceName());
105        Assert.assertEquals("0x0012", PolarH7Test.polarh7.gattLibrary.
106            getHandleHeartRateMeasurement());
107        Assert.assertEquals("0x0013", PolarH7Test.polarh7.gattLibrary.
108            getHandleHeartRateNotification());
109    }
110
111    @Test
112    public void testHeartRateControlByte() {
113        // heartrateData1
114        Assert.assertFalse(PolarH7Test.polarh7.is16BitHeartRateValue(
115            PolarH7Test.heartrateData1));
```

```
93     Assert.assertTrue(PolarH7Test.polarh7.isEnergyExpendPresent(
94         PolarH7Test.heartrateData1));
95     Assert.assertTrue(PolarH7Test.polarh7.
96         isSkinContactDetectionSupported(PolarH7Test.heartrateData1)
97         );
98     Assert.assertTrue(PolarH7Test.polarh7.isSkinContactDetected(
99         PolarH7Test.heartrateData1));
100    Assert.assertTrue(PolarH7Test.polarh7.isRRintervalDataAvailable(
101        PolarH7Test.heartrateData1));
102    // heartrateData2
103    Assert.assertFalse(PolarH7Test.polarh7.is16BitHeartRateValue(
104        PolarH7Test.heartrateData2));
105    Assert.assertTrue(PolarH7Test.polarh7.isEnergyExpendPresent(
106        PolarH7Test.heartrateData2));
107    Assert.assertTrue(PolarH7Test.polarh7.
108        isSkinContactDetectionSupported(PolarH7Test.heartrateData2)
109        );
110    Assert.assertTrue(PolarH7Test.polarh7.isSkinContactDetected(
111        PolarH7Test.heartrateData2));
112    Assert.assertFalse(PolarH7Test.polarh7.
113        isRRintervalDataAvailable(PolarH7Test.heartrateData2));
114    // heartrateData3
115    Assert.assertFalse(PolarH7Test.polarh7.is16BitHeartRateValue(
116        PolarH7Test.heartrateData3));
117    Assert.assertFalse(PolarH7Test.polarh7.isEnergyExpendPresent(
118        PolarH7Test.heartrateData3));
119    Assert.assertTrue(PolarH7Test.polarh7.
120        isSkinContactDetectionSupported(PolarH7Test.heartrateData3)
121        );
122    Assert.assertTrue(PolarH7Test.polarh7.isSkinContactDetected(
123        PolarH7Test.heartrateData3));
124    Assert.assertTrue(PolarH7Test.polarh7.isRRintervalDataAvailable(
125        PolarH7Test.heartrateData3));
126    // heartrateData4
127    Assert.assertFalse(PolarH7Test.polarh7.is16BitHeartRateValue(
128        PolarH7Test.heartrateData4));
129    Assert.assertFalse(PolarH7Test.polarh7.isEnergyExpendPresent(
130        PolarH7Test.heartrateData4));
131    Assert.assertTrue(PolarH7Test.polarh7.
132        isSkinContactDetectionSupported(PolarH7Test.heartrateData4)
133        );
134    Assert.assertTrue(PolarH7Test.polarh7.isSkinContactDetected(
135        PolarH7Test.heartrateData4));
136    Assert.assertFalse(PolarH7Test.polarh7.
137        isRRintervalDataAvailable(PolarH7Test.heartrateData4));
138    // heartrateData5
139    Assert.assertTrue(PolarH7Test.polarh7.is16BitHeartRateValue(
140        PolarH7Test.heartrateData5));
141    Assert.assertTrue(PolarH7Test.polarh7.isEnergyExpendPresent(
142        PolarH7Test.heartrateData5));
143    Assert.assertTrue(PolarH7Test.polarh7.
144        isSkinContactDetectionSupported(PolarH7Test.heartrateData5)
145        );
146    Assert.assertTrue(PolarH7Test.polarh7.isSkinContactDetected(
147        PolarH7Test.heartrateData5));
148    Assert.assertTrue(PolarH7Test.polarh7.isRRintervalDataAvailable(
149        PolarH7Test.heartrateData5));
150    // heartrateData6
151    Assert.assertTrue(PolarH7Test.polarh7.is16BitHeartRateValue(
152        PolarH7Test.heartrateData6));
153    Assert.assertTrue(PolarH7Test.polarh7.isEnergyExpendPresent(
154        PolarH7Test.heartrateData6));
```

```
124     Assert.assertTrue(PolarH7Test.polarh7.
125         isSkinContactDetectionSupported(PolarH7Test.heartrateData6)
126         );
127     Assert.assertTrue(PolarH7Test.polarh7.isSkinContactDetected(
128         PolarH7Test.heartrateData6));
129     Assert.assertFalse(PolarH7Test.polarh7.
130         isRRintervalDataAvailable(PolarH7Test.heartrateData6));
131     // heartrateData7
132     Assert.assertTrue(PolarH7Test.polarh7.is16BitHeartRateValue(
133         PolarH7Test.heartrateData7));
134     Assert.assertFalse(PolarH7Test.polarh7.isEnergyExpendedPresent(
135         PolarH7Test.heartrateData7));
136     Assert.assertTrue(PolarH7Test.polarh7.
137         isSkinContactDetectionSupported(PolarH7Test.heartrateData7)
138         );
139     Assert.assertTrue(PolarH7Test.polarh7.isSkinContactDetected(
140         PolarH7Test.heartrateData7));
141     Assert.assertTrue(PolarH7Test.polarh7.isRRintervalDataAvailable(
142         PolarH7Test.heartrateData7));
143     // heartrateData8
144     Assert.assertTrue(PolarH7Test.polarh7.is16BitHeartRateValue(
145         PolarH7Test.heartrateData8));
146     Assert.assertFalse(PolarH7Test.polarh7.isEnergyExpendedPresent(
147         PolarH7Test.heartrateData8));
148     Assert.assertTrue(PolarH7Test.polarh7.
149         isSkinContactDetectionSupported(PolarH7Test.heartrateData8)
150         );
151     Assert.assertTrue(PolarH7Test.polarh7.isSkinContactDetected(
152         PolarH7Test.heartrateData8));
153     Assert.assertFalse(PolarH7Test.polarh7.
154         isRRintervalDataAvailable(PolarH7Test.heartrateData8));
155     }
156     @Test
157     public void testHeartRateValue() {
158         Assert.assertEquals(Integer.valueOf(68), PolarH7Test.hrs1.
159             getHeartRate());
160         Assert.assertEquals(Integer.valueOf(68), PolarH7Test.hrs2.
161             getHeartRate());
162         Assert.assertEquals(Integer.valueOf(68), PolarH7Test.hrs3.
163             getHeartRate());
164         Assert.assertEquals(Integer.valueOf(68), PolarH7Test.hrs4.
165             getHeartRate());
166         Assert.assertEquals(Integer.valueOf(68), PolarH7Test.hrs5.
167             getHeartRate());
168         Assert.assertEquals(Integer.valueOf(68), PolarH7Test.hrs6.
169             getHeartRate());
170         Assert.assertEquals(Integer.valueOf(68), PolarH7Test.hrs7.
171             getHeartRate());
172         Assert.assertEquals(Integer.valueOf(68), PolarH7Test.hrs8.
173             getHeartRate());
174     }
175     @Test
176     public void testEnergyExpendedValue() {
177         Assert.assertEquals(Integer.valueOf(100), PolarH7Test.hrs1.
178             getEnergyExpended());
179         Assert.assertEquals(Integer.valueOf(100), PolarH7Test.hrs2.
180             getEnergyExpended());
181         Assert.assertEquals(Integer.valueOf(100), PolarH7Test.hrs5.
182             getEnergyExpended());
183         Assert.assertEquals(Integer.valueOf(100), PolarH7Test.hrs6.
184             getEnergyExpended());
```

```
159     }
160
161     @Test
162     public void testRRintervalValue() {
163         Assert.assertEquals("[901.3672, 863.28125]", PolarH7Test.hrs1.
164             getRRintervals());
165         Assert.assertEquals("[901.3672, 863.28125]", PolarH7Test.hrs3.
166             getRRintervals());
167         Assert.assertEquals("[901.3672, 863.28125]", PolarH7Test.hrs5.
168             getRRintervals());
169     }
```

Quelltext 22: Modultest für den Sensor Polar H7 und die Klasse AbstractHeartRateSensor

C. Modultest BLE113

```
1 package de.fhg.fit.biomos.sensorplatform.sensor;
2
3 import org.json.JSONObject;
4 import org.junit.Assert;
5 import org.junit.BeforeClass;
6 import org.junit.Test;
7
8 import de.fhg.fit.biomos.sensorplatform.gatt.BLE113lib;
9 import de.fhg.fit.biomos.sensorplatform.sample.PulseOximeterSample;
10 import de.fhg.fit.biomos.sensorplatform.util.SensorName;
11
12 /**
13 * The BLE113 with given firmware simulates a very simple Pulse Oximeter
14 * Sensor. Only the mandatory fields of this characteristic are
15 * implemented (Control Byte,
16 * SpO2PR-Normal - SpO2 and SpO2PR-Normal - PR). Furthermore, the checks
17 * for the control byte are not implemented. This test covers only
18 * the calculation of the
19 * SpO2PR Normal fields. The characteristic features a lot more
20 * information, which would require a "real" sensor, not a development
21 * board with custom firmware.
22 * For detecting which data would be available, all checks for the
23 * control byte are also mandatory.
24 *
25 * @see <a href= "https://www.bluetooth.com/specifications/gatt/viewer?
26 * attributeXmlFile=org.bluetooth.characteristic.
27 * plx_continuous_measurement.xml"> PLX
28 *      Continuous Measurement Characteristic</a>
29 *
30 * @author Daniel Pyka
31 *
32 */
33 public class BLE113Test {
34
35     private static final String pulseOximeterData1 = "00 00 62 00 3C";
36     private static final String pulseOximeterData2 = "00 00 5A 00 78";
37     private static final String pulseOximeterData3 = "00 00 4F 00 B4";
38
39     private static BLE113 ble113;
40
41     private static PulseOximeterSample pos1;
42     private static PulseOximeterSample pos2;
43     private static PulseOximeterSample pos3;
44
45     @BeforeClass
46     public static void setup() {
47         ble113 = new BLE113(SensorName.BLE113, BLE113lib.
48             DEFAULT_BDADDRESS, new JSONObject());
49         pos1 = ble113.calculatePulseOximeterData("2016-09-02T08
50             :45:30.555Z", ble113.gattLibrary.
51             getHandlePulseOximeterMeasurement(), pulseOximeterData1);
52         pos2 = ble113.calculatePulseOximeterData("2016-09-01T23
53             :12:55.378Z", ble113.gattLibrary.
54             getHandlePulseOximeterMeasurement(), pulseOximeterData2);
55         pos3 = ble113.calculatePulseOximeterData("2016-08-31T15
56             :47:08.160Z", ble113.gattLibrary.
57             getHandlePulseOximeterMeasurement(), pulseOximeterData3);
58     }
59
60     @Test
61     public void testControlByte() {
```

```
46     // always 00 00 for BLE113 firmware
47 }
48
49 @Test
50 public void testSp02NormalSp02() {
51     Assert.assertEquals(Integer.valueOf(98), BLE113Test.pos1.
52         getSp02());
53     Assert.assertEquals(Integer.valueOf(90), BLE113Test.pos2.
54         getSp02());
55     Assert.assertEquals(Integer.valueOf(79), BLE113Test.pos3.
56         getSp02());
57 }
58
59 @Test
60 public void testSp02NormalPulseRate() {
61     Assert.assertEquals(Integer.valueOf(60), BLE113Test.pos1.
62         getPulseRate());
63     Assert.assertEquals(Integer.valueOf(120), BLE113Test.pos2.
64         getPulseRate());
65     Assert.assertEquals(Integer.valueOf(180), BLE113Test.pos3.
66         getPulseRate());
67 }
```

Quelltext 23: Modultest für den Sensor BLE113 und die Klasse AbstractPulseOximeterSensor