Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Miloš Đorđević

Entwicklung eines webbasierten Software- und Informations-Managementsystems für medizinische Geräte

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Miloš Đorđević

Entwicklung eines webbasierten Software- und Informations-Managementsystems für medizinische Geräte

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Henning Dierks
Zweitgutachter : Prof. Dr. Robert Heß

Abgegeben am 22. Dezember 2016

Miloš Đorđević

Thema der Bachelorthesis

Entwicklung eines webbasierten Software- und Informations-Managementsystems für medizinische Geräte

Stichworte

C++, Qt, Embedded Linux, PHP, Datenbank, Kryptographie

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung eines Software-Managementsystems für medizinische Geräte. Unter Beachtung der Sicherheitsaspekte bei der Internetkommunikation, wird ein System geschaffen, welches den Medizingeräten Synchronisations- und Firmware-Update-Möglichkeiten bietet. Außerdem wird ein Notfallbeatmungsgerät, durch Modifizierungen, für die Nutzung neuer Features des entstandenen Systems vorbereitet.

Miloš Đorđević

Title of the paper

Development of a software and information management system for medical devices

Keywords

C++, Qt, Embedded Linux, PHP, database, cryptography

Abstract

This thesis deals with the development of a software management system for medical devices. Considering the security aspects of internet communication, a system which offers synchronisation and software update capability to the medical devices is created. In addition, an emergency respiratory device will be modified, to take advantage of a created management system.

Danksagung

Die vorlegende Bachelorthesis wurde in der Firma WEINMANN Emergency Medical Technology GmbH + Co. KG erstellt. Ich möchte mich bei dem Unternehmen für diese Möglichkeit bedanken.
Ausdrücklich bedanken möchte ich mich bei Herrn Dr. Nikolaus Voß für seine überaus engagierte und kompetente fachliche Betreuung, der sich trotz übervollen Terminkalenders dieser Aufgabe annehmen wollte.

Herrn Prof. Dr. rer. nat. Henning Dierks, von der HAW Hamburg, spreche ich meinen herzlichen Dank für die Unterschtützung, Betreuung und Erstprüfung der Arbeit.
Herrn Prof. Dr. Robert Heß, ebenfalls von der HAW Hamburg, möchte ich für die Zweitprüfung der Arbeit danken.

Meiner Ehefrau, Monika Đorđević, gebührt ein besonderes Dankeschön für ihre Geduld und Rückhalt, ohne die diese Arbeit sicherlich nicht möglich gewesen wäre.

Hamburg, im Dezember 2016
Miloš Đorđević

Inhaltsverzeichnis

Tabellenverzeichnis **7**

Abbildungsverzeichnis **8**

Listings **10**

1. Einführung **11**

2. Stand der Technik **13**
2.1. Stand der Technik allgemein 13
2.2. Stand der Technik in der Medizintechnik 15

3. Grundlagen **17**
3.1. Kryptosysteme 17
3.1.1. Symmetrisches Kryptosystem 17
3.1.2. Asymmetrisches Kryptosystem 18
3.1.3. Hybrides Kryptosystem 21
3.1.4. SSL/TLS 23
3.2. PHP 28
3.3. C++ 30
3.4. Qt 31
3.5. Datenbanksysteme 31
3.5.1. NoSQL 32
3.5.2. SQL 33

4. Analyse **34**

5. Design **38**
5.1. Serverdesign 39
5.2. Gerätedesign 45
5.3. Kommunikation und Sicherheit 49

6. Realisierung **51**
6.1. Realisierung unter Testumgebung 51

6.1.1. Realisierung des Servers 51
6.1.2. Modifizierung der Clientfirmware 60
6.2. Realisierung unter Produktivumgebung 64
6.2.1. Server-Portierung 64
6.2.2. Client-Portierung 67

7. Bewertung **72**

8. Ausblick **74**

9. Fazit **76**

Literaturverzeichnis **77**

A. Quellcode **79**

Tabellenverzeichnis

5.1. Geräteeigenschaften 42

7.1. Bewertung der Ergebnisse 72

Abbildungsverzeichnis

2.1. FOTA-System [12] 14
2.2. Aktuelle Update-Techniken mit Hilfe spezieller Software und Host-Rechner . . 16
2.3. Aktuelle Update-Techniken mit Speichermedien 16

3.1. Symmetrische Ver- bzw. Entschlüsselung basiert auf einem Schlüssel 18
3.2. Asymmetrische Verschlüsselung basiert auf einem Schlüsselpaar 19
3.3. Kommunikationssequenz bei dem Public-Key-Verfahren 20
3.4. Kommunikationssequenz bei dem hybriden Verschlüsselungsverfahren . . . 22
3.5. TLS Protokolle im Protokollstapel 23
3.6. TLS Handshake 24
3.7. Let's Encrypt Wachstum [9] 26
3.8. Mit Firefox geladene HTTPS-Seiten [9] 27
3.9. Anzahl der täglich ausgestellter Zertifikate [9] 27
3.10. Verbreitung serverseitiger Programmiersprachen für Webanwendungen [11] . 28
3.11. Beispiel für das Qt Signal-Slot Mechanismus 32

5.1. OTA-Management für Medizingeräte 38
5.2. Kommunikationsbeispiel 40
5.3. Update-Anforderung 41
5.4. ER Modell der Datenbank 43
5.5. Optimierung one-to-many und many-to-one Beziehungen 43
5.6. Referenzen zwischen den Geräte-, Geräteoption- und Kunden-Tabelle 45
5.7. Startmenü mit der Funktionskontrolle 46
5.8. Klassendiagramm der Funktionskontrolle mit Erweiterung 47
5.9. Betreibermenü 47
5.10. Klassendiagramm des Betreibermenüs mit Erweiterung 48

6.1. EER der realisierten Datenbank 53
6.2. Titel des Bildes 56
6.3. Flussdiagramm der Client-Funktion die die Synchronisation mit dem Server durchführt 62
6.4. Verbreitung der eingesetzten Webservern im Netz [11] 65
6.5. Make menuconfig Kernel 68

6.6. Make menuconfig Rootfilesystem (Buildroot) 69
6.7. Volle Sequenz bei einem Software-Update 70

Listings

6.1. MySql Komandozeilenbeispiel (neue Sitzung) 52
6.2. MySql Komandozeilenbeispiel (das Kommando *show databases*) 53
6.3. PHP Server 55
6.4. Serverfunktion *Check()* 57
6.5. Serverfunktion *insertFromSync()* 58
6.6. Serverfunktion *getNewestFwVersion()* 59
6.7. Gerätefunktion zum Senden einer HTTP-Anfrage 60
6.8. Ausgabe des Kernel-Ringpuffers 67

1. Einführung

Moderne elektronische Geräte sind ohne Software nicht mehr denkbar. In vielen Bereichen ist die Lebensdauer eines Systems mit der Erweiterung, Weiterentwicklung und Fehlerbehandlung der Software stark gekoppelt. Im Bereich der eingebetteten Systeme ist dies noch ausgeprägter. Ein aktuelles Beispiel wäre der Markt der Smartphones. Die Hersteller haben erkannt, dass sie die Kunden durch regelmäßige Aktualisierungen (Kernel-, ROM-, Sicherheits-, Service-Updates etc.) für sich gewinnen können. Für solche Rollouts werden Softwareverteilungssysteme genutzt, die eine kontrollierte Verteilung der Updatepakete ermöglichen (siehe Kapitel 2).

Im Bereich der Medizintechnik ist diese Vorgehensweise eher untypisch. Aktuelle Geräte der Notfallmedizin (Beatmung und Defibrillation) werden entweder über eine Verbindungsleitung mit dem Hostrechner/Server oder mittels eines Speichermediums aktualisiert. Obwohl einige Geräte über kabellose Schnittstellen verfügen (Bluetooth, WLAN, etc.), werden diese nicht für die Zwecke der Aktualisierung oder Service eingesetzt (siehe Kapitel 2). In der Medizintechnik ist bei Innovationen Vorsicht geboten, dennoch handelt es sich hierbei um ausgereifte Techniken und Vorgehensweisen die bei sorgfältiger Konzeptionierung auch in diesem Segment eingesetzt werden dürfen.

In dieser Arbeit wird ein Softwareverteilungs- und Informationsmanagementsystem entwickelt das:

- Geräte über Verfügbarkeit der Updates und Upgrades informiert
- Geräte mit neuer Firmware versorgt
- den Gerätezustand in der Gerätedatenbank aktuell hält
- das Gerät über Aktivierung neuer vorhandener Funktionalitäten informiert

Des weiteren wird ein Notfallmedizingerät so modifiziert und erweitert, dass es die Funktionalität des oben genannten Systems nutzen kann.

Zunächst wird der Stand der Technik in dem Segment der Softwareverteilungssysteme vorgestellt. Da die angewandten Techniken in der Medizintechnik etwas konservativ ausfallen, werden diese separat beschrieben. Im Anschluss werden einige, für die Arbeit notwendige Grundlagen vorgestellt. Die Analyse (Kapitel 4) befasst sich mit dem Anforderungsprofil

bezüglich der Funktionalität und der Anforderungen, die aus den Normen für medizinische Geräte resultieren. Danach werden, unter Beachtung der Sicherheitsstandards bei der Kommunikation, der Server und der Client modelliert. Im Kapitel 6 wird der Server und der Client umgesetzt. Damit die Realisierung beschleunigt wird, wurde diese zweistufig durchgeführt:

- in der Testumgebung
- in der Produktivumgebung

Zum Schluss werden die erreichten Ergebnisse bewertet und mögliche Erweiterungen des entwickelten Systems vorgeschlagen.

Das Ziel der Arbeit ist die Verbesserung der Software-Update-Prozesse in der Medizintechnik. Darüber hinaus soll eine Basis für ein zentralisiertes Warnsystem bei Gerätefehlern geschaffen werden, wodurch den Patienten mehr Sicherheit geboten werden kann.

2. Stand der Technik

Im Folgenden wird der Stand der Technik vorgestellt, welches allgemein eingesetzt wird und die Vorgehensweisen speziell in der Medizintechnik.

2.1. Stand der Technik allgemein

Kabellose Kommunikation ist bei aktuellen mobilen Geräten selbstverständlich. Diese wird nicht nur für die Kommunikation zwischen den Benutzern verwendet, sondern auch für die Sicherung der Benutzer- und Gerätedaten (E-Mails, Kontakte, Benutzerprofile usw.). Ein weiterer Verwendungszweck ist die kabellose Anpassung der Geräteeinstellungen, wie zum Beispiel die Accesspoint, MMS, etc. Hersteller wie Samsung, HTC, Apple und viele andere bieten eigene Software-Tools, die die Synchronisation der Geräte mit einem Rechner ermöglichen. Es werden Softwareverteilungssysteme von unterschiedlichen Herstellern angeboten. Unter anderem, sind speziell für Mobilgeräte folgende Verteilungssysteme zu finden:

- MDS Suite - BlackBerry
- Mobile Device Manager - Microsoft
- Microsoft Intune - Microsoft

Microsoft Intune bietet Unterstützung für iOS, Mac OS, Windows Mobile, Windows und Android.

Diese Methode ist unter den Namen OTA[1] bekannt. Mit einer Abwandlung der OTA ist es möglich Geräte-Softwareupdates durchzuführen. Diese Methode wird FOTA[2] genannt. Analog zu der FOTA existieren auch

- SOTA - **S**oftware-**O**ver-**t**he-**A**ir
- OAD - **O**ver-the-**a**ir (firmware) **d**ownloads
- OTAP - **O**ver-**t**he-**a**ir **p**rogramming

[1] **O**ver-**t**he-**A**ir
[2] **F**irmware-**O**ver-**t**he-**A**ir

- OTAU - **O**ver-**t**he-**a**ir **u**pdate

Aktuelle Geräte aus dem Mobile-Segment unterstützen FOTA und bieten dem Kunden die Möglichkeit, Software-Updates und -Upgrades durchzuführen. Betriebssysteme aus der Familie Windows, Unix, Linux etc., unterstützen ebenfalls diese Funktionalität.

Diese Technologie ist durch einige Patente geschützt. Viele dieser Patente gehören der Firma Bitfone Corporation, einer der ersten, die sich mit dem Thema FOTA beschäftigte.

Da das Prinzip viel Komfort, Sicherheit und Zeit-/Aufwandsersparnisse bietet, wird dieses Feature von einigen Herstellern angeboten. Bei einen handelt es sich um Eigenentwicklungen, bei den anderen um eingekaufte Module, die diese Funktionalität erfüllen. Hersteller wie Wind River, Zeeis, Red Bend, ITK Engineering und weitere bieten Lösungen an, die in denn Segment Automotive, Luft und Raumfahrt, Medizintechnik, Mobile, Embedded etc. eingesetzt werden können. In der Abbildung 2.1 sind die Hauptbestandteile der FOTA dargestellt. Typischerweise umfasst ein FOTA-System drei Hauptbestandteile:

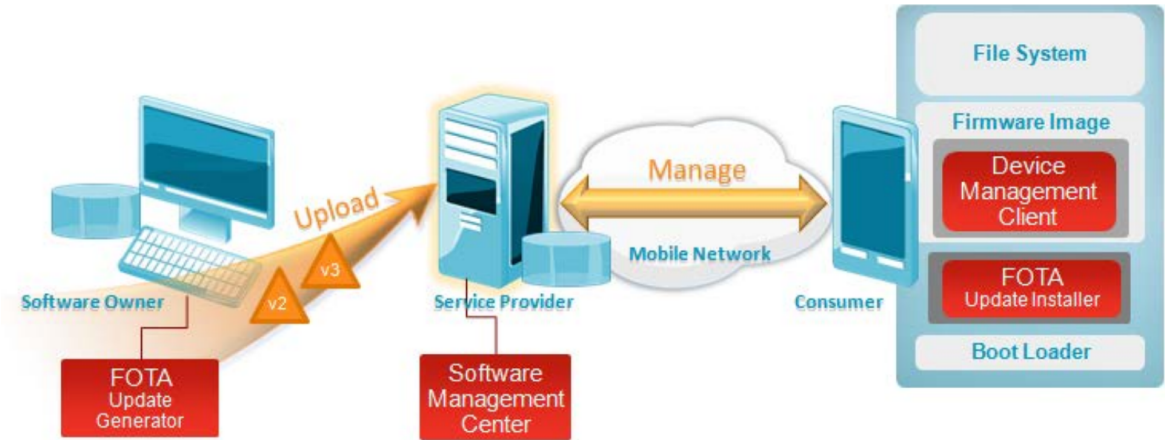


Abbildung 2.1.: FOTA-System [12]

- FOTA Update-Generator
- Software Managementsystem inklusive Kommunikationsprotokoll
- FOTA Update Installer

Der Update-Generator generiert eine Binärdatei, die auf das Zielgerät aufgespielt werden kann. Abhängig von dem Grad der Änderungen zu der aktuellen Firmware erstellt der Generator ein sog. Delta-Paket. Die Delta-Pakete beinhalten nicht die komplette Firmware, sondern nur die neuen oder geänderten Teile der Firmware. Auf diese Weise entstehen extrem kompakte Update-Dateien die effizient zum Client übertragen werden können.

Das Software Managementsystem verschickt die generierten Delta-Pakete zu den Clients. Dies geschieht unter Verwendung von speziell dafür entwickelten Protokollen. Das OMA DM[3] Standard ist das am häufigsten genutzte Protokoll, welches für die Kommunikation in dem Mobile-Sektor optimiert wurde.

Clientseitig existiert der Update Installer, der für die Updates und Upgrades zuständig ist.

2.2. Stand der Technik in der Medizintechnik

In der Medizintechnik wird FOTA nicht verwendet. Im folgenden werden System-Update-Techniken der verschiedenen Herstellern vorgestellt.

Der Medizintechnikhersteller ZOLL setzt für die Geräte-Updates und -Management ausschließlich das eigene Tool, ***Z**OOL **A**dministration **S**oftware* (ZAS) bzw. *RescueNet Code Review Software*, ein. Die Gerätekommunikation erfolgt über einen Kabel oder Infrarot-Modul. [20]

Die Flaggschiffe im Segment Monitoring/Defibrillation (X Serie) desselben Herstellers, verfügen über vielfältige Kommunikationsmöglichkeiten (Bluetooth, WiFi, USB-Mobilmodem und Ethernet). Diese werden für die Patientendatenübermittlung, aber nicht für die Firmwareupdates genutzt. [21]

Die Notfallmedizinprodukte des Herstellers Weinmann Emergency verfolgen einen anderen Ansatz bezüglich der Firmwareupdates. Sowohl im Segment der Defibrillation als auch in der Notfallbeatmung werden Weinmann-Geräte über ein externes Speichermedium (USB-Stick, SD-Karte) ohne zusätzlichen Host-Rechner mit Software-Updates versorgt. Der Updatevorgang wird am Gerät selbst gestartet. [18]

Die verwendete Techniken bei den Updates der Medizingeräte können in zwei Gruppen eingegliedert werden:

- Host + Kabel (oder ein Kommunikationsmodul anderer Art)
- Speichermedium

In beiden Fällen muss die Firmwaredatei von dem Hersteller bezogen werden. Abbildungen 2.2 und 2.3 visualisieren aktuelle Update-Techniken in dem Segment der Medizintechnikgeräte.

[3] **O**pen **M**obile **A**liance **D**evice **M**anagement

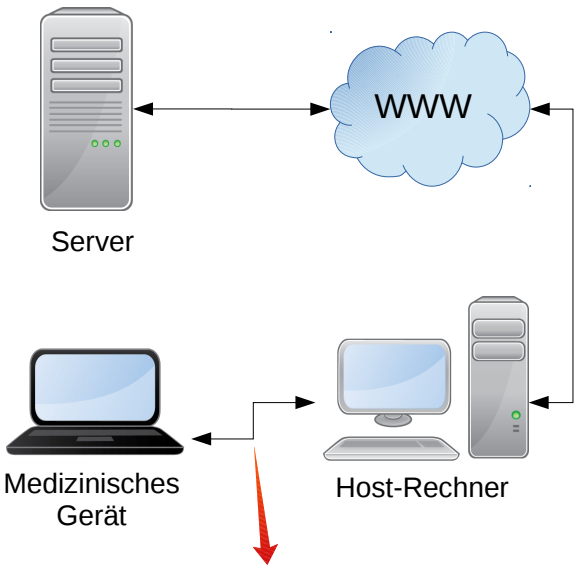


Abbildung 2.2.: Aktuelle Update-Techniken mit Hilfe spezieller Software und Host-Rechner

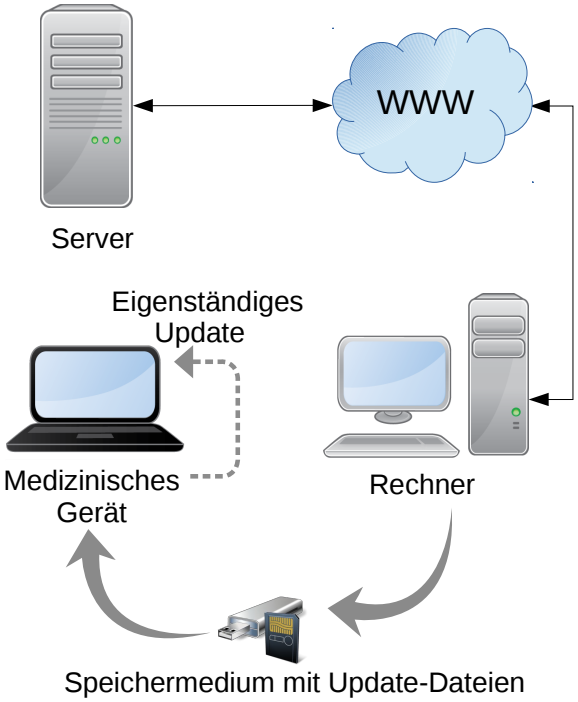


Abbildung 2.3.: Aktuelle Update-Techniken mit Speichermedien

3. Grundlagen

Diese Arbeit erfordert Kenntnisse in der Entwicklung der Server- und Client-Applikationen, außerdem die Programmierung von Datenbanken und deren Anbindungen, als auch die Netzwerkkommunikation und Kommunikationssicherheit (Kryptosysteme), sowie Kenntnisse über Linux-Kernel und Root-Filesysteme in der Embedded-Sparte. Da das Spektrum des Vorwissens etwas breiter ausfällt, werden relevante Themen kurz vorgestellt.

3.1. Kryptosysteme

Kryptosysteme dienen einer sicheren Informationsübertragung. Die Idee dabei besteht darin, dass die zu übertragende Information nicht im Klartext, sondern erst verschlüsselt und dann dem Kommunikationspartner übermittelt wird. Hierfür wurden verschiedene Verschlüsselungsverfahren entwickelt, die Vor- und Nachteile aufweisen. Diese können wie folglich klassifiziert werden:

- symmetrische Verschlüsselung
- asymmetrische Verschlüsselung

Da keines der beiden Verfahren aufgrund der mangelhaften Performance/Sicherheit, universal einsetzbar ist, werden aktuell hybride Verschlüsselungsverfahren eingesetzt. Diese entstehen durch Kombination des symmetrischen und asymmetrischen Verschlüsselungsverfahren. Der heutige Standard der sicheren Informationsübertragung heißt **T**ransport **L**ayer **S**ecurity, das auf der hybriden Verschlüsselung basiert. Folgende Abschnitte beschreiben die Grundbausteine des TLS-Protokolls.

3.1.1. Symmetrisches Kryptosystem

Symmetrische Verschlüsselung (auch Secret-Key-, Shared-Secret- oder Private-Key-Verfahren genannt) ist die älteste Verschlüsselungsart und wurde schon mehrere Jahrtausende vor Christi angewandt. Sie basiert auf einem Schlüssel, der sowohl für die Ver- als auch für die Entschlüsselung verwendet wird (Abbildung 3.1). Je nach Größe der zu ver-

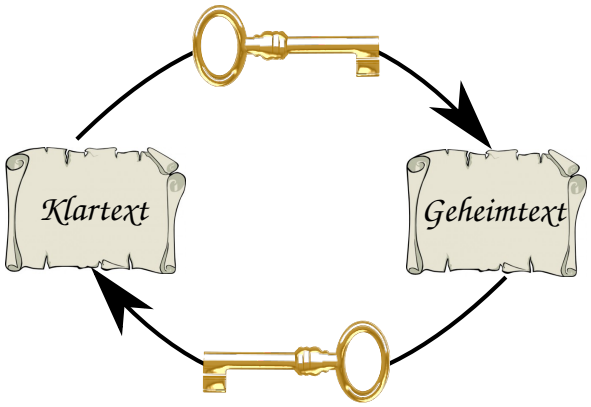


Abbildung 3.1.: Symmetrische Ver- bzw. Entschlüsselung basiert auf einem Schlüssel

schlüsselnden Information wird das Block- bzw. Stromchiffren basierte Verfahren eingesetzt. Bei dem Stromchiffren Verfahren wird, wie der Name verrät, Zeichen-basiert verschlüsselt. Es wird schnell klar, dass sich so ein Verfahren für große Dateien aufgrund der schlechten Performance nur bedingt eignet. Aus diesem Grund existiert noch das Blockchiffren-basierte symmetrische Verschlüsselungsverfahren. Dieses verschlüsselt die Information Block-basiert, das bedeutet mehrere Zeichen in einem Schritt, wobei die Blockgröße innerhalb des Geheimtextes konstant bleiben muss. Damit auch Daten, die die Blockgröße übersteigen, verschlüsselt werden können, wird die Blockchiffre mit dem Betriebsmodus kombiniert. Eine der Betriebsmodi wäre, dass die Information in mehrere Teile aufgesplittet und durch Padding auf die Blocklänge gebracht wird. Die Sicherheit der symmetrischen Verschlüsselung ist von der Geheimhaltung des Schlüssels abhängig. Da bei der Ver- bzw. Entschlüsselung ein und derselbe Schlüssel verwendet wird, ist es notwendig, dem Empfänger neben der Nachricht auch den Schlüssel zu überreichen. Demzufolge ist für den Schlüsseltransport ein sicherer Kanal erforderlich. Falls so ein Kanal existiert, könnte die Verschlüsselung entfallen und die Nachricht im Klartext gesendet werden. Das ungelöste Schlüsselverteilungsproblem ist der größte Nachteil des symmetrischen Kryptosystems. Das bekannteste und standardisierte Kryptosystem aus der Familie symmetrischer Kryptographie ist die Blockchiffre **A**dvanced **E**ncryption **S**tandard.

3.1.2. Asymmetrisches Kryptosystem

Ein sehr junges Gebiet der Kryptographie ist die asymmetrische Verschlüsselung (auch unter den Namen Public-Key-Verfahren bekannt). Im Gegensatz zu der symmetrischen Methode existiert bei der asymmetrischen Methode ein Schlüsselpaar. Dies hat zu Folge, dass zwei Schlüssel mit unterschiedlichen Funktionalitäten benötigt werden. Das Schlüsselpaar besteht aus einem öffentlichen (public-key) und einem privaten (private-key) Schlüssel. Mit

Hilfe des öffentlichen Schlüssels können Informationen nur verschlüsselt, aber nicht mehr entschlüsselt werden. Diese Chiffre kann nur durch Anwendung des privaten Schlüssels und nur auf diese Weise entschlüsselt werden. Der Private Schlüssel ist ausschließlich im Besitz des Empfängers und es besteht kein Grund, diesen bekannt zu machen. Jeder Sender ist in der Lage, mit dem öffentlichen Schlüssel des Empfängers die zu sendende Information zu verschlüsseln, aber nur der Inhaber des privaten Schlüssels, also der Empfänger, kann diese Nachricht in Klartext überführen (Abbildung 3.2). Eine vollständige Kommunikationssequenz ist der Abbildung 3.3 zu entnehmen.

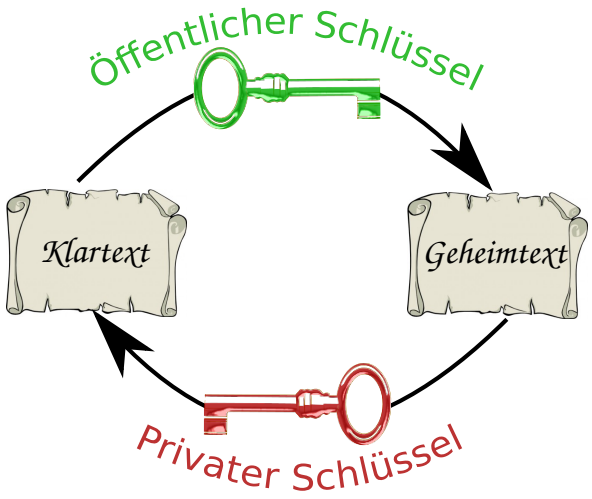


Abbildung 3.2.: Asymmetrische Verschlüsselung basiert auf einem Schlüsselpaar

Es ist erkennbar, dass bei dem asymmetrischen Verfahren sichere Kommunikationskanäle entfallen, da die Übertragung des Geheimschlüssels nicht stattfindet. Das Public-Key-Verfahren löst das Schlüsselverteilungsproblem und ist somit dem Private-Key-Verfahren, bezüglich der Sicherheit überlegen. Die Sicherheit des asymmetrischen Verfahrens beruht auf sog. Einwegfunktionen. Diese haben die Eigenschaft sich leicht berechnen zu lassen, dafür sind sie unheimlich schwer umzukehren.
Beispiel: Aus den Primzahlen 37 und 211 lässt sich das Produkt leicht berechnen, aber aus dem Produkt lassen sich, durch Faktorisieren, schwer die ursprünglichen Primzahlen ermitteln.

$$Multiplizieren(37, 211) \rightarrow 7807$$

$$Faktorisieren(7807) \rightarrow ?$$

Da es (noch) keine Verfahren und Algorithmen zur schnellen Faktorisierung gibt, zeichnet sich das Public-Key-Verfahren durch sehr hohe Sicherheit aus. Allerdings ist die asymmetrische Verschlüsselungstechnik viel langsamer als die symmetrische.

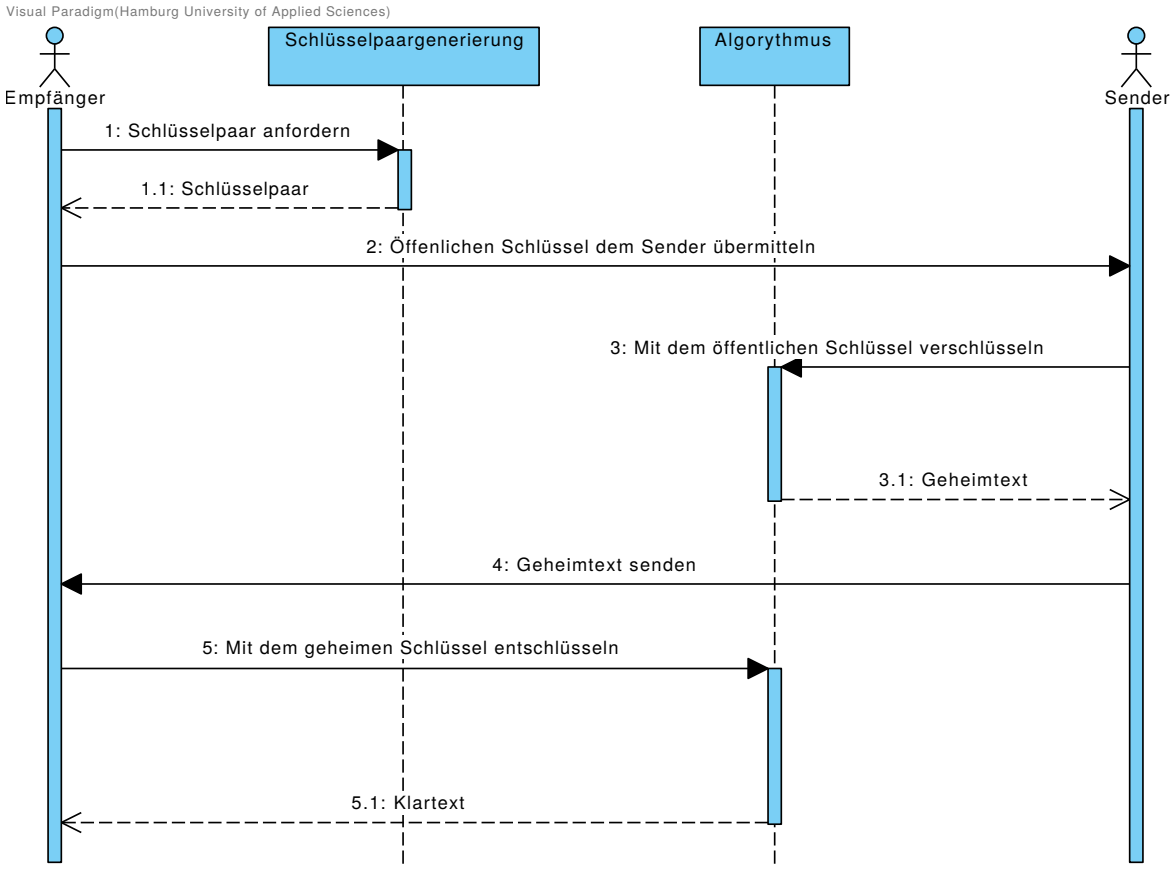


Abbildung 3.3.: Kommunikationssequenz bei dem Public-Key-Verfahren

3.1.3. Hybrides Kryptosystem

Eine Kombination aus symmetrischer und asymmetrischer Verschlüsselung wird hybride Verschlüsselung genannt. Diese Fusion löst das Schlüsselverteilungsproblem der symmetrischen Verschlüsselung und das Performanceproblem der asymmetrischen Verschlüsselung. Die Kommunikation startet wie bei asymmetrischer Verschlüsselung mit einer Schlüsselpaar-Generierung der Kommunikationsteilnehmer. Danach werden öffentliche Schlüssel ausgetauscht, wobei auf die Authentifizierung des Kommunikationspartner geachtet werden muss. Der Sender generiert, neben den Schlüsselpaar noch einen zufälligen, symmetrischen Schlüssel, der auch Session-Key genannt wird. Mit dem Session-Key werden die Nutzdaten verschlüsselt und der Session-Key selbst wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Nun kann die symmetrisch verschlüsselte Nachricht (Geheimtext) und der asymmetrisch verschlüsselte Session-Schlüssel dem Kommunikationspartner übermittelt werden. Der Empfänger ist in der Lage den Session-Schlüssel mit seinem Geheimschlüssel zu entschlüsseln und anschließend mit dem entschlüsselten Session-Schlüssel die Nutzdaten in Klartext zu überführen. In der Abbildung 3.4 ist eine Kommunikationssequenz bei dem hybriden Verschlüsselungsverfahren dargestellt. In der Regel sind die Nutzdaten relativ groß. Da diese symmetrisch verschlüsselt werden, sorgen die Vorteile des Private-Key-Verfahrens für eine gute Performance. Der Session-Key ist klein genug, so dass dieser mit dem langsamen Algorithmus der asymmetrischen Verschlüsselung verarbeitet werden kann, ohne dass diese Rechenzeit ins Gewicht fällt. Bei dem Session-Key handelt es sich um einen Schlüssel, der nur für eine Sitzung gültig ist. Dies hat zur Folge, dass selbst wenn der Session-Key bekannt wird, nur die Sicherheit der aktuellen Sitzung gefährdet ist. Da der Sitzungsschlüssel immer wieder neu generiert wird, muss die Generierung der Private-Public-Key-Paare nicht bei jeder Sitzung durchgeführt werden, sodass die Kommunikationsteilnehmer in der Regel die Schlüssel über längere Zeit nutzen können.

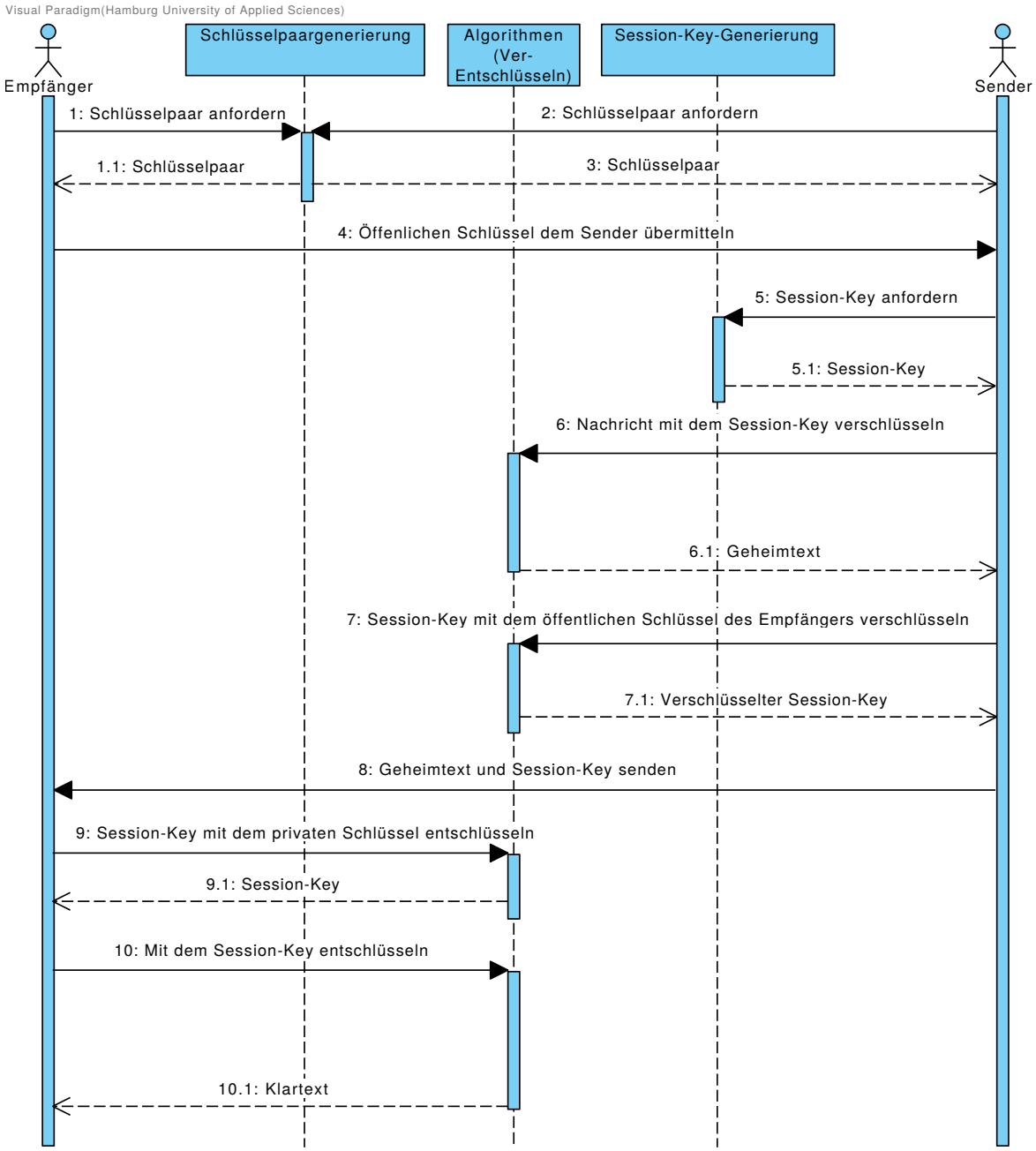


Abbildung 3.4.: Kommunikationssequenz bei dem hybriden Verschlüsselungsverfahren

3.1.4. SSL/TLS

Transport **L**ayer **S**ecurity ist ein hybrides Verschlüsselungsprotokoll, welches zur sicheren Internetkommunikation eingesetzt wird. Entstanden ist es im Jahr 1995 unter dem Namen **S**ecure **S**ockets **L**ayer. Ab der Version 3.1 (im Jahr 2006) in TLS umbenannt, beginnend mit der Version 1.0. Nichtsdestotrotz ist das Verschlüsselungsprotokoll unter den Namen SSL weitläufig bekannt. Weitgehend wird die Verschlüsselung mit HTTPS[1] eingesetzt und dient zur Abwehr verschiedener Angriffe (men-in-the-middle, Phishing, Replay etc.). Durch Kombination zweier unterschiedlicher Verschlüsselungsverfahren (asymmetrisch und symmetrisch) entstehen hybride Verschlüsselungen. Die Idee dabei besteht darin, die Vorteile des jeweiligen Verfahrens beizubehalten und dadurch die Sicherheit und Effizienz des neu entstandenen Verfahrens zu steigern (siehe Abschnitt 3.1.3).
Aktuell wird TLS-Verschlüsselung, bestehend aus RSA[2] und AES[3], für die sichere Internetkommunikation eingesetzt. Das TLS-Protokoll ist im OSI-Modell in der Sitzungsschicht (Schicht 5) angesiedelt. Es besteht aus zwei weiteren Schichten (Abbildung 3.5), wobei das TLS Record Protocol direkt auf die Transportschicht (Schicht 4) des OSI-Modells aufsetzt und zur Absicherung der Verbindung dient. Bevor die eigentliche Kommunikation stattfindet,

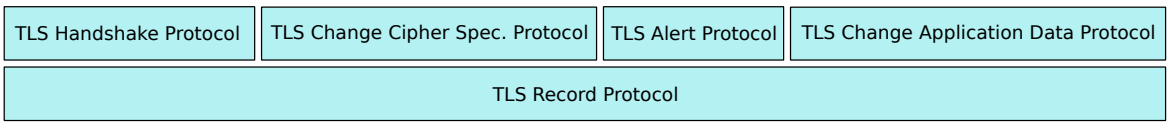


Abbildung 3.5.: TLS Protokolle im Protokollstapel

sorgt das Handshake Protokoll für die Authentifizierung der Kommunikationspartner (sowohl des Clients als auch des Servers) und für das Aushandeln der zu benutzenden Verschlüsselungsalgorithmen und Schlüssel. Die ganze Sequenz kann in vier Untersequenzen unterteilt werden (Abbildung 3.6).

Die erste Phase, und damit auch der TLS Handshake, beginnt mit einer „hello_Client“-Nachricht, die der Client dem Server verschickt. Diese besteht aus folgenden Parametern:

- die höchste Version des TLS-Protokolls, die von dem Client unterstützt wird
- eine 32 Byte lange Zufallszahl bestehend aus dem Zeitstempel (4 Byte) und der zuvor generierten Zufallszahl (28 Byte). Diese Zufallsinformation wird in der Phase 3 für die Generierung des „pre-master-secret“ benötigt.
- die Identifikationsnummer der Sitzung (Session-ID)

[1] **H**yper**T**ext **T**ransfer **P**rotocol **S**ecure ist ein abhörsicherer Internetkommunikationsprotokoll.
[2] **R**ivest, **S**hamir, **A**dleman ist ein asymmetrisches Verschlüsselungsverfahren.
[3] **A**dvanced **E**ncryption **S**tandard ist ein standardisiertes symmetrisches Verschlüsselungsverfahren.

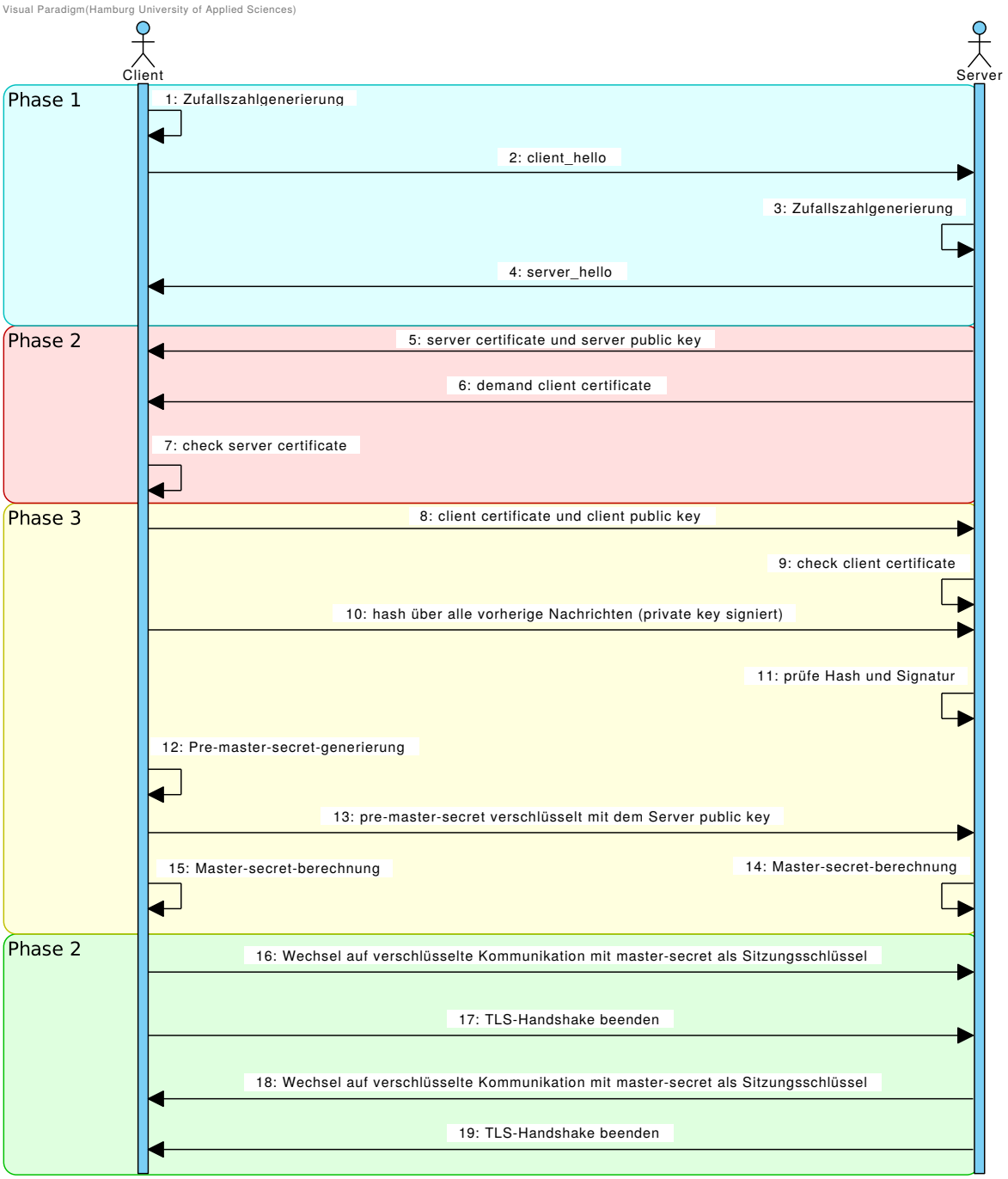


Abbildung 3.6.: TLS Handshake

- die zu verwendende Sammlung kryptographischer Verfahren (Cipher Suite). Jede Cipher Suite identifiziert eine Kombination aus vier Algorithmen die folgende Funktionalitäten abdecken:
  - Schlüsselaustausch (z.B. RSA)
  - Authentifizierung (z.B. RSA)
  - Hashfunktion (z.B MD5)
  - Verschlüsselung (z.B. AES)
- den gewünschten FQDN[4] für die Unterstützung von Server Name Indication. Dieser Part ist Optional und wird nicht näher erläutert.

Auf diese Nachricht antwortet der Server mit seiner „hello_Server“ Nachricht, womit die erste Phase des Handshakes beendet wird.
In der zweiten Phase identifiziert sich der Server gegenüber dem Client in dem das Server-Zertifikat (X.509v3-Zertifikat) übermittelt wird. Unter anderem ist in dem Zertifikat der öffentliche Schlüssel des Servers enthalten. Des weiteren stellt der Server eine Anfrage bezüglich des Client-Zertifikates an und wartet auf dessen Übermittlung. Die Phase wird mit der clientseitigen Prüfung des Server-Zertifikates beendet. Falls die Prüfung fehlschlägt, wird die Verbindung abgebrochen.
Zum Beginn der dritten Phase identifiziert sich der Client mit seinem Zertifikat, welches serverseitig überprüft wird.
Bei fehlerhaftem Client-Zertifikat wird die Verbindung abgebrochen, anderenfalls schickt der Client einen Hash-Wert, der sich aus allen vorherigen Nachrichten berechnen lässt und der mit dem privaten Schlüssel des Client signiert ist. Diese Nachricht wird ebenfalls von dem Server auf ihre Korrektheit geprüft. Wenn bis zu diesem Zeitpunkt, keine Fehler auftauchen und damit die Kommunikation immer noch steht, generiert der Client einen „Pre-Master-Secret“, verschlüsselt diesen mit dem öffentlichen Schlüssel des Servers und verschickt ihn. Da die Information mit dem öffentlichen Schlüssel des Servers chiffriert ist, kann diese nur von dem Server dechiffriert werden. Aus dem „Pre-Master-Secret“ können nun sowohl der Server als auch der Client den „Master-Secret“ berechnen. Das berechnete Geheimnis ist nichts anderes als der symmetrische Sitzungsschlüssel, mit dem von nun an die Nachrichten ver- bzw. entschlüsselt werden. In der vierten Phase wird die Kommunikationsart auf verschlüsselt umgestellt und der TLS-Handshake wird beendet.

Die Verschlüsselung der Internetkommunikation hat in den letzten Jahren einen stark zugenommen. Beginnend mit der Verschlüsselung der Google-Suchanfragen (in der BETA Version) im Jahr 2010, über YouTube Verschlüsselung (2011), bis zu standardmäßigen Verschlüsselung der Google-Suchmaschine (2013) hat sich nicht viel auf dem Gebiet getan.

[4] **F**ully **Q**ualified **D**omain **N**ame ist eine Bezeichnung für vollständige Domänennamen.

Die meisten Server waren immer noch zertifikatslos und hatten auch kein Interesse, dies zu ändern. Erst als Google im Jahr 2014 angefangen hat, HTTPS-Seiten in der Ergebnisliste höher zu ranken, fing die massive Umstellung an. Seitdem Let's Encrypt kostenlose Zertifikatsausstellung anbietet (Ende 2015), ist die Verbreitung des HTTPS-Protokolls massiv gestiegen. Abbildungen 3.7, 3.8 und 3.9 dürften für einen Überblick sorgen.

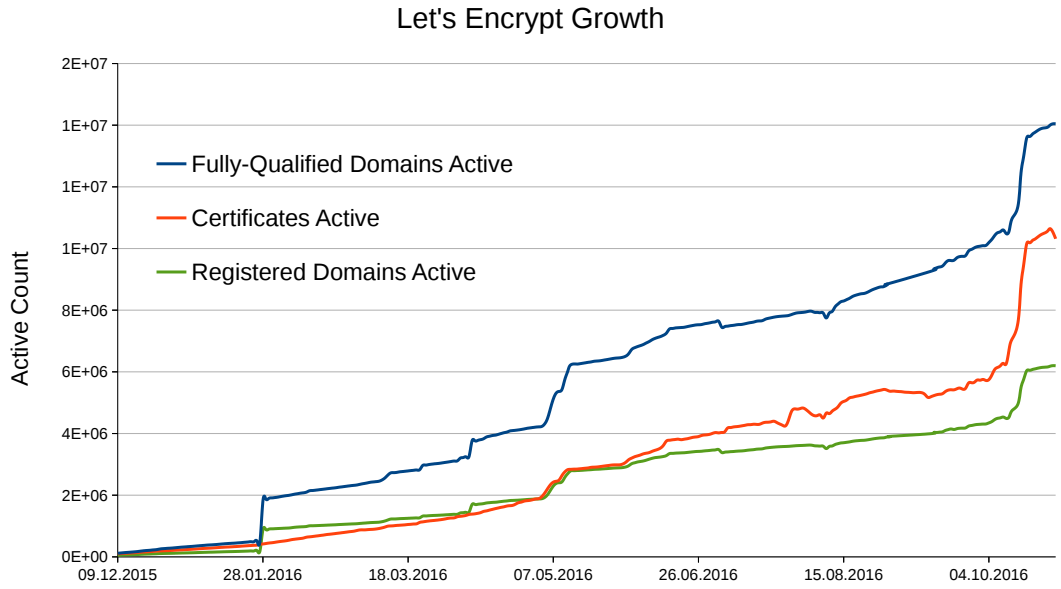


Abbildung 3.7.: Let's Encrypt Wachstum [9]

Die SSL-Zertifizierung, die früher eine Seltenheit war, ist durch die Machtdemonstration des Internetgiganten Google zum Quasi-Standard geworden. Aus den Statistiken lässt sich schließen, dass in der Zukunft die Internetkommunikation nur auf dem verschlüsselten Wege stattfinden wird.

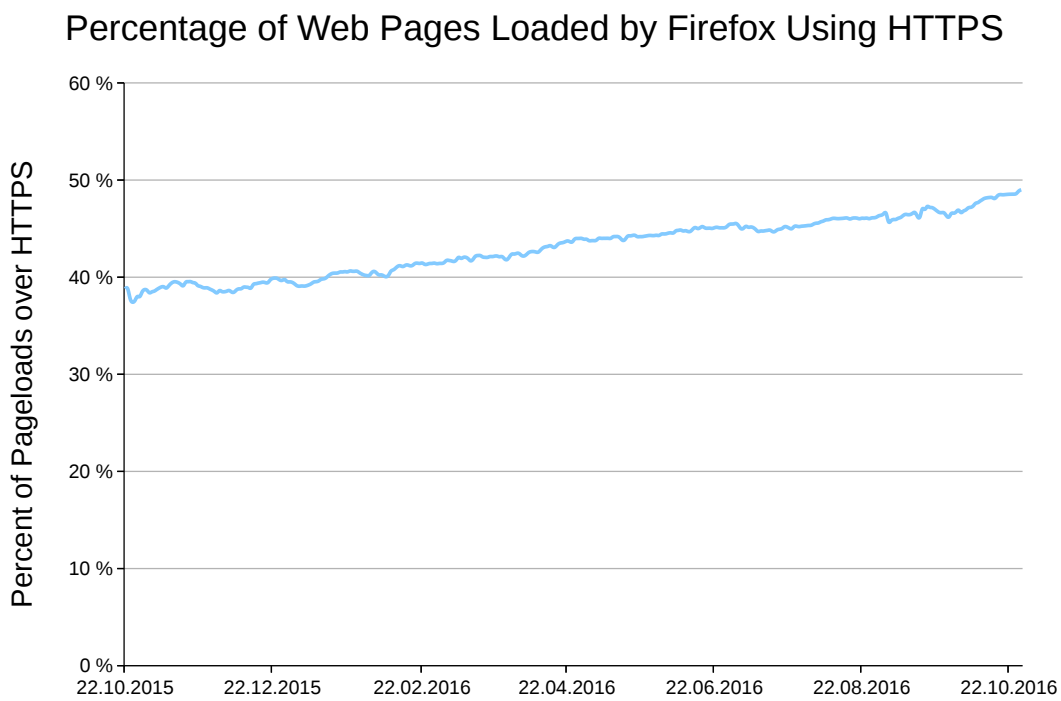


Abbildung 3.8.: Mit Firefox geladene HTTPS-Seiten [9]

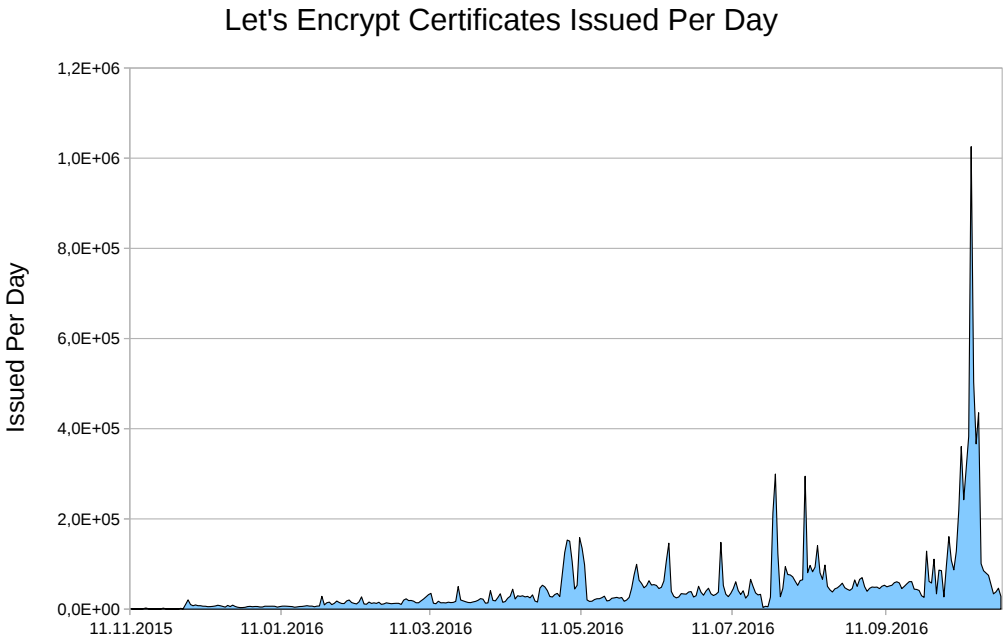


Abbildung 3.9.: Anzahl der täglich ausgestellter Zertifikate [9]

3.2. PHP

PHP (**H**ypertext**P**reprocessor, ursprünglich **P**ersonal **H**ome **P**age Tools) ist eine relativ junge Skriptsprache (1995), deren Syntax den Programmiersprachen C und Perl ähnelt. Hauptsächlich wird diese zur Fabrikation dynamischer Webseiten und/oder Teilen der Webseiten, aber auch für die Programmierung von Webanwendungen benutzt. Sehr wichtige und nützliche Eigenschaften dieser Programmiersprache zeichnen sich durch Unterstützung der Datenbank- und Internetprotokollanbindungen aus. Da viele Bibliotheken und Frameworks existieren, ist die Entwicklung relativ schnell und unkompliziert. Insbesondere wird PHP sehr oft als serverseitige Programmiersprache eingesetzt. Die enorme Verbreitung der Programmiersprache PHP im Websegment ist der Abbildung 3.10 zu entnehmen. Dieser Information zur Folge ist PHP aktuell die mit Abstand am häufigsten eingesetzte Programmiersprache für Webanwendungen. Dabei sind die Versionen der Sprache unterschiedlich stark vertreten. Die Version 5 wird am häufigsten (97%), die neueste Version 7 noch sehr selten (2%) eingesetzt. Dabei muss erwähnt werden, dass in den letzten 12 Monaten der Einsatz der Version 5 um ca 2% abgenommen hat, wogegen der Einsatz der Version 7 stetig zunimmt (1,9%). Da PHP 6 nie erschienen ist und dessen Entwicklung eingestellt wurde, wurden die geplanten Features der Version 6 als Erweiterungen der Version 5 eingepflegt. Erst 10 Jahre später (2014) ist die Version 7 erschienen, die eine mögliche Erklärung für die Verbreitungsdiskrepanz zwischen PHP 5 und PHP 7 ist.

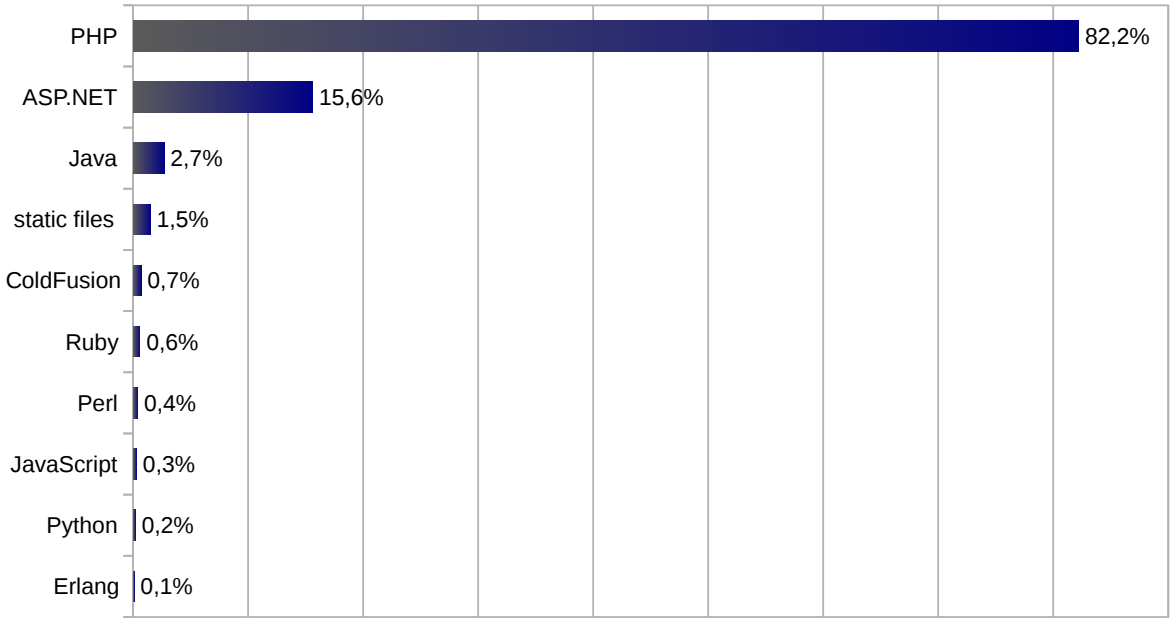


Abbildung 3.10.: Verbreitung serverseitiger Programmiersprachen für Webanwendungen [11]

Der PHP-Code wird nicht client- sondern serverseitig ausgeführt. Da es sich um eine Skriptsprache handelt, werden Quelltexte von dem PHP-Interpreter verarbeitet und dessen Ausgabe an den Client verschickt. Als Ausgabe des Interpreters sind nicht nur HTML- sondern auch PDF- oder Bilddateien möglich. Der Interpreter wird von einem Server-Dienst oder Server-Daemon (z.B. Apache[5], IIS[6]) ausgeführt, der durch ein Webinterface (z.B. CGI[7] oder ISAPI[8]) kommunizieren kann. Da der Interpreter von dem Server ausgeführt wird, wird dieser auch durch Aufrufe der PHP-Seiten belastet. Zudem werden die Seiten bei jedem Aufruf erneut abgearbeitet, was sich auf die Latenz und Performance des Servers negativ auswirkt. Aus diesem Grund werden aufgerufene Seiten zwischengespeichert, sodass der nächste Aufruf beschleunigt wird. Diese Vorgehensweise bildet das Fundament der PHP-Beschleuniger[9] und ist auch unter dem Namen Bytecode-Caching bekannt.

Ein nicht seltener Fall in der Programmierung der Webserver ist das Einbinden einer Datenbank. Das bedeutet, dass ein Bündel bestehend aus dem Betriebssystem, Webserver, Datenbank und Programmiersprache benötigt wird. Kombinationen, bei denen es sich um einen der Betriebssysteme Linux, Windows oder Mac OS X und Apache-Webserver, MySQL-Datenbank, PHP-Programmiersprache handelt, haben eigene Namen bekommen:

- LAMP - **L**inux **A**pache **M**ySQL **P**HP
- WAMP - **W**indows **A**pache **M**ySQL **P**HP
- MAMP - **M**ac OS X **A**pache **M**ySQL **P**HP

Für die Entwicklungszwecke sind fertig zusammengestellte Konfigurationen für alle drei Betriebssystemarten unter den XAMPP bekannt und lassen sich aus den Paketquellen einfach beziehen. Diese beinhaltet Apache, MariaDB, PHP und Perl.

[5] Apache HTTP Server ist der meistgenutzte opensource Webserver im Internet. Dabei unterstützt die Software eine Vielzahl von Betriebssystemen (Unix, Linux, Win32, NetWare etc.).

[6] **I**nternet **I**nformation **S**ervices ist eine Dienstplattform des Unternehmens Microsoft, die einen Netzwerkdateieinzugriff ermöglicht. Es werden lediglich hauseigene Betriebssysteme unterstützt.

[7] **C**ommon **G**ateway **I**nterface ist ein standardisiertes Dateninterface eines Webservers welches den Datenaustausch zwischen den Server und externer Softwareanwendungen ermöglicht.

[8] **I**nternet **S**erver **A**pplication **P**rogramming **I**nterface ist eine Microsoft Windows basierte Server-Dienste-Sammlung die von z.B. IIS verwendet wird.

[9] Anwendungen, die die Ausführung der PHP-Skripte beschleunigen können.

3.3. C++

C++[10] ist seit 1998 eine ISO genormte Programmiersprache, die als Erweiterung der Programmiersprache C entwickelt wurde. Im Gegensatz zu C handelt es sich bei C++ um eine objektorientierte Programmiersprache, die sich durch eine sehr große Anzahl an Bibliotheken auszeichnet (aktuell sind auf SorceForge über 3000 C++ Bibliotheken verfügbar).

Da das Kompilat der Sprache direkt von Maschinen gelesen werden kann, ist sie schneller als Skriptsprachen die immer einen Interpreter benötigen. Bei der Programmiersprache C++ wurde die Performance dem Komfort vorgezogen. Auch die bekannte STL[11] wurde mit dem Augenmerk „Effizienz“ designed. Außerdem sind die C++-Compiler für sehr gute Optimierung bekannt. Insbesondere können diese hervorragend „inlinen“[12].
Auf Grund der hohen Performance, wird sie oft bei der Entwicklung von Betriebssystemen, Treiber, Virtuellen-Systemen, Programmierung im Embedded-Bereich, Signalverarbeitung mit DSPs etc., eingesetzt. Der bekannteste C++ Compiler für Windows Betriebssysteme ist in der Microsoft Visual Studio IDE enthaltene C++ Compiler. Für (fast) alle andere Betriebssysteme (inklusive Windows) gibt es den g++ Compiler. Hierbei handelt es sich um einen der ältesten und verbreitetsten C++ Compiler. Der große Vorteil besteht darin, dass der g++ Compiler auf vielen Systemen vorhanden ist und somit die Entwicklung mit C++ keine aufwendige Vorbereitung benötigt. Im Gegensatz zu Java wird in C++ nativ keine Garbage Collection unterstützt, es ist jedoch möglich, diesen durch Erweiterungen nachzurüsten [3]).

Neben den C-Funktionalitäten bietet C++ in der ersten Version weitere Datentypen, Referenzen, Mehrfachvererbung, virtuelle Funktionen, Templates, Überladung von Funktionen und Operatoren, Namespaces, dynamische Speicherverwaltung, Inlinefunktionen usw. an. Um aktuelle Entwicklungen genügen Möglichkeiten zu bieten, wird die Sprache weiter entwickelt, sodass nach dem Jahr 2005 zwei weitere Standards veröffentlicht wurden. In der Version C++11 wurden anonyme Funktionen (Lambda-Funktionen), streng typisierte Enum, Auto-Datentyps, etc. nachgerüstet. In der Version C++14 wurden größtenteils die neuen Funktionalitäten aus den C++11 erweitert. Aktuell wird an dem C++17 Standard gearbeitet, welcher nahezu „feature-complete“ ist. Dieser wird 2017 veröffentlicht.

[10] ++ wird als der Inkrement-Operator benutzt, sodass C++ soviel wie „Inkrementiertes C“ bedeutet.
[11] **S**tandard **T**emplate **L**ibrary
[12] Der Inhalt einer Inline-Funktion wird vom Compiler an die Stelle des Aufrufs hin kopiert. Hierdurch wird die Programmausführung beschleunigt, da der Hin-/Rücksprung beim Funktionsaufruf eingespart wird.

3.4. Qt

Qt ist eine Klassenbibliothek, die eine sehr große Anzahl an Funktionalitäten für die Programmierung graphischer Benutzeroberflächen bietet. Diese ist plattformübergreifend und neben den Programmiersprachen Python, Ruby, C#, Java, PHP etc., auch für die Programmiersprache C++ verfügbar. Sie bietet Funktionen zur Internationalisierung, Datenbankanbindungen, Netzwerkkommunikation etc., aber auch Module für den Hardwarezugriff im Segment der mobilen Geräte (Smartphones, Tablets, Navigationssysteme usw.) wie z.B. Bluetooth, NFC, Sensoren etc. an. Der Qt-Präprozessor (der sog. MOC[13]) erzeugt einen C++-Standard konformen Code, der von den üblichen C++ Compilern (z.B. g++) übersetzt werden kann. Auf diese Weise wird die C++ Standard-Sprache um Qt-Funktionalität erweitert. Eine der zentralen Funktionalitäten der Qt-Bibliothek ist das sog. Signal-Slot-Mechanismus. Dieses ermöglicht die Kommunikation unter Objekten. Signale stellen eine Art Nachricht da, die von Objekten gesendet werden können. Optional können der Nachricht auch Attribute mitgegeben werden. Die Empfänger dieser Nachrichten werden Slots genannt. Diese sind mit „normalen“ Methoden vergleichbar und können auf die gleiche Art und Weise eingesetzt werden. Die ganze Funktionalität ist mit dem Observer-Pattern vergleichbar. Hierbei handelt es sich um ein Entwurfsmuster aus der Familie der sog. GoF[14]-Muster. In der Abbildung 3.11 ist eine Inter-Objekt-Kommunikation mittels Signal-Slot dargestellt. Das Objekt 1 kommuniziert mit den Objekten 2 und 4 über Signale 1 und 2. Das Signal 1 des Objekts 1 ist sowohl mit dem Slot 1 als auch mit dem Slot 2 des Objekts 2 gekoppelt. Wann immer das Objekt 1 Signal 1 aussendet, werden Slots 1 und 2 des Objekts 2 ausgeführt. In der gleichen Art und Weise kommunizierten die Objekte 1 und 3 mit dem Objekt 4. Das Signal-Slot-Mechanismus stellt eine Alternative zu der Callback-Technik. Qt-Widgets haben schon vordefinierte Signale, die benutzt werden können. Ein typisches Beispiel hierfür ist die Reaktion auf den Zustandswechsel eines GUI-Buttons oder das Ablaufen eines Timers. Ein Signal kann auch weitergeleitet werden, indem es mit einem anderen Signal statt mit einem Slot gekoppelt wird. Allen Objekten, die von QObject abgeleitet werden, steht diese Funktionalität zur Verfügung. Die Kombination der Signale und Slots stellt eine mächtige Qt-Komponente dar.

3.5. Datenbanksysteme

Datenbanksysteme dienen der elektronischen Datenverwaltung. In der Regel bestehen diese aus den Datenbankmanagementsystem und den Daten selbst. Das Datenbankmanagementsystem dient, wie der Name schon verrät, der Verwaltung der Daten. Es wird zwischen den relationalen und nicht-relationalen Datenbanksystemen unterschieden.

[13] **M**eta **O**bject **C**omplier
[14] **G**ang **o**f **F**our

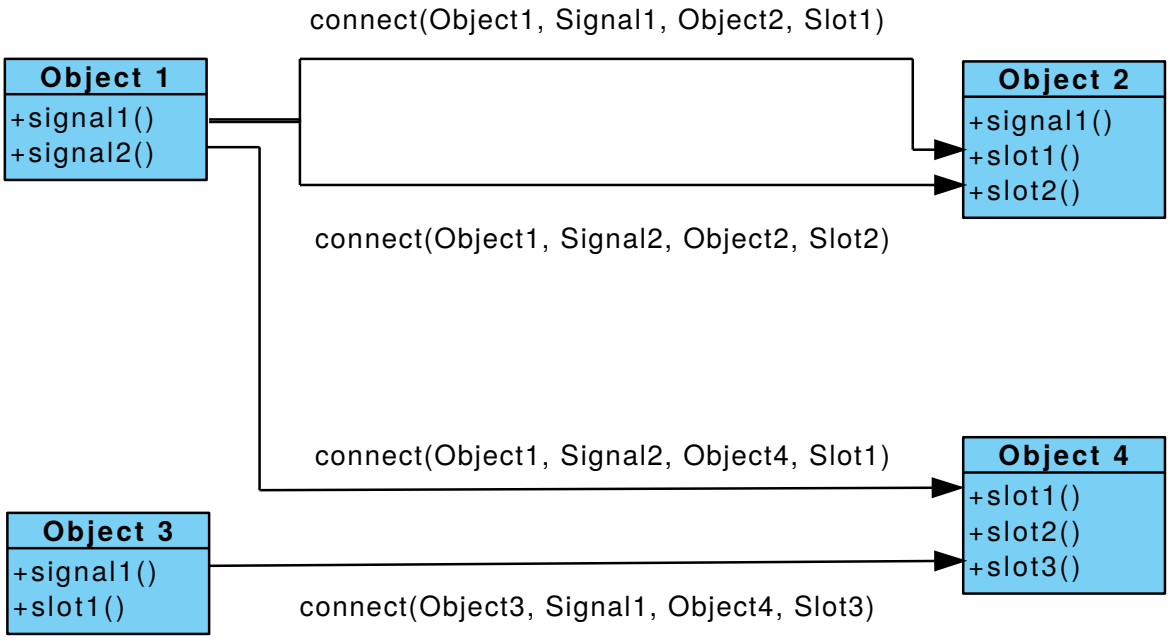


Abbildung 3.11.: Beispiel für das Qt Signal-Slot Mechanismus

3.5.1. NoSQL

NoSQL ist ein nicht-relationales Datenbanksystem bei dem verschiedene Datensicherungsmodelle zum Einsatz kommen (Dokument, Graph, Schlüssel-Wert etc.). Eine der Besonderheiten bestehen darin, dass die Informationen „on the fly“ hinzugefügt werden können und dass die „Zeilen“ nicht unbedingt die „Spalteninformationen“ beinhalten müssen. Eine sehr angenehme Eigenschaft der NoSQL Datenbanksystemen ist die horizontale Skalierbarkeit, was bedeutet, dass der Server aus mehreren Server besteht und bei Bedarf einfach neue dazugeschaltet werden können. Leider wird die ACID[1]-Eigenschaft auf Grund der Performance und der Skalierbarkeit teilweise oder gar vollständig aufgegeben. Ein NoSQL Datenbanksystem zeichnet sich durch hohe Flexibilität, Performance (insbesondere bei sehr großen Datenmengen) und Datensicherheit aus und soll dort eingesetzt werden, wo ein SQL System an seine Grenzen stößt. Die Verteilung der Daten auf mehrere Server schafft unter anderem Redundanzen, sodass diese eine Datensicherung überflüssig macht.

[1] **A**tomicity, **C**onsistency, **I**solation, **D**urability

3.5.2. SQL

SQL ist ein relationales Datenbanksystem, dessen Dateisicherungsmodell aus Zeilen und Spalten besteht. In den Spalten der jeweiligen Zeile werden Eigenschaften der Zeile selbst beschrieben. Das Tabellenschema muss vor der Nutzung fest definiert werden, sodass jede Zeile alle Spaltenwerte beinhalten muss. Die Änderung der Spaltenanzahl bedeutet Änderung der Datenbank und somit auch zwingender Weise Offline-Zeit des Systems. Mit wachsenden Datenmengen muss der Server auch „wachsen“, weil sich eine Verteilung der Daten auf mehrere Server als sehr schwierig und zeitintensiv erweist (vertikale Skalierbarkeit). Das SQL Datenbanksystem weist eine sehr hohe Transaktionssicherheit auf (ACID). Beispielsweise wird eine Transaktion entweder vollständig oder gar nicht durchgeführt. Relationale Datenbanksysteme sind unflexibel und bei großen Datenmengen nicht so performant wie die NoSQL Systeme, dafür ist die Konsistenz der Daten in der Datenbank garantiert.

4. Analyse

Es ist besonders schwierig, wenn nicht sogar unmöglich, ein fehlerfreies System zu entwickeln. Wichtig ist jedoch, sowohl die Software als Gesamtes, als auch die Software Teile zu klassifizieren, sie dementsprechend zu testen, zu verifizieren und zu validieren. Bei der Analyse einer Software wird klar, dass es unabhängige Teile gibt, die weniger kritisch für die Stabilität und Hauptfunktionalität des Systems sind, als die Kernmodule, die lieber nicht ausfallen sollten. Es wird, je nach Klassifizierung, dem einen Modul mehr Aufmerksamkeit geschenkt als dem anderen. Da Projekte normalerweise auch eine Deadline beinhalten, sind unendlich lange Tests, Prüfungen und Fehlersäuberung, inakzeptabel. Module mit einer hohen Risikoklasse müssen mit hoher Skepsis betrachtet werden. Deutlicher wird es anhand folgendes Beispiels: Der Absturz einer Taschenlampen App ist nicht so kritisch wie der Absturz der Software eines Medizingerätes, das gerade ein Menschenleben vor sicheren Tod bewahrt. Selbst bei solch einer Software, die besonders sicher laufen muss, gibt es Module, dessen Stabilität sich auf die Hauptaufgabe nicht auswirkt, z.B. ist es unwichtig, ob während eines Einsatzes die Anzeige der Uhrzeit eines Defibrillators ausgefallen ist, jedoch sehr kritisch, wenn die Energiedosierung für die Schockabgabe nicht oder fehlerhaft einstellbar ist. [4]

Trotz allen Vorsichtsmaßnahmen können Totalausfälle eines Systems nicht ausgeschlossen werden. Es muss also sichergestellt werden, dass solche Fehler verfolgbar sind. Unabhängig davon, aus welchem Grund das System ausgefallen ist (Mechanik, Elektronik etc.), ist es wichtig dass der Nutzer darüber informiert wird (akustisch, optisch), sodass weitere Schritte eingeleitet werden können [6]. Solche schwerwiegenden Fehler müssen so schnell wie möglich von dem Gerätehersteller behoben werden. Hierfür werden Logdateien geräteintern mitgeschrieben und im Fehlerfall als Wegweiser bei der Fehlersuche genutzt. Der jetzige Weg der Fehlerbehebung sieht folgendermaßen aus: Ein Systemfehler wird von dem Kunden bemerkt und an die Serviceabteilung des Herstellers gemeldet und beschrieben. Falls es sich tatsächlich um ein Gerätefehler handelt, müssen die dazugehörige Logdateien an den Hersteller übermittelt werden. Der Kunde muss also die Logdateien auf einen externen Datenträger exportieren, auf einen Rechner kopieren, ggf. verpacken/komprimieren und dann per Mail an die Serviceabteilung des Herstellers weiterleiten. Erst dann kann sich der Produktverantwortlicher, des Herstellers mit dem Problem des betroffenen Gerätes beschäftigen. Sobald ein Fehler erkannt und lokalisiert wurde, ist es meistens klar, ob der Fehler

gerätespezifischer Natur ist oder ob nur zuletzt aktualisierte Geräte oder gar eine ganze Produktionsserie davon betroffen sind. Folgende Fragen bleiben aber offen:

- Falls mehrere Geräte betroffen sind, wie viele sind es und wo befinden sie sich?
- Wurden sie schon eingesetzt?
- Werden potenziell in diesem Moment weitere Geräte mit der fehlerhaften Software bestückt?
- Wie viele Kunden müssen darüber informiert werden?

Manche dieser Fragen können schnell, manche innerhalb wenigen Tagen und manche gar nicht, beantwortet werden. Dabei ist es von höchster Bedeutung, in kürzester Zeit aktuelle Probleme zu lokalisieren, die Kunden darüber zu informieren, die Fehler zu beheben, die Korrekturen zu testen und alle betroffenen Geräte zu aktualisieren. Bei der Analyse solcher Szenarien wird schnell deutlich, dass durch Automatisierung dieser Vorgänge die Zeiten solch einer Fehlerbehandlung deutlich verkürzt werden könnte. Das bedeutet, dass sowohl die Kunden als auch der Hersteller von einem Überwachungssystem profitieren würden.

Es soll also ein System entwickelt werden, das möglicherweise noch vor dem Eintreten eines schwerwiegenden Fehlers die Produktverantwortlichen über bestimmte Unregelmäßigkeiten informiert, sodass der Hersteller auf diese rechtzeitig reagieren kann. Dies würde im besten Fall dazu führen, dass die Kunden über bestimmte Fehler vom Hersteller gewarnt werden, bevor diese aufgetreten, oder im schlimmsten Fall, dass der Kundenservice des Herstellers schon über den Fehler informiert ist, bevor der Kunde den Fehler meldet. Da die Serviceabteilung frühzeitig mit wichtigen Informationen versorgt wird, ist es möglich, dem Kunden schneller und zuverlässiger zu helfen. Des weiteren muss das System in der Lage sein, alle potenziell betroffenen Geräte und Kunden ausfindig zu machen und diese Liste den verantwortlichen Personen zu übermitteln. Nachdem der Fehler behoben wurde, wird die neue Firmware von dem System an die Endgeräte verteilt und der Kunde muss lediglich den Download und das Update der Firmware erlauben.

Es ist nicht nur in einem Fehlerfall nützlich, über solch ein System zu verfügen. Denn sowohl bei den Updates als auch bei Upgrades besteht die Gefahr, dass die Firmware irgendwelche unentdeckte Probleme verbirgt. Hierfür bietet sich eine kontrollierte Markteinführung an. Diese wäre mit Hilfe eines Softwareverteilungssystems sicherer und leichter durchzuführen. Es wäre möglich einen Rollout für die Geräte des Typs AB und Seriennummer YZ bis YZ + 100 freizugeben. Falls die Firmware doch fehlerhaft ist, befinden sich lediglich 100 Geräte im Feld, die „repariert“ werden müssen, was immerhin besser ist, als möglicherweise mehrere tausend Geräten einsammeln zu müssen. Die fehlerhafte Firmware wäre sofort gesperrt und würde die Kundengeräte nicht mehr erreichen können.

Heutzutage werden die Kompilate über Webseiten des Herstellers an Kunden verteilt und via Speichermedien oder mit Hilfe spezieller Updatetools aktualisiert (siehe Kapitel 2). Der Kunden sollte nach jedem Softwareupdate den neuen Gerätestand dem Hersteller melden. Dies hat zur Folge, dass die Hersteller auf die Rückmeldung des Kunden angewiesen sind, um die Gerätedatenbanken pflegen zu können. Falls keine Rückmeldung stattfindet, hat der Hersteller keine Möglichkeit mehr, an die Geräteinformationen zu kommen.

Ein weiterer Punkt sind kostenpflichtige Optionen, die der Kunde erwerben kann. Diese sind mit den kostenpflichtigen Apps für Betriebssysteme wie Android, IOS, Windows Mobile etc. vergleichbar. Falls der Kunde solch eine Option erwerben möchte, meldet er sich bei dem Hersteller und bekommt nach der Bezahlung einen Kode, der die Option freischaltet. Dieser wird in das Gerät eingegeben, wodurch gewünschte Optionen freigeschaltet werden. Die Eingabe selbst erfolgt meistens über einen Drehenkoder, Taster, Eingabefolien etc. mit denen alphanumerische Zeichenketten „eingedreht“ werden können. Dieser Vorgang hat erhebliches Optimierungspotenzial, zumal manuelle Eingaben alphanumerischer Zeichenketten unbequem sind und mit hoher Wahrscheinlichkeit fehlerhafte Eingaben aufweisen. Bei Verwendung des neuen Softwareverteilungssystems müsste der Kunde nur die gewünschte Option bei dem Hersteller buchen und sein Gerät mit der Gerätedatenbank synchronisieren. Diese Vorgehensweise ist für den Kunden wesentlich einfacher und komfortabler. Den Mitarbeitern der Serviceabteilung würde eine Aufgabe erspart bleiben, sodass sie die Zeit für andere Tätigkeiten nutzen könnten. Das bedeutet, dass sowohl der Kunde als auch der Hersteller von dem Feature profitieren würden.

Die Funktionalität des Systems ist beliebig erweiterbar. Insbesondere bietet sich das Monitoring und Fernbedienung des entfernten Gerätes durch einen Spezialisten an. Das bedeutet, dass ein spezialisierter Arzt die Möglichkeit hätte, in Echtzeit die Messwerte des Gerätes am Unfallort aus dem entfernten Krankenhaus mitzuverfolgen, den Sanitätern Instruktionen zu geben, oder gegebenenfalls, selbst einzugreifen. Es ist klar, dass hierbei Konnektivitätsprobleme auftreten könnten, denn möglicherweise gibt es in der Region des Unfallortes keine Internetverbindung oder diese weist eine zu hohe Latenz oder zu niedrige Geschwindigkeit auf etc. Möglicherweise müssten in bestimmten Regionen die Notfallwagen eine Internetverbindung via Satellit unterstützen, damit sie als Hotspots für die Medizingeräte eingesetzt werden können.

In dieser Arbeit wird die Entwicklung des Demonstrators in zwei Teilprojekte aufgeteilt und auf folgende Funktionalitäten beschränkt:

Serverseite:

- Synchronisierung der Geräteinformationen mit der Datenbank
- Synchronisierung der Logdateien des Gerätes mit der Datenbank
- Verteilung der Update-Dateien an Geräte

- Freischaltung der Geräteoptionen

Geräteseite:

- Anbindung eines WLAN Moduls
- Erweiterung der Firmware für die Synchronisierung der Geräteinformationen
- Erweiterung der Firmware für eine Synchronisierung der Logdateien des Gerätes
- Erweiterung der Firmware für den Download der Update-Datei
- Erweiterung der Firmware für die Aktivierung der Optionen

5. Design

Bei der Designspezifikation des Projektes wird anhand der Analyse ein Plan für die spätere Realisierung herausgearbeitet [7]. Im vorherigen Kapitel wurde deutlich, dass das Projekt physikalisch aus zwei Modulen besteht, der Server- und Geräteseite. Demzufolge werden die zwei Module unabhängig von einander entworfen und realisiert. Selbstverständlich muss die Kommunikationsschnittstelle zwischen dem Gerät und dem Server dabei berücksichtigt werden. Nichtsdestotrotz spricht man von zwei unabhängigen Modulen, die unabhängig von einander konzeptioniert und implementiert werden können. Idealerweise könnten die Teilprojekte an zwei Entwickler mit jeweils speziellen Kenntnissen aufgeteilt werden. Der eine müsste die Kenntnisse über die Programmiersprache und ihre Besonderheiten haben, die für das Zielgerät wichtig sind, wobei der andere die Server und Datenbanken-Kenntnisse aufweisen müsste. Der einzige Berührungspunkt der beiden Entwickler wäre das Kommunikationsinterface des Gerätes und des Servers.

Anhand der Abbildung 5.1 wird die grobe Funktionsweise erläutert.

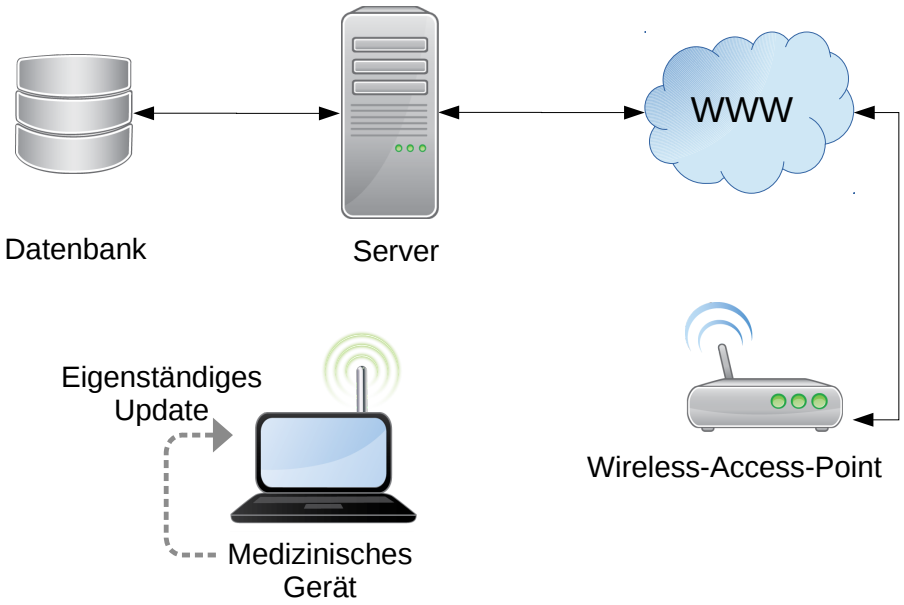


Abbildung 5.1.: OTA-Management für Medizingeräte

Das Gerät agiert als Client und kommuniziert via Internet mit dem Server. Der Server aktualisiert die Datenbank und hat die Möglichkeit das Gerät über neue Updates zu informieren. Die Manipulation der Datenbank wird ausschließlich von dem Server durchgeführt, sodass die Clients keine Möglichkeit haben direkt mit der Datenbank zu kommunizieren. Des weiteren ist die Authentifizierung, sowohl des Servers als auch des Clients, von großer Bedeutung bezüglich der Sicherheit.

5.1. Serverdesign

Dieses Unterkapitel beinhaltet die Modellierung des Servers selbst und den Aufbau der Datenbank, die die Geräteinformationen, Gerätelogdateien und Update-Dateien speichern soll. Als erstes sollen die Aufgaben und Funktionen des Servers definiert werden. Der Server ist nicht in der Lage, das Gerät fernzusteuern und kann nur die Daten und Dateien entgegennehmen, die vom Gerät gesendet werden. Diese Beschränkung wird auch durch das Kommunikationsinterface geschaffen und dient der Patientensicherheit. Es ist also keine serverseitige Gerätekonfigurationen/Manipulation und dadurch auch keine Patientengefährdung möglich. Die Kommunikation kann ausschließlich vom Gerät aufgebaut werden. Dies behebt die Gefahr, dass das Gerät in einem möglicherweise ungünstigen Moment unterbrochen wird, um sich mit dem Server zu synchronisieren. An welcher Stelle die Triggerung der Serververbindung stattfindet, wird im nächsten Kapitel näher erläutert.

Sobald die Verbindung hergestellt wird, muss der Server prüfen, ob der Client für die Kommunikation berechtigt ist. Bei fehlgeschlagener Authentifizierung des Clients wird die Verbindung gekappt, anderenfalls werden zuerst die Clientdaten wie Seriennummer, Software-/Hardwareversion, Gerätetyp etc. entgegengenommen und in der Datenbank gesichert.

Im zweiten Schritt werden die Logdateien empfangen und ebenfalls in der Datenbank gesichert. Hierbei besteht nicht die Notwendigkeit, jedesmal vollständige Logdateien zu übermitteln, wichtig sind nur die Teile der Dateien, die sich noch nicht in der Datenbank befinden. Da es sich bei den Logdateien ausschließlich um Textdateien handelt, die bekannterweise eine sehr hohe Kompressionsrate aufweisen, wäre eine weitere Möglichkeit, diese in komprimierter Form an den Server zu senden. Die dritte Variante könnte die Kombination der ersten beiden sein. Es muss also noch geprüft werden, welche Variante das beste Aufwand-Nutzen-Verhältnis aufweist.

Anhand der übermittelten Geräteinformationen wird der Client ggf. darüber in Kenntnis gesetzt, dass für diesen Gerätetypen eine neue Firmware existiert, ein Download wird jedoch nicht erzwungen. Wann die neueste Firmware heruntergeladen und installiert wird, entscheidet ausschließlich das Gerät bzw. der Nutzer. Hierbei handelt es sich wieder um eine Vorgehensweise, die der Sicherheit des Patienten dient. Ein Gegenbeispiel hierfür wäre das

bekannte quasi unkontrollierte Neustarten einer Windowsmaschine nach der Installation der neuesten Updates. Bei einem Notfallmedizingerät ist so ein Systemverhalten äußerst inakzeptabel. Ein Firmwareupdate des Systems während eines Rettungseinsatzes wäre für den Patienten fatal, möglicherweise sogar tödlich. Aus diesen Gründen werden alle Interaktionen mit dem Server clientseitig getriggert. Folgende Abbildung visualisiert einen möglichen, wie oben beschriebenen Kommunikationsablauf.

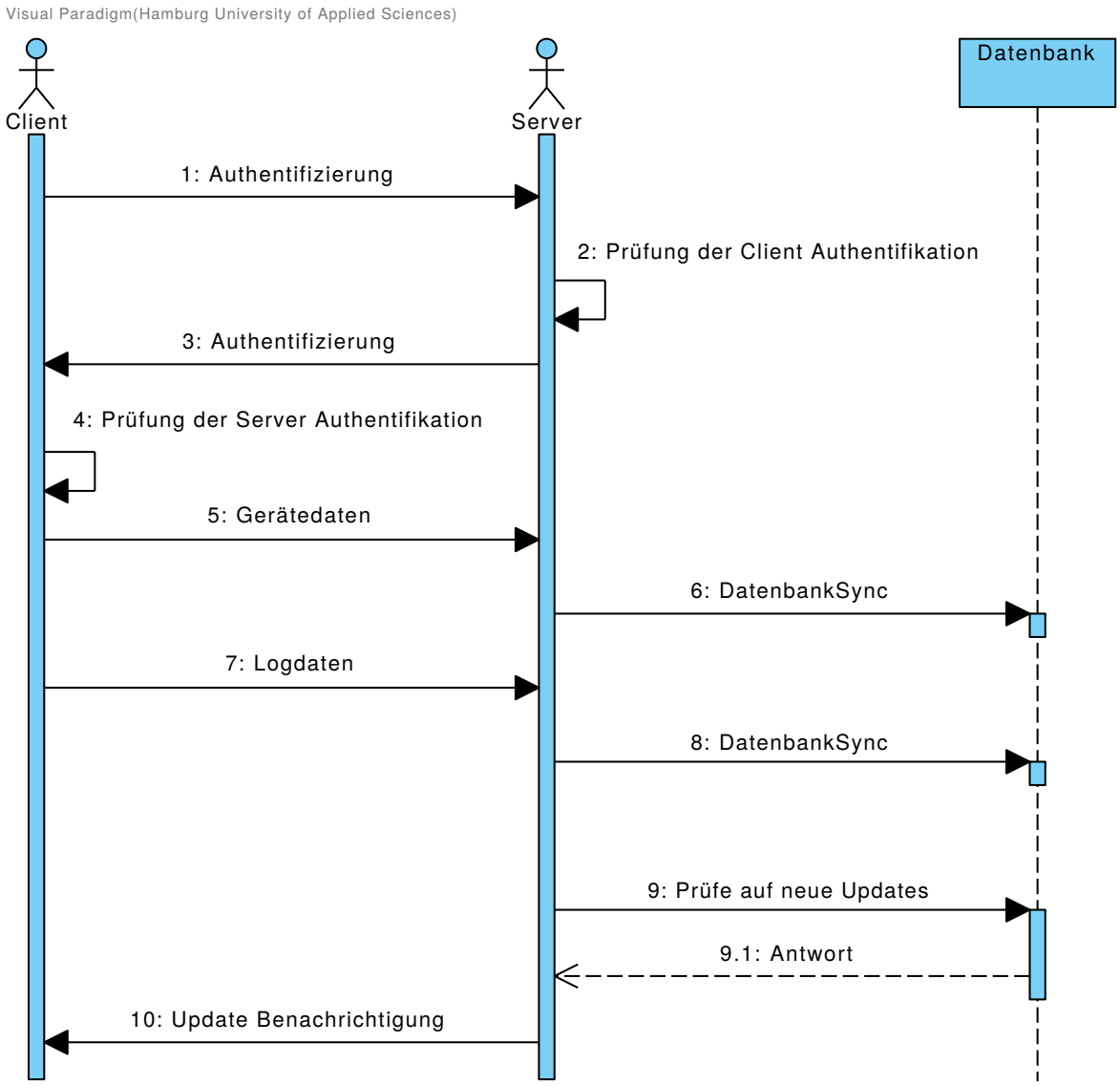


Abbildung 5.2.: Kommunikationsbeispiel

Anhand des Kommunikationsbeispieles in der Abbildung 5.2 wird klar, dass sowohl server- als auch clientseitig eine Authentifizierung des Kommunikationspartners stattfindet. Dies ist insofern wichtig, als dass der Server keine Updatedateien an unberechtigte Clients verteilen

darf. Andererseits ist es noch wichtiger, dass der Client keine falsche, gar fremd manipulierte Firmware herunterlädt und diese später auch installiert.

Nach einer erfolgreichen Synchronisierung mit dem Server wird das Gerät ggf. über eine neuen Firmware informiert. Wie schon erwähnt, wird diese nicht sofort heruntergeladen. Hierfür ist eine Anforderung der Firmware notwendig, wichtig ist jedoch, dass das Gerät in Kenntnis über ein anstehendes Softwareupdate gebracht wurde. Ob und wann die Updatedatei angefordert wird, ist ausschließlich clientseitig gesteuert. Nähere Informationen über das Firmwareupdate werden im nächsten Kapitel beschrieben. Eine Anforderung der neuen Firmware könnte wie in Abbildung 5.3 gezeigt aussehen.

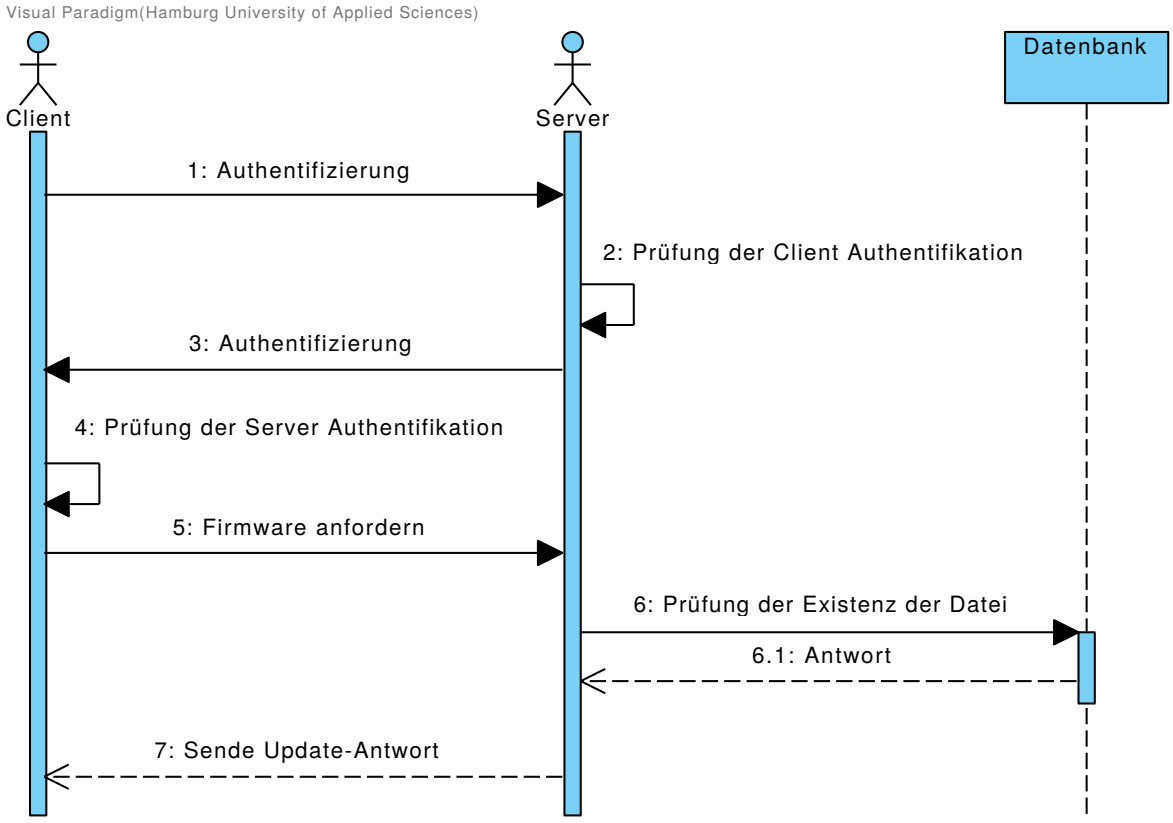


Abbildung 5.3.: Update-Anforderung

In den Abbildungen 5.2 und 5.3 sind auf der Serverseite Funktionen zu finden, die einen Datenbankzugriff beinhalten (DatenbankSync(), SendeUpdatedatei() etc.). Der Server stellt eine indirekte Verbindung zwischen den Clients und der Datenbank her. Direkte Zugriffe, etwa Tunnel-Kommandos, sollen vermieden werden.
Die Struktur der Datenbank ist nicht besonders komplex, sollte aber gut durchdacht aufgebaut werden, damit spätere Erweiterungen mit wenig Aufwand durchgeführt werden können.

Im ersten Schritt werden alle Informationen gesammelt, die für ein Gerät wichtig sind. In der Tabelle 5.1 wurden beispielhaft einige Informationen aufgelistet.

Eigenschaft	Beschreibung
Seriennummer	Eindeutige Seriennummer eines Gerätes eines Typs
Geräteidentifikation	Eindeutiger Name eines Gerätes eines Typs
Softwareversion	Die aktuell installierte Softwareversion
Hardwareversion	Die aktuell bestückte Hardwareversion
Gerätetyp	Der Typ des Gerätes
Kunde	Kundeninformationen
Optionen	Geräteoptionen, die Freigeschaltet sind
Logdateien	Logdateien, die vom Gerät mitgeschrieben werden

Tabelle 5.1.: Geräteeigenschaften

Anhand der Informationen aus der Tabelle 5.1 lässt sich das, in der Abbildung 5.4 gezeigte, ER Model extrahieren. Dieser Schritt ist sehr hilfreich bezüglich der Übersicht, Struktur und Optimierung.

Hierbei wurde darauf geachtet, dass vorhandene Entitäten so nah wie nötig durch Attribute und Schlüsselattribute beschrieben werden. Das ER Modell hebt einige Optimierungsmöglichkeiten hervor, denn alle one-to-many und many-to-one Relationen weisen Optimierungspotenziale auf. Anhand des folgenden Beispiels soll die Vorgehensweise verdeutlicht werden. In der Abbildung 5.5 sind Relationen zwischen Entitäten *Device* $\leftrightarrow$ *Customer* und zwischen *Device* $\leftrightarrow$ *Device option* in einem ER Model dargestellt.

Die rot bzw. blau gekennzeichnete Polygone markieren Bereiche, die bei der Umwandlung zusammengefasst werden können. Alle Attribute bleiben nach der Optimierung bestehen und der primäre Schlüssel der n-Seite der Kardinalität wird übernommen. Als Fremdschlüssel wird der Primärschlüssel der Entität auf der 1-Seite der Kardinalität übernommen. Das Ergebnis der Optimierung der zwei Fälle aus der Abbildung 5.5 sieht wie folgt aus:

Device(Id, Serial number, ..., Customer id)

mit dem Fremdschlüssel

Device.Customer id $\rightarrow$ *Customer.Id*

und

Device option(Option id, State, Unlock code, Device id)

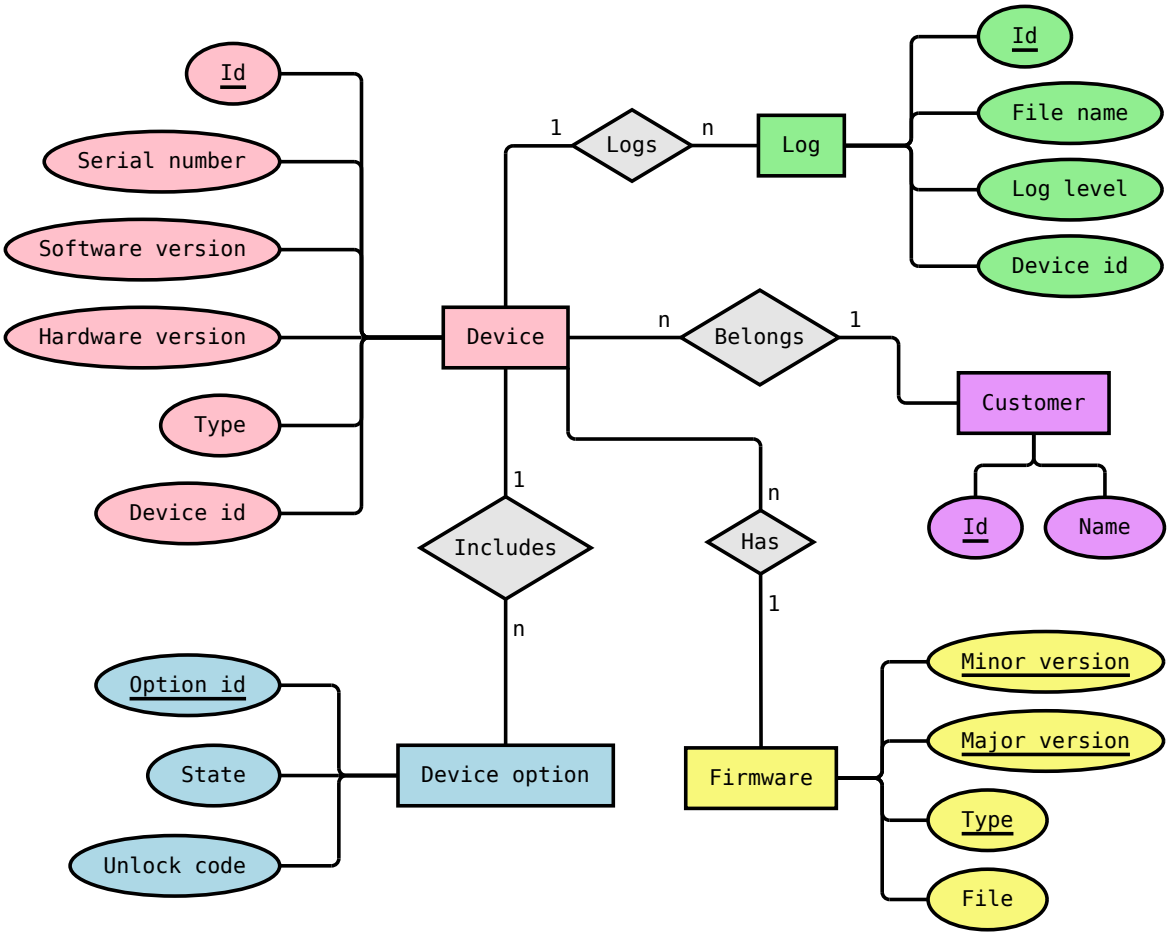


Abbildung 5.4.: ER Modell der Datenbank

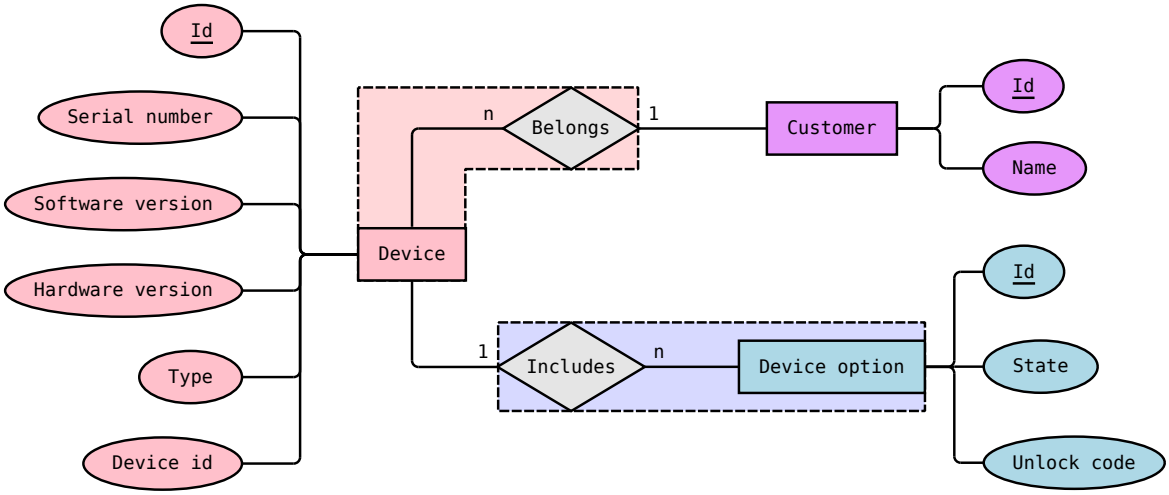


Abbildung 5.5.: Optimierung one-to-many und many-to-one Beziehungen

mit dem Fremdschlüssel

$$\textit{Device option.Device id} \rightarrow \textit{Device.Id}$$

Aus dem ER Modell (Abbildung 5.4) lassen sich Datentabellen und Verlinkungen zwischen diesen definieren. Das Ziel hierbei ist es, die Aufteilung so zu wählen, dass die logische Abhängigkeiten in den jeweiligen Tabellen erhalten bleiben. Eine mögliche Tabellenaufteilung wäre

- Gerät
- Geräteoption
- Kunde
- Firmware
- Log

Es muss eine Verbindung zwischen den genannten Tabellen hergestellt werden. Beispielsweise muss die Gerätetabelle die Kundentabelle referenzieren, weil ein Kunde mehrere Geräte besitzen kann. Des weiteren werden auf diese Weise redundante Daten gemieden. Wenn ein Kunde etwa hundert oder hunderte von Geräten in seinem Besitz hat, taucht der Kundenname bzw. Kundeninformationen nicht in der Gerätetabelle hundertfach auf. Es werden lediglich in der Gerätetabelle die Referenzen auf den einen Eintrag in der Kundentabelle angelegt. Noch ein Vorteil besteht darin, dass Änderungen bezüglich eines Kunden keinerlei Änderungen in der Gerätetabelle mit sich bringen, sondern nur die Kundentabelle tangieren. Analog dazu muss die Gerätetabelle von der Optionstabelle referenziert werden, da eine Option in mehreren Geräten aktiviert sein darf. Das EER[1]-Diagramm in die Abbildung 5.6 dürfte die beschriebene Vorgehensweise verdeutlichen.

Damit sofort klar wird, welcher Eintrag aus welcher Tabelle wo hinkommt, wurden die Fremdschlüsseln hervorgehoben und farblich gekennzeichnet. Als Ergebnis der Referenzierung entsteht eine Gerätetabelle, die als Kundeneinträge den automatisch inkrementierten primären Schlüssel der Kundentabelle beinhaltet. Ebenso finden sich in der Geräteoptionstabelle die primären Schlüsseln der Gerätetabelle.

Da die Datenbankstruktur definiert wurde, kann das Serververhalten bei der Kommunikation mit einem Gerät, das noch nicht in die Datenbank aufgenommen wurde, konzeptioniert werden. Durch die Geräteattribute (Seriennummer, Geräte_id und Gerätetyp) ist die Eindeutigkeit eines Gerätes gewährleistet. Es besteht also keine Gefahr zwei oder mehr Datenbankeinträge zu haben, bei denen Zweifel bestehen, um welches Gerät es sich handelt. Wenn

[1] **E**nhanced **E**ntity-**R**elationship ist ein erweitertes **E**ntity-**R**elationship Model das zur Modellierung der Datenbanken benutzt wird.

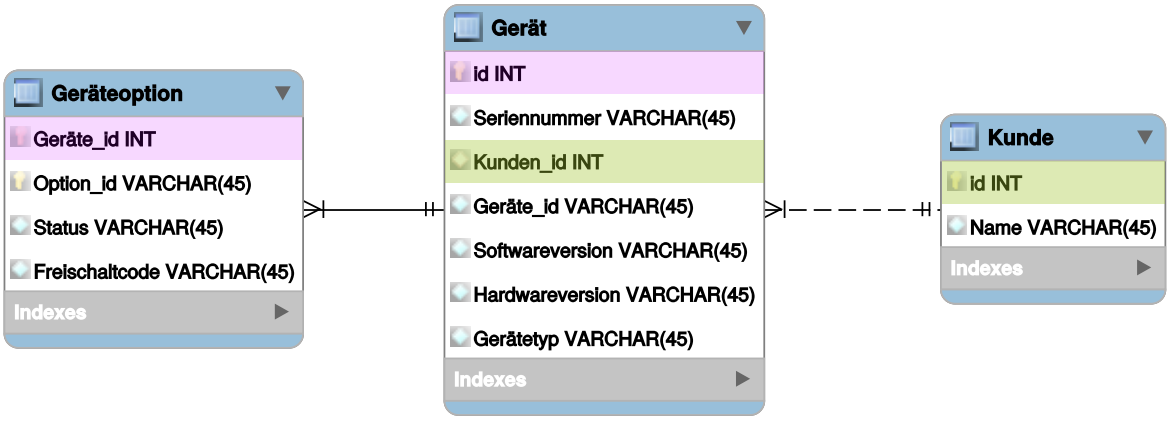


Abbildung 5.6.: Referenzen zwischen den Geräte-, Geräteoption- und Kunden-Tabelle

also ein Gerät die Verbindung zum Server triggert, wie in der Abbildung 5.2 dargestellt, wird ein Datenbankeintrag des Gerätes in die Gerätetabelle erstellt. In der Geräteoptionstabelle werden so viele Einträge generiert, wie es die Anzahl der möglichen Optionen für das Gerät vorschreibt. Anhand des Status der Option ist erkennbar, ob diese installiert, aktiviert oder noch nicht verfügbar ist. Das Konzept der Geräteoptionen wird in der Sektion 5.2 näher erklärt. Die Kundeninformationen, werden ebenfalls von dem Gerät übermittelt und in der Kundentabelle eingepflegt. Falls der Kunde schon vorhanden ist, wird dessen Identifikationsnummer in die Gerätetabelle eingetragen, ohne dass die Kundentabelle angefasst wird. Bei weiteren Verbindungen werden statische Informationen (Softwarenummer, Gerätetyp etc.) nicht mehr angefasst, denn das Ziel solcher Verbindungen ist es, die dynamische Gerätedaten (Softwareversion, Logdateien etc.) auf dem Server aktuell zu halten.

5.2. Gerätedesign

Anders als bei der Serverkonzeptionierung, müssen bei der geräteseitige Konzepterstellung gewisse Randbedingungen beachtet werden. Da das Gerät schon existiert und nicht in Rahmen dieser Arbeit entwickelt wird, müssen die neuen Funktionen an die Gegebenheiten angepasst werden.
Es handelt sich um ein Linux-basiertes eingebettetes System. Die Hauptfunktionalität des Gerätes ist die Patientenbeatmung in Notfallsituationen. Das System beinhaltet zwei Prozessoren, einen für die Aufgaben der Beatmungsmaschine und einen für die Aufgaben der graphischen Darstellung und der Benutzerinteraktion mit dem System. Für die Entwicklung des Upgrades ist nur der GUI-Controller, also der Hauptcontroller von Bedeutung, da die Hauptapplikation von diesem ausgeführt wird. Die Hauptapplikation ist ein CMake basiertes Projekt unter Verwendung der Programmiersprache C++ und des Qt-Frameworks. Um das

gegebene System an die neue Anwendung anzupassen, muss die Applikation selbst sowie der Linux-Kernel angefasst werden. Die GUI muss um die neuen Funktionalitäten, die die Kommunikation mit dem Server ermöglichen, erweitert werden und im Kernel müssen die Voraussetzungen für solch eine Kommunikation erfüllt werden, etwa die notwendigen Module einbinden.
In der Sektion 5.1 wurde darauf hingewiesen dass die Server-Client-Kommunikation ausschließlich von dem Client getriggert werden kann. Insbesondere ist es wichtig, dass dies keineswegs während eines Einsatzes passieren darf. Laut Gebrauchsanweisung des Herstellers sind die Benutzer (Sanitäter, Ärzte etc.) dazu verpflichtet, vor jeder Geräteübernahme, zB. bei jedem Schichtwechsel, eine sogenannte Funktionskontrolle durchzuführen. Hier-

Abbildung 5.7.: Startmenü mit der Funktionskontrolle

bei handelt es sich um einen Test der einzelne Teilsysteme (Knöpfe, Drehencoder, visuelle und akustische Signalisierung, Funktionalität der Pneumatik bzw. des Luftgebläses usw.) prüft und den Benutzer über die Einsatzfähigkeit der Systems informiert. Bei der Funktionskontrolle ist das Gerät nicht im Einsatz und hat mit hoher Wahrscheinlichkeit den Zugang zu einem kabellosen Netzwerk. Diese Stelle eignet sich besonders gut für die Platzierung der neuen Funktion, die für die Synchronisierung mit dem Server dient. Die Zusammenfassung des Gerätezustandes, die nach Durchführen einer Funktionskontrolle dem Benutzer angezeigt wird, kann um die Informationen des anstehenden Softwareupdates erweitert werden, sodass das gegebene Gerätekonzept erhalten bleibt. In diesem Funktionsabschnitt wird nur der Marker über neuer Software gesetzt. Das erweiterte Klassendiagramm (Abbildung 5.8) zeigt grob die vorhandene Klassenstruktur inklusive Erweiterungsklasse *DatenbankSynchronisierung*. Das Update selbst kann nur in Betreiber- oder in Service-Modus durchgeführt werden. Hierbei handelt es sich um spezielle, passwortgeschützte Modi, die vom Betreiber oder dem Service betreten werden können. Diese Modi beinhalten einen Menüpunkt, in dem ein Softwareupdate gestartet werden kann. Durch die Funktionserweiterung wären die zwei

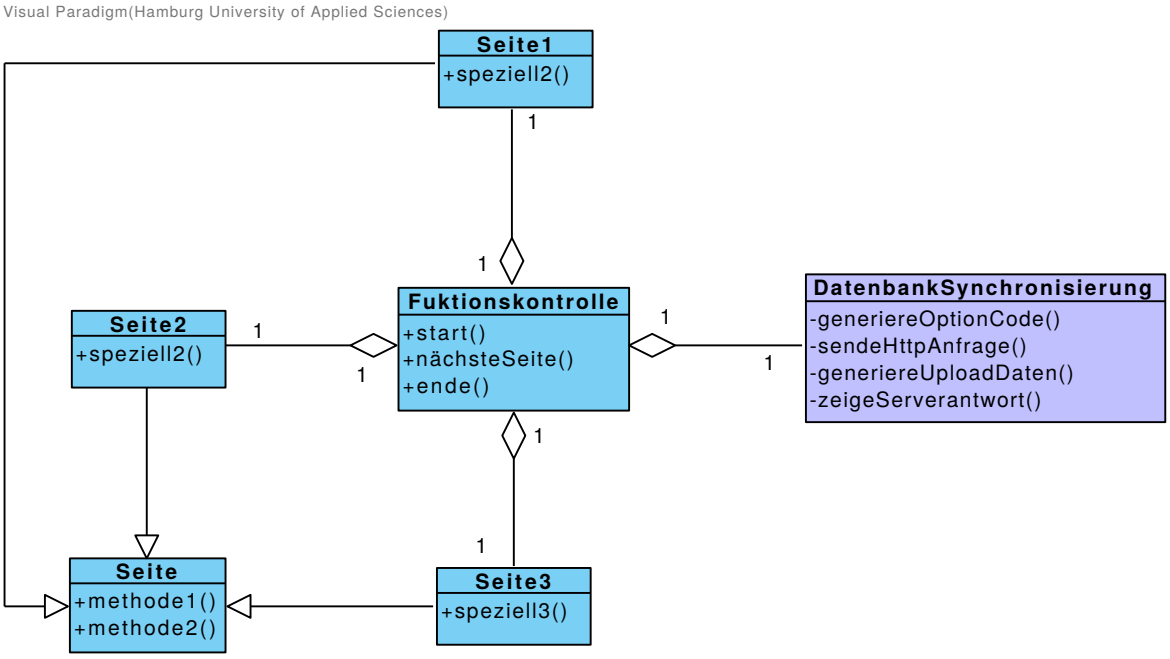


Abbildung 5.8.: Klassendiagramm der Funktionskontrolle mit Erweiterung

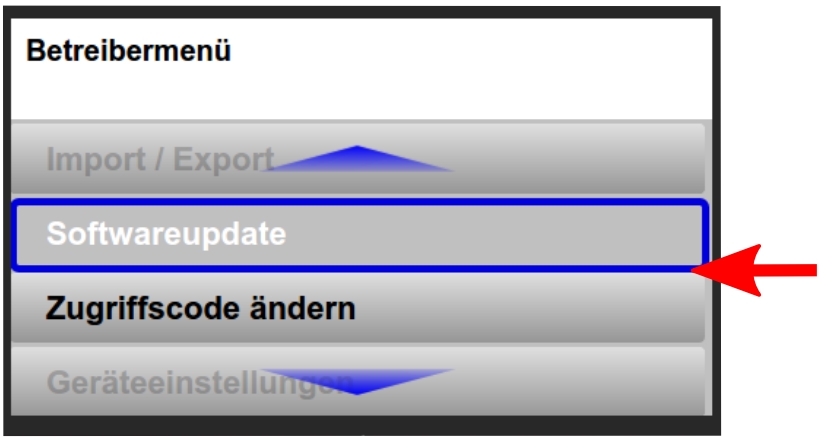


Abbildung 5.9.: Betreibermenü

Menüs um einen Unterpunkt erweitert, unter dem die binäre Updatedatei heruntergeladen und installiert werden kann. Der rote Markierungspfeil in der Abbildung 5.9 zeigt eine mögliche Stelle für die Platzierung des neuen Untermenüs. Sowohl die Menüstruktur als auch die Menülogik bleiben trotz der Funktionserweiterung erhalten, sodass der Benutzer, der das alte System kennt, auch das neue ohne Probleme verstehen und nutzen kann. Damit der Fortschritt des Prozesses dem Nutzer mitgeteilt werden kann, muss ein Fortschrittsbalken des Download-Prozesses implementiert werden.
Auch bei der Aktivierung neuer Optionen des Gerätes muss der Benutzer keine neuen Menüs oder Ähnliches kennenlernen. Die Freischaltung findet automatisch, innerhalb der Synchronisierung statt, die während der Funktionskontrolle läuft. In der Abbildung 5.10 ist eine mög-

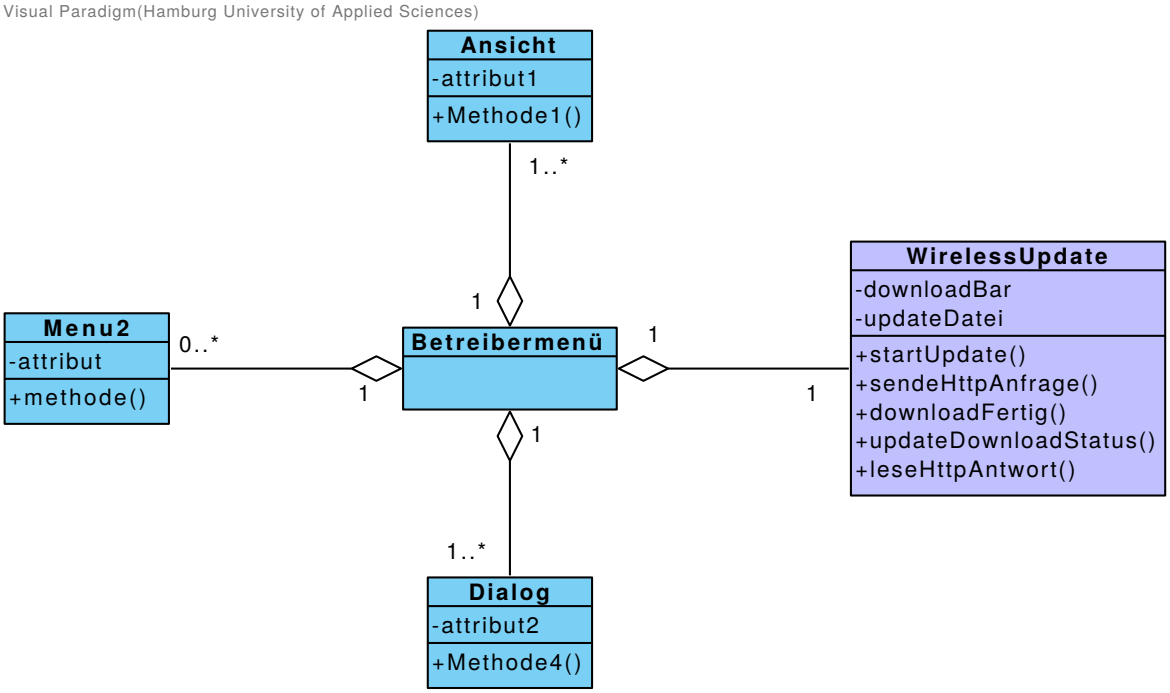


Abbildung 5.10.: Klassendiagramm des Betreibermenüs mit Erweiterung

liche Erweiterung dargestellt. Nach wie vor werden freigeschaltete Optionen im passwortgeschützten Betreibermenü aktiviert.
Offen ist nur noch die Ansteuerung eines WLAN Moduls. Hierfür muss der Linuxkernel neu konfiguriert und gebaut werden. Das Konfigurationsmenü des Kernels bietet die Möglichkeit, einzelne Module fest einzukompilieren oder als Modul während der Laufzeit zu laden. Das Zielgerät ist nicht mit einem WLAN Modul ausgestattet, dafür hat es aber freie USB-Anschlüsse. Diese könnten entweder direkt oder mit Hilfe eines aktiven USB-Hubs zum Anschließen eines WLAN-Sticks genutzt werden. Die Alternative mit einem aktiven USB-Hub wird dann gewählt, wenn die Stromaufnahme des WLAN-Sticks die Stromabgabe des inte-

grierten USB-Ports des Zielgerätes übersteigt. Gewiss ist jedenfalls, dass die Ausstattung des Gerätes mit einem WLAN-Modul möglich ist.

5.3. Kommunikation und Sicherheit

Die Kommunikation zwischen den Clients und dem Server ist eine Internetkommunikation. Die Vorteile bestehen darin, dass der Hersteller alle Geräte, solange diese eine Internetverbindung haben, unabhängig von der geographischen Position, pflegen, warten und synchronisieren kann. Theoretisch könnte jeder anderer Teilnehmer im Netz sowohl mit dem Server als auch mit den Clients kommunizieren. Ohne Authentifizierung der Kommunikationspartner und Verschlüsselung der Kommunikationsdaten wäre das ganze System bezüglich der Sicherheit unbrauchbar. Es gäbe weder die Garantie, dass die Gerätedaten in der Datenbank tatsächlich von dem Gerätetyp X mit der Seriennummer Y stammen, noch dass die gerade heruntergeladene Firmware tatsächlich von dem Server des Herstellers kommt. Anwendung der Phishing[1]- oder Man-in-the-Middle-Angriffe[2] wären äußerst erfolgreich und der Einsatz des Systems damit ausgeschlossen. Ein weiterer Aspekt wäre, dass die Client-Server-Kommunikation im Klartext durchgeführt wird, was den Angreifern die Mühe der Dekodierung erspart. Die Folge solcher Angriffe könnte verheerende Ausmaße annehmen. Von Geräteabstürzen, über falsche Parametrierung der Beatmung/Defibrillation, bis zu Remote-Steuerung durch unbefugte Personen, ist alles denkbar. Diese Problematik besteht in vielen anderen Anwendungsbereichen wie z.B. Online Banking, Einkäufe über das Internet, etc. in den der Kommunikationspartner verifiziert und der Kommunikationskanal verschlüsselt werden muss.

Für solche Anwendungsfälle wird aktuell HTTPS[3]-Protokoll benutzt. Hierbei handelt es sich um ein Kommunikationsprotokoll aus der Internetprotokollfamilie und dient der verschlüsselten Datenübertragung. HTTPS nutzt das HTTP Schema mit dem Unterschied, dass die Daten SSL/TLS[4] verschlüsselt werden. Hierbei handelt es sich um ein hybrides Verschlüsselungsprotokoll. Der Kommunikationspartner wird mittels SSL-Handshakes (Zertifikat) authentifiziert (siehe Kapitel 3.1.4). Im nächsten Schritt wird mit Hilfe asymmetrischer Verschlüsselung der symmetrische Sitzungsschlüssel ausgetauscht, mit dem die Kommunikationsdaten verschlüsselt werden. Solch ein Verfahren macht den Austausch des Sitzungsschlüssel

[1] Ein Versuch durch Fälschung der Webseiten, E-Mails etc. an Benutzerdaten zu kommen.

[2] Ein Kommunikationsteilnehmer der den beiden Kommunikationsseiten den jeweils anderen Kommunikationspartner vortäuscht.

[3] **H**yperText **T**ransfer **P**rotocol **S**ecure

[4] **S**ecure **S**ockets **L**ayer / **T**ransport **L**ayer **S**ecurity - SSL wurde in der Version 3.1 im Jahr 1999 umbenannt in TLS 1.0. Nichtsdestotrotz ist die Verschlüsselungsmethode unter den Namen SSL bekannter und wird weiterhin so benutzt.

sicher und durch die symmetrische Verschlüsselung der Nutzdaten ist die Kommunikation auch schnell.

Unter Verwendung dieses Verfahrens werden die Sicherheitslücken geschlossen und das System kann bedenkenlos eingesetzt werden.

6. Realisierung

Die Realisierung des Systems wird wie das Design in zwei Teilprojekte aufgeteilt. Zum einen wird die Realisierung des Servers mit allen notwendigen Funktionalitäten und zum anderen die Realisierung der Geräteerweiterung, hinsichtlich der Nutzung neuer Features, die von dem Server angeboten werden, beschrieben. Fragen, die zum Zeitpunkt des Designs nicht vollständig beantwortet werden konnten, etwa Performance-Unterschiede bei der Synchronisation der Logdateien oder Anbindung des WLAN-Sticks, werden in diesen Kapitel näher untersucht und geprüft, welche Implementierung/Realisierung mehr und welche weniger vorteilhaft ist. Es werden auch genutzte Entwicklungsumgebungen und Frameworks mit allen für das Projekt relevanten Eigenschaften beschrieben. Die komplette Entwicklung wird auf einem Linux-System durchgeführt. Damit die Realisierung strukturiert und schnell durchgeführt werden kann, wird diese in zwei Schritte aufgeteilt. Im ersten Schritt wird das System auf dem *localhost* entwickelt, also ohne den physikalischen Server und Client und im nächsten Schritt wird die Integration der Systeme durchgeführt. Die Aufsplittung in die Test- und Produktivumgebung soll die Implementierung und das Debuggen deutlich erleichtern und beschleunigen.

6.1. Realisierung unter Testumgebung

6.1.1. Realisierung des Servers

Die Funktionalität des Servers kann in zwei Gruppen aufgeteilt werden, der Kommunikation mit dem Gerät und der Manipulation der Datenbank. Bei dem Design des Systems wurde auf diese Aufteilung verzichtet, bei der Realisierung wird diese jedoch der besseren Übersicht halber eingehalten.

Serverdatenbank

Die ersten Entscheidungen müssen bezüglich der Datenbankwahl getroffen werden: SQL oder NoSQL? Ein grober Vergleich der beiden Datenbanksysteme kann die klaren Vor- und Nachteile des jeweiligen Systems hervorheben (siehe Kapitel 3.5).

Nach der Gegenüberstellung der zu Verfügung stehenden Datenbanksystemen wird klar, dass die Wahl eines SQL Datenbanksystems, für diesen Anwendungsfall, einige wichtige Vorteile mit sich bringt. Da es sich nicht um große Datenmengen handelt, spielt die Cloud-Aufteilung und die höhere Performance des NoSQL Systems bei großen Datenmengen eine unwesentliche Rolle. Dagegen ist die Transaktionssicherheit, also Atomarität und Datenkonsistenz von fundamentaler Bedeutung.

Eine der beliebtesten SQL Datenbanksystem ist MySQL. Da diese oft in der Kombination mit einem Linux, Windows oder Mac OS Betriebssystem, Apache Webserver und der PHP Webprogrammiersprache eingesetzt wird, existieren schon vorkonfigurierte Pakete, die diese Komponenten beinhalten (siehe Kapitel 3.2). Da es sich hierbei um ein weitverbreitetes, quellen-offenes System handelt, steht der Entwicklung eine große Community zur Verfügung, die bei der Einarbeitung, Umsetzung und der Fehlersuche zeitsparend sein könnte. Die Datenbankzugriffe selbst können ganz einfach via Kommandozeile in der SQL-Syntax durchgeführt werden. Die Installation und Konfiguration des MySQL Datenbanksystems ist recht einfach und kann direkt aus der Paketquelle mit de Ausführung folgender Zeile bezogen werden

apt-get␣install␣mysql-server

Nach der Installation und der Passwortvergabe kann mit *mysql␣-u␣root ␣-p*, einschließlich der Eingabe des bei der Installation gewählten Passwortes, das Tool gestartet werden. Das Argument *-u␣root* gibt den Benutzer *root* an und das Argument *-p* gibt an, dass eine Passworteingabe erforderlich ist. Jetzt kann auf der Kommandozeile in der SQL-Syntax eine neue Datenbank erstellt, neue Tabellen angelegt und andere Datenbankmanipulationen vorgenommen werden. Im Listing 6.1 ist eine erfolgreiche Anmeldung an die MySQL-Kommandozeile dargestellt.

```
misko@mdj:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25
Server version: 5.7.13-0ubuntu0.16.04.2 (Ubuntu)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights
   reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
   affiliates. Other names may be trademarks of their respective owners

Type 'help;' or '\h' for help. Type '\c' to clear the current input
   statement.

mysql>
```

Listing 6.1: MySql Komandozeilenbeispiel (neue Sitzung)

Listing 6.2 zeigt die Ausgabe des *show databases*-Kommandos.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| wm                 |
+--------------------+
5 rows in set (0,00 sec)

mysql>
```

Listing 6.2: MySql Komandozeilenbeispiel (das Kommando *show databases*)

Es besteht keine Notwendigkeit, weitere Tools bei der Entwicklung zu involvieren. Falls es doch komplex und unübersichtlich wird, steht einem das Oracle-Tool MySQL Workbench zur Verfügung. Hierbei handelt es sich um ein sehr hilfreiches und mächtiges Werkzeug, das ein intuitives graphisches Interface anbietet.

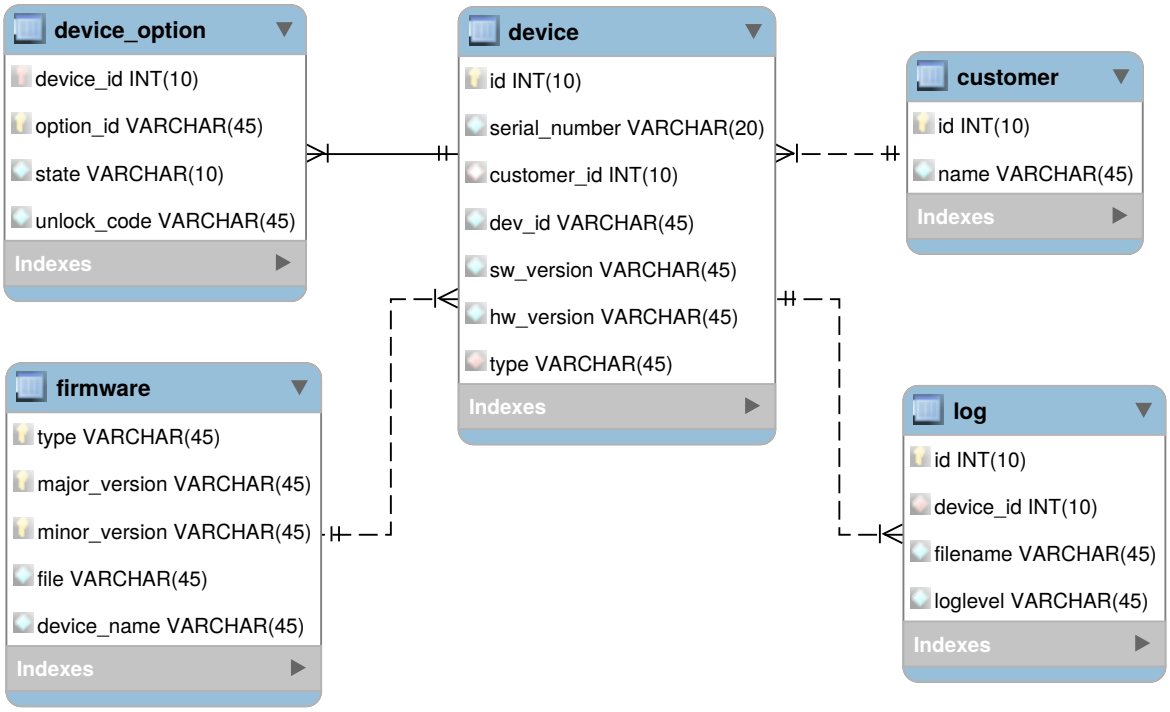


Abbildung 6.1.: EER der realisierten Datenbank

Die Datenbanktabellen werden wie in dem Abschnitt 5 beschrieben, realisiert und das EER-Diagramm wurde in der Abbildung 6.1 dargestellt.

Server

Für die Realisierung des Servers wurde die Programmiersprache PHP gewählt. Die Implementierung solcher Serverapplikationen kann auch in C++, Java o.ä. durchgeführt werden, dennoch ist PHP ein Quasi-Standard geworden. Insbesondere erweisen sich die Datenbank- und die HTTP-Zugriffe als sehr komfortabel. Noch bequemer wird die Entwicklung bei Verwendung eines PHP-Frameworks. Welches von den vielen Frameworks genutzt werden soll, ist eine Aufwand-Nutzen-Frage, die jeder Entwickler projektabhängig beantworten muss. Unter den bekanntesten PHP-Frameworks zählen

- Laravel
- Yii
- Symfony
- Phalcon
- Codeigniter
- Zend PHP Framework

Einige der Frameworks sind sehr mächtig und bieten einen breiten Spektrum an Funktionalitäten an. Für die Realisierung des Servers ist ein einfaches, schlankes Framework mit einer kurzen Einarbeitungszeit, bestens geeignet. Es wäre kontraproduktiv ein Framework zu nutzen, bei dem die Einarbeitung und Konfiguration teurer als die eigentliche Implementierung ist. Aus diesen Gründen wurde der CodeIgniter Web Framework gewählt. Es handelt sich hierbei um ein schlankes (ca. 2 Megabyte), MVC[1]-Pattern orientiertes PHP-Framework. Die Konfiguration beschränkt sich auf das Setzen der Basis-URL-Adresse des Servers. Aus praktischen Gründen bietet es sich an, während der Entwicklung, hierfür den *localhost:xxxx* zu wählen, wobei der *localhost* der Adresse *127.0.0.1* und *xxxx* der gewählten Portnummer (z.B. *4000*) entspricht. Wenn das PHP-CLI[2] per Kommandozeile mit dem Kommando *php␣-S␣localhost:4000* gestartet ist, kann mit einem Browser die Testseite aufgerufen werden. Hierbei bedeutet das Argument *-S*, dass der Webserver an der übergebenen Adresse und Port gestartet werden soll. Wenn der Webserver zu Entwicklungszwecken an dem lokalen Rechner ausgeführt wird, können die Servermeldungen einfacher mitverfolgt werden, da diese als Terminalausgabe erscheinen. Das bringt bei der Fehlersuche erhebliche Erleichterungen mit sich. Das Listing 6.3 zeigt eine Terminalausgabe des laufenden PHP-Server.

[1] **M**odell-**V**iew-**C**ontroller
[2] **C**ommand **L**ine **I**nterface

```
misko@mdj:~$ php -S localhost:4000
PHP 7.0.9-1+deb.sury.org~xenial+1 Development Server started at Tue Aug
    16 15:47:29 2016
Listening on http://localhost:4000
Document root is /home/misko
Press Ctrl-C to quit.
[Tue Aug 16 15:48:12 2016] 127.0.0.1:55844 [200]: /svn/HomeControl/
    Bachelor_mdj/WMService/index.php/SyncDeviceController/check
[Tue Aug 16 15:48:12 2016] 127.0.0.1:55846 [200]: /svn/HomeControl/
    Bachelor_mdj/WMService/index.php/SyncDeviceController/sync
```

Listing 6.3: PHP Server

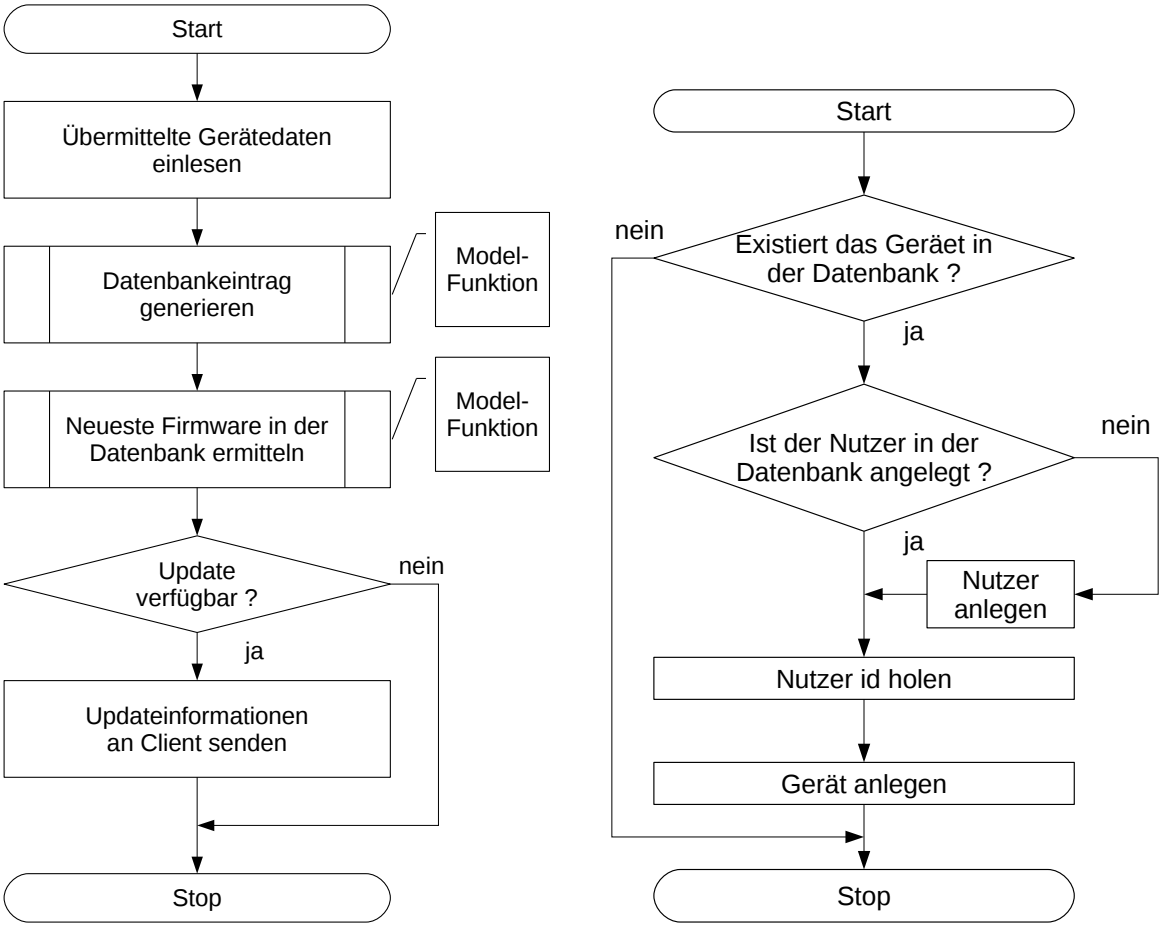
Anhand der Ausgabe ist erkennbar, dass der Server gestartet wurde, dass sich das Wurzelverzeichnis unter */home/misko* befindet, dass die Basis-URL *http://127.0.0.1:4000/svn/HomeControl/Bachelor_mdj/WMService* ist und dass es erfolgreiche Zugriffe (HTTP Statuscode 200) auf *index.php/SyncDeviceController/check* und *index.php/SyncDeviceController/sync* gab. In einem Fehlerfall würden sich die Pfade der aufgerufenen Seite rot statt grün einfärben und in den eckigen Klammern würde der entsprechende HTTP Statuscode stehen. Die gesamte Funktion des Servers, die zur Kommunikation mit dem Client gebraucht wird, kann zuerst in drei Unterfunktionen aufgeteilt werden

- Check() → Überprüft anhand der übergebenen Daten ob
  - sich das Gerät in der Datenbank befindet und fügt dieses ggf. hinzu
  - ein Update verfügbar ist und teilt dies dem Client mit
  - eine Option freigeschaltet werden soll und teilt dies dem Client mit
- Sync() → Synchronisiert die Clientdaten mit der Datenbank
- Download() → Überträgt die von dem Client angeforderte Updatedatei.

Diese Funktionen werden in den Controller der Servers untergebracht. Alle Funktionalitäten, die einen Datenbankzugriff beinhalten, werden im Modell implementiert.

Die Abbildung 6.2 visualisiert detailreich den serverseitigen Ablauf, der zum einen die Controller-Funktion, die die Synchronisation der Clientdaten mit der Datenbank durchführt und zum anderen den Ablauf der Modell-Funktion, die anhand der Gerätedaten einen Datenbankeintrag generiert. Bei dem Funktionsaufruf handelt es sich um eine HTTP-Anfrage des Gerätes. Aus der Abbildung 6.2(a) ist erkennbar, dass alle Datenbankzugriffe in Modell-Funktionen ausgelagert wurden. In der Abbildung 6.2(b) wird genauer die Generierung des Datenbankeintrags dargestellt. Die Suche nach der neuesten Firmware beinhaltet lediglich SQL-Befehle, sodass keine Notwendigkeit nach einem Ablaufplan besteht.

Anhand der in der Abbildung 6.2 vorgestellten Programmablaufpläne kann die Synchronisationsfunktionalität des Servers implementiert werden. Auch wenn keine Notwendigkeit für die Benutzung einer IDE[1] besteht, wird aus Komfortgründen Eclipse mit PHP-Plugin benutzt.



(a) Controller Funtion *Check()* (b) Modell Funktion *Datenbankeintrag Generieren()*

Abbildung 6.2.: Flussdiagramm der Funktion *Check()*

[1] **I**negrated **D**evelopment **E**nvirement

```
4  public function check()
5  {
6          $this->load->model('Insert_model', 'im');
7          $this->load->model('GetInfoModel', 'gim');
8          $devData = array
9          (       // save all received parameters
10                 'serial' => $this->input->post('serial'),
11                 'devId' => $this->input->post('devId'),
12                 'swVersion' => $this->input->post('swVersion'),
13                 'hwVersion' => $this->input->post('hwVersion'),
14                 'type' => $this->input->post('type'),
15                 'customer' => $this->input->post('customer'),
16                 'options' => $this->input->post('options'),
17         );
18         $insertedDevId = $this->im->insertFromSync($devData);
19         if ( $insertedDevId )
20                 echo "inserted␣into␣database\n";
21         else
22                 echo "device␣exists␣in␣database\n";
23
24         $dbData ['swVersion'] = $this->gim->getNewestFwVersion(
               $devData['type']);
25         // sync options
26         $lines = explode(';', $devData['options']);
27
28         foreach ( $lines as $line )
29         {       // extract single option parameters
30                 $lineSplit = explode('␣', $line);
31                 $optionId   = $lineSplit[0];
32                 $unlockCode = $lineSplit[1];
33                 $state      = $lineSplit[2];
34                 $ret = $this->gim->getOption($devData['devId'],
                       $optionId, $unlockCode);
35                 log_message('error', "ret=".$ret);
36                 if ( $ret == null ) // insert new option
37                         $this->im->insertOption($devData['serial'],
                               $devData['devId'], $optionId, $unlockCode,
                                $state);
38                 else
39                 {       // sync old options
40                         $optCode = $this->im->syncOptionState(
                               $insertedDevId, $optionId, $state);
41                         if( $optCode )
42                                 echo($optCode.";");
43                 }
44         }
45         if ( $devData['swVersion'] < $dbData ['swVersion'] )
46                 echo 'update';
47 }
```

Listing 6.4: Serverfunktion *Check()*

Im Listing 6.4 ist die Implementierung der Funktion *check()* dargestellt. In den Zeilen 8-17 werden die übergebenen Geräteparameter in das Feld *devData* gesichert. Anschließend wird das Feld als Parameter an die Modellfunktion *insertFromSync()* übergeben. Damit diese Funktion benutzt werden kann, wurde in der Zeile 6 das Model *Insert_model* geladen und mit dem Alias *im* versehen. Ob es sich bei dem Client um eine Erstverbindung handelt, wird in die Variable *inserted* (Zeile 18) gespeichert und anhand des Wertes dieser Variable wird eine Meldung geschickt (Zeilen 19-22). Als nächstes werden die Optionen synchronisiert. Dafür wird der Parameter *options* aus dem Feld *devData* mit Hilfe der Funktion *explode()* in die Bestandteile zerlegt. Wenn sich der Zustand der Option geräteseitig geändert hat, sprich eine Option wurde aktiviert bzw. deaktiviert, wird die Datenbank dementsprechend Synchronisiert. Falls sich der Zustand einer Option datenbankseitig geändert hat, wird das Gerät mit der Datenbank synchronisiert. Als letztes wird überprüft, ob die Gerätesoftwareversion aktuell ist, anschließend wird eine entsprechende Benachrichtigung dem Client gesendet.

Der Aufruf der *Modelfunktion insertFromSync()* benötigt eine detaillierte erklärung.

```
83  public function insertFromSync($devData = array())
84  {
85      $devId = null;
86      // generate database request for given device data
87      $query = $this->db->select('id')
88                  ->where('dev_id', $devData['devId'])
89                  ->where('serial_number', $devData['serial'])
90                  ->get('device');
91
92      if( $query->num_rows() == 0)
93      {   // device don't exist in database
94          $customerId;
95          // generate database request for given customer
96          $query = $this->db->from('customer')
97                      ->where('name', $devData['customer'])
98                      ->get();
99
100         if ( $query->num_rows() )
101         {   // customer exists, get the id
102             $customerId = $query->row()->id;
103         }
104         else
105         {   // customer don't exist, so add him
106             $this->db->set('name', $devData['customer'])
107                         ->insert('customer');
108             // get the generted customer id
109             $customerId = $this->db->insert_id();
110         }
111         // now add the new device in the database
112         $this->db->set('serial_number', $devData['serial'])
113                     ->set('customer_id',    $customerId)
114                     ->set('dev_id',         $devData['devId'])
```

```
115                     ->set('sw_version',     $devData['swVersion'])
116                     ->set('hw_version',     $devData['hwVersion'])
117                     ->set('type',           $devData['type'])
118                     ->insert('device');
119             $devId = $this->db->insert_id();
120         }else{ // device exists, get the id
121             $devId = $query->row()->id;
122         }
123         return $devId;
124     }
```

Listing 6.5: Serverfunktion *insertFromSync()*

Listing 6.5 zeigt den Ausschnitt der Quelldatei *Insert_model.php* der die Implementierung der Funktion *insertFromModel()* darstellt. In der Zeile 87 wird in der Datenbank, genauer in der Gerätetabelle, nach einem Eintrag gesucht, dass die übergebene Daten beinhaltet. Falls so ein Eintrag existiert, gibt die Funktion false zurück und wird beendet. Anderenfalls muss ein neuer Eintrag generiert werden. Zuerst muss jedoch festgestellt werden, ob der übergebene Kunde vorhanden ist, oder ob dort auch ein neuer Eintrag eingepflegt werden muss. Für den Fall, dass es sich um einen existierenden Eintrag handelt, wird die Kunden-Identifikationsnummer in die Variable *customerId* gesichert (Zeile 102), sonst wird eine neue Kundentabellenzeile generiert und die neue Kunden-Id wird gesichert (Zeile 106-109). Nachdem die Geräteinformation vervollständigt wurde, kann der eigentliche Gerätetabelleneintrag eingepflegt werden (Zeile 112-118). In der Zeile 119 wird nur noch der Rückgabewert auf die Id des eben getätigten Eintrags gesetzt.

Anstehend ist noch die Erklärung der Funktion *getNewestFwVersion()* (Listing6.6).

```
22  public function getNewestFwVersion($type)
23  {
24      $major = null;
25      $minor = null;
26      // select the highest major and minor number in the database
27      $query = $this->db->select('major_version, minor_version')
28          ->where("device_name",$type)
29          ->order_by('major_version, minor_version','DESC')
30          ->get('firmware', 1);
31      // extract the major and minor version numbers
32      if ( $query->num_rows() )
33      {
34          $major = $query->row()->major_version;
35          $minor = $query->row()->minor_version;
36      }
37      // put them together (e.g. 2.1)
38      return $major.'.'.$minor;
39  }
```

Listing 6.6: Serverfunktion *getNewestFwVersion()*

Der Kern der Funktion besteht in der Datenbankanfrage (Zeile 27-30), die aus der Firmwaretabelle die (zum Gerätetyp passende) Firmware mit der höchsten Versionsnummer ausfindig macht. Die Anfrage in CodeIgniter-Syntax ist folgender SQL-Anfrage äquivalent

```
SELECT major_version, minor_version
FROM wm.firmware WHERE device_name = 'DeviceType'
ORDER BY minor_version DESC, minor_version DESC LIMIT 1;
```

6.1.2. Modifizierung der Clientfirmware

Geräteseitig gilt es eine Erweiterung zu implementieren, die die Netzwerkfähigkeit und Serverkommunikation ermöglicht. Da es sich um das Upgrade eines bestehenden Systems handelt, existieren Vorgaben, die eingehalten werden müssen, etwa die Programmiersprache, Framework, Toolchain, Betriebssystem etc. Programmiert wird in der Sprache C++ unter Verwendung der Qt Bibliothek in einer CMake-orientierten Projektstruktur. Die Cross-platform CMake ermöglicht das Bauen der Software für verschiedene Plattformen. Diese angenehme Eigenschaft wird für die Entwicklung der GUI-Komponenten genutzt, sodass zur Testzwecken die Firmware nicht auf das Zielgerät aufgespielt werden muss, sondern in der X11 Umgebung gestartet werden kann. Auf diese Weise können ohne vorhandene Zielhardware, die Teilergebnisse und Fortschritte begutachtet werden. Die Kompatibilität mit dem eingebetteten System kann in bestimmten Abständen geprüft werden, z.B. wenn ein Meilenstein erreicht wurde.

Die Implementierung der Gerätesoftware kann also ohne der Hardware durchgeführt werden. Da der Server unter localhost gestartet werden kann, ist es möglich, die Gerätesoftware auf der lokalen Maschine zu testen. So wird die Anbindung des WLAN Moduls, SSL-, Komprimierungs- sowie anderen benötigten Bibliotheken zurückgestellt und die eigentliche Funktionen der Geräte-Software unabhängig vom Zielsystem vorgezogen. Anhand des Flussdiagramms in der Abbildung 6.3 und des Listings 6.7 wird die Client-Server-Kommunikation vorgestellt.

```
79 void FunctionTestDatabaseSync::sendHTTPRequest(){
80     QEventLoop eventLoop;
81     QNetworkAccessManager* mgr = new QNetworkAccessManager(this);
82 // "quit()" the event-loop @ network request "finished()"
83     CONNECT(mgr, SIGNAL(finished(QNetworkReply*)),
84             &eventLoop, SLOT(quit()));
85
86     QUrl serviceUrl = QUrl(check);
87     QNetworkRequest request(serviceUrl);
88
89     QSslConfiguration config = request.sslConfiguration();
```

```
90      qDebug() << "protocol = " << config.protocol();
91
92  /* upload device information */
93      const IoManager::Versions& v = IO.getVersions();
94      QByteArray postData;
95      postData.append(QString("serial=2219&"));
96      postData.append(QString("devId=simulation&"));
97      postData.append(QString("swVersion=1.0&"));
98      postData.append(QString("hwVersion=%1/%2&").arg(v.mbRev).arg(v.
            mbAssembly));
99      postData.append("type=MEDUXXX Standard&");
100     postData.append("customer=MEDUCUSTOMER&");
101     postData.append(QString("options=%1").arg(generateOptionCodes(
            QString("%1").arg(v.deviceId))));
102     DEBUG_INFO("%s", postData.data());
103     QNetworkReply* reply = NULL;
104     reply = mgr->post(QNetworkRequest(serviceUrl), postData);
105     eventLoop.exec();// mgr emited finished()
106     if(reply){
107         QString answer(reply->readAll());
108         if (answer.contains("update"))
109             GETSETTING(newFw)->set(1);
110         else
111             GETSETTING(newFw)->set(0);
112
113         QStringList unlockCodes = answer.split(";");
114         if(! unlockCodes.isEmpty())
115                 unlockCodes.removeLast();
116         foreach (QString code, unlockCodes)
117             Std2Options::addOption(code.toLatin1().data(), false);
118     }
```

Listing 6.7: Gerätefunktion zum Senden einer HTTP-Anfrage

Die Instanz *mgr* von Typ *Zeiger auf QNetworkAccessManager* (Zeile 81) dient zum Senden der Netzwerkanfragen und Empfangen der Antworten. Eines der Eigenschaften des Qt-Frameworks ist das sog. Signal-Slot-Mechanismus. Hierbei handelt es sich um eine der zentralen Funktionalitäten des Frameworks. Es dient zu Objektkommunikation und kann mit dem Observer-Pattern verglichen werden. Ein konkretes Beispiel findet sich in der Zeile 83 wo eine Verbindung zwischen dem Objekt *mgr* und *eventLoop* hergestellt wird. Immer wenn der *QNetworkAccessManager mgr* ein Signal *finished()* aussendet, wird die Funktion (Slot) *quit()* der *QEventLoop eventLoop* ausgeführt. In der Zeile 86 wird die URL der Serverzieladresse instanziiert, sodass bei der Anfrage die Serverfunktion *check()* ausgeführt wird. Es müssen noch die zu sendende Daten in ein Bytefeld zusammengepackt werden. Dies geschieht in den Zeilen 95-101. Zu Demonstrationszwecken wurden einige Geräteinformationen als Dummy-Werte übergeben (Softwareversion, Gerätetyp, Kundenname etc.). Jetzt kann der Post-Befehl, an der zuvor definierten URL mit den zusammengesetzten Daten ge-

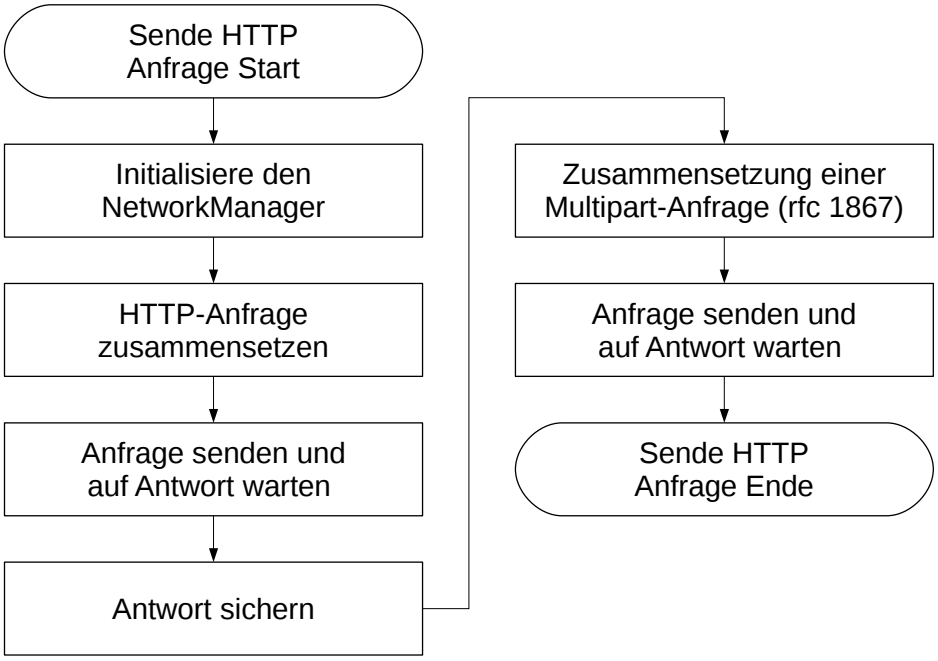


Abbildung 6.3.: Flussdiagramm der Client-Funktion die die Synchronisation mit dem Server durchführt

sendet und die Antwort in die Variable *reply* von Typ *QNetworkReply* gesichert werden. Da die Ausführung des Post-Befehls und die Serverantwort etwas Zeit in Anspruch nehmen, muss das Programm solange warten (Zeile 105). Sobald die Serverantwort zu Verfügung steht, wird der *NetworkManager mgr* das Signal *finished()* senden, welches mit dem Slot quit() des eventLoop verbunden ist. Danach wird der Code weiter ausgeführt und die Antwort des Servers wird ausgewertet. Falls eine neue Firmware zu Verfügung steht, wird diese Information in die persistente Variable *newFw* gesichert. Außerdem wird noch geprüft ob serverseitig neue Optionen freigeschaltet wurden (Zeile 113-117). Es muss noch erwähnt werden, dass neue Optionen von dem Server nur freigeschaltet, aber nicht aktiviert werden können. Nach Freischaltung neuer Optionen lassen sich diese in dem passwortgeschützten Betreibermenü aktivieren.

Der zweite Teil der Synchronisation besteht aus dem Komprimieren und Hochladen der Logdateien des Gerätes. In dem Abschnitt 5.1 stellte sich die Frage bezüglich der Performance beim Hochladen der Logdateien. Es standen drei Möglichkeiten zu Verfügung

- immer alle Dateien komprimiert versenden
- nur Änderungen der Dateien unkomprimiert versenden
- nur Änderungen der Dateien komprimiert versenden

Sowohl für den Client als auch für den Server wäre die einfachste Variante, wenn der Client immer alle Dateien in komprimierter Form versenden würde. Durch diese Vorgehensweise müsste keine der beiden Seiten Dateimanipulationen vornehmen. Die zweite Erkenntnis besteht darin, dass die Komprimierung eines Ordners mit X Dateien effizienter ist als die Komprimierung einzelner Dateien. Diese Tatsache lässt sich dadurch begründen, dass die Abkürzungen, die für die erste Datei in das Komprimierungs-Dictionary vermerkt wurden, für alle nachkommende Dateien mitgenutzt werden können. Bei sequenzieller Komprimierung ist das nicht der Fall. Die Größe aller Logdateien in komprimierter Form beträgt etwa 200 KByte. Die Übertragung solcher Pakete ist bei aktuellem Stand der Technik in Bruchteilen einer Sekunde abgearbeitet. Aus diesem Grund wird die einfachste Variante umgesetzt und bewusst ein Optimierungspotenzial für kommende Stufen gelassen.

Die Qt-Bibliothek bietet als Teil der Klasse *QByteArray* die Funktionen *qCompress* und *qUncompress* an. Leider sind Archive, die mit diesen Funktionen erstellt wurden, nicht mit den gängigen Tools, wie zum Beispiel WinZIP, WinRAR oder 7-Zip kompatibel. Eine der Lösungen stellt die QuaZIP-Bibliothek da. Hierbei handelt es um einen C++ Wrapper der bekannten zlib-Bibliothek. Um diese benutzten zu können, muss der Ordner mit den Quelldateien in die Top-Level-CMake-Datei mit *include_directories(quazip)* angegeben werden. Hierdurch werden die Quellen der Bibliothek bei dem Build-Process mitberücksichtigt. Als nächstes muss dem Linker die Bibliothek bekannt gemacht werden, damit bei dem Link-Vorgang gegen diese gelinkt werden kann. Dies wird in der selben Datei mit der Zeile *set (LINK_LIBRARIES ${LINK_LIBRARIES} ... quazip_static libz.a)*. Da es sich bei der QuaZIP um eine Bibliothek handelt, die auf die zlib-Bibliothek aufbaut, muss diese im System vorhanden sein und gegen diese ebenfalls gelinkt werden. Optional kann die Anwesenheit der zlib als Voraussetzung für das Build-Process mit *find_package (ZLIB REQUIRED)* angegeben werden. Nun können mit Hilfe der Funktion *static bool compressDir (QString fileCompressed, QString dir=QString(), bool recursive=true)* ganze Ordner komprimiert werden. Da der Parameter *recursive* ein Default-Wert hat, der für die Rekursion bei der Komprimierung sorgt, könnte der Aufruf wie folgt aussehen:

```
compressDir(NameDesArchivs, DerZuKomprimiernderOrdner)
```

6.2. Realisierung unter Produktivumgebung

In dem Abschnitt 6.1 wurden alle Funktionalitäten auf der Client- und auf der Server-Seite implementiert. Hiermit wurde auf der einen Seite die Machbarkeit und das Konzept des Systems geprüft, auf der anderen Seite ist der entstandene Quelltext auf die Zielsysteme leicht portierbar.
Nun gilt es den Localhost-Server auf einen „richtigen“, von außen erreichbaren Server umzusiedeln, sodass dieser für die Clients immer erreichbar ist. Auf der Seite des Clients muss noch die Kommunikationshardware, sowie alle notwendigen Bibliotheken und Treiber eingebunden werden.

6.2.1. Server-Portierung

„Server-Portierung“ ist möglicherweise eine unpassende Bezeichnung für die Arbeitsschritte die, bezüglich des Servers, noch umgesetzt werden müssen. Hierbei handelt es sich vielmehr um die Installation und Konfiguration eines Webservers. Dennoch ist dieser Schritt sehr wichtig, und vor allem notwendig für die Erreichbarkeit des Software- und Informations-Managementsystems.

Es existieren sehr viele Webserver, die für diesen Anwendungsfall eingesetzt werden können:

- Apache
- Cowboy
- Node.js
- Apache Traffic Server
- IdeaWebServer
- Tomcat
- Google Server
- LiteSpeed
- Microsoft-IIS
- Nginx

Bei Betrachtung der Statistik der Webserververbreitung (Abbildung 6.4) wird es deutlich, dass nicht alle Webserver gleichermaßen eingesetzt werden. Der am meisten eingesetzte Webserver ist der Apache HTTP Server, welches ein quelloffenes und freies Softwareprodukt der Apache Software Fondation ist. Dieser wird oft in Kombination mit einem Linux Betriebssystem, MySQL Datenbank und PHP Webprogrammiersprache eingesetzt. Diese Konstellation ist auch unter dem Akronym LAMP bekannt und ist als Gesamtpaket verfügbar. Für die Umsetzung des Servers wird LAMP eingesetzt.

Mittlerweile existieren viele Host-Service-Anbieter, sodass der Aufbau eigener Server-Hardware und damit verbundene Kosten entfallen. Der Service besteht darin, dass einem ein virtueller Server (vServer) mit einstellbaren Parametern (Anzahl der CPU-Kerne, Arbeitsspeicher und Massenspeicher-Kapazität) zur Verfügung gestellt wird. Hierbei handelt es sich um eine preisgünstige Lösung, die die Vorteile in Form von Skalierbarkeit und Erreichbarkeit des Servers mit sich bringt.

Mit der Anpassung der Host-Adresse des PHP-Servers ist Server-Portierung abgeschlossen. Ab diesem Zeitpunkt können die Server-Änderungen via ssh-Verbindung vorgenommen

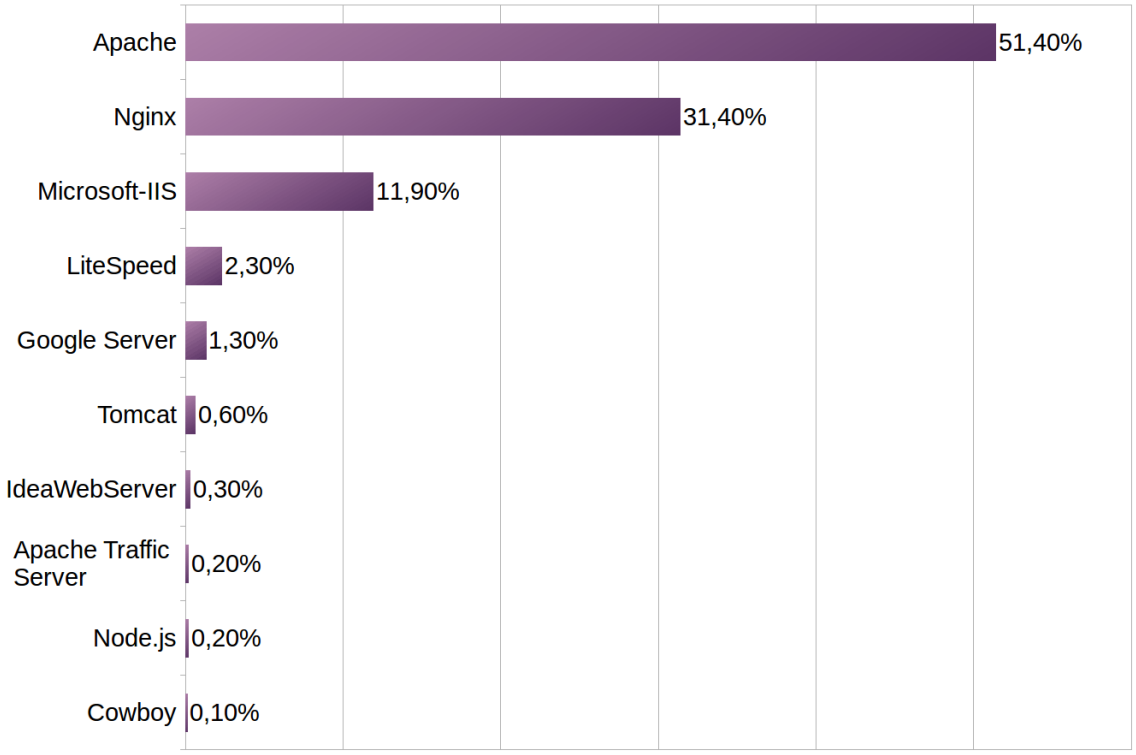


Abbildung 6.4.: Verbreitung der eingesetzten Webservern im Netz [11]

werden. Hierdurch können nur noch Terminal-Editoren (vim, nano und co.) eingesetzt wer-

den. Bei kleinen Änderungen und Anpassungen ist die Vorgehensweise akzeptabel, doch bei ernsthaften Problemen sorgen die Tools wie Eclipse, Sublimetext o.ä. für ein effizienteres Arbeiten. Hierfür existieren einige Tools, die das Mounten entfernter Dateisysteme unterstützten. Konkret wurde das Problem mit dem Linux-Tool SSHFS gelöst. Nach der Installation kann auf der Kommandozeile mit folgendem Befehl das Dateisystem des Servers auf dem lokalen Rechner eingehängt werden.

sshfs␣Benutzername@Servername:/pfad_auf_dem_server/␣ /Mountstelle

Nun wirken sich alle Änderungen in dem Ordner „Mountstelle“ auf die Serverdateien aus.

Diese Vorgehensweise ist universal einsetzbar und bietet sich bei der Manipulation entfernter Zielsysteme an.

6.2.2. Client-Portierung

Die Portierung der Client-Software fordert etwas mehr Aufwand als die des Servers. Im Allgemeinen werden die Quellen, aus den der Simulator gebaut wurde, mit einer anderen Toolchain crosskompiliert, sodass die dadurch entstandene Binary mit der Architektur des Clients kompatibel ist. Damit der Kompiliervorgang fehlerfrei abgeschlossen werden kann, muss das Betriebssystem die benötigten Bibliotheken beinhalten. Die verwendete Komprimierungsbibliothek *quazip* wurde statisch gelinkt, sodass diese auch ohne Systemunterstützung genutzt werden kann. Im Gegensatz dazu muss das System eine der Implementierungen der Verschlüsselungsbibliothek SSL zu Verfügung stellen.

Openssl ist eine der bekanntesten SSL-Implementierungen. Diese kann bei der Konfiguration des Root-Filesystems des Clients eingebunden werden. Außerdem können Implementierungen der Algorithmen für symmetrische (AES) und asymmetrische (RSA) Verschlüsselung im Linux-Kernel aktiviert werden. Diese können sogar abhängig von der Prozessorfamilie durch vorhandene Hardware beschleunigt werden. Um diese Konfigurationen des Kernels und Filesystems durchzuführen, können die *.config* Dateien des jeweiligen Systems direkt angepasst werden. Eine wesentlich komfortablere Variante ist die Verwendung der schon vorhandenen Konfigurationstools. Durch Verwendung dieser werden letzten Endes dieselben *.config* Dateien manipuliert mit dem Unterschied, dass die Änderungen mit Hilfe eines pseudo-graphischen[1] Menüs viel einfacher vorgenommen werden können. *Make menuconfig* ist einer der Tools, das für die Konfiguration, sowohl der Kernels als auch des Buildroots verwendet werden kann. Abbildung 6.5 zeigt, welche Module der Krypto-Bibliothek des Kernels aktiviert sind.

```
/ # dmesg | tail
usb 1-2: USB disconnect, device number 2
usb 1-2: new high-speed USB device number 3 using atmel-ehci
usb 1-2: New USB device found, idVendor=0b05, idProduct=1723
usb 1-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-2: Product: 802.11 bg WLAN
usb 1-2: Manufacturer: Ralink
usb 1-2: reset high-speed USB device number 3 using atmel-ehci
ieee80211 phy0: rt2x00_set_chip: Info - Chipset detected - rt: 2573, rf:
    0002, rev: 000a
ieee80211 phy0: Selected rate control algorithm 'minstrel_ht'
usbcore: registered new interface driver rt73usb
```

Listing 6.8: Ausgabe des Kernel-Ringpuffers

Neben den notwendigen Bibliotheken müssen im Kernel auch bestimmte Treiber aktiviert werden, die die Anbindung der Hardware für die kabellose Netzwerkkommunikation unterstützen. Hierfür sind die Informationen über der vorhandener Hardware notwendig. Diese

[1] Textbasierte Menüs, die in einem Terminal dargestellt werden können.

können aus dem Ringpuffer des Client-Kernels, mit dem Kommando *dmesg*, extrahiert werden. Anhand des Listings 6.8 können notwendige Hardwareinformation gewonnen werden.

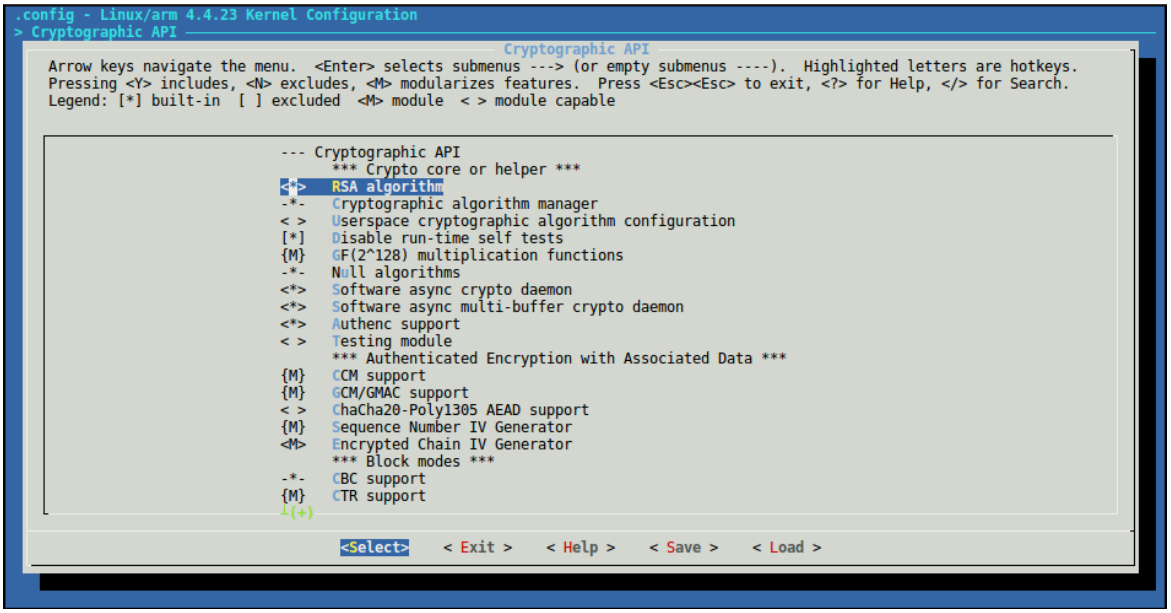


Abbildung 6.5.: Make menuconfig Kernel

Nun ist es klar, dass im Kernel, unter dem Pfad Device Drivers -> Network device support -> Wireless LAN -> Ralink driver support, Ralink-Treiber aktiviert werden müssen. Hiermit ist die Kernelvorbereitung abgeschlossen.

Die Openssl-Bibliothek lässt sich analog der Kernel-Konfiguration im Rootfilesystem aktivieren (Abbildung 6.6).

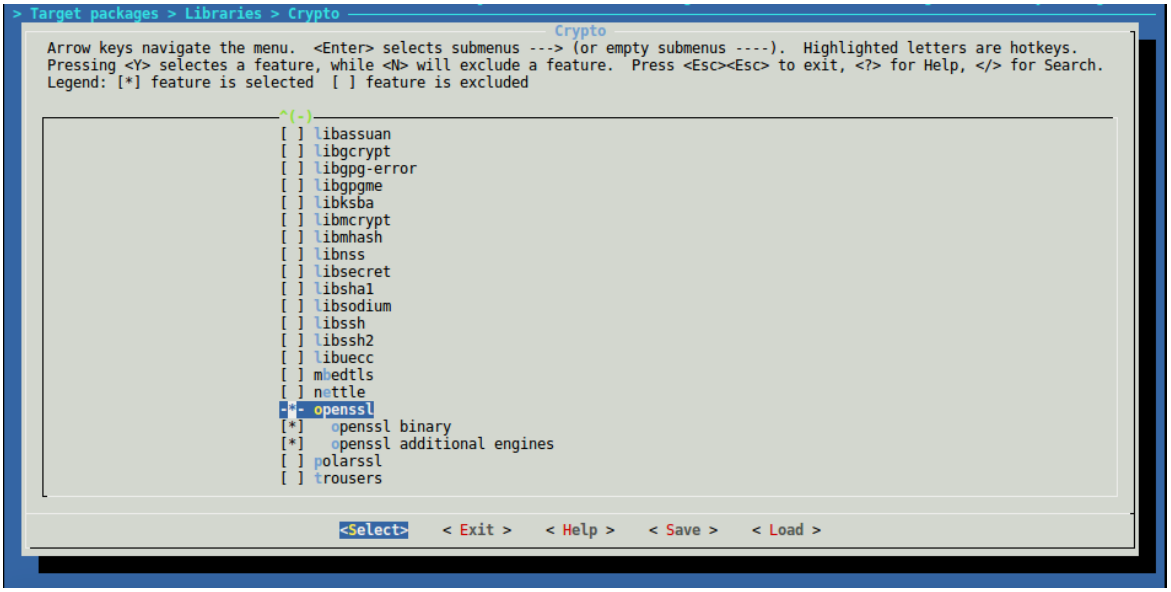
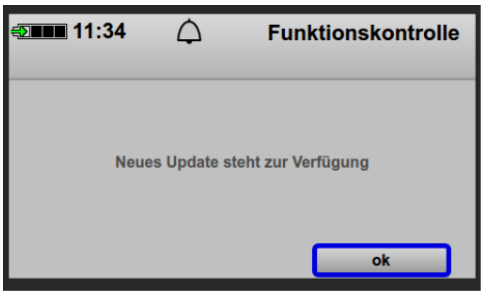


Abbildung 6.6.: Make menuconfig Rootfilesystem (Buildroot)

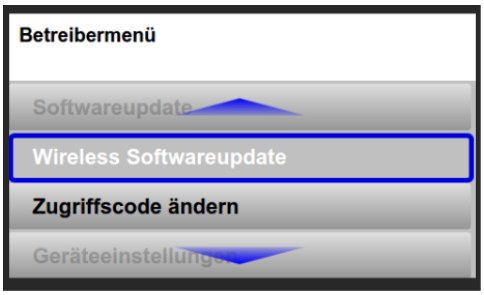
Sobald alle Änderungen des Kernels und Rootfilesystems vorgenommen wurden, können diese gebaut und zusammengeführt werden.

Prinzipiell ist der Client nun fähig, über den WLAN-Modul mit der „Außenwelt“, via Internet zu kommunizieren. Es muss lediglich der *wpa_supplicant* Daemon gestartet werden und mit Hilfe des DHCP-Clients *udhcpc* eine neue IP-Adresse bezogen werden. Optional kann die Verbindung mit mit *ping server_name* getestet werden. Hiermit wurde die Portierung des Clients abgeschlossen, sodass die Testphase unter Produktivumgebung beginnen kann.

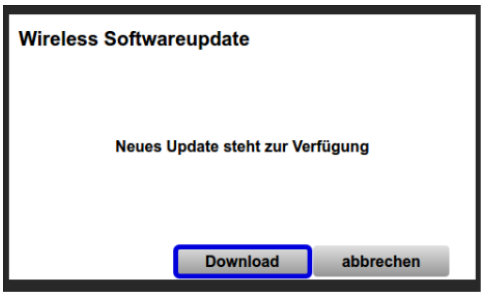
Abbildung 6.7 zeigt den sequenziellen Ablauf bei der Verwendung des webbasierten Software- und Informations-Managementsystems.



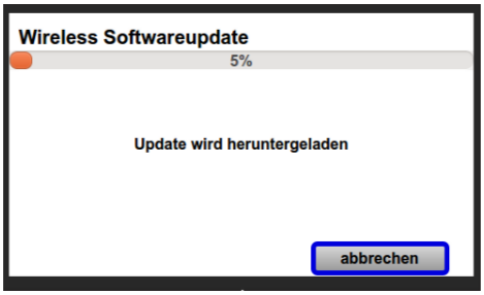
(a) Ergebnis der Synchronisation mit dem Server in der Funktionskontrolle



(b) Wireless Update im Betreibermenü



(c) Im Betreibermenü kann die neue Software heruntergeladen werden



(d) Der Download-Vorgang

(e) Nach dem Download kann die neue Software installiert werden

Abbildung 6.7.: Volle Sequenz bei einem Software-Update

Besonderheiten der Client-Portierung

Bei dem Client handelt es sich um ein linux-basiertes eingebettetes System. Dies bedeutet nicht, dass alle Funktionalitäten des Kernels und des Rootfilesystems „out of the box“ zu Verfügung stehen. Es handelt sich um ein optimiertes System, welches nur die notwendigen Funktionalitäten unterstützt. Fehlende Bibliotheken können nicht einfach via „apt-get install“ nachgerüstet werden. Alle „Extras“ müssen in den Quell- oder Konfigurationsdateien von Hand aktiviert werden. Hinzu kommt die Erschwernis, dass es nicht immer klar ist welche Module aktiviert werden müssen, damit die gewünschte Funktionalität unterstützt wird. Fraglich ist auch ob die benötigte Bibliothek im Kernel/Rootfilesystem vorhanden ist oder ob diese erst besorgt und die Applikation gegen diese statisch gelinkt werden muss. Da es sich aber nicht um eine Standard-X11-Applikation handelt, ist es sehr wahrscheinlich, dass die Bibliothek nur als Quelldateien zu Verführung steht, sodass diese vorerst mit der passenden Toolchain übersetzt werden muss.

Auch die Verwendung der Qt-Bibliothek kann nicht als “straightforward“ angenommen werden. Damit Ressourcen nicht verschwendet werden, enthält die Embedded-Variante nicht den vollen Funktionsumfang. Demzufolge impliziert ein fehlerfreier Kompiliervorgang der Desktop-Version der Applikation nicht einen fehlerfreien Kompiliervorgang der Embedded-Version der Applikation. Ein Beispiel hierfür ist die Verwendung des Fortschrittsbalken beim Herunterladen einer neuen Firmware. Die Verwendung der Klasse QProgressBar funktionierte in der Desktop-Variante auf Anhieb, wohingegen die besagte Klasse in der Embedded-Variante nicht enthalten war. Damit die Embedded-Applikation wieder fehlerfrei gebaut werden konnte, musste die Qt-Bibliothek neu konfiguriert und gebaut werden.

Die Anpassung des Kernels, Buildroots und allen notwendigen Bibliotheken sollte achtsam angegangen werden, denn ein Clean-Build der gesamten Firmware für das Zielgerät dauert in etwa 4 Stunden.

7. Bewertung

Die Bewertung der Ergebnisse erfolgt durch den Vergleich der vorgegebenen Ziele und der tatsächlich umgesetzten Funktionalitäten. Die Tabelle 7.1 verschafft einen Überblick über vorhandene und angeforderte Funktionalitäten des Systems. Es wurden alle Anforderungen

Anforderung	**Status**
Server	
Client-Informationen entgegen nehmen	✓
Client-Dateien entgegen nehmen	✓
Clientdaten in einer Datenbank sichern	✓
Verschlüsselte Kommunikation mit dem Client	✓
Neue Client-Optionen freischalten	✓
Binärdateien an Clients übermitteln	✓
Client	
Kabellose Internetanbindung	✓
Verschlüsselte Kommunikation mit dem Server	✓
Client-Informationen an Server übermitteln	✓
Client-Dateien an Server übermitteln	✓
Dateien komprimieren	✓
Binärdateien herunterladen	✓
Anpassung der Client-GUI	✓

Tabelle 7.1.: Bewertung der Ergebnisse

erfüllt. Nichtsdestotrotz handelt es sich weder bei dem Server noch bei dem Client um eine serienreife Software die live geschaltet werden darf. Vielmehr wurde durch die Umsetzung des Konzeptes, die Machbarkeit des Systems, inklusive Client-Demonstrators überprüft.

Die Umsetzung aller Anforderungen hat sich nicht als trivial erwiesen. Insbesondere ist ein breites Wissensspektrum für die Umsetzung gefordert:

- Server
  - PHP
  - CodeIgniter
  - Webserver
  - Datenbanken
- Client
  - C++
  - Qt
  - CMake
  - Konfiguration des Linux-Kernels
  - Konfiguration der Rootfilesystems
  - Qt Config & Build
  - Subversion (Versionsverwaltungstool)
- Kommunikation
  - Kryptographie allgemein
  - SSL/TLS Verwendung mit Qt/PHP
  - Netzwerkkommunikation
  - Allgemeine Kommunikation mit dem Zielgerät/Server (Avahi, ssh, Remote-Mount)

Es ist ersichtlich, dass es sich um verschiedene Gebiete handelt, die nicht unbedingt mit einander eng verwandt sind.

8. Ausblick

Sowohl die Server- als auch die Client-Seite weist sinnige, teilweise notwendige, Optimierungs- und Erweiterungsmöglichkeiten auf.

Server

Serverseitig ist ein Webaccess notwendig, damit Geräteoptionen aktiviert werden können, ohne dass der direkte Datenbankzugriff erforderlich ist. Eine Webseite, auf der die Datensätze angezeigt werden, ist ebenfalls von großer Bedeutung. In allgemeinen muss geprüft werden, welche Funktionalitäten der Server beinhalten müsste, durch die die Service-Arbeitsabläufe optimiert werden können. Die Gerätedateien wie Log- und Debug-Dateien werden aktuell einfach in der Datenbank abgelegt. Dies ist zwar ein Fortschritt im Vergleich zu den Lösungen, die aktuell Medizintechnikhersteller anbieten, dennoch lässt sich der Server um hilfreiche Funktionen erweitern, wie zum Beispiel:

- automatische Dekomprimierung der Geräte-Dateien
- automatische Auswertung der Log- bzw. Debug-Dateien
- automatische Benachrichtigung des Produktverantwortlichen (z.B. per E-Mail) über bestimmte Fehlverhalten der Clients

Es müsste auch eine Möglichkeit geben, Restriktionen bezüglich des Softwareupdates einzuführen, durch die Clients von bestimmten Typen und mit bestimmter Seriennummer von Softwareupdates mit bestimmter Version ausgeschlossen werden können. Die Motivation für solch einen Feature könnten unterschiedliche Sicherheits- bzw. Norm-Vorgaben sein, die auf internationaler Basis auseinander driften können.

Client

Auf der Seite des Clients können einige Teile der Initialisierung automatisiert werden. Hierzu zählt die treiberseitige Anbindung des WLAN-Moduls. Aktuell muss der WLAN-Stick nach

dem Bootvorgang von Hand eingebunden werden. Anschießend muss der DHCP-Client-Deamon gestartet werden. Des weiteren beinhaltet die Client-Software keinen Netzwerk-Manager, mit dessen Hilfe nach Netzwerken gesucht werden kann. Das Gerät versucht sich immer in ein Netzwerk zu registrieren, dessen SSID und Key vordefiniert wurden. Solch ein Verhalten ist für Demonstrationszwecke akzeptabel, für die Praxis und später für die Seriensoftware äußerst unbrauchbar.
Eine wesentlich wichtigere Voraussetzung ist die Netzwerkfähigkeit der Clients in Form von onboard WLAN- bzw. LTE-Modulen. Diese war bei dem Gerät, dessen Software im Rahmen dieser Arbeit modifiziert wurde, nicht vorhanden. Das Problem wurde durch die Erweiterung der Peripherie des Gerätes über einen USB-Debug-Adapter gelöst.

Wie schon erwähnt, handelt es sich um den Aufbau eines Demonstrators, sodass sowohl die Verifizierung als auch die Validierung der Software entfallen ist.

9. Fazit

Durch diese Arbeit wurden die Vorteile, die ein Software- und Informations-Managementsystems bietet, dargestellt. Insbesondere könnte das Medizintechniksegment, durch Einführung dieser Techniken profitieren. Eine Zentralisierung bei der Verwaltung der Medizingeräte könnte sowohl den Herstellern als auch den Kunden und damit auch den Patienten große Vorteile und Risikominimierung bieten. Die Nutzung eines solchen Systems schließt das Auftreten eines Gerätefehlers nicht aus, es unterstützt vielmehr den Hersteller das Wiedereintreten des Fehlers zu verhindern. Es kann für eine höhere Effizienz bei der Pflege der Serienprodukte sorgen. Dies umfasst auch kürzere Latenzzeiten zwischen den Eintreten des Fehlers und dessen Lokalisierung und Behebung.

Diese Basisentwicklung kann durch Implementierung weiterer Features erweitert werden. Denkbar ist auch die Erweiterung in der Richtung der Telemedizin, die die Anwesenheit eines Arztes am Unfallort überflüssig macht, ohne dass die Qualität der Patientenbehandlung beeinflusst wird.

Trotz des erfüllten Anforderungsprofils darf das entwickelte System nicht als ein Serienprodukt betrachtet werden. Es soll vielmehr als ein Fundament für die Weiterentwicklung dienen, welches in der Medizintechnik für mehr Sicherheit sorgt.

Mehr Sicherheit in der Medizintechnik bedeutet mehr Sicherheit für den Patienten.

Literaturverzeichnis

[1] BAUER, Friedrich L.: *Entzifferte Geheimnisse. Methoden und Maximen der Kryptologie*. Springer, 2000. – ISBN 3-540-67931-6

[2] BLESS, Roland: *Sichere Netzwerkkommunikation. Grundlagen, Protokolle und Architekturen*. Springer, 2005. – ISBN 3-540-21845-9

[3] BOEHM, Hans-J.: *A garbage collector for C and C++*. 2003. – URL https://www.hboehm.info/gc/. – Eingesehen am 22.11.2016

[4] BUNDESMINISTERIUM DER JUSTIZ UND FÜR VERBRAUCHERSCHUTZ: *Medizinproduktegesetz - MPG*. 2016. – URL http://gesetze-im-internet.de/mpg/. – Eingesehen am 21.10.2016

[5] CPLUSPLUS.COM: *cplusplus.com*. 2016. – URL http://www.cplusplus.com/. – Eingesehen am 22.10.2016

[6] DIN EN 60601-1: *Medizinische elektrische Geräte – Teil 1: Allgemeine Festlegungen für die Sicherheit einschließlich der wesentlichen Leistungsmerkmale*. Dezember 2013

[7] DIN EN 62304: *Medizingeräte-Software – Software-Lebendzyklus-Prozesse*. Oktober 2016

[8] GRADY BOOCH, Ivar Jaconson: *The Unified Modeling Language User Guide*. Addison-Wesley, 1999. – ISBN 0-201-57168-4

[9] INTERNET SECURITY RESEARCH GROUP: *Let's Encrypt*. 2016. – URL http://letsencrypt.org/stats. – Eingesehen am 10.10.2016

[10] JOAN DAEMEN, Vincent Rijmen: *AES: The Advanced Encryption Standard*. Springer, 2002. – ISBN 3-540-42580-2

[11] Q-SUCCESS: *W3Techs*. 2016. – URL http://www.W3Techs.com/. – Eingesehen am 09.11.2016

[12] RED BEND SOFTWARE: *FOTA Best Practices*. 2013. – URL http://redbend.com/data/upl/whitepapers. – Eingesehen am 01.11.2016

[13] SINGH, Simon: *Geheime Botschaften*. dtv, 2001. – ISBN 3-423-33071-6

[14] STEVE BURNETT, Stephen Paine: *Kryptographie RSA Security's Official Guide*. mitp-Verlag, 2001. – ISBN 3-8266-0780-5

[15] STROUSTRUP, Bjarne: *Die C++-Programmiersprache*. Addison-Wesley, 2009. – ISBN 978-3-8273-2823-6

[16] THE APACHE SOFTWARE FOUNDATION: *The Apache Software Foundationg*. 2016. – URL http://www.apache.org/. – Eingesehen am 22.10.2016

[17] THE QT COMPANY LTD: *Qt 4 Documentation*. 2016. – URL http://doc.qt.io/qt-4.8/. – Eingesehen am 22.10.2016

[18] WEINMANN EMERGENCY: *Gebrauchsanweisungen*. 2016. – URL http://weinmann-emt.de. – Eingesehen am 21.10.2016

[19] WOLF, Jürgen: *Qt 4.6 - GUI-Entwicklung mit C++*. Galileo Computing, 2010. – ISBN 978-3-8362-1542-8

[20] ZOLL MEDICAL CORPORATION: *ZOLL AED Pro Bedienerhandbuch Rev G*. 2012. – URL http://zoll.com. – Eingesehen am 01.11.2016

[21] ZOLL MEDICAL CORPORATION: *ZOLL X Series Bedienerhandbuch Rev F*. 2012. – URL http://zoll.com. – Eingesehen am 01.11.2016

A. Quellcode

Der Anhang zu Arbeit befindet sich auf DVD und ist einzusehen bei Prof. Dr. rer. nat. Henning Dierks.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 22. Dezember 2016	
Ort, Datum	Unterschrift