Bachelorarbeit

Informatik-Ingenieurwesen

Reverse Engineering eines Kaffeevollautomaten

von

Niklas Joachim Eberhard Krüger

Februar 2019

Betreut von

Florian Meyer
Institut für Telematik, Technische Universität Hamburg

Erstprüfer	Prof. Dr. Volker Turau Institut für Telematik Technische Universität Hamburg
Zweitprüfer	Florian Meyer Institut für Telematik Technische Universität Hamburg

Institut für Telematik
Technische Universität Hamburg
TUHH

Eidesstattliche Erklärung

Ich, NIKLAS JOACHIM EBERHARD KRÜGER (Student im Studiengang Informatik-Ingenieurwesen an der Technischen Universität Hamburg, Matr.-Nr. 21491319), versichere an Eides statt, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, 22. Februar 2019

Niklas Joachim Eberhard Krüger

Inhaltsverzeichnis

Abkürzungsverzeichnis

EEPROM Electrically Erasable Programmable Read-Only Memory 1
RAM Ramdom-Access Memory 1
MCU Microcontroller Unit 5
SPI Serial Peripheral Interface 5
UART Universal Asynchronous Receiver Transmitter 5
API Programmierschnittstelle 9
POSIX Portable Operating System Interface 11
JSON JavaScript Object Notation 11
HTML Hypertext Markup Language 12
CSS Cascading Style Sheets 12
JS JavaScript 12
AJAX Asynchronous JavaScript and XML
CDN Content Delivery Network

Kapitel 1

Einführung

1.1 Motivation

Heutige Geräte versprechen viel Komfort und eine einfache Handhabung. Über das Internet werden die Geräte zunehmend vernetzt und smart. Diese Arbeit befasst sich mit einer Kaffeemaschine und in erster Linie mit ihrem Speicher. Dieser merkt sich nicht nur Betriebszustände, sondern auch Einstellungen, Zählstände und evtl. vieles mehr. Diese Informationen können nicht nur gelesen, sondern zum Teil auch verändert werden. Moderne Funktionen stecken ohne eine entsprechende graphische Schnittstelle aber bereits in älteren Maschinen, unter der Voraussetzung über das entsprechende Wissen zu verfügen.

!!!ToDo!!! Deutlich mehr ausholen: Was ist IoT? Warum braucht man das? Warum ist das relevant für diese Arbeit? Und dann zur Aufgabenstellung überleiten.

1.2 Aufgabenstellung

Anhand eines *Jura Impressa S9 Kaffeevollautomaten* soll der Speicher untersucht werden. Welches Wort / Byte / Bit speichert welche Information? Welche Bedeutung haben diese auf den Betrieb? Hierfür sollen Skripte erstellt werden und der Speicher systematisch untersucht werden.

Wenn die Denkweise der Kaffeemaschine bekannt ist, sollen Werte im Electrically Erasable Programmable Read-Only Memory (EEPROM) abgefragt, aber auch gezielt verändert werden, sowie Statusinformationen aus dem Ramdom-Access Memory (RAM) ausgelesen werden. Die erhaltenen Rohinformationen werden nach den gewonnen Erkenntnissen aufbereitet und stehen so für weitere Projekte zur Verfügung.

Als kleine Demonstrationen entsteht am Ende ein Programm, welches Profile anlegen kann, sodass jeder Nutzer auf Knopfdruck einen Kaffee nach seinen Lieblings Präferenzen zubereitet bekommt.

!!!ToDo!!! Aufbau der Arbeit beschreiben: In Kapitel 2 werden grundlegende Begriffe erklärt und ein Einblick in den Stand der Technik gegeben...

Kapitel 2

Grundlagen

2.1 Begrifflichkeiten

Der Titel dieser Arbeit lautet `Reverse Engineering eines Kaffeevoll-automaten`, aber hinter der Terminologie des Begriffs *Reverse Engineering* steckt ein ganzes Spektrum an Bedeutungen. E. J. Chikofsky und J. H. Cross differenzieren in ihrem Paper [CC90] die Begriffe *Forward Engineering*, *Reverse Engineering*, *Redocumentation*, *Design Recovery*, *Restructuring*, und *Reengineering*.

Zugrunde liegt ein Produkt, welches während seiner Entwicklung mehrere Lebenszyklen durchlaufen hat. Kassem A. Saleh beschreibt in seinem Buch [Sal09] ausführlicher Entwicklungsaktivitäten, wie die Anforderungsanalyse, das Design, die Implementation, die Tests, die Installation und den Einsatz. Während der Implementation nennt er Vorgehensmodelle der Softwareentwicklung, wie das Wasserfallmodell, den Prototypenbau, das Spiralmodell, den Objektorientierten Ansatz, das inkrementelle und iterative Modell, sowie das agile Modell. Jede (Hardware- und) Softwareentwicklung durchläuft dabei, unabhängig vom Modell, die verschiedenen Entwicklungsaktivitäten. Das voranschreiten zur nächsten Stufe, bis zur Fertigstellung des Produkts, stellt das *Forward Engineering* dar.

Der zweite Begriff des *Reverse Engineering* führt in die Gegenrichtung. Aus dem implementierten Produkt wird auf das Design, bzw. aus dem Design auf die Spezifikationen geschlossen. Dabei werden zum einen die Komponenten und ihr Zusammenspiel identifiziert, zum anderen wird aber immer eine höhere Abstraktionsebene, eine vorherige Stufe, rekonstruiert. Ein wichtiges Zitat, welches in Abschnitt 6.3 aufgegriffen wird, lautet: „Reverse engineering in and of itself does not involve changing the subject system or creating a new system based on the reverse-engineered subject system. It is a process of examination, not a process of change or replication.“[CC90]

Redocumentation arbeitet auf einer Ebene und bringt primär eine andere Darstellung. Das Paper nennt „dataflow“, „data structure“ und „control flow“ als Beispiele, die über Werkzeuge wie Diagramm Generatoren, Syntaxhervorhebung und Querverweis Generatoren erzeugt werden können. Das Kernziel sei es die Zusammenhänge und Ablaufpfade hervorzuheben.

Breiter angelegt ist das *Design Recovery*, das für die Designwiederherstellung externe Informationen, Schlussfolgerungen, und Unschärfelogik mit einbezieht, um sinnvolle Abstraktionen auf höherer Ebene zu identifizieren, welche nicht aus dem System selbst hätten gewonnen werden können.

Restructuring umfasst das Nachbauen einer Darstellung innerhalb einer Ebene. Funktionalität und Semantik bleiben erhalten, während die Darstellung umgestaltet wird. Als Beispiel wird die Umstellung von unstrukturiertem Spaghetti Code zu strukturiertem „goto“ freien Code genannt. Der Begriff umfasst aber auch Datenmodelle, Entwurfsmuster und Anforderungsstrukturen. Dabei genügt das Wissen über die Struktur, ohne die Bedeutung dahinter zu verstehen, beispielsweise können „if“ und „case“ Ausdrücke ineinander überführt werden, ohne zu verstehen wann welcher Fall eintritt. Normalerweise werden daher ohne Anpassung der Spezifikation keine Veränderungen vorgenommen. *Restructuring* ist oft eine Form der präventiven Instandhaltung.

Zuletzt wird der Begriff *Reengineering* oder auch „Renovation and Reclamation“ als nachträgliche, neu erstellte Form beschrieben. Dafür geht *Reverse Engineering* zum abstrakten Verständnis dem *Forward Engineering* oder *Restructuring* voraus. Beim *Reengineering* sind Anpassung der Spezifikation und Veränderungen durchaus möglich. Ähnlich zum *Restructuring* verändert das *Reengineering* die unterliegende Struktur ohne die Funktionalität zu beeinträchtigen. Jedoch passiert es selten, dass beim *Reengineering* keine weiteren Funktionalitäten hinzugefügt werden. Damit ist der Begriff *Reengineering* allgemeiner gefasst als das *Restructuring*. Aber *Reengineering* ist kein Überbegriff für *Reverse Engineering* und *Forward Engineering*, nur weil es beides beinhaltet. Beide Disziplinen entwickeln sich unabhängig vom *Reengineering* weiter.

Auch K. A. Ingle [Ing94] grenzt das Reverse Engineering von anderen Konzepten ab. Das *Reengineering* wird, im Gegensatz zu dem *Reverse Engineering*, beschrieben als Neustrukturierung des Programmcodes ohne funktionale Änderungen am System. Diese Beschreibung passt zum dem Begriff des *Restructuring* aus dem Paper von E. J. Chikofsky und J. H. Cross [CC90].

Beim *Simultaneous Engineering* greifen während einer Produktentwicklung mehrere Arbeitsabläufe frühzeitig verzahnt ineinander, um einen Zeitgewinn gegenüber eines traditionellen Projektablaufs zu erzielen. Die schnelle und einfache Reproduzierbarkeit sei beispielsweise nicht das direkte Ziel des *Reverse Engineering*, sondern ein Ziel des *Simultaneous Engineering*, schreibt K. A. Ingle. Nichts desto trotz könne aber die Idee des *Simultaneous Engineering* in das *Reverse Engineering* aufgenommen werden.

Eine weitere Differenzierung findet zwischen dem *Hardware-* und dem *Software Reverse Engineering* statt. *Software Reverse Engineeging* verfolge das Ziel aus dem Programm Code ein übergeordnetes Design oder Spezifikationsangaben zu extrahieren. *Hardware Reverse Engineeging* hingegen befasst sich mit dem Fertigungs- oder Produktionssystem. Das Buch von 1994 sah aber schon ein Zusammenwachsen beider Bereiche durch den Einsatz und

die Einbettung von immer mehr Software in Hardwaresystemen kommen. Die Relevanz des Reverse Engineering würde hoffentlich weiter steigen um keine vorhandenen Softwaresysteme neu bauen zu müssen. K. A. Ingle schreibt: „The aim of reverse engineering is to increase productivity through improved documentation.“[Ing94]

Diese Arbeit greift den Gedanken auf und versucht den Einfluss und die Bedeutung einzelner Speicherstellen für weitere Arbeiten zu dokumentieren.

2.2 Stand der Technik

Eine Bachelorarbeit der Universität Magdeburg zum Thema „Reverse-engineering a De'Longhi Coffee Maker to precisely bill Coffee Consumption“[Off18] behandelt eine De'Longhi Caffee Maschine. Das Ziel ist es den Verbrauch exakt zu bestimmen und das System damit um ein Abrechnungssystem zu erweitern. Dafür wird eine Microcontroller Unit (MCU) über das Serial Peripheral Interface (SPI) zwischen Master und Slave Einheiten geschaltet und Informationen aus dem proritären De'Longhi Protokoll ausgelesen. Das Ergebnis der Arbeit ist unter anderem ein Verständnis über das interne Bus Protokoll der Maschine.

Ein Buch von Eldad Eilam [Eil05] stellt den Begriff des Reverse Engineering ähnlich wie zu Beginn in Abschnitt 2.1 dar, geht aber im Folgenden tiefer auf das Reverse Engineering von Low-Level Software ein. Ein Hauptaugenmerk des Buches liegt auf auf der sicherheitsrelevanten Seite, sowie die Vorgänge und Lücken im Low-Level Bereich.

Diese Arbeit hingegen nutzt als Ansatz eine gegebene serielle Schnittstelle des Jura Kaffeevollautomaten mit einem ebenfalls propritären Universal Asynchronous Receiver Transmitter (UART) Protokoll. Dabei wird nicht die Hardware geöffnet und beispielsweise die Firmware disassembliert und analysiert.

!!!ToDo!!! Erweitern um smart appliances (s. 3x Paper)

Kapitel 3

Hardware und Software

Dieses Kapitel führt die gegebenen und verwendeten Hard- und Software Komponenten in dieser Arbeit auf.

3.1 Der Kaffeevollautomat

Der „Jura Impressa S9“, siehe Abbildung 3.1(a), ist ein Kaffeevollautomat mit fünf Kaffeebezugstasten: Spezialkaffee, 1 große Tasse Kaffee, 2 große Tassen Kaffee, 1 kleine Tasse Kaffee und 2 kleine Tassen Kaffee. Auf der rechten Seite befinden sich Bedienelemente für heißes (Tee-)Wasser und Wasserdampf zum Milch Aufschäumen. Der Kaffeevollautomat wird für seinen Betrieb direkt mit der Netzspannung versorgt.

3.1.1 Aufbau und Verkabelung

Hinter der linken Wartungsklappe an der Vorderseite des Kaffeevollautomaten befindet sich neben mehreren Menü-Tasten eine serielle Schnittstelle. Über ein eigenes UART Protokoll kann hierüber mit der Maschine kommuniziert werden.

Abbildung 3.1(b) illustriert die Pinbelegung. Die 5 Volt Leitung kann für ein autark laufendes Projekt genutzt werden. In dieser Arbeit bezieht der Arduino seine Versorgungsspannung über den am USB Kabel befindlichen Computer. Von `TX` nach `RxD` werden Befehle an den Kaffeevollautomaten verschickt. Auf der Rückrichtung von `TxD` nach `RX` werden Antworten des Kaffeevollautomaten gelesen. `GND` ist abschließend die gemeinsame Erdung und Bezugsleitung für die serielle Kommunikation. Auf der Abbildung kreuzen sich die Leitungen `RxD` und `GND` am Stecker auf Seiten des Kaffeevollautomaten. `GND` ist über eine schwarze Markierung gekennzeichnet.

Ein Arduino Uno übersetzt als „Man in the middle“ die bekannten Kommandos in das Format der Kaffeemaschine. Die serielle Kommunikation läuft über die Pins Nummer 12 und 13. Über eine weitere serielle Verbindung per USB lässt sich der Arduino ansteuern.

Aus Sicht des Computers ist der Arduino ein Gerätelaufwerk unter `/dev/ttyACM0` mit einer Baudrate von 9600. Diese Zahl findet sich zu Beginn des Arduino Uno Skripts wieder.

(a) Der ganze Kaffeevollautomat

(b) Pinbelegung zum Arduino Uno

Abbildung 3.1: Jura Impressa S9

3.1.2 Serielle Kommunikation

Diese Arbeit baut auf das „CoffeeMachine“ Projekt [Kö18] auf und nutzt das Arduino Skript als Grundlage der Kommunikation weiter, ebenso werden Kommandos und erste Speicherstellen aus der Weboberfläche aufgegriffen. Das Arduino Skript kodiert die UART Kommandos von und zu dem Kaffeevollautomaten.

Die Kommandos werden zeichenweise ASCII-Zeichen für Zeichen übertragen. Dafür wird ein an den Kaffeevollautomaten adressiertes Byte (ein ASCII-Zeichen), bestehend aus acht Bits, in je vier Bytes aufgeteilt. Die dritte und sechste Stelle der neuen Bytes repräsentieren je zwei Bits des ursprünglichen Bytes. Bei der Übertragung an den Kaffeevollautomaten werden die restlichen Bits mit Nullen aufgefüllt. Abbildung 3.2 veranschaulicht dies an dem ersten Byte des Einschaltbefehls, die entsprechende ASCII-Kodierung ist in Abbildung 5.1 der zweiten und dritten Spalte zu entnehmen. Die Kodierung der, von dem Kaffeevollautomaten kommenden, Bytes erfolgt analog. Laut „Protocoljura“[1] bestehen die irrelevanten Bits der ankommenden Bytes aus einer Null und weiteren fünf Einsen pro Byte.

Vier Bytes kodieren auf diese Weise ein ASCII Zeichen und werden fortan als Gruppe bezeichnet. Zwischen jeder Gruppe gibt es eine Verzögerung von 8ms. Diese Verzögerung begrenzt hauptsächlich die Übertragungsgeschwindigkeit, was in Abschnitt 6.1.1 diskutiert wird.

[1] http://protocoljura.wiki-site.com/index.php/Protocol_to_coffeemaker

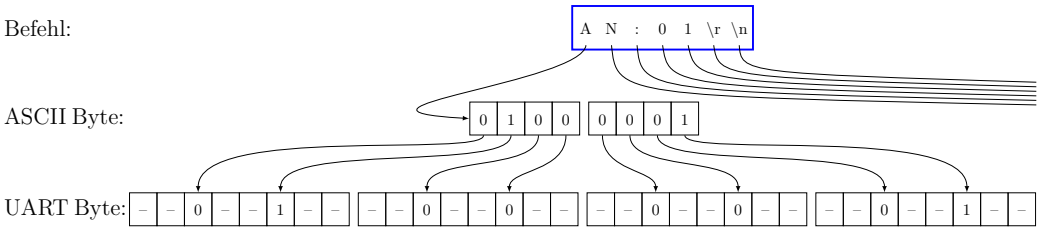


Abbildung 3.2: Umrechnung

3.1.3 Kommandos

Die Tabelle 3.1 zeigt die Befehlsgruppen, sowie ausgewählte Befehle, die der Kaffeevollautomat versteht. Diese Befehle stammen aus dem „CoffeeMachine“ Projekt [Kö18].

Befehle beginnen entweder mit zwei Großbuchstaben gefolgt von einem Doppelpunkt und i.d.R. einer zweistelligen Hexadezimalzahl, oder sie beginnen mit einem Fragezeichen gefolgt von i.d.R. zwei ASCII Zeichen. Positionsnummern, egal ob Speicherposition, Betriebszustands-, Bezugstasten- oder Steuerungskomponenten-Nummer, werden durch zweistellige Hexadezimalzahlen repräsentiert und reichen von $0_{16} = 0_{10}$ bis $\mathrm{FF}_{16} = 255_{10}$.

Zwei Ausnahmen des ersten Typs sind z.B. `TY:` zum Abfragen des Maschinen Typs und `WE:00,01FF` zum Schreiben des Wertes $01\mathrm{FF}_{16} = 511_{10}$ an die Position 00. Eine Ausnahme des zweiten Typs ist z.B. `?D1DISPLAY␣` und `?D2␣␣TEST␣␣`. Der volle Umfang möglicher Displayzeichen, sowie deren Benutzung wird in Abschnitt 5.2.1 ausgeführt.

Einige Befehle, die aus weiteren Projekten mit Maschinen der S-Reihe bekannt sind, sind in dieser „Jura Impressa S9“ leider nicht implementiert. Dazu zählen `IC:` zum Auslesen aller Eingaben, `FA:0A` eine unbelegte Bezugstaste, `CM:` für weitere Status Informationen, `CS:` für Sensor Informationen und `PM:` ein Befehl um Musik abzuspielen. Weitere Befehlsgruppen werden in der `README.rst` eines Github Repositories aufgeführt[2].

Die jetzt vorhandene Umgebung kann bereits über den seriellen Monitor der Arduino IDE genutzt werden.

3.2 Libraries und Frameworks

Diese Arbeit nutzt „libraries“ und „frameworks“. „libraries“ sind thematisch gebündelte Funktionen und Routinen. Sie bieten eine Programmierschnittstelle (API), die das eigene Programm ansprechen kann.

[2] https://github.com/PromyLOPh/juramote

Kommando	Beschreibung	Rückgabewert
Betriebszustand (AN:<id>)		
AN:01	Einschalten	ok:
AN:02	Ausschalten	ok:
AN:03	Display Test	ok:
AN:<id>	u.v.m.	ok:
Bezugstaste (FA:<id>)		
FA:02	Gerät spülen	ok:
FA:0C	Spezialkaffee	ok:
FA:<id>	u.v.m.	ok:
Steuerungskomponenten (FN:<id>)		
FN:<id>	Pumpen, Heizung, u.v.m	ok:
Eingabestatus*		
IC:	Eingaben auslesen*	
Spiele Musik (easter egg)*		
PM:	Play music*	
Speicherzugriff		
RE:<address>	Liest 2 Byte EEPROM Speicher	re:[0000-FFFF]
WE:<address>,<value>	Schreibt 2 Byte in EEPROM	ok:
RT:<address>	Liest eine Zeile EEPROM	rt:[0-F 64x]
RR:<address>	Liest eine Zeile RAM	rr:[0-F 32x]
Sperrung		
?M3	Aktiviert den Inkassomodus	?ok
?M1	Deaktiviert den Inkassomodus	?ok
Display		
?D0	Standard Displaytext zurück setzen	?ok
?D1[A-Z 8-11x]	Displaytext Zeile 1	?ok
?D2[A-Z 8-11x]	Displaytext Zeile 2	?ok
Aktion an der Kaffeemaschine		
	1 kleiner Kaffee (Produkt 1)	?PAE
	2 kleine Kaffees (Produkt 2)	?PAF
	1 großer Kaffee (Produkt 3)	?PAA
	2 große Kaffees (Produkt 4)	?PAB
	Spezialkaffe (Produkt 7)	?PAG
	Dampf (Produkt 6)	?PAI
	1 Dampfportion (Produkt 5)	?PAJ
	1 Tasse Tee (Produkt 8)	?PAK
*** Nicht alle Befehle sind in der Jura Impressa S9 implementiert, siehe weitere Geräte der S-Reihe**		

Tabelle 3.1: Befehlsübersicht der Jura Kaffeevollautomaten (S-Reihe)

„frameworks“ sind noch mehr und bieten ganze Programmiergerüste und Routinen. Sie rufen wenn nötig selbst vorgesehene Funktionen auf und folgen damit dem Paradigma des „Inversion of Control“.

In der aktuellen Version bieten beide nicht nur leicht zugängliche Aufrufe, sondern sind auch sicher und robust in ihrem Gebiet.

3.2.1 Serielle Kommunikation

Bei dem Versuch die Grätedatei direkt anzusprechen kam es zu Problemen, die in Abschnitt 6.1.2 erörtert werden. Für die zuverlässige serielle Kommunikation kommt daher eine geeignete Library zum Einsatz.

libserial

Das selbst entwickelte C++ Programm nutzt „liberial“[3] um sicher und zuverlässig über das Gerätelaufwerk mit dem Arduino (und letztlich dem Kaffeevollautomaten) zu kommunizieren. Diese Bibliothek bietet eine objektorientierte Schnittstelle für alle Portable Operating System Interface (POSIX) Systeme und ist unter Linux über die Paketverwaltung installierbar.

3.2.2 Speicher- und Austauschformat

Sowohl bei der Untersuchung des Speichers, als auch am Ende für das aufbereitete Ergebnis, werden Informationen gesammelt, verglichen und öffentlich zugänglich gemacht. Als beliebtes und flexibles Speicherformat wird JavaScript Object Notation (JSON) verwendet. Es bietet viele Datenformate und ist unbegrenzt verschachtelbar. In dieser Arbeit werden hauptsächlich Zeichenketten, Zahlen, Unter-Objekte und -Arrays verwendet.

Darüber hinaus kann es leicht vom C++ Programm, der Webseite, oder auch für viele weitere Projekte genutzt werden.

libjsoncpp

Die „libjsoncpp“[4] bietet dem C++ Programm eine Lese- und Syntaxanalyse-Funktion zum Aufnehmen eines JSON Ausdrucks, sowie eine Möglichkeit ein JSON Objekt kompakt zusammengefasst oder leserfreundlich aufgefächert in einen Ausgabestrom zu schreiben. Als Ausgabestrom sind reine JSON Dateien nach einer Speicherauszugs Aufnahme oder die Standardausgabe im Gebrauch als API vorstellbar.

[3] https://github.com/crayzeewulf/libserial
[4] https://en.wikibooks.org/wiki/JsonCpp

3.2.3 Webseite

Eine kleine Webseite soll am Ende die ausgelesenen Werte visuell anschaulich präsentieren. Die JavaScript-Bibliothek jQuery und das Framework Bootstrap helfen dabei dies zügig und ansprechend umzusetzen.

Bootstrap

Bootstrap[5] ist ein von den Twitter Entwicklern begonnenes Projekt um ursprünglich intern die Verwaltungswerkzeuge zu vereinheitlichen. Daraus ist aber ein ganzes System an Design Elementen geworden, welches heute sehr populär ist. Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript (JS) werden zusammen eingesetzt und bieten Entwicklern eine Gitteranordnung für eine Mobile- und die Desktop-Ansicht, Inhaltselemente wie Tabellen und Abbildungen oder auch einzelne Komponenten wie Steckkarten, Prozentanzeigen und Knöpfe.

jQuery

jQuery[6] ist eine freie JavaScript-Bibliothek, die in der „slim“-Variante bereits über Bootstrap eingebunden ist. Um später wirklich mit dem Kaffeevollautomaten interagieren zu können, wird unter anderem AJAX aus dem vollständigen jQuery Paket benötigt (auch wenn daraus später ein synchrones JavaScript mit JSON wird).

Über das CDN eingebundene Packet stehen ab jetzt einfache Programmierschnittstellen zum modifizieren der HTML Elemente, ein erweitertes Event-System, sowie AJAX-Funktionalitäten bereit.

Intro.js

Zu guter Letzt folgt mit Into.js[7] noch eine Schritt-für-Schritt-Anleitung, die dem Seitenbenutzer eine kurze Einführung in den Aufbau und die Bedienung der Webseite geben soll.

[5] https://getbootstrap.com/
[6] https://jquery.com/
[7] https://introjs.com/

Kapitel 4

Methodik und Implementierung

4.1 Vorgehen

Die Speicher der Kaffeemaschine, der EEPROM und der RAM, werden Zeilenweise von einem Programm ausgelesen. Dieses Programm setzt die einzelnen Informationen zu einem gesamten Speicherauszug zusammen. Aufeinander folgende Speicherauszüge können dann auf Veränderungen verglichen werden.

Von Hand werden nun Veränderungen angestoßenen, die möglichst elementar sein sollten, um gezielten Aktionen die Werteänderungen bestimmter Speicherzellen zuordnen zu können.

Der nötige Zugriff zum Auslesen kann auf zwei Arten erfolgen: direkt am Speicherstein auf der Hauptplatine oder seriell über die vorhandene UART Schnittstelle, siehe Abschnitt 6.1.1. Im Folgenden erfolgt die Kommunikation über die UART Schnittstelle und mithilfe der *libserial*-Library in einem eigens dafür entwickelten C++ Programm.

4.2 Das C++ Programm „./JuraCoffeeMemory“

Für diese Arbeit wurde ein C++ Programm entwickelt, das die Kommunikation und die Aufschlüsselung der Antworten übernommen hat.

4.2.1 Makefile

In dem Projekt Ordner befinden sich auf der Hauptebene mehrere `CPP`- und `HPP`-, eine `H`- sowie eine `Makefile`-Datei. Sind die in der `Readme.md` genannten Abhängigkeiten und Voraussetzungen erfüllt, kann das Projekt mit dem Befehl `make`, ohne Zusatzangaben gebaut werden. Am Ende sollte dann auf der Hauptebene eine ausführbare Datei namens `./JuraCoffeeMemory` herauskommen. Es werden neben Objekt-Dateien auch noch einige weitere Tools im Unterordner `./tools/` gebaut, auf die später in Abschnitt 4.3 eingegangen wird.

Der Befehl `make clean` entfernt die Objekt-Dateien, mittels `make dist-clean` werden ebenfalls die ausführbaren Programme entfernt und der Projekt Ordner in seinen Ursprungszustand zurück versetzt.

4.2.2 Interaktives Menü

Startet man in einem Terminal das Hauptprogramm `./JuraCoffeeMemory` wird das Hauptmenü angezeigt. Abbildung 4.1(a) visualisiert diesen Zustand. Das Listing 4.1 zeigt einmal schematisch all Menüoptionen.

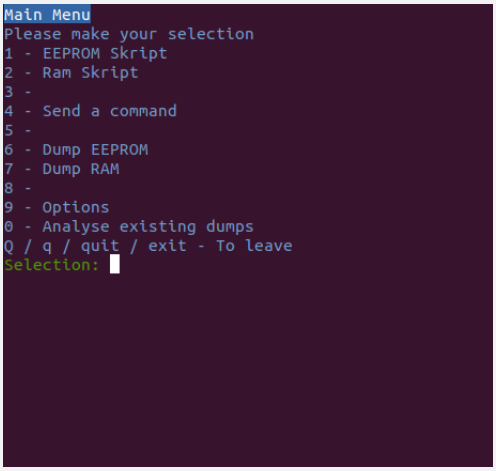
```
Main Menu
|-- 1: EEPROM Skript
|-- 2: Ram Skript
|-- 4: Send a command
|-- 6: Dump EEPROM
|-- 7: Dump RAM
|-- 9: Options
|   |-- 1: Device path (/dev/ttyACM0)
|   |-- 2: EEPROM log file path (data/eeprom.json)
|   |-- 3: Ram log file path (data/ram.json)
|   `-- 4: Back
|-- 0: Analyse existing dumps
|   |-- Analyse Dumps Menu
|   |   |-- data/xxx.json
|   |   |   `-- Dumps
|   |   |-- C: Clear window
|   |   `-- Q: Quit
|   `-- Q: Quit
`-- Q / q / quit / exit: To leave
```

Listing 4.1: Menü-Baum ./JuraCoffeeMemory

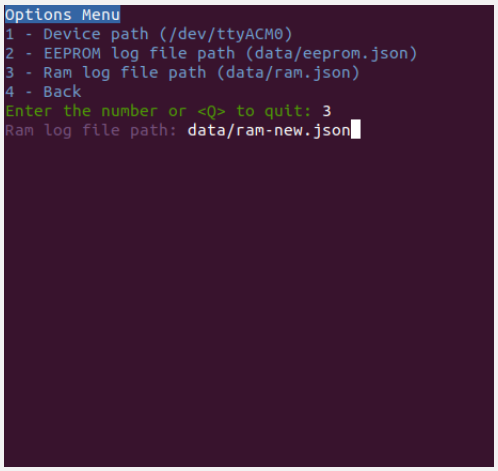
!!!ToDo!!!...

Farbcodierungen

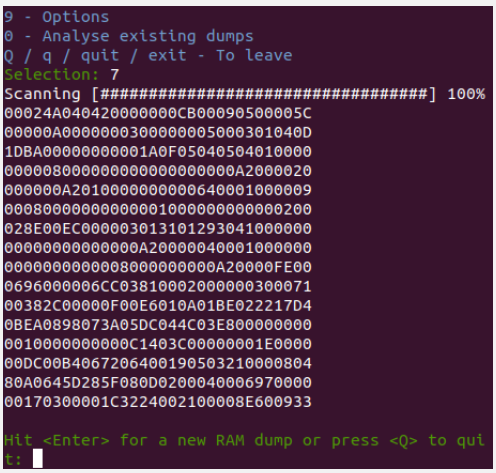
Normale Ausgaben sowie Benutzer Eingaben erscheinen in weißer Schrift. Weiße Schrift auf blauem Grund stellt Menü Überschriften dar. In blauer Schrift sind Menüs dargestellt. Grüne Schrift vor der blinkenden Eingabemarke fordert zur Eingabe auf. Der Text beschreibt valide Eingaben. Abschließend beschreibt roter Text einen Fehler. Die aufgetretene Position wird in weißer Schrift auf rotem Grund hervorgehoben.



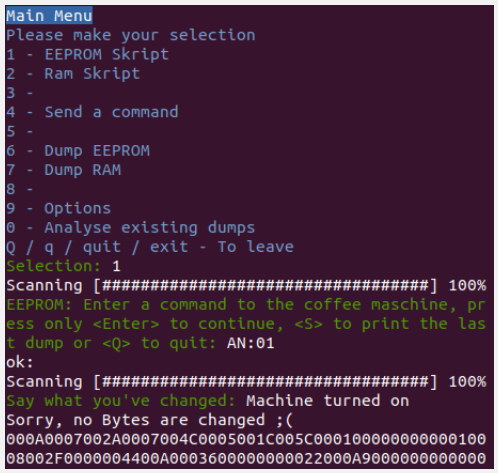
(a) Hauptmenü



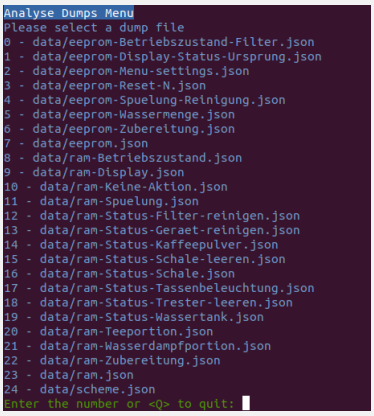
(b) Optionen ⇝ Log Pfad ändern



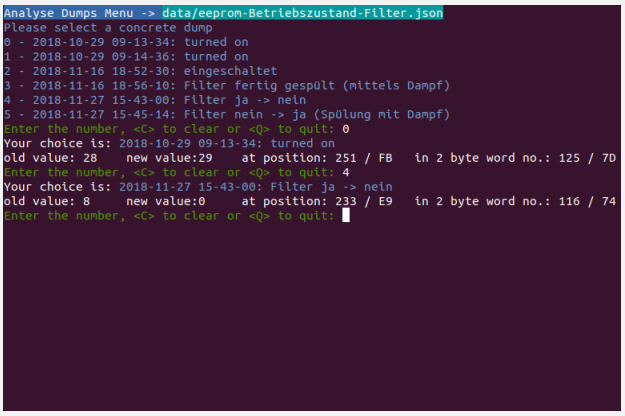
(c) RAM Speicherauszug



(d) EEPROM Skript



(e) Auswertungen



(f) Speicherauszug einsehen

Abbildung 4.1: Interaktives Menü des C++ Programms „./JuraCoffeeMemory“

4.2.3 Die API nach außen

Das Hauptprogramm kann aber auch über vier Parameter aufgerufen werden. Die Ausgaben sind dann im fehlerfreien Fall farblose JSON Zeichenketten oder einfacher Text in die Standardausgabe.

Mögliche Fehlermeldungen sind für Entwickler in der `Readme.md` dokumentiert. Die Fehler Ausgabe erfolgt wie im interaktiven Menü mit Farben. Für Entwickler bietet sich daher der Rückgabewert des Programms an.

./JuraCoffeeMemory command Von der Standardeingabe wird eine Zeichenkette erwartet, die an den Kaffeevollautomaten weiter gegeben wird. Mögliche Eingaben sind die Befehle aus der Tabelle 3.1. Die Antwort des Kaffeevollautomaten erscheint als einfacher Text.

./JuraCoffeeMemory eepromWrite Bei diesem Aufruf wird von der Standardeingabe ein JSON Objekt erwartet. Alle dort genannten Bezeichnungen werden abgearbeitet und die neuen Werte im Speicher des Kaffeevollautomaten hinterlegt. Auf der obersten Ebene müssen sich aus `EEPROM_Status::getEntriesEEPROM()` bekannte Bezeichnungen befinden, die ein Unterobjekt mit der Bezeichnung `value` sowie je einem ganzzahligen Wert beinhalten. Listing 4.2 veranschaulicht dies an einem Beispiel.

Treffen die Bedingungen zu, wird ein Schreib-Befehl an den Kaffeevollautomaten gesandt. Die Antworten werden in einer Zeichenkette mit der verwendeten Bezeichnung und dem Text „###“ als Abstandshalter festgehalten und auf der Standardausgabe ausgegeben. Die Antwort auf das Beispiel Listing 4.2 steht in Listing 4.3.

```
{
  "powder_quantity_special_coffee": {
    "value": 11
  },
  "water_quantity_special_coffee": {
    "value":  380
  }
}
```

Listing 4.2: Beispiel einer JSON Eingabe für./JuraCoffeeMemory eepromWrite

```
ok: powder_quantity_special_coffee###ok: water_quantity_special_coffee###
```

Listing 4.3: Antwort auf das Beispiel der JSON Eingabe

./JuraCoffeeMemory eeprom Nach ungefähr fünf Sekunden erhält man die aktuellen Einstellungen und Zählstände des EEPROM in einem kompakten JSON Objekt. Im Gegensatz

zum EEPROM Skript werden nur die nötigen Speicherstellen abgefragt. Die Bezeichnung der Speicherstellen, die auch der Name eines JSON Eintrags ist, befindet sich mit der zugehörigen Adresse in `EEPROM_Status::getEntriesEEPROM()`. Eine Adresse besteht aus der Position eines Wortes sowie einer oder beider Bytes. Von Hand sind dort zum Teil weitere Hintergrundinformationen, wie Standardwerte über die `[N]`-Taste des Kaffeevollautomaten, minimale und maximale Schranken, der Wert für einen deaktivierten Zustand oder überhaupt ausschließlich zugelassene Werte hinterlegt.

./JuraCoffeeMemory ram Ähnlich zum Abfragen des EEPROM wird hier aber der RAM in ungefähr drei Sekunden an den entscheidenden Stellen ausgelesen. Die Assoziation einer Bezeichnung mit ihrer Speicherposition geschieht in `RAM_Status::getEntriesRAM()`. Hier wird ein Byte und ggf. mehrere zusammenhängende Bits als ein Eintrag abgelegt. Als Ausgabe erfolgt ein kompaktes JSON Objekt.

4.2.4 ...

!!!ToDo!!!

Abbildung 4.1(b) Abbildung 4.1(c) Abbildung 4.1(d) Abbildung 4.1(e) Abbildung 4.1(f)

!!!ToDo!!!

./RAM.cpp ./JsonFile.cpp ./Storage.cpp ./JuraCoffeeMemory.cpp ./tools/libserial-test/main.cpp ./tools/display.cpp ./tools/formate-dump.cpp ./tools/jsoncpp-test/main.cpp ./tools/display-screen-saver.cpp ./tools/Ideen/Ctrl-D_stop.cpp ./tools/Ideen/byte-bit.cpp ./tools/Ideen/progressbar.cpp ./tools/Ideen/byte-bit-2.cpp ./tools/Ideen/lock-file.cpp ./RAM_Status.cpp ./EEPROM.cpp ./EEPROM_Status.cpp ./SerialConnection.cpp

EEPROM: Handbuch einmal durch gegangen und alle Menüoptionen einmal eingestellt, bzw. Schale (Hardware) bewegt. Bei Skalen immer Minimum, Maximum, Anzahl der Schritte, sowie den Standardwert [N] Aktionen in vielen Einzelschritten aufgenommen: eine Reinigung => welche Zähler werden zurückgesetzt Speicher zu einem späten Zeitpunkt gezielt beschrieben => Bezugszähler im Menü entschlüsselt: nicht das eine Wort mit dem gleichen Wert, sondern die Summe aus 5+1 Zählern => 2 Trester Zähler entdeckt, werden beim Entnehmen der Schale tatsächlich zurückgesetzt => Filterwechsel: 500 = 50l => 0,11 Zähleinheiten => Reinigung

RAM: mehrere Speicherauszüge in der eingeschalteten Ruheposition (Kaffee bereit) erstellt und regelmäßige Unregelmäßigkeiten festgestellt. Diese Bytes im Kommenden ausgeblendet. Aktionen ausgelöst, min. 3 Speicherauszüge erstellt und auf Gemeinsamkeiten verglichen. Dafür auch Herdware gebaut (s. nkH2 Platinen Foto für Schale)

Für den Schluss ACHTUNG: während einer Übertragung ist das interne Bus System gehemmt (Display vollzieht keine Änderung), aber auch Gegenrichtung: wechselnde Displaytexte sorgen für Verzögerungen bei der Übertragung

4.3 Weitere kleine Tools

!!!ToDo!!! befinden sich im Unterordner "tools/". - display -> unterstützter Zeichensatz - display-screen-saver -> Easter Egg - formate-dump -> Aufschlüsselung eines Speicherauszuges (inkl. autom. Erkennung eines EEPROM / RAM Auszugs anhand der Größe)

Display: Speicherort unbekannt, über die Befehle (?D0, ?D1xxx, ?D2xxx) systematisch alle ASCII Zeichen gesandt. Ab dez 128 Darstellung einer Zahl in mehr als einem Kästchen => Speicherüberlauf, nur das normale (nicht erweitere) ASCII Alphabet. (war dem wirklich so?, noch einmal testen!) (Für Kapitel 5: 0-3x und 9x-127 ggf. verpixeltes kanji, aber keine lateinischen Buchstaben oder arabische Zahlen sowie ASCII bekannte Sonderzeichen mehr! Mittelteil ist in der Tabelle dargestellt)

4.4 Die Webseite

!!!ToDo!!! befindet sich im Unterordner "website/".

Kapitel 5

Ergebnisse

5.1 Bedeutung der Speicherstellen

!!!ToDo!!!

Um den Speicher zu verstehen, klären wir vorab das Rückgabeformat der Speicherauszugs Befehle, sowie den Aufbau des Speichers der Jura Impressa S9. Dieser Kaffevollautomat besitzt einen RAM für die Status, Messwerte und Zwischenberechnungen im Betrieb und einen EEPROM für Einstellungen und Zählstände, die auch nach einer Stromunterbrechung erhalten bleiben.

5.1.1 EEPROM

!!!ToDo!!! Der Electrically Erasable Programmable Read-Only Memory (EEPROM) umfasst 512 Bytes. Über den Befehl `RT:<address>`, siehe Tabelle 3.1, lässt sich eine Zeile EEPROM Speicher abfragen. Die Adresse reicht von 0x00 bis 0xF0 in sechzehner Sprüngen. Als Antwort erhält man hinter dem kleingeschriebenen Kommando eine Zeichenkette bestehend aus 64 Hexadezimalzeichen. Da immer zwei Hexadezimalzahlen eine Byte (8 Bit mit Werten von 0-255) repräsentieren umfasst eine EEPROM Zeile 32 Bytes. Über den Befehl `RE:<address>` können direkt Wörter im EEPROM abgefragt werden. Die Adresse reicht von 0x00 bis 0xFF. Die kleinste adressierbare Einheit, das Wort, besteht daher aus zwei Bytes, also vier Hexadezimalzeichen.

!!!ToDo!!! Abbildung aus dem Antrittsvortrag: Seite 8/13

5.1.2 RAM

!!!ToDo!!! Der Ramdom-Access Memory (RAM) hingegen umfasst 256 Bytes. Der Lesebefehl für eine RAM Zeile lautet: `RR:<address>`. Die Adresse reicht hier von 0x00 bis 0xFF, sodass hier die kleinste Einheit ein Byte, bestehend aus zwei Hexadezimalzeichen, ist.

!!!ToDo!!! Abbildung aus dem Antrittsvortrag: Seite 9/13

5.2 Aktionsauswirkungen

!!!ToDo!!! Reinigung setzt etliche Zähler zurück; Entnahme der Schale setzt min. 2 Trester zähler zurück; Bits im RAM variieren auch im ausgeschalteten Betriebszustand, einige Informationen nur im eingeschalteten Zustand auslesbar, ander aber auch immer (solange Maschine mit Strom versongt wird)

5.2.1 Display

!!!ToDo!!! !!!ToDo!!! Display, siehe Abbildung 5.1

Aufbau der Befehle, siehe Abschnitt 3.1.3

Ausgabe mittels: `cd JuraCoffeeMemory -> make -> ./JuraCoffeeMemory -> 4 - Send a command -> ?D1xxx -> <Strg>+D -> ./tools/display`

ASCII			Kaffeevollautomat	ASCII			Kaffeevollautomat
hex	dez	Zeichen	Display	hex	dez	Zeichen	Display
	0 – 31	…	…	0x41	65	A	A
0x20	32	Leerzeichen	Leerzeichen	0x42	66	B	B
0x21	33	!	_	0x43	67	C	C
0x22	34	"		0x44	68	D	D
0x23	35	#		0x45	69	E	E
0x24	36	$		0x46	70	F	F
0x25	37	%		0x47	71	G	G
0x26	38	&		0x48	72	H	H
0x27	39	’		0x49	73	I	I
0x28	40	(		0x4A	74	J	J
0x29	41	)		0x4B	75	K	K
0x2A	42	*		0x4C	76	L	L
0x2B	43	+	+	0x4D	77	M	M
0x2C	44	,	,	0x4E	78	N	N
0x2D	45	-	-	0x4F	79	O	O
0x2E	46	.	.	0x50	80	P	P
0x2F	47	/	/	0x51	81	Q	Q
0x30	48	0	0	0x52	82	R	R
0x31	49	1	1	0x53	83	S	S
0x32	50	2	2	0x54	84	T	T
0x33	51	3	3	0x55	85	U	U
0x34	52	4	4	0x56	86	V	V
0x35	53	5	5	0x57	87	W	W
0x36	54	6	6	0x58	88	X	X
0x37	55	7	7	0x59	89	Y	Y
0x38	56	8	8	0x5A	90	Z	Z
0x39	57	9	9	0x5B	91	[	
0x3A	58	:	:	0x5C	92	\	
0x3B	59	;	;	0x5D	93	]	
0x3C	60	<	<	0x5E	94	^	
0x3D	61	=	=	0x5F	95	_	
0x3E	62	>	>	0x60	96	‘	
0x3F	63	?	?		97-127	a-z { | } ~	…
0x40	64	@					

Tabelle 5.1: Verfügbarer Zeichensatz des Displays

Kapitel 6

Diskussion und Ausblick

6.1 Probleme

6.1.1 Kommunikation mit der Kaffeemaschine

Der direkte Zugang ist wahrscheinlich am schnellsten, im laufenden Betrieb aber schlecht möglich, da die Speicheradressen aktiv abgefragt werden müssten und damit Kurzschlüsse in der Elektronik produziert werden. Die serielle UART Schnittstelle ermöglicht das Abfragen im laufenden Betrieb, also auch im RAM. Jedoch benötigt das Auslesen seine Zeit (7-9 Sekunden für den RAM und 10-15 Sekunden für den EEPROM). Gerade im RAM gibt es viele Veränderungen im Ruhezustand, sodass man auf diesem Weg nie einen zusammenhängenden Speicherauszug zu einem festen Zeitpunkt auslesen kann.

6.1.2 Serielle Kommunikation

Gerätedatei direkt ansprechen Um aufbauend Skripte ausführen zu können war der erste Versuch den Dateideskriptor `/dev/ttyACM0` sowohl lesend als auch schreibend zu nutzen. Zunächst funktionierte dies sehr gut, bis nach wenigen Tagen kaum reduzierbares Fehlverhalten auftrat. Gerade Antworten der Kaffeemaschine kamen nur noch in Bruchstücken an. Dies wurde bei der Einrichtung bereits im Projekt „CoffeeMachine“[Kö18] in der Readme.md für die Ausführung auf einem Raspberry Pi beschrieben.

Versuche dies direkt zu lösen, in dem ein funktionierende Umgebung nachgebaut wurde, schlugen fehl. !!!ToDo!!! Unternommene Befehle und Versuche aufzeigen

C++ und die *libserial*-Library Für eine sichere Verbindung wird in dem C++ Programm auf die im Linux Repository erhältliche Library *libserial* zurückgegriffen. Beim Verbindungsaufbau über die *connect()* Methode der *SerialConnection* Klasse wird die Schnittstelle initialisiert. Hier werden der Gerätepfad, die Baudrate und eine erwartete Mindestlänge festgelegt.

Kommandos gehen zuverlässig an den Arduino raus und ausgelesene Antworten kommen nun im Ganzen an. Der *read* Befehl ist ein blockierender Aufruf, der das Programm anhält bis eine Antwort vorliegt. Dies stellt für die Speicherauslesung keine Problem da, da die ordnungsgemäßen Befehle immer ein `ok:` oder `xx:0-F` Speicherauszug zurück geben.

6.1.3 Unbekannte Speicherorte

Unbekannter Speicherort des Einstellungsmenü-Zählerstands (überlebt Stromtrennung?!) und des Standarddisplaytextes (im RAM?!); dafür Blick auf den Mikrocontroller in der Hardware [Hol17] und externes Auslesen ggf. erforderlich!

6.2 Aussagekraft der Ergebnisse

EEPROM Menüeinstellungen recht sicher, einige partielle Zubereitungszähler aber weniger (Firmware unbekannt)

RAM mehrere Status Bits einigermaßen sicher (vor allem die im ausgeschalteten Betriebszustand); durch viele unregelmäßige Veränderungen wurden aber mehrere Bytes außer acht gelassen, die an bestimmten Bit Positionen wichtige Flags enthalten könnten!

Es fehlen (wichtige?!), im Benutzerhandbuch dokumentierte, Meldungen wie: Gerät verkalkt, Störung 2, Störung 8, sowie etliche Menümeldungen während der internen Programmabläufe.

6.3 Einordnung zur Terminologie des Reverse Engineerings

Diese Arbeit besteht im wesentlichen aus den zwei Teilen, den Speicher zu analysieren und zu verstehen, bzw. das Wissen darüber in einer neuen Oberfläche einfach zur Verfügung zu stellen. Ein kurzer Rückblick in Abschnitt 2.1 erinnert an die Begriffe *Forward Engineering*, *Reverse Engineering*, *Redocumentation*, *Design Recovery*, *Restructuring*, und *Reengineering*.

Für den ersten Teil wurden eine gegebene serielle Schnittstelle, sowie bereits implementierte Befehle ausgenutzt, um den Speicher auszulesen. Viele Aufnahmen, mit kleinen Veränderungen zwischendurch, ließen Rückschlüsse auf die Bedeutung der Werte an bestimmten Speicherstellen zu. Gerade mit den Überlegungen, was während einer Aktion am Kaffeevollautomaten alles im Speicher passiert, lässt sich diese Tätigkeit dem Begriff *Reverse Engineering* zuordnen. Aus dem fertig implementierten und gegebenen System werden Rückschlüsse auf das Design der Maschine gezogen. *Redocumentation* trifft aufgrund fehlender Spezifikationen und Quellcodeauszügen wenig zu. Betrachtet man aber das Benutzerhandbuch und Interneteinträge über Erfahrungen mit der Maschine als externe Informationen, sowie Schlussfolgerungen

und Unschärfelogik über den RAM, kann dieser Part auch als *Design Recovery* über den Speicher angesehen werden.

Mit diesem Wissen des ersten Teils wurde dann im zweiten Teil etwas Neues geschaffen. Das Abfragen der aktuellen Einstellungen und Zustände bietet eine neue Oberfläche, die als *Redocumentation* oder *Restructuring* angesehen werden kann. Für die *Redocumentation* spricht das neue Ausgabeformat auf der Ebene der Implementierung. Ein eigenes Gerät ist zur Alternative (Ausnahmen siehe 6.2) für das kleine Display des Geräts geworden und gibt den aktuellen Zustand in ausführlicherer Weise aus. Ohne eine präventive Instandhaltung beabsichtigt zu haben ist *Restructuring* aber auch zutreffend, da Wissen aus dem ersten Teil hier eingeflossen ist. Wenn die Bedeutung bestimmter Speicherstellen nicht bekannt gewesen wäre, hätten die eingestellten Werte keine Bedeutung gehabt, was für den Begriff *Restructuring* spricht.

Für eine individuelle Zubereitung, wird nur die Datengrundlage kurz zuvor mit einem Impuls von außen verändert, der Kaffee wird dabei nach wie vor von den Standard Abläufen der Kaffeemaschine zubereitet. Die neue Oberfläche kann man als *Restructuring* oder dem allgemeineren *Reengineering* angesehen werden. Da die neue Oberfläche, über die serielle Schnittstelle in Verbindung mit einem eigenen Endgerät, aber sehr von den Bordmitteln der Maschine abweicht passt vielleicht das *Reengineering* etwas besser. Die Funktionalitäten der Maschine sind aber immer noch die Gleichen, das Programmmenü muss nun jedoch nicht mehr betreten werden, um die allgemeinen Geräte Einstellungen für den eigenen Kaffee anzupassen.

Im Detail ist die genau Zuordnung der Begrifflichkeiten immer eine persönliche Auslegung dieser Begriffe. Sehr zutreffend ist das Zitat aus Abschnitt 2.1, denn diese Arbeit hat den Speicher intensiv betrachtet und mittels der Ergebnisse aus dem zweiten Teil um neue Perspektiven bereichert, ohne das Ziel verfolgt zu haben die Maschine zu klonen oder umzubauen.

6.4 Zusammenfassung

!!!ToDo!!! Conclusion !!!ToDo!!!

6.5 Ausblick

!!!ToDo!!! Mit unendlicher Zeit und unendlich vielem Geld wäre folgendes noch möglich: evtl. !!!ToDo!!! Firmware aus ROM auslesen und disassemblieren -> Vorgänge wirklich verstehen Für folgendes hat die Zeit gefehlt / der Schwerpunkt ging nicht in diese Richtung: Display Texte und Menü-Zählerstand wurden im RAM nicht wiedergefunden, ...

Literaturverzeichnis

[CC90] CHIKOFSKY, E. J. ; CROSS, J. H.: Reverse engineering and design recovery: a taxonomy. In: *IEEE Software* 7 (1990), Jan, Nr. 1, S. 13–17. http://dx.doi.org/10.1109/52.43044. – DOI 10.1109/52.43044. – ISSN 0740–7459 3, 4

[Eil05] EILAM, Eldad: *Reversing : secrets of reverse engineering*. Indianapolis, Ind. : Wiley, 2005 http://www.gbv.de/dms/ilmenau/toc/480966761eilam.PDF. – ISBN 0764574817 9780764574818 5

[Hol17] HOLTEK SEMICONDUCTOR INC. (Hrsg.): *HT46F46E/HT46F47E/HT46F48E/HT46F49E Cost-Effective A/D Flash MCU with EEPROM*. Rev. 1.50. Hsinchu, Taiwan: Holtek Semiconductor Inc., April 2017. http://www.holtek.com.tw/documents/10179/11842/HT46f4xev150.pdf. – http://www.holtek.com/productdetail/-/vg/46f4xe 24

[Ing94] INGLE, Kathryn A.: *Reverse engineering*. New York u.a. : McGraw-Hill, 1994 http://www.loc.gov/catdir/toc/mh022/94018447.html. – ISBN 0070316937 9780070316935 4, 5

[Kö18] KÖSTLER, Maximilian: *CoffeeMachine*. Git Repository. https://collaborating.tuhh.de/e-17/General/CoffeeMachine. Version: Januar 2018. – nicht öffentlich 8, 9, 23, 29

[Off18] OFF, Fabian: *Reverse-engineering a De'Longhi Coffee Maker to precisely bill Coffee Consumption*, Universität Magdeburg, Diplomarbeit, Februar 2018 5

[Sal09] SALEH, Kassem A.: *Software engineering*. Boca Raton, Fla. : Ross, 2009. – ISBN 1932159940 9781932159943. – Literaturangaben 3

Anhang A

Inhalt der CD

Inhalt	Beschreibung
thesis.pdf	Diese Arbeit in digitaler Form
JuraCoffeeMemory/	Das entwickelte Projekt
JuraCoffeeMemory/arduino/arduino.ino	Das Arduino Skript aus [Kö18]
JuraCoffeeMemory/data/	Die aufgenommenen Speicherauszüge im JSON Format
JuraCoffeeMemory/doxygen/	Die Doxygen Dokumentation
JuraCoffeeMemory/manual_jura/	Handbuch und Abbildung des Kaffeevollautomaten aus [Kö18]
JuraCoffeeMemory/result/	Ergebnisse der Auswertung
JuraCoffeeMemory/tools/	Kleine Helfer-Programme und Codeschnipsel
JuraCoffeeMemory/website/	Die Webseite zur Visualisierung

Tabelle A.1: Inhalt der CD