Karlsruhe Institute of Technology

Institut für Technische Informatik
Chair for Embedded Systems

Improving Usability and Reliability of an IoT-based Controller for a Coffee Machine

Bachelorarbeit
von

Simon Korz

an der Fakultät für Informatik

Betreuer:	Prof. Dr. Jörg Henkel
Betreuende Mitarbeiter :	Dr.-Ing Lars Bauer
Betreuende Mitarbeiter :	Dr.-Ing Farzad Samie
Tag der Anmeldung:	11.11.2019
Tag der Abgabe:	10.03.2020

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig angefertigt und mich keiner fremden Hilfe bedient habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichtem oder unveröffentlichtem Schrifttum entnommen sind, habe ich als solche kenntlich gemacht.

Karlsruhe, 10.03.2020

Abstract

Wenn es zur Implementierung neuer und innovativer Technologien für bisher analoge Prozesse kommt, dann kann dies häufig zu einer tatsächlichen Verminderung der Benutzbarkeit für die Benutzer des Systems führen. Der offensichtliche Vorteil, eine Strichliste, die dazu verwendet wird Benutzer anhand ihres Kaffeekonsums aus einer handelsüblichen Kaffeemaschine, mit einem elektronischen, automatisierten System zu ersetzen, liegt darin, die manuelle Arbeit des Schreibens und Zählens von Strichen zu sparen. Wenn das System allerdings derart unzuverlässig funktioniert, dass es einem Benutzer nicht mehr möglich ist die Kaffeemaschine zu verwenden, dann erfüllt die Implementierung eindeutig nicht die Erwartung, den Prozess für den Benutzer zu verbessern. In dieser Thesis wird eine bereits existierende IoT-basierte Anwendung, die zur Erfüllung der zuvor genannten Aufgabe entworfen wurde, untersucht und die mit deren Implementierung verbundenen Schwierigkeiten hervorgehoben, sowie die Verbesserungen beschrieben, die durchgeführt wurden um einen Punkt zu erreichen an dem die Benutzbarkeit des neuen Systems einen klaren Vorteil gegenüber der manuellen Ausführung der Aufgabe darstellt.

Abstract

When it comes to implementing new and innovative technologies for previously analogue processes, often enough there is an actual decrease in usability for the users of the system. The most apparent benefit of replacing a tally sheet that is used to charge users according to their coffee consumption from an off-the-shelf coffee machine, with an electronic, automated system, lies in eliminating the manual work of writing and counting strikes. However, when the system operates unreliably, to such an extent that the user has no chance of using the coffee machine, the implementation clearly does not meet the expectations of improving the process for the user. Examining an existing IoT-based appliance that was designed for the before mentioned task, this thesis will highlight the difficulties of such an implementation and describe the improvements that were made to reach a point where the utilization of the new system is a clear advantage over performing the task manually.

Contents

List of Figures

Acronyms

DBMS Database Management System

DSI Display Serial Interface

FSM Finite-state machine

GPIO General-purpose input/output

HTTP Hypertext Transport Protocol

IPC Inter Process Communication

IoT Internet of Things

LAMP Linux Apache MySQL PHP

NFC Near Field Communication

PWM Pulse Width Modulation

RFID Radio-Frequency Identification

SPI Serial Peripheral Interface

SoC System on Chip

UID Unique Identifier

1 Introduction

This thesis documents the improvements to an existing system that was designed to replace a tally sheet for coffee consumption. The system is connected to an off-the-shelf coffee machine and automates the registration of every coffee or hot water dispensed.

There are two systems that were used before[1]. For simplicity, in this text they will be named version 1 (V1) and version 2 (V2), respectively. Version 3 (V3) is the resulting system with the improvements described in this thesis. While version 1 reportedly has yielded very good results, it had to be adjusted due to a change in the coffee machine model. In V2, subsequently, the system was almost completely redesigned and rebuilt for the new coffee machine. However, the developed design and implementation are incomplete and did not produce satisfying results. There was an initial list of bugs and possible improvements for V2 and also newly requested features for V3. Throughout the progress of this work, the list has been reiterated and further specified. A final version of it, containing an analysis of all the previously known and newfound problems and shortcomings of V2, can be found in section 3.

This work aims to improve V2 by resolving all bugs and additionally increase its usability and reliability. Making the transition as smooth as possible, without disruptions to the daily use, is a high priority. In an attempt to reutilize as much of the design ideas and code of the previous works as possible, the implementation evolved out of the code base of V2. However, a large part of the existing components had to be redesigned and newly implemented.

In the ISO norm [ISO18], **usability** is defined as the "extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use." In subsection 3.1 this definition will be applied to evaluate the system. Later it will be explained that **reliability**, in the sense of high availability and fault tolerance, is a direct consequence of the requirements for usability. Put simply, the system has to be designed and implemented in a way that whenever a user wants to have a coffee or hot water, the system is available and fully functional and must never run into errors which inhibit the proper usage of the coffee machine. Reliability also plays an important role in the **Internet of Things** (IoT) [Pok19]. The concept of IoT can generally be described as the upgrade of formerly analog, offline devices and plainly physical objects, to highly interconnected, digital devices and is still one of the biggest trends in IT [MRT15, AGM+15]. Although, the approach taken in the system examined here uses many of the key enablers of the IoT, it can be argued that it does not fully deserve to be called an IoT appliance. Missing key characteristics are, the inter-connectivity between multiple devices and systems, the pervasiveness of all parts, i.e., the complete physical integration of the added hardware into the coffee machine, and gathering data from sensors for later processing. The distinction of "IoT-based" is made here to indicate that the system uses a combination of technologies which are considered IoT enablers and could easily serve as a foundation for an IoT appliance.

- Access to a controlling server via HTTP
- NFC reading RFID tag attached to physical object (e.g. coffee mugs, key chain)
- Support for a mobile application[2]

[1] Cf. [Mod14] and [Amm17].

[2] The mobile application will not be further examined in this thesis. Support for it was dropped in V3, as there were no users.

- Reading sensor data
- High availability

In the next three parts of this thesis, first an overview of background knowledge is given, then the problems in V2 and their solutions in V3 are discussed, and at the end a short conclusion will be drawn.

2 Background

2.1 The Previous System (V2)

The system, as it was used before this work started, included the following hardware components:

- Coffee machine Saeco Café Royal SUPE016RE
- Raspberry Pi 3 Model B v1.2
- Adapter board between the coffee machine and the Raspberry Pi
- RFID reader module by JoyIT, based on the NXP MFRC-522 frontend
- Buzzer connected to the Raspberry Pi
- 7” Touchscreen
- Ethernet cable
- Official Raspberry Pi micro-USB power adapter, 5.1V, 2.5A
- Shoe box, converted into a case for all components apart from the coffee machine

The coffee machine used, does not provide an interface to access its sensor data. Therefore, in the previous works, an adapter board employing optocouplers was designed in order to read and manipulate the coffee machine’s internal sensors. This was done by hijacking the cables coming from and going to the coffee machine’s control board. The three sensors available to the system now are:

- **water flow:** The water flow through the coffee machine’s pump is measured by a flow meter. The signal captured is a square wave signal, oscillating between logic high and low voltages. It has varying frequencies, dependant on water flow speed. A higher frequency implies more water output during a certain time interval. Time intervals vary in duration, since they depend on the user’s preference for the amount of water. Note that while the current coffee machine has only one flow meter, the source code has support for two water flow signals. The reason is that the coffee machine model which was used before has two separate water circuits for coffee and hot water, i.e., two separate signals available that help distinguish between coffee and hot water dispensing. With the new model, the adapter board did not change and the two different connectors still exist. However, both transmit the same input signal.

- **grinder:** The grinder signal is similar to the water flow signal, in that it is also a square wave signal. Interestingly, it can only be measured after the coffee machine has already finished grinding. It then occurs for less than a second on average. Further analysis of the grinder signal is done in subsubsection 3.3.8.
- **water level/blocking:** The water level signal is toggled every time the water contained in the fresh water tank sinks below a certain level. By intercepting the signal an empty water tank can be simulated to the coffee machine. This will effectively block the dispensing of water and coffee by putting the machine into "no water mode". The system can leverage this mechanism to lock and unlock the coffee machine. However, the system then needs to inform the user when the water level is actually low, because the text on the coffee machine's display that says "FILL WATERTANK" will only indicate that the machine is locked and might be misleading otherwise.

2.1.1 The Raspberry Pi and GPIO

The Raspberry Pi 3 Model B is a single-board computer integrating a Broadcom system on chip (SoC). Its credit card sized form factor, numerous I/O capabilities, extensive documentation and the relatively cheap price of €35 are the reason why it is a popular choice for all kinds of maker projects and teaching.

The relevant parts of the specification for the Raspberry Pi model used are:

- Broadcom BCM2837, quadcore Cortex-A53 (ARMv8) 64-bit SoC @ 1.2GHz
- 1GB LPDDR2 SDRAM
- 100 Base Ethernet
- Extended 40-pin GPIO header
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input

A complete specification can be found in the appendix A.2.

The general-purpose input/output (GPIO) header provides 40 input and output pins that each can be configured and controlled independently by the user at run time. The output voltage can be set to high and low. When set to high, the voltage has a minimum value of 3.0V and cannot be higher than 3.3V. When set to low, the voltage has a maximum rating of 0.14V. In input mode, gpio pins read voltages higher than 1.6V as high and voltages lower than 0.9V as low. For each pin an interrupt can be set in the ARM, triggered by high/low levels or rising/falling edge. There are also internal pull-up and pull-down resistors of $50 - 60k\Omega$ that can be enabled optionally. Furthermore, alternative functions can be assigned to the pins so they can be used for, e.g., PWM, I2C or SPI buses, and serial connections. In total, the GPIO interface makes the Raspberry Pi highly versatile and opens up numerous possibilities for connecting it to all kinds of periphery[3].

[3]See [Fou] and [Cor] for a detailed description of the Raspberry Pi's GPIO capabilities.

The manufacturer of the Raspberry Pi provides Raspbian as the default operating system. It is a Debian Linux derivative, especially designed for the Raspberry Pi. It makes use of `systemd`, which, among other tasks, is in charge of managing system services. Although many options exist, the de facto standard and best supported programming language for the Raspberry Pi is Python. As a result, there are several libraries available for accessing the Raspberry Pi's GPIO from Python. Three of the more important ones are:

- `RPi.GPIO`, which is classically the most commonly used, but only offers basic functionality.
- `gpiozero`, which offers specialized high level objects representing different hardware components like buttons and LEDs, and is thus the easiest to use for simple tasks.
- `pigpio`, which is more sophisticated with a greater support for low level features like wave patterns, PWM and glitch/noise filters for edge detection. Unlike the other two, it runs as a system daemon that is accessed by clients via Unix sockets and also allows remote connections.

2.1.2 Connection between Components

A user authenticates with an RFID tag. The ID from the RFID tag is read by the RFID reader. The RFID reader is connected to the Raspberry Pi via the GPIO pins and is controlled using the SPI bus protocol. After reading the ID, the system checks if there is a user in the database that registered with this ID. If that is the case, the user will be logged in and the coffee machine will be unlocked. The buzzer is used to give audible feedback when a user successfully logs in.

The touchscreen is directly plugged into the Raspberry Pi's Display Serial Interface (DSI). Raspbian has built-in support for the screen and does not require special configuration, allowing to use the provided desktop environment out-of-the-box.

The Raspberry Pi is connected to the local network through ethernet and also has internet access. However, a stable connection cannot always be guaranteed.

Although, a case to contain all the components was designed in V2, it was never finished, and thus the parts were assembled into a shoe box.

2.1.3 Accounting Server

The accounting server has a record of all registered users and their associated RFID tags. It logs every order, every payment to recharge balance and every withdrawal of money for consumables such as coffee beans and milk. There is one person responsible for creating accounts, registering RFIDs and entering withdrawals and recharges into the database. Each user's current balance is calculated by summing up all withdrawal and recharge events from the past. The events are kept in one large table in the database, whose history goes back to 2012. This was the the year when the accounting server was created and that was before the work on V1 started.

The accounting server is accessed across the local network via an HTTP interface. It operates on a typical LAMP stack. A list of all registered users and their associated RFIDs can be obtained with:

```
http://i80web2.ira.uka.de/coffee/getusers.php?secret={some-secret}
```

Information about a single user by RFID can be obtained with:

`http://i80web2.itec.kit.edu/coffee/getuser.php?rfid={some-rfid}`

If there is a user associated to the given RFID, the response includes `user_id`, `balance` and `token`, if there is no associated user, the response contains the string "unknown token/rfid". A `token` is an alphanumeric identifier for a user account.
An order is submitted to the accounting server with:

`http://i80web2.itec.kit.edu/coffee/buy.php?rfid={some-rfid}&black={true/false}&water={true/false}&cheated={true/false}`

There is a graphical interface for direct access by users where they can see an overview of their consumption, a list of every user's balance, activities like withdrawals and deposits to the treasury, and a ranking of "good milk guys" and "bad milk guys". The latter are statistics based upon the ratio of milk brought to the office and milk consumed.

2.2 Concurrency in Python, IPC and Multiprocessing

Due to a mechanism in Python called Global Interpreter Lock (GIL), in one python process only a single thread can execute at a time. Other threads need to wait until the running thread releases the lock. This limitation requires the usage of Python's `multiprocessing` library to utilize more than one CPU core and run applications truly in parallel. It allows to create processes with a similar interface as used for threading.

To communicate between these processes there are two types of channel integrated with the `multiprocessing` module.

- Pipes and
- Queues

For direct communication between two processes, pipes are the simplest solution. On Unix, pipes simply act as a higher level interface for sockets. They open an either uni- or bidirectional connection between two processes, returning a connection object that can be used to transmit data from one end to the other. While sending or receiving on both ends at the same time is possible, it will most certainly corrupt data when trying to send from more than one thread or process on the same end of a pipe at the same time. In other words, this implementation of pipes is not thread-safe. However, queues offer a layer of abstraction, adding synchronized access on top of pipes and hereby allow process- and thread-safe IPC.

Concurrency is also an important aspect of graphical user interface (GUI) programming. If, e.g., a button press starts a long running calculation, the whole GUI would become irresponsible while the calculation is in progress. Therefore, separate threads should be used for rendering the GUI and performing background tasks. In Qt5, which is the GUI framework used in this work, there is the concept of slots and signals for thread safe asynchronous communication. Signals with attached parameters can be emitted and will trigger the execution of connected slots, i.e., functions that accept the passed parameters.

2.3 3D Printing

For printing the 3D models that were designed in V2 and V3 the *formlabs Form 2* 3D printer was used. This printer uses the stereolithography (SLA) printing technique where

a “laser [is used] to cure solid isotropic parts from a liquid photopolymer resin” [for]. SLA belongs to the more expensive methods for 3D printing but therefore promises a much higher precision and resolution. The printer has a resin tank with an elastic, transparent floor. The tank is located on top of a glass window that shields the inner parts of the printer. On the inside of the printer there is a stationary laser that is redirected into the resin tank by a high precision mirror system. Since the laser has to pass the glass window and the floor of the resin tank, it reacts highly sensitive to impurities, such as dust and fingerprints, between these layers. Refractions would impose potentially grave damage to printed models. The model is growing layer by layer on the build platform, which is located above the resin tank and oriented upside down. The platform is lowered into the resin tank for every new layer. Whenever the platform sinks into the resin, it lightly pushes onto the elastic resin tank floor to ensure only a thin layer of resin gets cured. Unfortunately, this process causes the resin tank to wear out quickly. As a result, the resin tank has to be replaced after a printing volume from 1-1.5 liters of resin.

The 3D model was designed with the open source tool Blender. Since Blender is a very advanced computer aided design tool, where focus lies more on creating 3D models for animations, it is hard for beginners to use it for 3D printing. However, with the “3D-Print Toolbox” add-on, Blender offers tools for checking and repairing the designed models.

2.4 Terms

This is a description of the terms used in the context of this thesis.

Order An order is the selection of a coffee with milk, a coffee without milk, or hot water and always connected to a specific user.

Ordering Process The process of placing an order that starts with the user authenticating and ends after the order was submitted to the accounting server.

System The hardware and software that constitute the appliance. It includes the Raspberry Pi, the RFID reader, the buzzer, the amplifier circuit and all programs running on the Raspberry Pi. The accounting server and the coffee machine are explicitly excluded from this definition.

3 Problem Analysis and Solutions

3.1 Improving Usability

In the introduction a definition of usability was given. Starting by specifying the user, goal and context of use, and continuing with the requirements for effectiveness, efficiency and satisfaction, the definition can be utilized as a guideline for evaluating the system.

User The majority of users are employees at the chair, i.e., the professor, post docs, the secretary, and research assistants. A small number of users are students either writing a thesis or working as assistants. Most users only use the coffee machine to order coffee. Hot water, however, is ordered very rarely.

Goal The user’s goal is to buy a coffee or hot water. The coffee can be with milk or without milk. The user adds milk manually and has to register their choice with the system.

Context Users engage with the system in various situations, but usually only less than a minute. Some possible situations can be distinguished here:

1. The user is actively using the time to get a coffee as a short break.
2. The user is in a conversation with a colleague.
3. The user is marveling about a hard to solve work related problem.

Furthermore, it has to be taken into account that the screen will likely be at a distance of more than 60cm from the user, which means that the font size should be significantly larger than the 11-points standard used in many operating systems. It is also important to note that, because the screen is a touchscreen and interaction is haptic, there are certain constraints for the sizes of controls. They have to be large enough to be easily usable. The time when users order coffee varies from day to day and cannot be specified in a fixed interval. Nevertheless, most orders occur between 10am and 6pm from Friday to Saturday. While it cannot be ruled out to receive an order at, e.g., 4am, it is very unlikely.

Effectiveness The interactions with the system have to be leading towards the user's desired goal. If the interactions do not contribute to achieving the goal, the system is not effective.

Efficiency There should be as few steps as possible required to achieve the goal. The total time needed for interaction with the system should be kept minimal.

Satisfaction The user should be satisfied with the overall experience, have a positive attitude towards the system and feel comfortable using it. Sources of frustration have to be eliminated. Satisfaction can be increased, for instance, with an appealing graphical design, the right use of colors and visuals, but also by making it easy to learn how to use the system. That means visual and audible feedback should be used to guide the user through the ordering process to ultimately achieve the goal.

In summary, the system must not unnecessarily obstruct and distract the user and it must always focus on its main purpose, which is simplifying the tally sheet approach. For the evaluation of V2 the list of problems and shortcomings is used. The results are then taken into account for creating a new design for V3.

3.2 Analysis of Problems and Shortcomings in the Previous System

There are several problems which have to be fixed and a few additional requirements that have to be met.

GUI01 Bad visibility of important information When the user logs in the only visual feedback is their name and balance appearing on the screen. Furthermore, this information only takes up a small portion of the screen and is not in the center of attention. The user can easily be unaware the changes on the screen, especially when distracted. There is no explicit hint that the coffee machine is ready to use and the user has to guess, which makes first use unnecessarily difficult. The entertainment content, which is irrelevant at the beginning of the ordering process, takes up most part of the available space on screen. This is highlighted in Figure 1. Applying

the requirements for usability stated before, it becomes clear that this design is not ideal.

GUI02 Milk preference controls to small It can be difficult for the user to select if they want a coffee with or without milk, because the controls are only about 3.9mm in height on screen, and therefore, making it hard to tap them with the average fingertip. When the user needs several attempts to set the milk choice, this is ineffective and can lead to frustration.

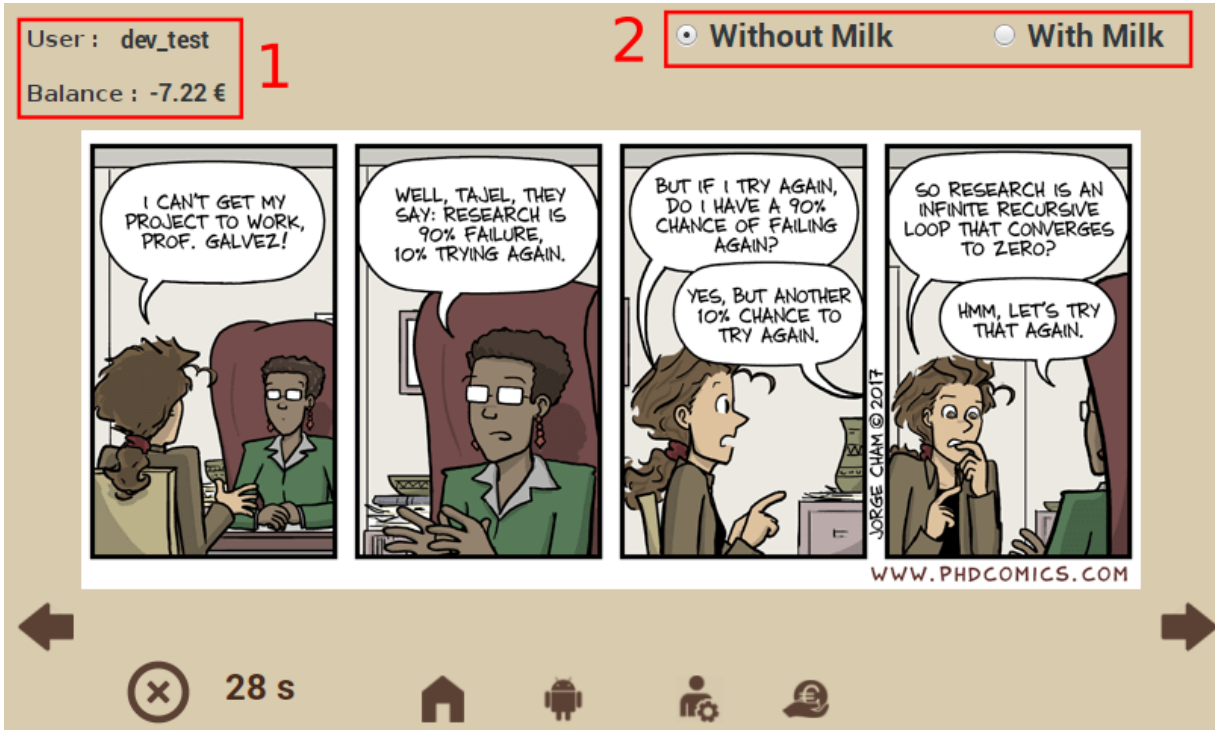

Figure 1: Screenshot of the order view in V2, taken from [Amm17].
1) The information displayed after a user successfully authenticated and the coffee machine is unlocked.
2) The controls for selecting a milk option.

GUI03 Saving milk preferences not working The system should store if the user wants their coffee with or without milk. Even though it was described as a feature in V2, the storage and retrieval of the preference was left unfinished.

Saving the milk preference increases efficiency by not prompting users for their milk choice on every order, saving them an additional interaction step. Nevertheless, it has to be easy to change the preference during every order.

GUI04 Entertainment content is not working Entertainment content (comics) is not shown anymore. The script responsible for fetching content on the accounting server stopped working, because of changes on the website(s) the content images were downloaded from.

The rationale behind showing entertainment content is to offer the user a form of engagement while waiting for the coffee machine to finish brewing. If working properly, this feature can contribute to increase the overall user satisfaction.

GUI05 Information for unknown RFID tag disappears to fast When a users tries to login with an unknown RFID tag a message is shown. It informs about the necessary steps to take in order to use the coffee machine. However, the message disappears after a very short time, making it difficult for the user to fully read it. If the user removed their RFID card in the meantime, they have to place it on the RFID reader again to finish reading the message. This is not effective.

GUI06 Screensaver confuses users and fails to show if water is empty or not When the screensaver is activated, the screen turns completely black. The user cannot easily tell if the system is operating correctly or not. Sometimes the screensaver does not deactivate when a user wants to order a coffee. Especially, when the water tank is empty, there is no indication on the screen. This circumstance was found to confuse users. From the usability point of view, this problem can frustrate the user or even stop them from accomplishing their goal, i.e., using the coffee machine, and therefore must be avoided.

BUG01 Coffee recognized as hot water When users order coffee, the system regularly fails to recognize it correctly and instead registers hot water. This bug has the highest priority, since it can cause the treasury a financial deficit.

BUG02 Rinsing after wake up charges the user with hot water After waking up from power save mode, the coffee machine might perform a rinsing. The signal read by the system has the same characteristics as the signal for hot water. Therefore, it is possible that the user is falsely charged for hot water. A mechanism to detect this situation should be implemented.

BUG03 The GUI freezes regularly, requiring a restart The GUI freezes from time to time, only showing a log-out button, but no logged-in user. Measures should be taken that make it impossible for the GUI to freeze. Again, if the GUI freezes, the system is not usable.

BUG04 Free coffee when water tank is refilled When the water tank is refilled the coffee machine stays unlocked for about a second. During this time frame a user could select a product without logging in and get coffee for free. The system should only ever be unlocked when a user is signed in or maintenance mode is activated.

BUG05 Free coffee when canceling while coffee is ground When the user is logged in and selects coffee on the coffee machine. The coffee machine starts grinding the coffee beans, but the system only receives a signal after the grinding finished. Before the grinding is recognized the user can press the cancel button and log out. The coffee dispensing, however, will proceed without being registered by the system. If the system recognizes a `coffee_ground` signal without a user currently logged in, it should assume that, in case a user was logged in a few seconds before, it was them who ordered a coffee. The system should register the coffee as a cheating attempt.

BUG06 Offline mode not functional The system should be able to operate independently, without network connection, for a certain period of time. The mechanism implemented in V2 is only partially functional. The SQL query supposed to store offline orders in the database fails, because it tries to insert data into non-existent columns. Resolving this bug is of high priority for improving reliability.

BUG07 RFID reader not responsive One of the challenges described in [Amm17] was that the RFID reader is not responding from time to time. This problem still occurs. The RFID reader is essential for using the coffee machine and has to work reliably.

BUG08 3D-Model not printable In V2 a 3D model for a case to contain all the hardware was designed. The model was split up into three components, the top, the bottom, and the screen frame, to be able to fit into the 3D printer and later be assembled to form the case. However, the screen frame was the only part that was completed and usable. The other two parts had design errors. The bottom part was missing big enough holes for cables, the floor layer, with 1.28mm, was too thin and broke easily, and the walls, with 6.21mm, were very thick, and thus, wasting material. The top part, which was supposed to sit on top of the bottom part and hold the screen frame, was not printable due to floating planes and broken geometry, which can be seen in Figure 2. Even if it was printable, the design would have been problematic, because the RFID reader would have been located right in front of the screen, resulting in cups with RFID tags attached, or big key chains blocking the view on the screen. Furthermore, there were no structures to hold the Raspberry Pi or the RFID reader in place.

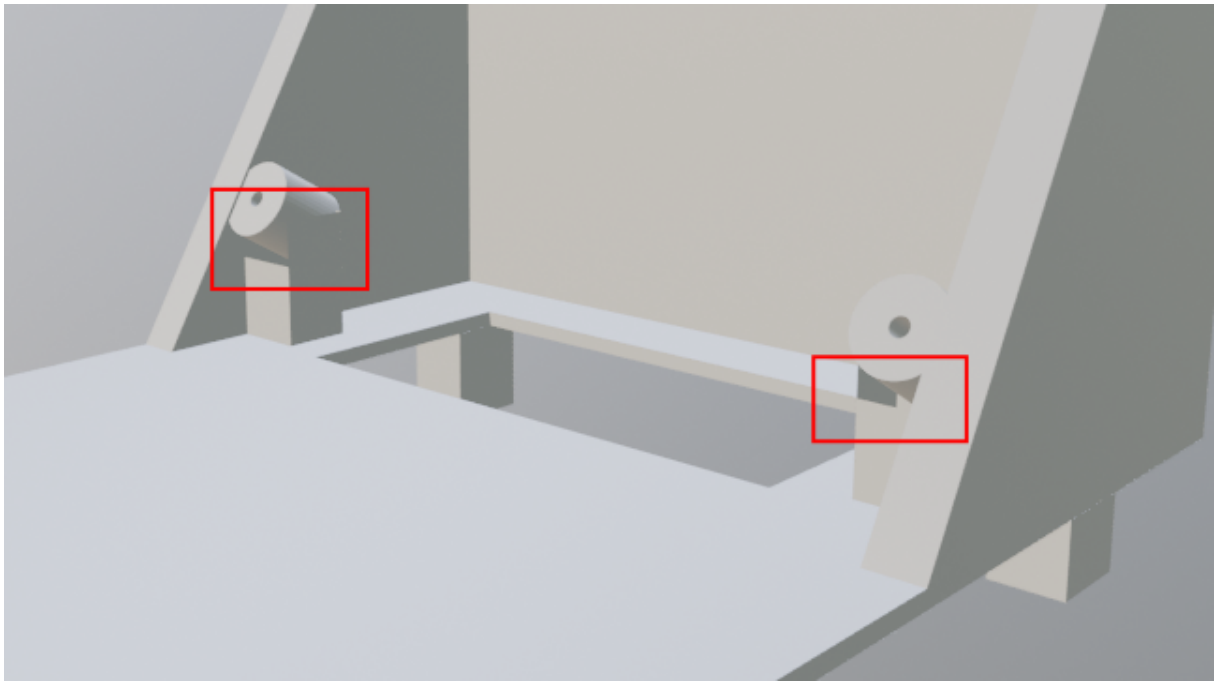

Figure 2: Top part of the 3D model for V2. Broken geometry that made the model unprintable is highlighted.

F01 No support for new KIT-Card A common way to authenticate with the device is to use the NFC-enabled KIT-Card[4]. Throughout the past year all KIT-Cards were replaced by a newer model, which uses the MIFARE DESFire EV1 integrated circuit for NFC. The new model should be supported in V3.

[4]Every employee and every student at KIT has a KIT-Card. It is used, for example, as an access key to buildings, to rent books at the library and to pay meals in the canteen.

F02 High CPU usage During normal operation with no user interaction one core of the CPU is constantly at around 95+% utilization[5]. This load appears to be very high when taking the available processing power into account. The problem lies in how the sensor data is read from the coffee machine. The signals are read and processed inefficiently at a frequency of 1kHz. See an excerpt of the code in Figure 3. The load should be reduced to save power and avoid negative effects on hardware health.

F03 Warm-up requires the user to log in several times The session expires 90 seconds after login. This time interval is not long enough to warm up the coffee machine after a long idle time, e.g., after being idle over night. Heating time, according to the manual, is approximately 2 minutes.

F04 Logging not sufficient The existing logging system prints messages to the console and several log files. However, the information is not clear, often missing information about its origin and timestamp. This makes it very difficult to reason about the possible causes of a problem. The system should have a proper logging mechanism to allow better debugging, easy locating of errors and thereby improve the overall reliability.

F05 Text files are used for IPC Reasoning that the performance of pipes or sockets is not sufficient for IPC [Amm17, 3.3.5], V2 uses simple text files in the working directory for communicating between processes. An example of it can be seen in Figure 3. The arguments made against pipes that "due to the different processes cycles, this solution was slowing the system" and sockets "blocking solutions will stop the system", do not convince. Especially, because using text files in the filesystem is essentially the same as using named pipes in Unix with the disadvantage of being unmanaged by the kernel and causing unnecessary writes to the flash drive. In total, the way IPC is implemented here, makes communication between processes opaque and difficult to reason about.

F06 Buzzer has only one sound For a richer user experience and better audiovisual feedback, more sounds should be available.

F07 The system has poor code quality that makes it hard to maintain Code is mostly undocumented, function and variable names are not always chosen meaningfully. Compare Figure 3.

[5]Performance measurements of V2 can be found in Table 1.

```
def printLog(array):
    ...
    if not os.path.isfile("/home/pi/CoffeeMachine/UI/maintenance.txt"):
        global previousGrinder
        if GPIO.input(grinderPin) == 0 and previousGrinder == 1 and not currentOrder == '
            coffee':
            if os.path.isfile("/home/pi/CoffeeMachine/UI/order.txt") :
                with open("/home/pi/CoffeeMachine/UI/order.txt", "r") as j:
                    temp = j.readline()
                if temp == "water\n":
                    os.remove("/home/pi/CoffeeMachine/UI/order.txt")
            with open("/home/pi/CoffeeMachine/UI/order.txt", "a+") as f:
                f.write("coffee\n")
                f.close()
            with open("/home/pi/CoffeeMachine/UI/stop.txt", "a") as f:
                pass
            currentOrder = 'coffee'
            previousGrinder = 1
        elif GPIO.input(grinderPin) == 1 and previousGrinder == 0:
            previousGrinder = 1
        elif not (GPIO.input(waterFlow1Pin) == previousWaterFLow1) or not (GPIO.input(
            waterFlow2Pin) == previousWaterFLow2):
            previousWaterFLow2 = GPIO.input(waterFlow2Pin)
            previousWaterFLow1 = GPIO.input(waterFlow1Pin)
            idleCount = 0
            if not os.path.isfile("/home/pi/CoffeeMachine/UI/order.txt"):
                with open("/home/pi/CoffeeMachine/UI/order.txt", "a+") as f:
                    f.write("water\n")
                currentOrder = 'water'
            elif not os.path.isfile("/home/pi/CoffeeMachine/UI/unlock.txt") and os.path.isfile("/
                home/pi/CoffeeMachine/UI/order.txt") and currentOrder == 'water':
                with open("/home/pi/CoffeeMachine/UI/unlock.txt", "a") as f:
                    pass
```

Figure 3: Code excerpt from V2's `inputGPIO.py`. The name of the function is misleading. Here the `coffee_ground` and `water_flow` signals are detected, then the type of order is determined and at last unlocking of the coffee machine is initiated. The function is run in a loop with 1ms sleep after each execution.

3.3 Design and Implementation of a New Solution (V3)

In the previous section the flaws of V2 become clear. Here the measures taken for improvement in V3 will be elaborated.

3.3.1 System Architecture

The finite state machine (FSM) is the core logic component of the system and is in charge of the control flow. There are eight possible states the FSM can be in. The states are mapped to the steps of the ordering process. Signals read from the coffee machine can trigger the defined state transitions. As a result of the circumstances explained in subsection 2.1, the transitions `flow_1_started` and `flow_2_started` are triggered by the same signal, as are `flow_1_stopped` and `flow_2_stopped`.

The existing states are:

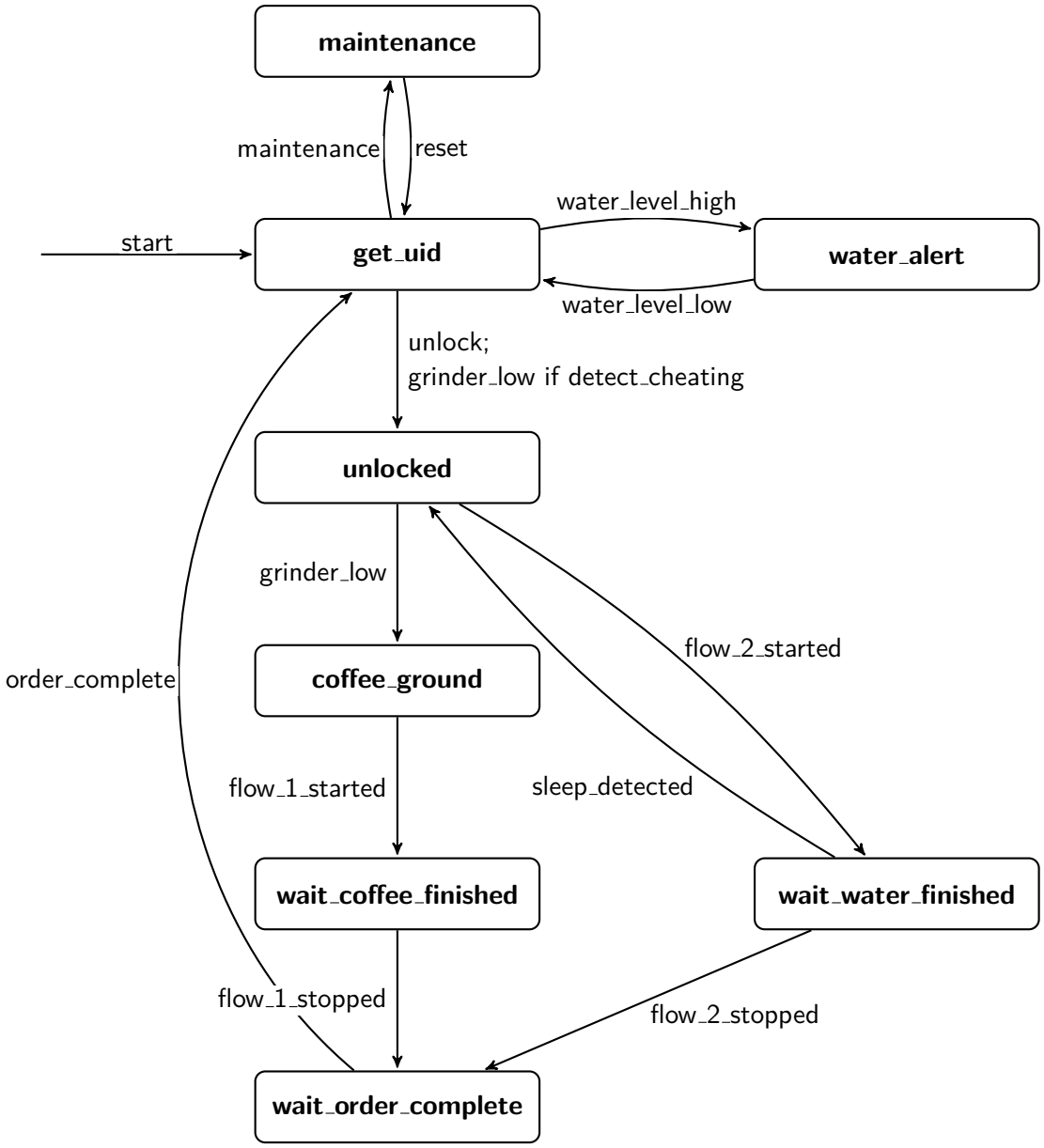

Figure 4: The finite state machine. It is the core logic component of the system.

get_uid The RFID reader continuously tries to read an RFID from a card or tag. If an RFID is read the system will try to retrieve an associated user account, if possible[6], from the accounting server, otherwise from the local database. If there is an associated user for the given RFID, the unlock transition will be triggered, if not, a message will be displayed, explaining to the user how to create an account. While in this state, the GUI will display entertainment content.

unlocked The coffee machine is unlocked by unblocking the `water_level` signal. The user is now able to use the coffee machine to order coffee or hot water. The touchscreen will now display the user's name, balance and last order. Also the milk preference, which was stored from the last order, is now preset. The user can change it until reaching the state `wait_order_complete`.

[6]The network connection could be down. See subsubsection 3.3.9

wait_water_finished The `flow_2_started` event was detected, signalling the start of hot water dispensing. On the touchscreen the user can now see the choice made. Because of the problems with the sensors described before in **BUG01** and later in subsubsection 3.3.8, the GUI will ask the user for confirmation if they really ordered water. The question can be answered with "Yes" or "No". The default is to assume that the user wanted coffee, because a false positive of water inflicts a direct financial loss. If "Yes" is selected the order will count as hot water. If "No" is selected the order will count as coffee and the user has the option to toggle the milk choice button.

After a certain time without use, the coffee machine will enter *power save mode*, also referred to as sleep. The idle time before sleep can be set in the coffee machine's menu [Sae]. Currently, it is set to 3 hours. To leave power save mode, the coffee machine's menu button has to be pressed. The coffee machine will now warm up for about 2 minutes, depending on ambient temperature, and may or may not perform a rinsing afterwards. The system has to distinguish between a valid rinsing and a user ordering hot water. Therefore, it needs to detect if the coffee machine was in power save mode. This is done by checking for the last time an order was submitted, whenever entering the `wait_water_finished` state. If the last order was more than 3 hours ago, assume the coffee machine was in power save mode and trigger the `sleep_detected` transition.

coffee_ground The `grinder_low` signal was detected. It is certain now that the user ordered coffee. The system will remain in this state until the water flow starts.

wait_coffee_finished The `flow_1_started` event was detected, signalling the start of coffee dispensing. If the user presses the "OK" button, the entertainment content will be shown and a note that the system will be ready for a new order in a few moments. At the same time the transition to `wait_order_complete` is initiated.

wait_order_complete Once reached this state, the system offers the user one last chance to change the milk choice. The user can confirm the order by pressing the "OK" button and will be presented with the entertainment content while waiting for the dispensing to finish. The order will only be completed and passed to the accounting server after the dispensing finished and an additional timer of 20 seconds ends. With the timer in place the user can still view their order and reconsider their milk choice after coffee was dispensed.

maintenance Maintenance mode was activated by a user who entered the passcode to access the admin menu. In this mode the coffee machine is unlocked and sensors are ignored. The state can be reset to `get_uid` from the admin menu.

water_alert This state is entered when the `water_level_high` event was detected. The naming refers to the logic signal from the coffee machine that switches to high when the water level in the water tank is low. Although, the signal could be toggled while dispensing coffee, the only valid transition can be started from the `get_uid` state. Still, it is important to enter the `water_alert` state after the ordering process completed, otherwise users would not get informed about the empty water tank. To keep the FSM simple, a flag variable was used instead of modeling additional states. The flag is set and reset whenever the signal is toggled. If the flag is set on entering `get_uid`, the transition to the `water_alert` state is made immediately.

3.3.2 Process Organization and IPC

Because the system has to render the GUI, read from the RFID reader and the coffee machine's sensors and play a sound on the buzzer almost at the same time. it improves performance and responsiveness when these tasks are split up into several processes. Due to the limitations of Python, using threading would not be an option here. For communication between processes, Python's Pipes are used and replace text file method described in **F05**. With the pipes a bi-directional connection between every subprocess and the main process, as is depicted in Figure 5, is established. There is a predefined set of messages in `util.py`. The messages are further separated into the two types "CMD", for command and "E", for event. Generally, if the message is a command, this indicates that it is sent by the main process and received by the subprocess. Events, on the other hand, are received by the main process and sent by the subprocess, potentially triggering a state transition in the FSM.

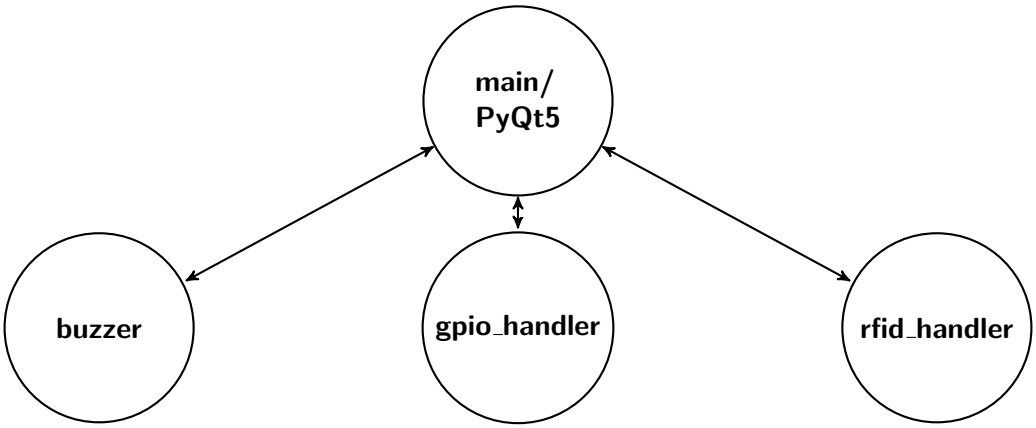

Figure 5: Relation between the four processes.

Messages shared by all subprocesses:

- CMD_PAUSE: Pause execution of subprocess until CMD_RESUME is received.
- CMD_RESUME: Resume execution of subprocess.

Messages specific to GPIOHandler:

- CMD_LOCK: Lock coffee machine.
- CMD_UNLOCK: Unlock coffee machine.
- E_FLOW_1_STARTED: The start of flow meter 1 was detected.
- E_FLOW_1_STOPPED: The stop of flow meter 1 was detected.
- E_FLOW_2_STARTED: The start of flow meter 2 was detected.
- E_FLOW_2_STOPPED: The stop of flow meter 2 was detected.
- E_WATER_LEVEL_HIGH: The water level signal was triggered to high.
- E_WATER_LEVEL_LOW: The water level signal was triggered to low.
- E_GRINDER_LOW: The grinder signal was triggered to low.

Messages specific to RFIDHandler:

- E_GOT_ID: Signal that a user id was read. It has to be directly followed by a message string containing the id just read.

Messages specific to Buzzer:

- CMD_BUZZ: Make buzzing sound.

The ability to pause and resume the execution of processes contributes to a lower resource usage. The RFID reader is not required to check for new RFIDs once a valid RFID was read, so the process can sleep while the FSM is not in the `get_uid` state.

The processes are now managed as a `systemd service`. It replaces `Cron`, which is another service on Linux systems to allow scheduling tasks. Unlike `cron`, `systemd` automatically detects crashes and can initiate a restart. Furthermore it allows easy stopping and restarting of services, and the program's console output is automatically redirected to syslog. The configuration file can be found in the appendix A.3.1.

3.3.3 Logging

The insufficiencies of V2's logging system are described in **F04**. Because of the importance for debugging, the logging system was the first topic worked on. It is indispensable for identifying the causes when encountering errors and receiving bug reports. Python offers great support for logging with the `logging` module from its standard library. To meet the requirements of this software it was adjusted to achieve that: Different log levels can be set for each imported module individually; Every log entry has the source module attached to it; Every log entry has the exact timestamp attached to it; All the logs are redirected to Raspbian's `syslog` and thus can be easily searched with standard tools. The operating system will also take care of automatically rotating logs, archiving and eventually deleting old entries. Additionally, having all the logs in one single place, makes following the timeline of events when debugging a lot easier.

3.3.4 The New GUI Design

Taking into account the problems described in 3.2, the GUI was redesigned with a focus on usability. To address **GUI01**, elements in the GUI such as the current user name, balance and milk choice were reorganized to center focus on them. Additionally, the exact time, the product choice and the price of the last order were added to the view. This information is not only "nice to have", but also helped with finding errors, since it allowed users to easily check if their last order was registered correctly. The entertainment content will only be shown before the user logs in and then again after the ordering process ends. To make it easier for first-time users, every step of the ordering process will cause visual feedback on the screen. When a user successfully logged in they will see a confirmation that the coffee machine was unlocked. The milk preference is already loaded and can be changed now or after selecting a product on the coffee machine. At this point the user can still cancel the order by pressing the "Cancel" button.

The timeout after login that was causing problem **F03** was increased to 3 minutes and thereby adds a 1 minute buffer on top of the approximate heating time. Furthermore, it is reset every time the screen is touched. Like this, the user will not have to log in several times while the coffee machine is heating.

To solve **GUI02**, the size of all buttons was increased to at least 21.68mm in height and width, which is large enough to easily tap them. This change increases effectivity and decreases frustration that occurred when more than one try was needed to change the milk choice. By using the standard style for buttons they have a familiar look and are easily recognizable as such. This is especially an improvement towards using only icons that do not look like they are clickable/touchable, as it was done in V2.

Once the product choice is recognized, the user will see both a symbolic and a textual representation of the order. The symbolic representation is either coffee beans or water drops, and a crossed out or checked milk container depending on milk choice. The textual representation is a summary of the order, showing the choice and total price. See Figure 7.

Freezing of the GUI, as described in **BUG03**, was found to be the result of the system being trapped in one or more undefined situations that occurred in edge cases. As a side effect from the implementation of the state machine in V3 these problems now can be ruled out.

In order to resolve **GUI06**, the screen saver was deactivated permanently.

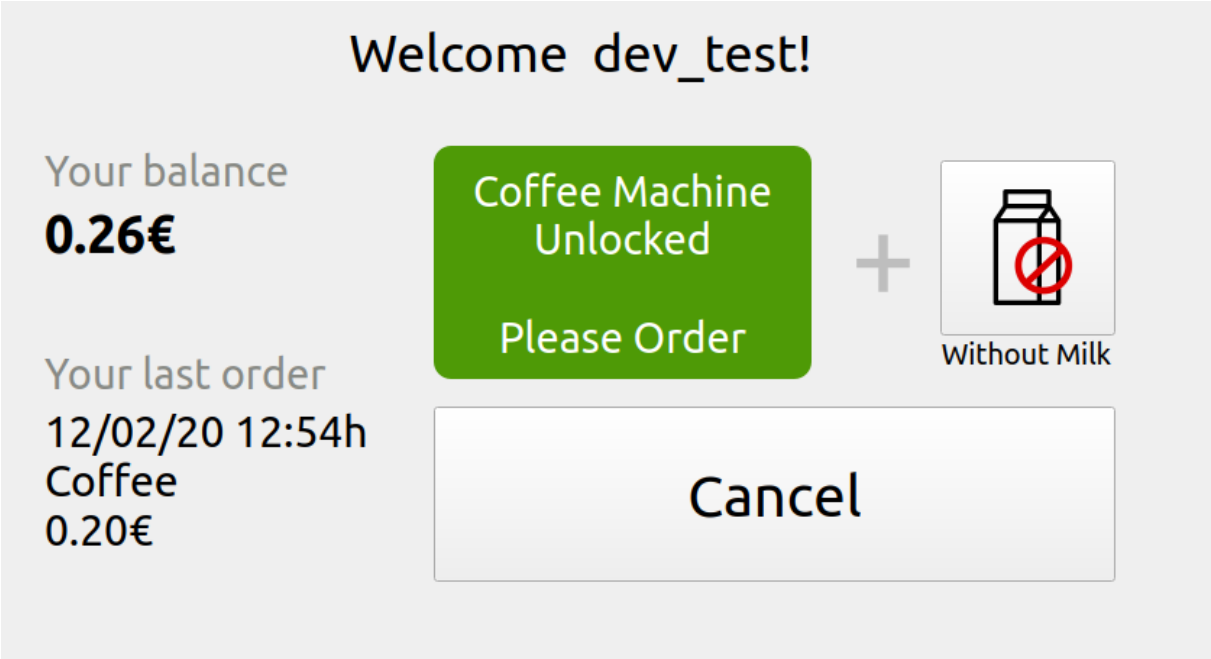

Figure 6: The GUI when the user successfully logged in.

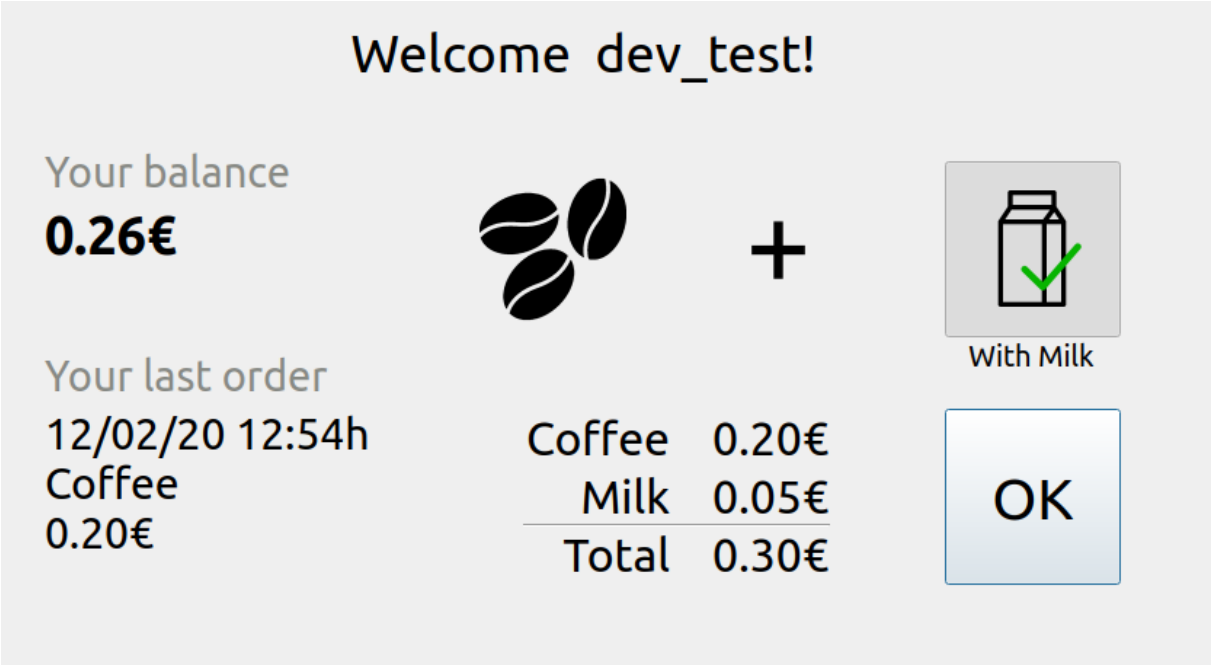

Figure 7: The GUI after the user selected coffee on the coffee machine. The "With Milk" button was toggled to add milk to the order.

3.3.5 Dispensing Limit

By setting a dispensing limit, users are encouraged to pay their debts earlier. The dispensing limit is reached when the balance of an account falls below a certain threshold. There are two stages, which can be configured individually. The first stage is merely a warning that is displayed in the GUI every time a user logs in, right before selecting a product. But once the second stage is reached, the user will not be able to unlock the coffee machine without recharging their account balance before. Instructions on how to recharge their balance are shown to the users in this case, too.

Although a mechanism limiting dispensing like this existed in V1, there is no sign of an implementation or description of such a mechanism in V2.

The accounting server has basic support for such a mechanism but there is no properly designed interface[7] to access the information. Because changes to the accounting server are meant to be kept minimal, the better solution was to implement a standalone mechanism on the Raspberry Pi. In the appendix A.1.2 there is an explanation on how to set the thresholds for the two stages.

3.3.6 Replacing the Buzzer

The previously used buzzer was replaced by one of a different type to allow varying tones to be played (**F06**). The new buzzer mainly consists of a piezo christal that moves to either one position when it is pulled to high or low. One can hear a single clicking sound when the input changes from low to high. Now, by applying a square wave input signal with a frequency in the audible spectrum - and the spectrum permitted by the buzzer's physical constraints - a single note can be played. For example, switching the signal on and off with a frequency of 440Hz results in the note A4 being played. A frequency of 220Hz would result in an A3, 880Hz would be an A5. Since the wave signal is synthesized in software, occasional deviances of sound depending on processor load will occur.

At first, the new buzzer's volume was not satisfactory when directly attached to the Raspberry Pi's GPIO output, because the Raspberry Pi's GPIO output voltage is 3.3V at maximum. Increasing the buzzer's input voltage directly results in a higher volume. Therefore, in order to get a higher volume, it was sufficient to create a CMOS circuit for switching the 5V available from the Raspberry Pi's power lane as the buzzer's input voltage. There is a schematic of the amplifier circuit in Figure 8.

The new sounds for login and logout are a low tone followed by a high tone for login and a high tone followed by a low tone for logout. The logout sound is also used for signalling that the water tank was successfully refilled. When the account is locked, because of the dispensing limit a special sound is played.

3.3.7 Debugging the RFID Reader

With the help of the logging system the problems with the RFID reader, described in **BUG07**, could be narrowed down to be related to the software library, `mfrc522`, used for connecting to the device. The logs proofed that all other parts of the system were executing as expected. Especially, the loop in which the code for accessing the library was ran every half second, always returned without an error, invalidating the initial theory that the process was killed or blocking at some point. This led to the key observation that after reinitializing the library, the reader was fully functional again. Consequently, the simplest solution was to use a timer to completely reinitialize the RFID reader every five minutes. Further investigation into the topic could start with the "power reduction modes", which are mentioned in the modules data sheet [NXP16].

The KIT-Card issue **F01** was resolved by simply updating the library.

[7] The current state for each user would have to be parsed from the different colors and text in an HTML document.

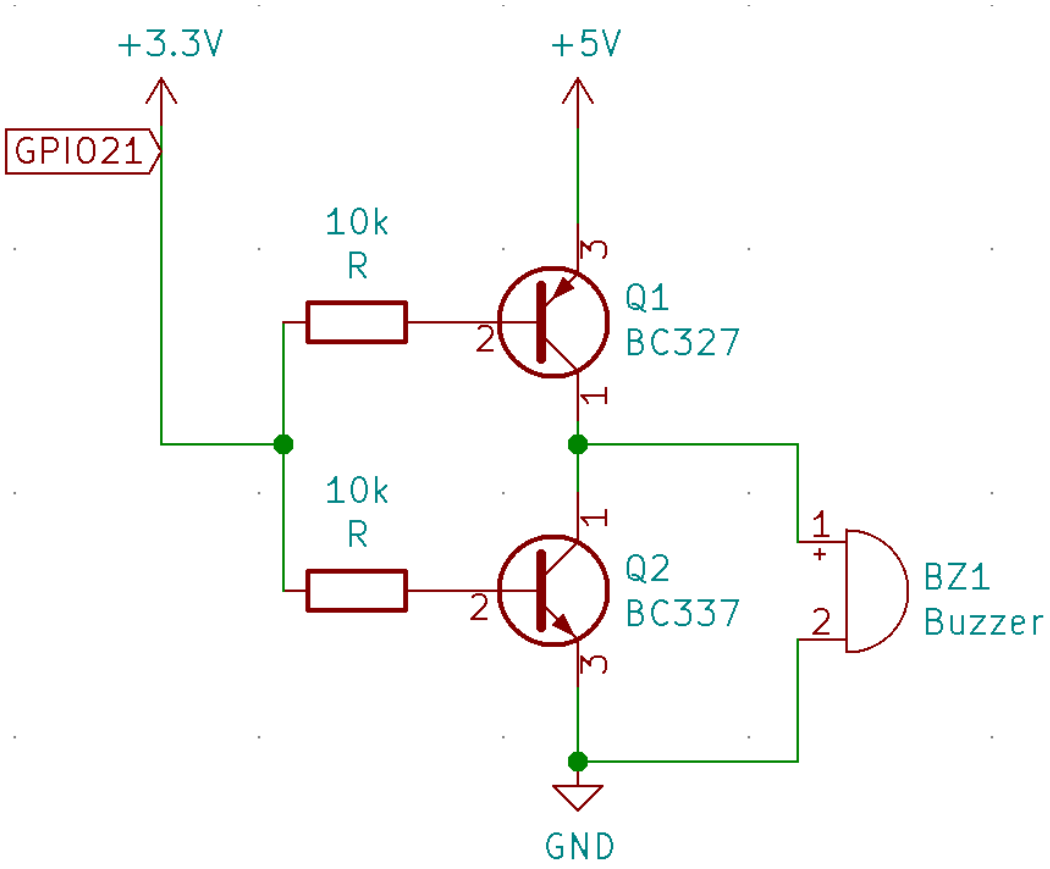

Figure 8: Schematic of the amplifier built for the buzzer.

3.3.8 Changes in GPIO Handling

To reduce the high CPU usage mentioned in **F02**, the method to read the GPIO signals was changed. Instead of polling at a sample rate of 1kHz in less performant Python code, the responsibility for polling was completely moved to the GPIO library. By setting up interrupts to trigger callbacks, hardware capabilities for polling can be leveraged and, in consequence, significantly reduce CPU usage. In the process of rewriting GPIO handling, RPi.GPIO was replaced by pigpio.

After implementing the logging system an important observation was that the `water_alert` and `coffee_ground` events were triggered seemingly at random, without relation to any user interaction. Sometimes an event could be triggered by switching the light off and on again in the same room the system is set up. This indicates a high sensitivity to noise from the power supply. To avoid problems caused by falsely recognized signals, a glitch filter was applied to every signal read from the machine. The glitch filter effectively filters noise from signals by only toggling the signal if the input was in a stable state for a fixed period of time. Measurements were performed throughout a month to determine a suitable parameter for the filter. Switching the GPIO library from RPi.GPIO to pigpio, made using a glitch filter an easy task, as it already comes implemented with pigpio and can be activated optionally. The parameters for the signals were set to 3500µs for water_flow, 10000µs for water_level and 30µs for grinder. The values were determined by measuring and observing each signal. The sample rate of pigpio is set to 200kHz, i.e., a signal is polled every 5µs. Signals with a lower resolution than that will not be recognized.

Sometimes the grinder signal is not recognized, which is one of the reasons that

BUG01 occurred. The noise filter as a possible cause for the bug could be ruled out after measuring the signal with the filter deactivated. Keeping in mind that the lowest resolution signal that gets recognized has a period length of 10µs, and that typical period lengths for the coffee ground signal are about 100µs, it seems very unlikely that a too low sample rate could be the problem. Unfortunately, the exact reason for this problem could not be found nor explained by software related issues. Which leaves a fault in the adapter board as the most probable cause for the problem.

3.3.9 Operation in Offline Mode

To reflect **BUG06**, the system was designed with possible network outages in mind. So when the accounting server cannot be reached, a user is still able to use the coffee machine. Furthermore, the system will handle network errors gracefully and will not crash.

For the offline mode a local copy of user data and a log of orders is stored in an SQL database running locally on the Raspberry Pi. The DBMS used is *MariaDB*. Figure 9 gives an overview of the database schema, showing the three tables required. Every order is linked to the RFID tag, that was used to authenticate. RFID tags are called `uid` in the database. A user can have more than one uid and therefore the `uids` table relates uids to actual user ids in the `users` table.

The entries in the database are created whenever a user tries to authenticate with the system. Then the data is fetched from the accounting server and updated in the database. Consequently, only if the user logged into the system with a newly registered RFID tag at least once, a copy of the account information is stored locally.

To be able to synchronize the orders with the accounting server once the network connection is up again, there is an `is_synchronized` flag stored with every order. In case the order cannot be passed to the accounting server, the flag is set to `False`. Only after successfully pushing the order to the accounting server, the flag is set to `True`. To push all unsynchronized orders to the accounting server, the scheduler periodically initiates the synchronization process. The limitations when operating in offline mode are:

1. There is no support for recharging a user's balance.
   While there is a copy of the user's balance which is updated for every order made, this copy does not reflect any updates to the balance on the accounting server. Only when the internet connection is back online, the system will fetch the updated balance from the accounting server.

2. Users need to have used their RFID tag at least once before.
   If a user did not use their RFID tag before, the RFID tag will appear as unregistered.

3. No updates of entertainment content are possible.

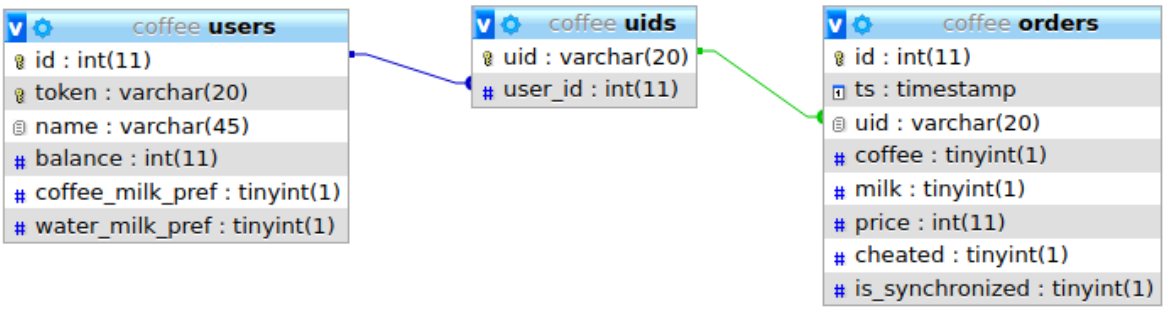

Figure 9: Database schema for offline mode and milk preference.

3.3.10 Storing the Milk Preference

For the feature as described in **GUI03**, the assumption is made that a user habitually drinks their coffee either with or without milk. This habit would only change rarely and the most probable cause would be that there is no milk left. Currently, the milk preference is set after every order and, in consequence, just reflects the last choice made. Thus, the simplest way to implement the feature would be retrieving the milk choice from the last order. Nevertheless, the preference is stored separately in the users table in the local database (Figure 9). This leaves room for further improvement of the feature, maybe by adding a separate menu to change the preference.

Initially, it was assumed that users who drink coffee sometimes also take hot water to make tea and possibly want to drink it with milk. Therefore, a milk preference for hot water was added. Later this idea was dropped for three reasons: 1. Users were accustomed to make the milk choice before selecting a product on the coffee machine and it would not have been possible to pre-load the milk choice before the system knows if the user gets coffee or hot water; 2. There was no known user who drank their tea with milk; 3. Non trivial changes in the accounting server would have to be made to be able to register hot water with milk. Nonetheless, the code and the database column for this feature were not removed.

3.3.11 Entertainment Content

Along with the new design of the GUI, the entertainment content feature was overhauled, also adressing **GUI04**. While the system is in the state `get_uid`, the user can view images for entertainment. These images were only comics in V2 and now are complemented by a news feed in V3 . Completely new are the ability to use two-finger swipe for zooming and panning the view, and an automatic slideshow. A screenshot of the GUI displaying entertainment content can be found in the appendix A.5. The software will load any image, with a supported file extension, located in `./res/images/`. After displaying an image for 90 seconds, the next image will be shown automatically. The supported file extensions are `.jpg, .jpeg, .png, .gif`. There is no support for animated GIFs.

To always display new content, images are downloaded from different services automatically. In order to integrate better with the image viewing mechanism, news articles are simply rendered from an RSS text feed into an image. A limit for the maximum number of images to keep from each service can be set. All images beyond that limit will be deleted after an update, oldest first. The system is kept simple to allow for easy extension with new services.

3.3.12 Scheduler

It is necessary to periodically synchronize offline orders, download images for entertainment content and afterwards reload the image viewer. Therefore a simple scheduler was implemented. It is run in a separate thread, using only one single timer. All registered tasks will be triggered by the same timer, every 3 hours. A useful feature is to be able to pass a condition for the execution of the scheduled tasks. This makes it possible to only allow the scheduler to run a task when there is no order currently in progress, avoiding additional load that could have a negative impact on the ordering process.

3.3.13 Cheating Protection

In V2 there were two ways of tricking the system into unlocking the machine without registering the order. The first bug that can be exploited when refilling the water tank is already described as **BUG04**. Rewriting the GPIO handling and IPC removed latencies that were causing this issue.

BUG05 describes the second bug that a user could exploit to get free coffee. It was possible by ordering a coffee and then immediately pressing the cancel button in the GUI. This is dealt with by saving the timestamp when a user logs in and then, if a `coffee_ground` event occurs while in the state `get_uid`, checking if the last login was within a time frame of 270 seconds. The time frame is the sum of the current cancel timeout for logging the user out after 180 seconds, and a 90 seconds buffer where coffee could be ground. If the last login was within that time frame, the user is logged in again and the system will proceed as with a normal order. The only difference will be an additional note that will be sent to the accounting server, documenting the cheating attempt.

3.3.14 Code Quality

While implementing new features and reorganizing code, special attention was directed to creating documentation and writing clean code. As a result, **F07** could be improved.

3.4 An Improved Design for the 3D-Printed Case

The first step in the design process is to create the model in Blender. One has to keep in mind that, depending on the 3D printer's dimensions, a model has to be sliced into several smaller parts to fit. In this case it is three different parts.

For the new design the problems found in **BUG08** were fixed. The thickness for all outside walls was set to 3mm uniformly. This measure was found to result in a stable structure that does neither break nor deform too easily. It also does not waste as much material. The top part slides into the bottom part from the rear and is locked into place with a screw. Little cylinders that fit exactly into the Raspberry Pi's screwing holes are located on the floor of the bottom part. The USB and LAN sockets fit exactly into the excavations in the rear side. Together with the top part pushing down on the Raspberry Pi, the board is held in place without any additional screws. A socket for screwing the RFID reader to the top part from below was added, too. On the upper side of the top part the RFID symbol is engraved to show users where to place their RFID. On the rear side of the top part there are openings for air circulation. On the rear side of the bottom part there is one hole for the LAN cable and another opening where one part of the adapter board can fit through. After receiving user feedback about key chains with RFID chips

attached falling down from the top, a barrier was added around the top surface to prevent this from happening. The final 3D model can be seen in Figure 10.

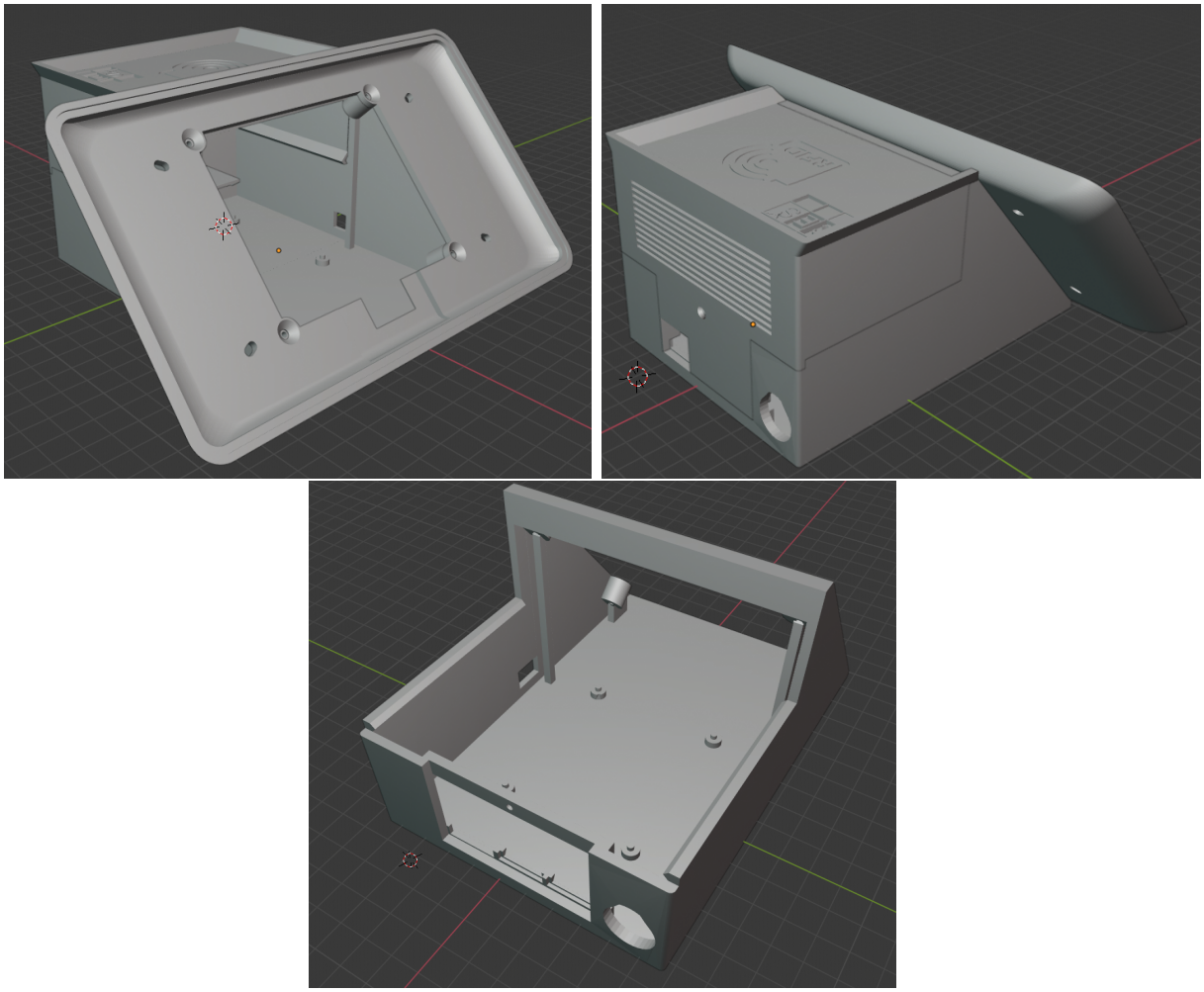

Figure 10: The final 3D model of the case for V3.

Once the model has the desired shape, the mesh has to be cleaned up and checked for errors. In Blender 2.8+ this is done by selecting "Mesh > Clean Up > Merge by Distance", "Mesh > Clean Up > Delete Loose" and "Mesh > Normals > Recalculate Outside". These commands will remove obsolete geometry and recalculate the inside and outside of the mesh. The check that follows afterwards is the most important one, as it checks for solidity. The most common problem is non-manifold geometry, internal faces, areas with no thickness or disconnected vertices, as these make it impossible to calculate the models volume unambiguously. In initial iterations of the design process, the model appeared to be solid, when in fact it was not. This led to parts not being connected, components falling apart and walls being hollow and filled with uncured resin. Therefore, the "3D-Print Toolbox" add-on has to be used to analyze the mesh for potential problems. The tool will highlight ambiguous geometry that has to be corrected manually most of the times.

The next step consists of exporting the model from Blender and importing it into formlab's *PreForm* tool using the STL file format. The vendors software can automatically repair a few problems in the model if they are not too grave. A preview of the model is shown and it has to be processed for printing. Supporting structures are added

automatically to prevent overhanging parts from falling. This can be seen in Figure 11 The structures can be removed after printing. However, they will cause a bumpy surface on planes that they are attached to. In consequence, when placing the structures, which is done implicitly by changing the models orientation, less visible contact surfaces should be preferred. Ideally, supporting structures are not attached to small parts that could accidentally be torn of when removing the structures. The models orientation directly influences the amount of structures, material and printing time needed. It is a good practice to experiment with different orientations to preview the calculated supporting structures and material requirements.

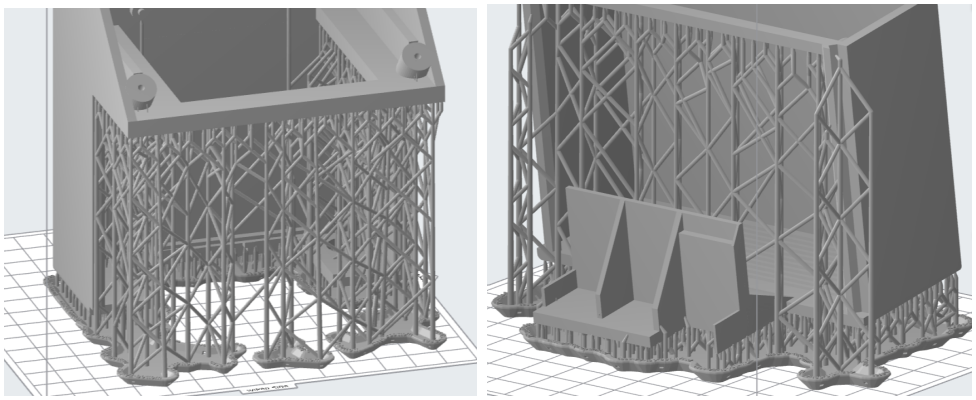

Figure 11: 3D models of top and bottom part with supporting structures.

4 Results

4.1 Improvement of Usability

Measuring the improvement of usability can be difficult to put in numbers. In theory usage studies would be performed, interviewing every user or letting them fill out a questionnaire. However, in this environment it would be impractical to perform a study like this. In consequence, the improvement of usability is only measured in terms of changes made to reflect the requirements for usability that were examined in subsection 3.1. Out of the 21 problems and shortcomings listed in subsection 3.2, only 3 are not directly related to usability considerations. These three problems are F02, F04 and F07; CPU usage, logging and code quality. More importantly, the remaining part of issues that had negative effects on usability was resolved.

Furthermore, it can be argued that because of the iterative evaluation of design changes and the integration of user feedback during the development process, usability and overall user satisfaction with the system were naturally improved.

4.2 Improvement of Reliability

To measure the improvement of reliabilty in numbers, statistics from V2 would be required. Apart from subjective opinion from users, unfortunately, there is not much data on improvement. Although informal and hard to quantify, it should be mentioned that users said they were able to rely on the system again in a way that they could be sure they can use it any time they want, without running into problems. Also it is worth to mention that the consecutive uptime of the system in V3 significantly increased when compared

to V2, which indicates that there was no need for manually restarting the system when in an error state. However, no guarantees can be made for a certain availability.

4.3 Improvement of Performance

To measure and compare the performance of both versions, the system was rebooted and then it ran for one hour before the statistics were retrieved with the Unix command `ps axo pcpu,pmem,args`. During the time of measurement no orders were taken. The results can be seen in Table 1 and Table 2. It is easy to conlude that the changes made in 3.3.8 to improve the CPU usage were effective. While in V2 CPU usage for the GPIO and buzzer functionality alone was at 98.1%, it dropped to a cumulated value of 6.3% in V3.

% CPU	% memory	python module
98.1	1.8	inputGPIO.py (GPIO & buzzer)
12.4	2.1	inputGPIO.py (RFID)
0.1	10.2	main.py (GUI)
0.0	1.6	inputGPIO.py (locking)

Table 1: Performance measurements of version V2, one hour after boot.

% CPU	% memory	python module
5.9	0.1	pigpiod (GPIO)
16.3	3.9	main.py (RFID)
0.7	10.6	main.py (GUI)
0.4	4.6	main.py (GPIO & locking)
0.0	4.4	main.py (buzzer)

Table 2: Performance measurements of version V3, one hour after boot.

% CPU is the “cpu utilization of the process in ”##.#” format. Currently, it is the CPU time used divided by the time the process has been running (cputime/realtime ratio), expressed as a percentage.” [LJS+18], and % memory is the “ratio of the process’s resident set size to the physical memory on the machine, expressed as a percentage.”.

4.4 3D Printing Cost Calculation

In the end, the amount of resin needed for the final 3D model printed were: For the top part, volume without support structures 65.63ml, total volume 90,79ml. For the bottom part, volume without support structures 96.88ml, total volume 150.08ml.

The Form 2 Resin Tank is sold for €65.45 and has to be replaced after a printing volume of 1-1.5 liters. 1 liter of the Grey Resin[8], which was the type of resin used, costs €160.65. These are the prices from the manufacturer’s online shop plus german VAT. A conservative cost calculation for printing would be:

$$\frac{€65.45 + €160.65}{1000ml} = €0.2261/ml$$

With the cost per ml the costs for the different parts can be calculated:

[8]Product number: RS-F2-GPGR-04

- The top part, with a volume of 90.79ml, costs €20,53.
- The bottom part, with a volume of 150.08ml, costs €33.93.
- The screen frame, with a volume 53.60ml+[9], costs €12.12.

This results in a total of €66.58 in printing costs for the final model. Taking into account all the failed printing attempts during development of the model, including the models for V2, the total material cost could be estimated to lie above €300.

5 Conclusion

In the introduction the goal of this thesis was outlined as documenting the improvement of V2's usability and reliability. The design and implementation changes with the greatest positive impact on these two aspects are: Controlling core logic with the FSM, using interrupts instead of polling for GPIO, having a more user-friendly design for the GUI. To evaluate the different design choices made, feedback from the users was gathered during daily usage and later utilized to optimize the design in several iterations. The new 3D printed case gives the whole appliance a more elegant look that users appreciate. Throughout the thesis the importance of usability during the design process was illustrated. Although the results for usability and reliability are vague, it can be confidently concluded that the final product has now reached a point were it absolutely is an advantage over using a paper tally sheet.

6 Future Work

Debugging the hardware to find what is causing the coffee grinder signal not to be recognized should be considered the most important topic to work on. The workaround that was created for V3, does prevent financial loss, but otherwise is not very satisfying.

Another category of features could be implementing more warnings in the GUI for people with undesirable behavior, such as the "bad milk guys" or people who leave it to the next person to refill the water tank. For a greater effect, an alarm sound could be played with the buzzer, once a certain warning is displayed. Additionally, these warnings could be sent out automatically to every user's email address.

Because the system is switched on 24/7, it is not ideal that the screensaver is currently deactivated permanently. A screensaver that only gets enabled outside of the normal usage hours, and only if the water tank is not currently empty, could be developed. Ideally, the screensaver would be disabled by touching the screen.

To take the idea of saving power a step further, the Raspberry Pi could be equipped with a relais for 330VAC power to control the coffee machine. Like that, the coffee machine could be switched off completely at night hours, only switching on when required. This might also make it possible to control the coffee machine's heating. Another interesting feature to improve usability and save time could evolve out of this. Users could have the possibility to go on a website served by the accounting server and initiate the heating of the coffee machine before they go to the room where the coffee machine is located.

[9]For the screen frame only the volume without supporting structures is available, as it was already printed for V2.

References

[AGM+15] Ala AlFuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols and applications. *IEEE Communications Surveys and Tutorials*, 17:Fourthquarter 2015, 11 2015.

[Amm17] Karim Ben Ammar. Design and implementation of an IoT-based control device for an off-the-shelf coffee machine, 2017.

[Cor] Broadcom Corporation. BCM2837 ARM Peripherals. `https://github.com/raspberrypi/documentation/files/1888662/BCM2837-ARM-Peripherals.-.Revised.-.V2-1.pdf`.

[for] formlabs. formlabs Form2. `https://formlabs.com/3d-printers/form-2/`. Accessed on 2020-02-19.

[Fou] Raspberry Pi Foundation. GPIO. `https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/`. Accessed on 2020-02-19.

[ISO18] ISO. ISO 9241-11:2018 Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts, 2018.

[LJS+18] Brank Lankester, Michael Johnson, Michael Shields, Charles Blake, David Mossberger-Tang, and Albert Cahalan. ps(1) Linux man page, 2018.

[Mod14] Tobias Modschiedler. Upgrading an off-the-shelf coffee machine with RFID-based access control, 2014.

[MRT15] Somayya Madakam, R Ramaswamy, and Siddharth Tripathi. Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications*, 3:164–173, 04 2015.

[NXP16] NXP. MFRC-522 Standard performance MIFARE and NTAG frontend. `https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf`, 2016. Accessed on 2020-02-19.

[Pok19] Slavko Pokorni. Reliability and availability of the Internet of things. *Military Technical Courier*, 67:588, 06 2019.

[Sae] Saeco. Saeco Royal Professional Operating Instructions. `https://www.download.p4c.philips.com/files/0/0351.0us.77g/0351.0us.77g_dfu_aen.pdf`. Accessed on 2020-02-19.

A Appendix

A.1 Settings

The system has to be restarted after making any changes described here. Use `sudo systemctl restart coffee-v3`

A.1.1 How to Change Prices

It is not sufficient to change prices in the accounting server's database. Price updates are not automatically fetched by the system, because they do not occur frequently enough. To change the prices, the values of the according constants in `order.py` have to be adjusted.

A.1.2 How to Change Dispensing Limit

To change the thresholds, which the user's balance has to fall below before

1. a warning about the account being locked is shown and
2. the account is locked,

adjust the constants `DISPENSING_WARN` and `DISPENSING_DENY` in `main.py`. Both positive and negative values are allowed. However, `DISPENSING_WARN` should always have a greater value than `DISPENSING_DENY`.

A.1.3 How to Change Admin Passcode

The source code for the admin login can be found in `src/gui/admin_login.py`. There the passcode is stored in the variable `self.password` and can be changed to any series of numbers.

A.2 Raspberry Pi 3 Model B V1.2 Specifications

- Broadcom BCM2837, quadcore Cortex-A53 (ARMv8) 64-bit SoC @ 1.2GHz
- 1GB LPDDR2 SDRAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- Extended 40-pin GPIO header
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input

A.3 System configuration

A.3.1 Systemd Service

```
[Unit]
Description=coffee-v3

[Service]
User=pi
Environment="DISPLAY=:0"
Environment="XDG_DATA_DIRS=/usr/local/share:/usr/share/raspi-ui-overrides:/usr/share:/usr/
    share/gdm:/var/lib/menu-xdg"
Environment="XDG_RUNTIME_DIR=/run/user/1000"
Environment="XAUTHORITY=/home/pi/.Xauthority"
WorkingDirectory=/home/pi/coffee-v3/Thesis_Simon/coffee-v3
ExecStart=/usr/bin/python3 src/main.py
Restart=always

[Install]
WantedBy=graphical.target
```

Figure 12: The configuration file for the systemd service.

A.3.2 Security

The Raspberry Pi has a static and publicly accessible IPv4-Address. It is a so called exposed host, not protected by a network firewall. To lower the risk of unauthorized access, the following measures were taken: *fail2ban* bans ips that fail to authenticate with ssh several times in a row. *iptables* were configured to only accept connections from inside KIT-Network and drop all other connections. *unattended-upgrades* is used to automatically install security updates.

A.4 Accounting Server

A.4.1 Fixing Problems after Update

The statistics diagram on the accounting server's website stopped working properly. This was related to an operating system upgrade on the server which updated php and mysql versions.

The new version of php is 7.2. It was php5.* before. Statements had to be replaced as follows to run the code without errors:

```
# pre php7
mysqli_escape_string($string)

# php7
mysqli_escape_string($link, $string)
```

The new version of the MySQL DBMS is 5.7.28, before it was below 5.7.5. MySQL 5.7.5 and newer activate the ONLY_FULL_GROUP_BY mode by default. In this mode all queries are checked for functional dependence. MySQL rejects queries for which the select list, HAVING condition, or ORDER BY list refer to nonaggregated columns that are neither named in the GROUP BY clause nor are functionally dependent on them.

A.4.2 Repairing Order Log on Accounting Server

There was a bug in the new software that caused all orders to be processed as “Coffee with Milk”. This condition lasted about one month before it was detected and fixed. During that time users were charged more money for their black coffees and hot waters. Luckily, there is a local copy of the orders on the Raspberry Pi. This could be used to repair the false data. First the data in the affected time period were extracted from the databases. Then a Python script was used to merge the data and create an SQL query for every corresponding database row.

```
# accounting server database
SELECT * FROM coffee.protokoll WHERE id >= 20643 AND id <= 20974 AND
   Kommentar LIKE '%rfid%' AND UserID != 124;

# local database
SELECT * FROM coffee.orders where id >= 10 and id <= 277 and uid !=
   0123;
```

Ultimately, after performing a database backup via the `mysqldump` CLI tool, the generated sql update statements were applied to the accounting server database.

```
SET sql\_mode=(SELECT REPLACE(@@sql\_mode,'ONLY\_FULL\_GROUP\_BY
   ','''));
```

A.5 Figures

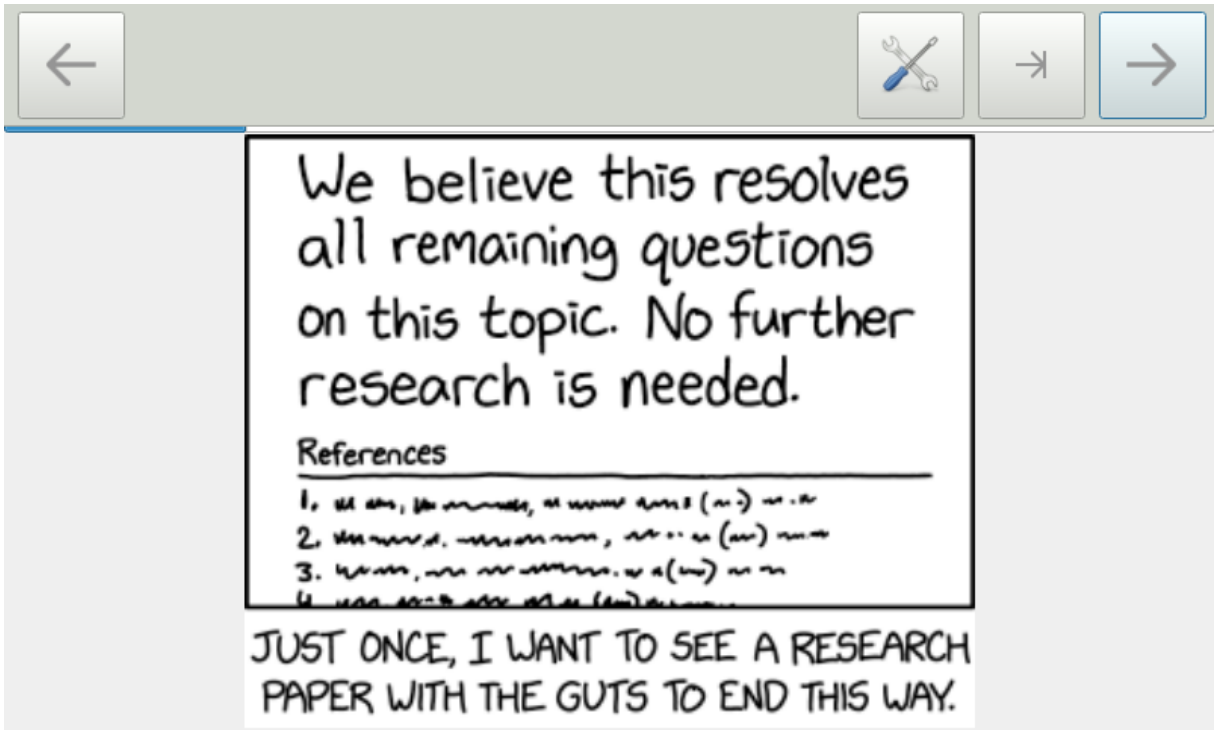

Figure 13: Entertainment content in the GUI.