Karlsruher Institut für Technologie

Institut für Technische Informatik (ITEC)
Lehrstuhl für Eingebettete Systeme (CES)
Prof. Dr. Jörg Henkel

Upgrading an off-the-shelf coffee machine with RFID-based access control

Studienarbeit
von

Tobias Modschiedler

an der Fakultät für Informatik

Tag der Anmeldung: 20.03.2014
Tag der Abmeldung: 20.06.2014

Betreuer:
Prof. Dr. Jörg Henkel

Betreuende Mitarbeiter:
Dipl.-Inform. Artjom Grudnitsky
Dr.-Ing. Lars Bauer

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, 20.06.2014

Contents

List of Figures

Acronyms

RFID Radio-Frequency Identification

NFC Near Field Communication

UID Unique Identifier

SPI Serial Periphal Interface

UART Universal Asynchronous Receiver Transmitter

1 Introduction

A coffee machine is an important social facility in an academic institute. Various groups of people use it every day: academic employees, professors, students, and other employees all like to get a coffee (or other hot beverage requiring hot water) during their breaks from work.

Of course, this facility requires organizational effort. Coffee beans and other supplies have to be obtained and the machine has to be cleaned regularly. These tasks can easily be distributed among a small group of dedicated coffee enthusiasts. However, another more complex and more error-prone issue needs to be tackled in order to make a service like this sustainable: the fair distribution of all costs to the whole user base, including proper accounting and charging.

This work describes the design and implementation of an interactive access control upgrade to an off-the-shelf coffee machine, thereby automating the accounting system. The main aim is making accounting as easy as possible while providing a user-friendly interface, thereby minimizing the probability of errors and resulting financial losses.

1.1 Requirements

The following points are stipulated for a successful implementation:

- The upgraded coffee machine shall be locked when no user is authenticated, i.e. it shall not be possible to get any product from it.
- For authentication, RFID tokens shall be used. They are cheap and easily replaceable. Additionally, every KIT member already holds an RFID-based student/employee card which can be used for this purpose.
- The authentication system shall use an existing accounting system (see section 2.2) to get user information. Whenever the machine produces a beverage, this information shall be submitted back for accounting.

  - Even if the accounting system is not reachable (e.g. due to a network outage), known users should be able to unlock the coffee machine.
- The human interface should be easy to use and visually appealing.
- The coffee machine shall only be modified as minimally as possible. All modifications should be reversible so the machine may still be serviced by the manufacturer in case of a malfunction.

1.2 Structure of this work

Chapter 2 shows the existing hardware and software which is used as a starting point for this work: the coffee machine and the pre-existing accounting system. In chapter 3, the design of the access-control upgrade is presented, giving an overview of the hardware used and the software developed in this work. Subsequently, chapter 4 details the concrete implementation of these components. Chapter 5 concludes the work by summarizing and giving an outlook.

2 Background

2.1 Coffee machine

The coffee machine that is being upgraded in this work is a *Saeco Royal Professional* (SUP016R) as pictured in figure 2.1. This is a fully automatic coffee machine, i.e. the user just has to push a button to make the machine grind the required amount of coffee beans and complete the brewing process. Apart from coffee, the machine can also deliver hot water and steam.

Figure 2.1: Saeco Royal Professional (from http://www.philips.de/c-p/RI9914_01/royal-kaffeevollautomaten)

To implement the upgrade described in this work, the machine needs to be locked until a user has authenticated, i.e. it must not be possible to get any product beforehand. Additionally, access to information about the machine state is required to detect e.g. the start and end of a grinding/brewing process. Unfortunately, the machine does not offer a diagnostic interface or other means to monitor or control the brewing process from the outside. Furthermore, Saeco does not provide any detailed documentation about the control electronics used in its products.

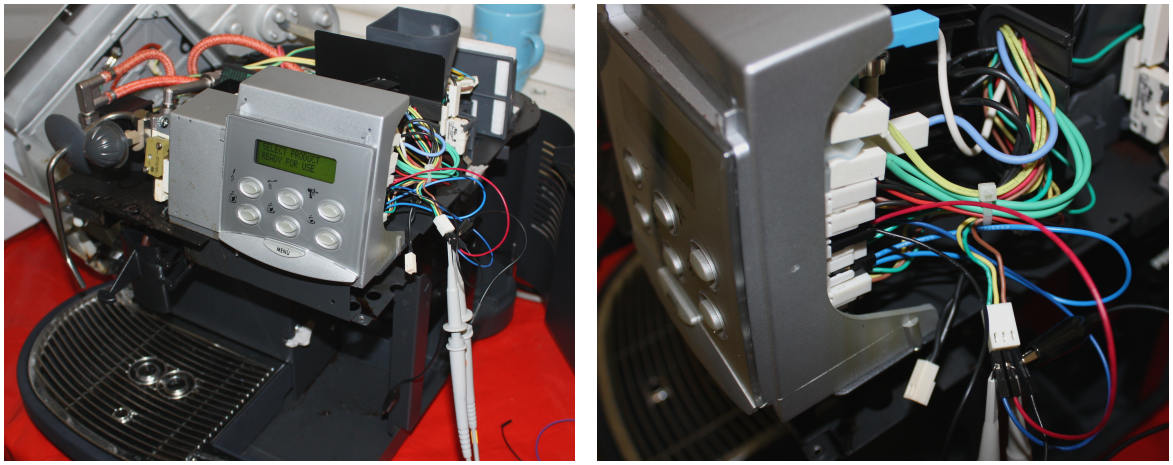

Figure 2.2: Opened Saeco Royal Professional with oscilloscope probes attached to one of the two water flow meters

As a result, some reverse engineering of the control circuit is necessary to find out which useful signals are obtainable without significantly modifying the machine. The machine can be opened easily without causing any damage. Figure 2.2 shows the opened machine.

The various sensors of the machine are connected to the control circuit using detachable connectors. Luckily, these are located at the edge of the control board and therefore easily accessible. In figure 2.2, two oscilloscope probes attached to one of the connectors can be seen.

2.1.1 Electrical signals/connections

No detailed documentation about the coffee machine control circuits is available from Saeco, so the required information needs to be obtained by reverse engineering. Using an oscilloscope in combination with a digital multimeter, the accessible signals can be observed during a grinding/brewing process. Assisted by information found on the internet (e.g. [Fri07]) describing similar control boards, the following functions can be assigned to the different connectors:

Two wires (black/black): Water level sensor, implemented by a reed switch that is is operated by a magnet floating in the water tank.

Two wires (green/black): End-of-grinding sensor, implemented by a micro switch that is closed when the chamber for ground coffee is full.

Three wires (green/yellow/brown, bottom): Water flow sensor for the water circuit used for brewing coffee, using a turbine with a hall effect sensor.

Three wires (green/yellow/brown, top): Second water flow sensor for the water circuit used for making hot water and steam.

The other connectors in the vicinity lead to micro switches used to track the state of various mechanical components (e.g. position of the brewing unit) as well as temperature sensors and power supply lines. These are not interesting for this work.

It would be useful to gain access to the circuits of the buttons offered by the coffee machine, e.g. to monitor user input or disable them. Unfortunately, they are mounted directly on the inaccesible side of the control board and therefore cannot easily/reversibly be intercepted. For this reason, the button signals are not directly used in this work.

The signals found on the four connectors listed above are used to implement the control concept as described in section 3.2.1. The following sections describe their exact functions and electrical properties. An overview of the brew process with regard to these signals is given in figure 2.3.

Flow sensors

The two flow sensors use three pins each. Their function is:

green wire	Supply Voltage (5 V)
brown wire	Ground
yellow wire	Output

According to [Fri02], either the device *Allegro UGN3140U* or *Micronas HAL 506K* is used. These sensors use an open-collector/open-drain output transistor so that their output is pulled to ground when a magnetic field is sensed (see [All93]/[Mic10]). Combined with a turbine that rotates a permanent magnet, this can be used as a flow sensor.

On the Saeco control board, these sensor outputs are connected to a microcontroller input together with a 4.7 kOhm pull-up resistor. This is pictured in figure 2.4. By probing this signal with an oscilloscope during the brewing process, a square wave with a frequency of 13–14 Hz can be observed.

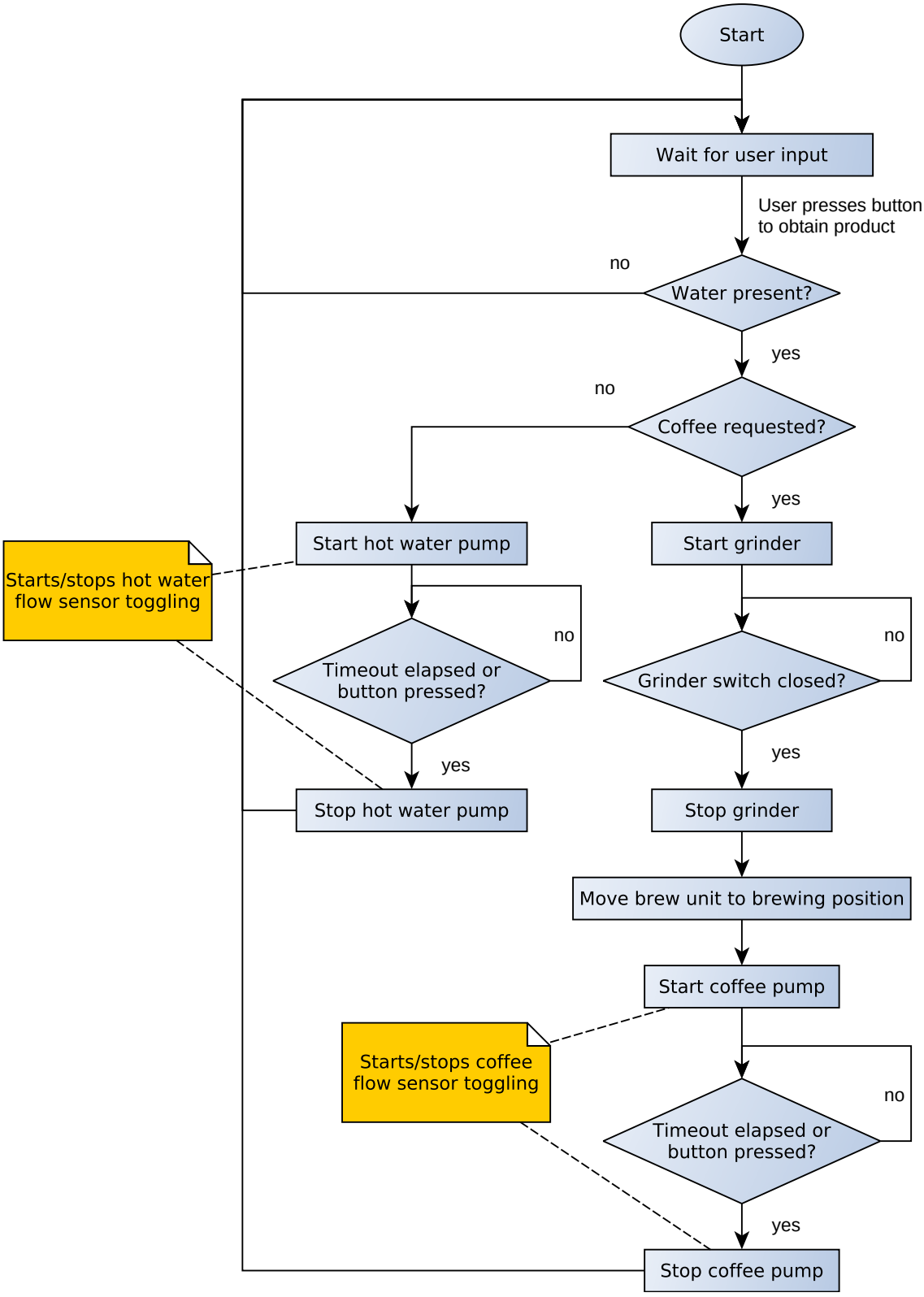

Figure 2.3: Flowchart of a brew process

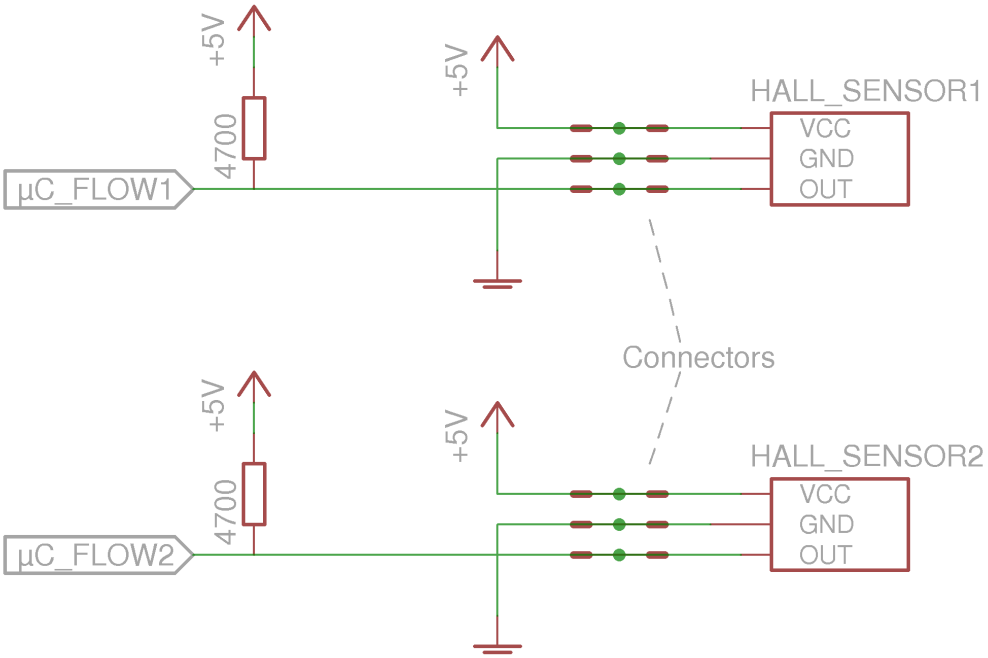

Figure 2.4: Schematic of the flow sensors attached to the microcontroller on the Saeco control board

Water/Grinder switches

These two switches are wired to pull a microcontroller pin to ground when operated. As pictured in figure 2.5, they share one connection by using different series resistors. This needs to be considered when reading the respective values (see 3.2.1).

The grinder switch is operated when the chamber for ground coffee is full. This way, the machine knows when to stop the grinder. The size of the chamber is mechanically adjustable to obtain the desired amount of ground coffee. When the machine transports the ground coffee to the brewing position (about one second later), the switch is released.

For detecting the water level, a reed switch is used. At one side of the water tank, there is a little chamber containing a permanent magnet. It will rise with the water level, but is blocked by the ceiling of the chamber, which is lower than the maximum fill level. At this height, the reed switch is positioned in the machine. It is closed by the magnet when enough water is left. Once the water level drops far enough, the magnet no longer operates the switch so that the machine can detect a low water level. The machine will then disable coffee/water delivery until water is refilled.

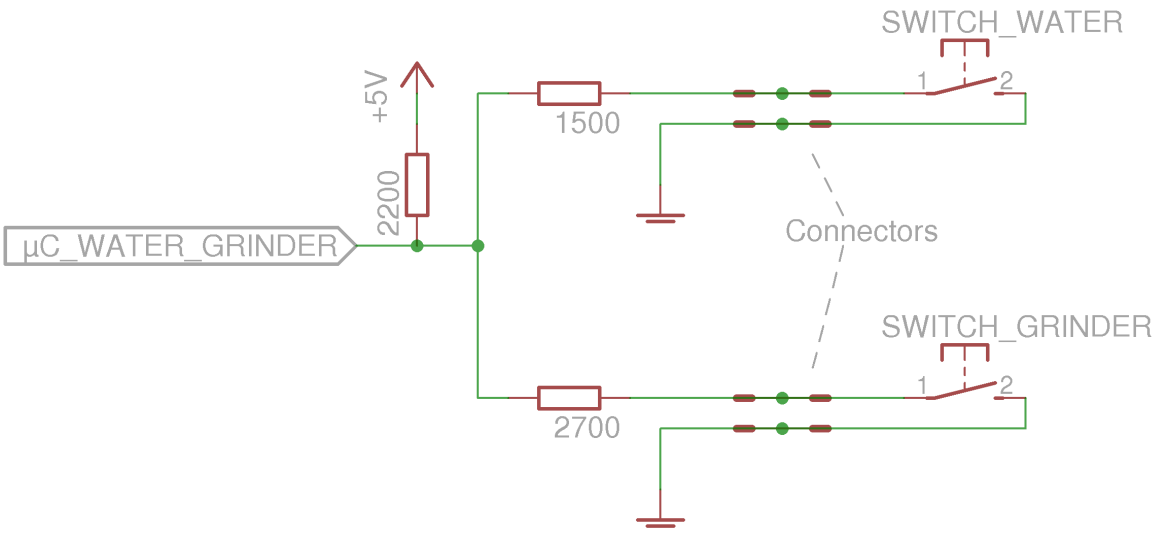

Figure 2.5: Schematic of the water level and end-of-grind sensors attached to the microcontroller on the Saeco control board

2.2 Existing accounting system

In an effort to efficiently distribute the costs between the coffee machine users, a web based accounting system had been developed previously[1]. Every user has to register with this system and deposit money on their prepaid account by cash.

When getting a coffee, the user has to record this transaction by themselves. Three methods of doing this are implemented:

Web interface: Every user is assigned a unique *token* that can be used to buy coffee via a PHP-based web frontend.

Android application: The users can also install a custom application on their personal Android phone which uses a web API to communicate with the accounting system.

Tally sheet: As a "low-tech" variant, there are printed tally sheets with a line for each registered user. A payment can be done by putting down a mark in this line. The sheets are evaluated regularly and the data is added to the electronic system.

Each method differentiates between a black coffee and a coffee with milk added, as the milk supply is also organized with this system.

Additionally, the web interface provides statistics and a log of the coffee consumption. It also displays the current account deposit.

[1] completely independent of this work; by a CES PhD student (Sebastian Kobbe)

Unfortunately, there is a problem with this system: It is very easy for the user to unintentionally or intentionally *not* log a consumed coffee, leading to financial losses and subsequently to higher coffee prices for every user. This constitutes one of the main reasons for setting up this work.

Additional reasons include:

- The laborious counting of the tally sheets can be omitted.
- More accurate statistics are possible.
- Users can be prevented from accumulating a larger negative balance (or from getting a negative balance at all).

The system implemented in this work integrates with the existing “backend”, replacing the existing payment methods.

3 Design

3.1 Overview

The system developed in this work basically consists of three parts:

- the coffee machine itself
- a user interface, including:
  - graphical output of information (e.g. account balance)
  - user input (e.g. milk choice and authentication)
- an interface to the existing accounting system

3.1.1 User Interaction

The basic user interaction with this system is depicted as a flowchart in figure 3.1.

When no user has authenticated yet, the coffee machine is locked, i.e. it is not possible to get any products. If someone wants to get a coffee or hot water, they have to put their RFID token in front of a reader. The system checks with the accounting system if this is a registered user and gets account information which is displayed to the user. Then the machine is unlocked and the user can obtain the desired product by using the coffee machine buttons. The machine's state is tracked to detect the user's choice and the end of the brew process. Information about the user's choice is submitted back to the accounting system so the account can be charged appropriately. Afterwards, the machine is locked again.

Before triggering the brew process, the user has to make their choice regarding milk by using a separate button[1]. This information is needed to charge the account correctly. The choice is saved for each user and used as a default value for their next transaction.

[1]Milk is not dispensed by the coffee machine, but can be obtained from a fridge nearby.

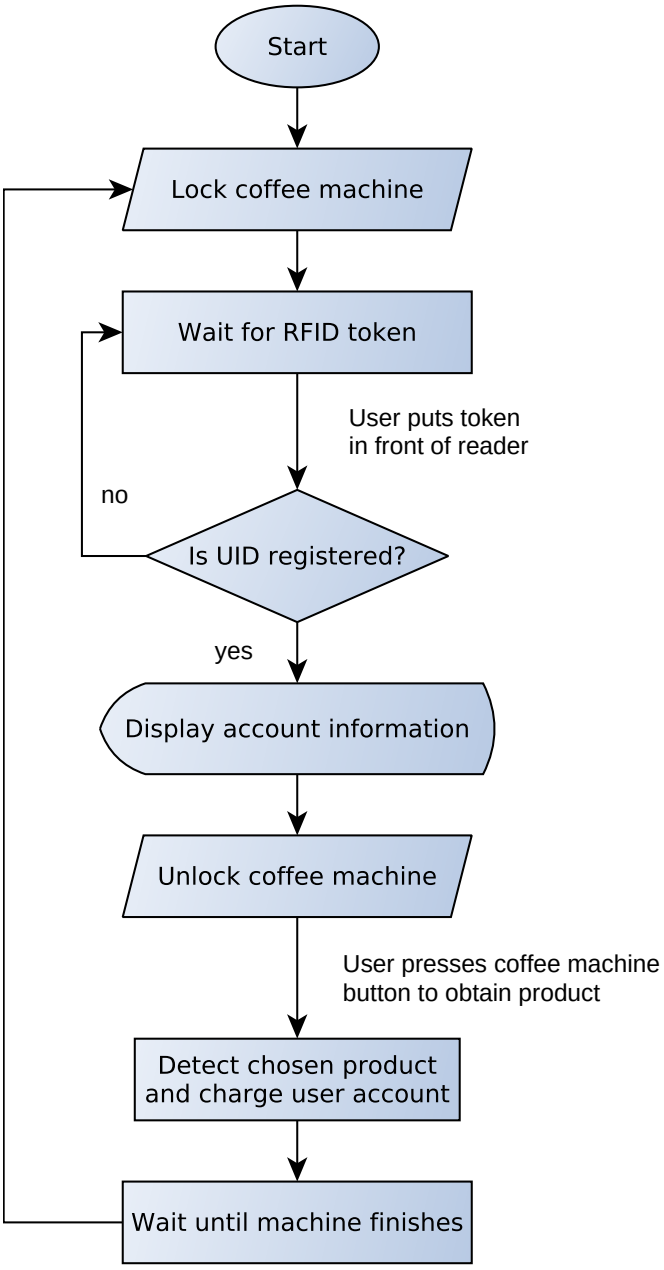

Figure 3.1: Flowchart of the basic user interaction

New users have to visit the accounting administrator to get their token registered. Before that, they should put the token in front of the reader once so that its identification code can be logged. This simplifies the registration process for the administrator.

Maintenance

The coffee machine needs to be cleaned/decalcified regularly. These processes are largely automated. To allow the process to complete without being interrupted by the new control system, a special *maintenance mode* is introduced. It essentially disables any locking and tracking of the coffee machine state. Additionally, a warning notice is displayed on the user interface.

When someone wants to clean or decalcify the machine, they have to enter this mode by using a special RFID token. Then the process can be started as usual by using the coffee machine interface and filling in the cleaner or decalcifying solution. When the process has finished, the maintenance mode can be exited by reading the token again.

3.2 Hardware

As a base processing platform, an Android based tablet computer is used. This has several advantages:

- integrated display with high resolution
- integrated input device (touch sensor)
- integrated networking hardware
- small form factor
- relatively low cost (used device available)

A disadvantage is the lack of hardware interfaces to connect the coffee machine signals described in section 2.1 and the RFID reader for authentication. Only a USB interface is available. This problem can be solved by using additional hardware as described in the next sections.

The concrete tablet model used here is the *Yarvik GoTab Zetta Tab466EUK*. It features an ARM Cortex A8 based SoC running at 1.2 GHz, 1 GB DDR SDRAM and a 9.7 inch display (1024x768) with a capacitive touch sensor. Android versions up to 4.1.2 are available. A modified version of the firmware

(CyanogenMod) is used, making it easy to obtain superuser privileges needed for the application (see section 4.4).

3.2.1 Interface to Coffee Machine

To enable the Android software to communicate with the low level signals found in the coffee machine, a separate microcontroller board is used, namely the popular *Arduino Leonardo* (see figure 3.2). This board is based on the 8-bit *Atmel ATmega32U4* controller. It allows direct connection of the coffee machine signals to the microcontroller pins. On the other hand, it offers USB connectivity for communication with the Android tablet. Software can use a virtual serial port to exchange information over USB.

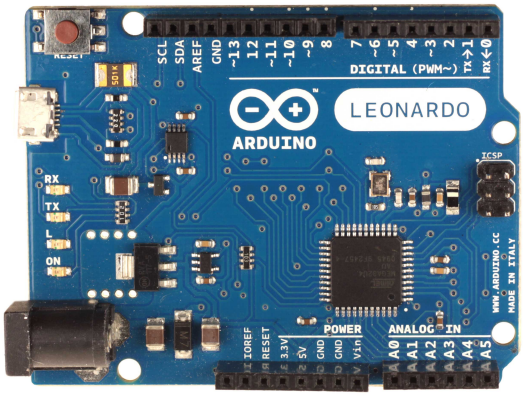

Figure 3.2: Arduino Leonardo with Atmel ATmega32U4 (from http://arduino.cc/en/Main/ArduinoBoardLeonardo)

To actually electrically connect the coffee machine signals to this board, an adapter consisting of two boards connected by a cable is designed. One of them is placed inside the coffee machine. It uses connectors that fit the ones used in the coffee machine, so the machine does not need to be permanently modified. The other board just uses a pin header to connect to the Arduino board.

Locking the Coffee Machine

To prevent users from using the machine without authenticating and paying, it has to be locked. As the buttons cannot easily be intercepted, this has to be achieved in a different way.

Locking can be realized by fooling the machine into assuming there is no water left in the tank. To achieve this, the water switch (described in section 2.1.1) can just be disconnected from the machine's control circuit. The adapter board includes an additional optocoupler whose transistor is inserted into the connection to the switch. When disabled, the machine reacts exactly as if the water level was low.

The disadvantage of this approach is that the coffee machine display shows the message "Fill Water Tank" when the machine is locked. Any other feasible locking methods would either display another wrong message or delay the process significantly (cutting power). Therefore, this drawback is accepted.

When disconnecting the water sensor line to lock the machine, a peculiarity needs to be considered: The coffee machine does not react to a low water level if it occurs *while* the pump is active. Even after the current process is finished, it does not inhibit starting a new one. Instead, it assumes there is enough water left for at least the next product. To work around this problem, i.e. to still lock the machine successfully in any situation, the sensor line has to be pulled to ground (simulate a closed water switch) until the current process is finished.

Detection of Coffee Machine State

After the machine is unlocked, the user has to press a button on the machine, depending if they want to get coffee or hot water. Again, this information cannot easily be extracted directly. To detect the chosen product, the system waits until either the grinder switch indicates that a coffee is being made or the flow sensor for hot water is activated. After that, the system waits until the respective flow sensor is not toggling any more to detect the end of the process. Then the machine is locked again.

By using the grinder switch to detect a coffee being made (instead of the coffee flow sensor), the coffee machine can complete a *rinsing* process before making the actual product. For rinsing, only the coffee flow sensor is activated. As a result, the user is not charged wrongly.

A detailed description of the adapter boards can be found in section 4.2.

3.2.2 RFID Reader

For user identification and authentication, RFID tokens are used. Consequently, the control system has to include an RFID reader. To allow the KIT

student/employee cards to be used, this reader has to support 13.56 MHz tokens.

As the design already includes an Arduino board, a readily available Arduino-compatible extension board (so-called *shield*) can be used for this purpose. For this work, the *Seeed Studio NFC Shield V2.0* (see figure 3.3) was chosen. It uses an external antenna connected by a cable, allowing a more flexible placement in an enclosure.

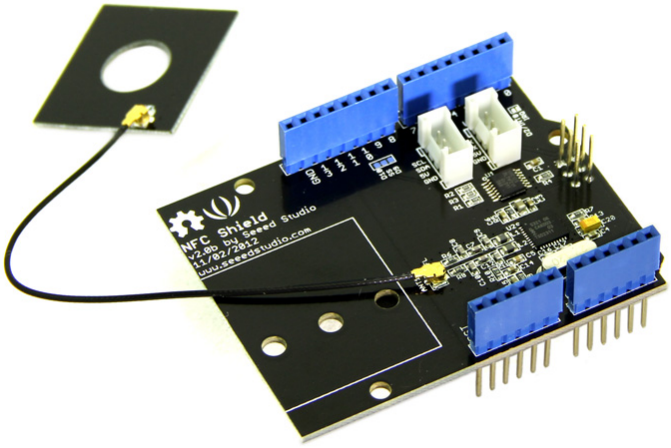

Figure 3.3: Seeed Studio NFC Shield V2.0 with detachable antenna[2]

This shield uses the integrated transceiver module *NXP PN532* for implementing RFID communication. It communicates with the Arduino microcontroller using an SPI interface and can read up to two RFID tokens at a time.

3.2.3 Network Interface

The Android tablet includes a wireless network interface which can in principle be used to communicate with the backend accounting system. Unfortunately, the wireless reception at the designated coffee machine location is very unreliable. For this reason, an additional wired network interface (connected to the tablet by USB) is included in the design. The chosen device is the *Digitus DN-10050-1*.

As the tablet only includes one USB *host* interface, a USB hub is required to connect both the Arduino board and the network interface. Here, a *Digitus DA-70220* is used. This device also includes a separate power supply, relieving the tablet from providing power to the USB devices.

The two additional USB components are pictured in figure 3.4.

(a) Network interface DN-10050-1[3]

(b) USB hub DA-70220[4]

Figure 3.4: Additional USB components

3.3 Software

An overview of the information flow between the components described in the previous section is shown in 3.5. It is realized by the Arduino and Android software and will be detailed in the following sections.

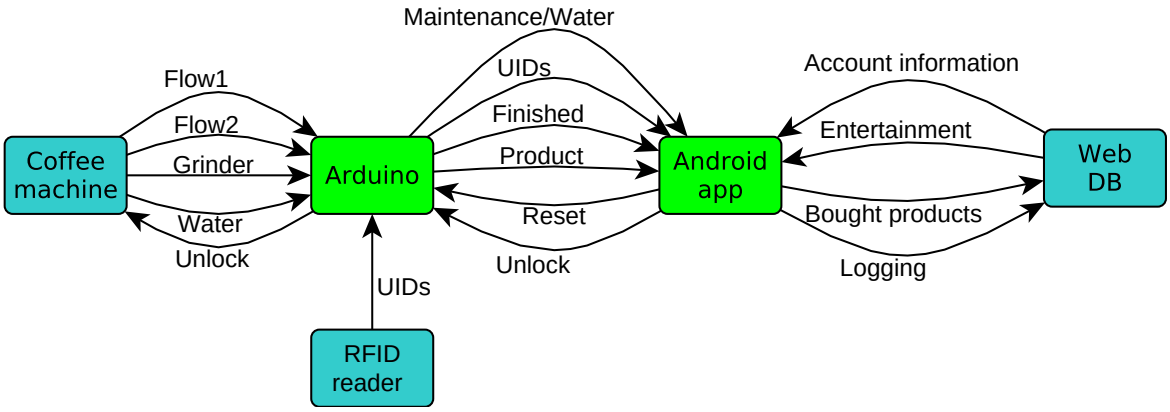

Figure 3.5: Information flows in the envisioned system

3.3.1 Arduino

The Arduino microcontroller is responsible for three tasks:

- handling the RFID reader input

[2]from http://www.seeedstudio.com/depot/nfc-shield-v20-p-1370.html
[3]from http://www.digitus.info/en/products/network/fast-ethernet-network/network-interface-cards/r-10100-mbps-network-usb-adapter-dn-10050-1/
[4]from http://www.digitus.info/en/products/accessories/usb-hubs/usb-20-4-port-hub-da-70220/

- locking/unlocking the coffee machine and tracking its state
- communicating with the Android application

The controller sends the *UID* (Unique Identifier) of any RFID token to the Android application, which decides if the coffee machine should be unlocked and sends the appropriate command back. Then, state tracking as described in section 3.2.1 is done and the Android application is notified of any changes.

There is one special case: The maintenance card UID (hard coded) is handled indepently from the Android application. When it is read, the maintenance mode is entered, i.e. the machine is unlocked without any state tracking. It will only be re-locked when the maintenance card is read again. The Android application will just be notified of these events. This approach has the adavantage that the maintenance mode doubles as an “emergency mode”. In case the Android tablet stops working, the coffee machine can still be unlocked this way.

Additionally, the controller sends information about the water level to the Android application so that a notice can be displayed. This allows the users to distinguish if the machine really needs to be refilled or it is just locked.

Implementation details of the Arduino software are described in section 4.3.

3.3.2 Android

The Android application (Java) has to handle the following tasks:

- providing a graphical user interface
- communicating with the backend accounting system
- communicating with the Arduino software
- maintaining an offline database with user information

The graphical user interface can be subdivided into two parts: the interface for actually operating the access control system, and an area where entertainment content is provided. This entertainment function was added to make coffee breaks even more enjoyable by showing e.g. popular web comics. The tablet can get this content over its network interface and display it using a browser engine.

The actual access control interface instructs the user to start a transaction by putting their RFID token in front of the reader. After receiving a UID from the Arduino software and checking with the backend if the token belongs to a registered user, it sends a message to the Arduino software to unlock the coffee machine. It then displays account information and accepts user input

regarding the milk choice. When it receives a message specifying the chosen product, it charges the user's account in the backend system and waits until the next message (end of process) arrives.

To communicate with the existing accounting system, the network interface is used to perform HTTP requests to the server to obtain user information and charge the user's account. Additionally, user and transaction information has to be cached locally to allow operation in the case of a network outage. The cache is refreshed periodically to pick up new users.

Furthermore, the application displays messages when the machine's water tank has to be refilled or the maintenance mode is activated.

Implementation details of the Android software are described in section 4.4.

3.3.3 Communication Protocol

For communication over the serial connection between the two devices, a simple ASCII-based protocol was designed. Every command consists of a three letter prefix, followed by a colon (0x3A) and the description. It is ended by a linefeed character (0x0A). This way, the communication is human-readable and easy to debug. The protocol overhead compared to a binary approach is not relevant in this application, as the message frequency is very low.

The Android application can send the following commands to the Android software:

`CMD:BEG\n`	Begin a new transaction (reset)
`CMD:UNL\n`	Unlock the coffee machine

For the opposite direction, the following commands can be sent:

`UID:[uid1]<,uid2>\n`	Up to two RFID UIDs that were read
`RES:AOK\n`	Response to commands
`RES:COF\n`	User gets a coffee
`RES:WAT\n`	User gets hot water
`RES:DON\n`	Machine is finished
`STA:MNT\n`	Maintenance mode entered
`STA:WAT\n`	Water tank has to be filled
`STA:NOR\n`	Normal operation resumed

A normal transaction for a registered user getting a coffee would proceed as follows:

1. Arduino sends `UID:ABCD1234`
2. Android checks UID, sends back `CMD:UNL`
3. Arduino unlocks, waits for grinder switch, sends `RES:COF`
4. Arduino waits for flow sensor, sends `RES:DON` and locks

4 Implementation

The whole resulting system is pictured in figure 4.1. To the left of the coffee machine (on top of the microwave oven), the Android tablet can be seen inside its specially designed enclosure, mounted on a VESA display stand. On the right side, there is a smaller enclosure containing the remainder of the components: the Arduino board with attached NFC shield and the connection to the coffee machine, the USB hub, and the USB network interface.

4.1 Mechanical Installation

The tablet enclosure is specially designed and built for this work by a CES employee (Peter Kretzler). Figure 4.2 shows a 3D model of its base. The materials used are PVC for the base and acrylic for the front plate. It is produced on a CNC mill and can be wall-mounted or put on a VESA display stand. The tablet's mechanical buttons are covered by the enclosure to prevent the users from closing the Android application. Moreover, the cables for power and USB are routed through tight openings so they cannot be removed without opening the enclosure.

As an enclosure for the other components, the off-the-shelf *Strapubox Modulgehäuse TYP 518* is used. Figure 4.3 shows it opened and upside down with the USB hub taken out. The Arduino board is located right below the NFC shield (black PCB), and the RFID antenna is taped to the bottom (i.e. top of the enclosure). On top, the Arduino-facing part of the adapter detailed in the next section is visible.

4.2 Adapter Coffee Machine/Arduino

The adapter between the coffee machine control circuit and the Arduino board (as introduced in 3.2.1) makes the low-level electrical signals found in the machine available to the access control system. It consists of two separate PCBs connected by a cable. One of them is placed directly into the machine, and the other one is plugged into the Arduino board.

Figure 4.1: Whole system including coffee machine and access control upgrade

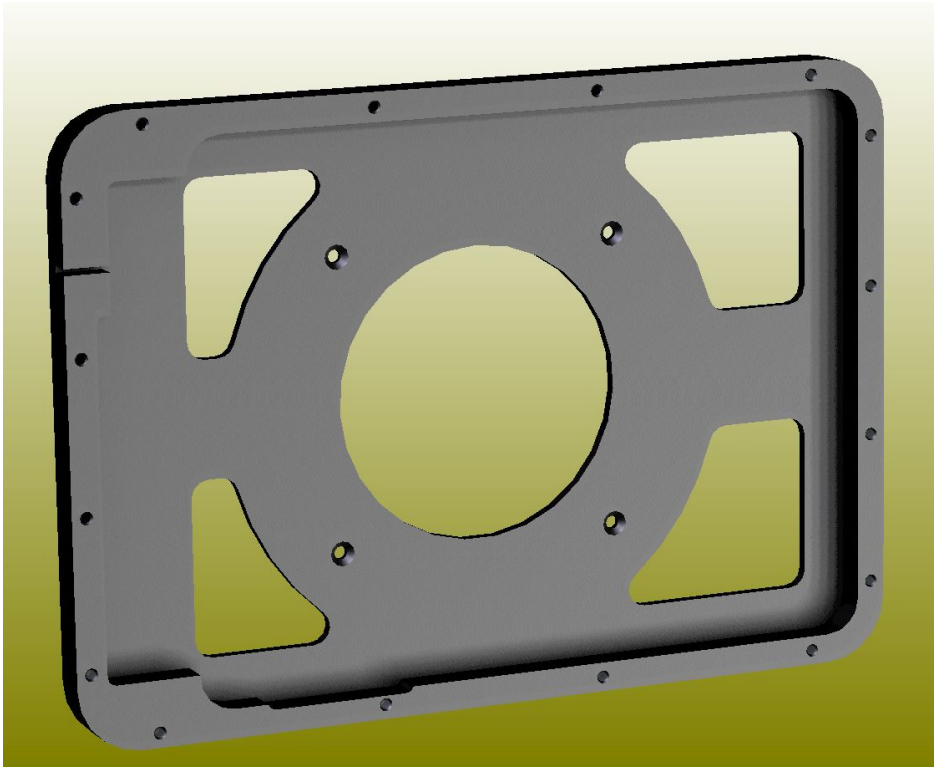

Figure 4.2: 3D model of the tablet enclosure base

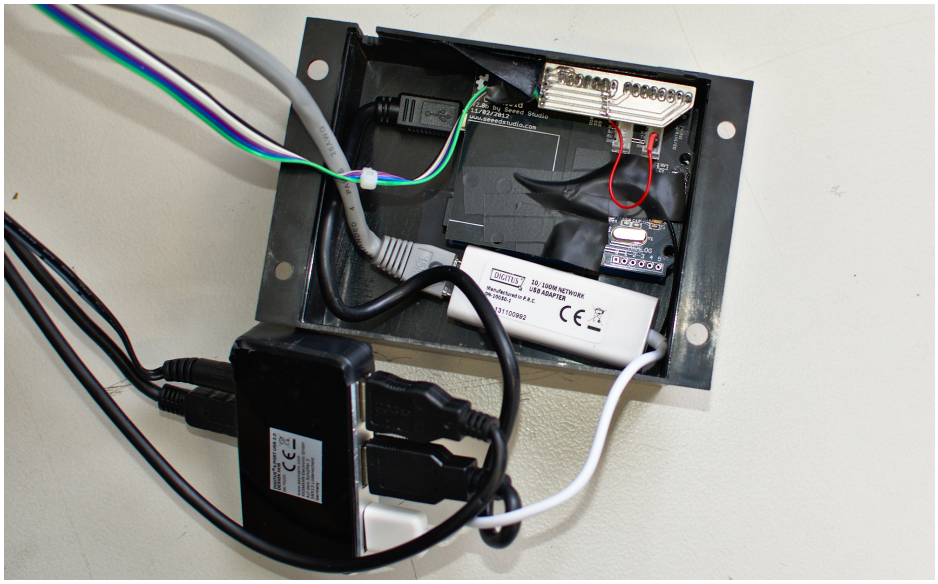

Figure 4.3: Opened RFID reader enclosure with other components: Arduino, NFC shield, adapter board, USB hub, ethernet adapter and cables

Adapter Board Inside the Machine

The machine-facing part uses optocouplers to lead the machine signals out and the unlock signal from the Arduino in. They galvanically isolate the coffee machine from the outside and thereby prevent accidental damage (e.g. by ESD) to the machine. The schematic for this board is pictured in figure 4.4. Outside the dashed line, the relevant parts of the coffee machine circuitry are included.

The optocoupler devices used here are one four-channel *Avago ACPL-247* for the four signals from the coffee machine and one single-channel *Avago ACPL-217* for the unlock signal from the Arduino.

The resistors in series to the optocoupler's inputs limit the current to the optical element (LED). The current needs to be high enough to allow the photo transistor to switch properly, but also low enough to not interfere with the operation of the circuit.

The supply voltage for the optocouplers driven by the coffee machine circuitry is taken from the Vcc line to the flow sensors. Here, an input current of about 1 mA has been chosen, therefore 3.9 kOhm resistors are used. The phototransistor pulls the output line to ground when the LED is switched on. Together with a pull-up resistor on the Arduino side, the states can be distinguished.

For the unlock signal, the Arduino provides the input current. In order to allow the coffee machine circuitry to draw a higher current through this

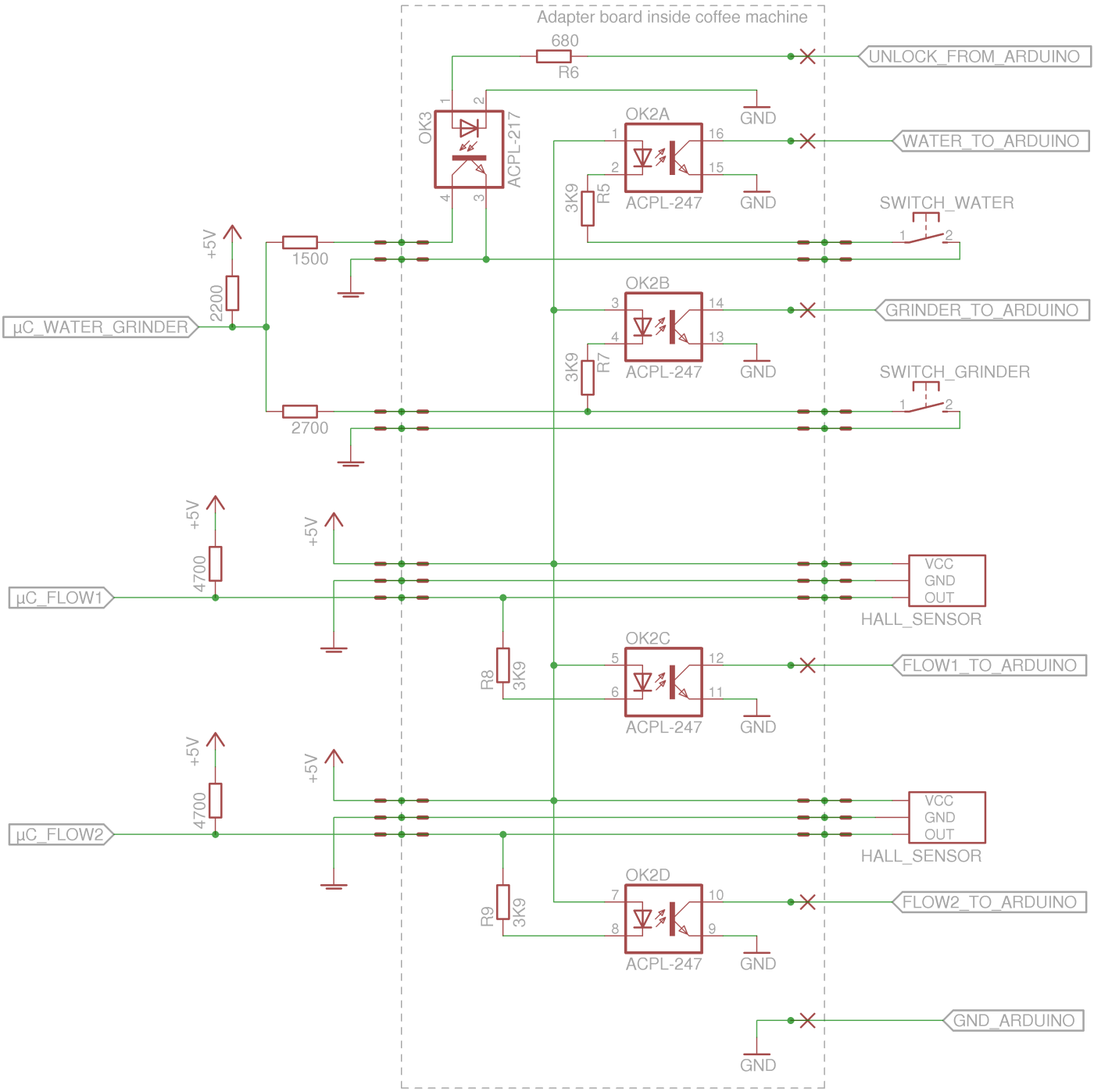

Figure 4.4: Schematic of the machine-facing part of the adapter (inside dashed line)

phototransistor, a higher input current of about 7 mA has been chosen. Therefore, a 680 kOhm resistor is used here.

As explained in section 3.2.1, the Arduino unlock signal does not only need to be able to disconnect the water sensor line (simulating an opened switch), but also simulate a closed switch. For this reason, the optocoupler's photo transistor is connected directly between the sensor line and ground instead of including the water switch. The machine's control board is thereby effectively disconnected from the water switch. Consequently, the Arduino software has to "relay" the actual water level information when needed so the machine still knows when there is no water left.

A difficulty arises for reliably detecting the water and grinder signals on the Arduino: As shown in the schematic, they are connected to the same line on the Saeco controller board, just using different series resistors. In a state where the water sensor line is pulled to ground (unlocked/water present), a schematic as pictured in figure 4.5 results. Now, when the grinder switch is open, an erroneous forward current of about 0.26 mA flows through the LED. Therefore, the photo transistor already partially conducts, allowing a small collector current. On the Arduino, this state still needs to be reliably distinguished from the grinder switch also being closed, resulting in a forward current of 1 mA as designed (and therefore a higher possible collector current). This can easily be achieved by using strong enough pull-up resistors on the Arduino inputs. The pull-ups included in the ATmega controller are too weak (up to 50 kOhm), therefore additional 10 kOhm resistors are included on the Arduino adapter board (see next section).

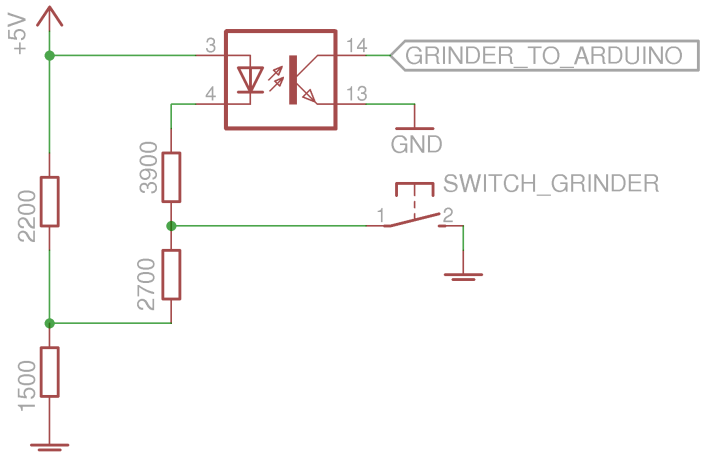

Figure 4.5: Schematic of problematic section (while water switch closed)

The completed PCB can be seen in figure 4.6. The sensors are connected on top of the board, and the cables on the left are plugged into the coffee

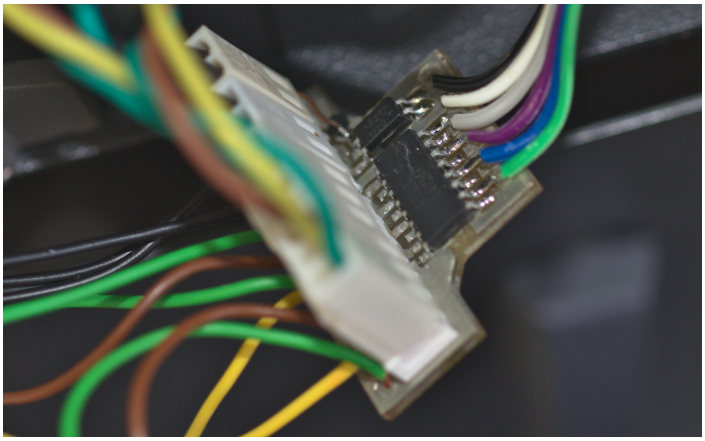

Figure 4.6: Adapter connected to the coffee machine

machine control board.

Adapter Board on the Arduino

The Arduino-facing part just uses pin headers to connect to the Arduino board (actually the NFC shield, which is placed on top of the Arduino board and extends its connectors). Additionally, pull-up resistors (10 kOhm) have been added to be used instead of the weaker built-in pull-ups of the ATmega microcontroller. This allows reliable operation with the optocoupler's photo transistors as explained above. Its schematic is pictured in figure 4.7.

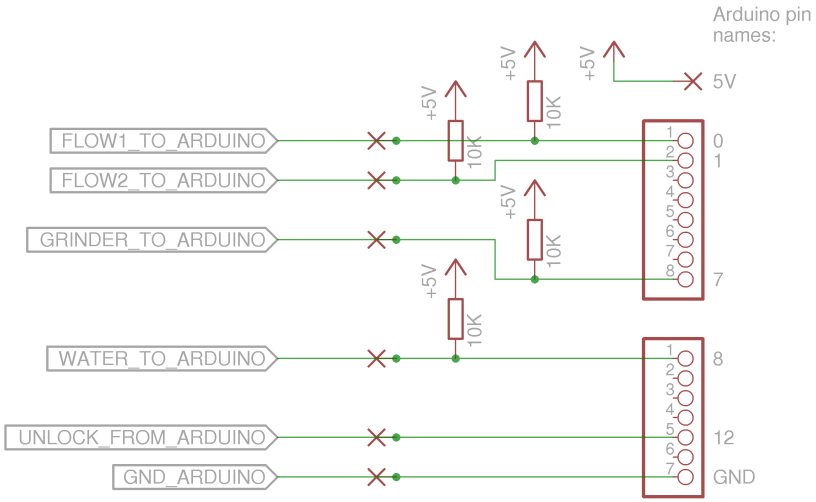

Figure 4.7: Schematic of the Arduino-facing part of the adapter

4.3 Arduino Software

The software running on the Arduino microcontroller is written in C++ using the Arduino libraries. It is coded as a state machine as depicted in figure 4.8 (Mealy model).

Due to the design of the adapter to the coffee machine, the signals from the coffee machine use negative logic (i.e. when the water/grinder switch is closed, the corresponding pin is low). In contrast, the unlock pin uses positive logic.

For communication with the PN532 RFID chip over SPI, the software uses code adapted from the library provided by Seeed Studio[1]. It is stripped down to remove unneeded code and modified to use interrupts instead of polling and accept more than one token at the same time (the maximum number supported by the PN532 is two).

The flow sensor inputs also use the interrupt capabilitites of the controller by incrementing a counter on every signal edge. To detect the start of toggling, the counter has to be above a certain threshold. Stopping is detected when it stays below a threshold during a certain time span. These numbers are determined experimentally.

As described in section 3.2.1 and 4.2, it is necessary to simulate a closed water switch (implying enough water is present) until after the brew process is finished to actually lock the coffee machine. Additionally, the software periodically does this every second for a duration of 100 ms in the `GETUID` state to ensure reliable locking.

Since the detection of the user requesting a coffee works by intercepting the end-of-grind signal, it is possible that this signal is triggered in abnormal states, i.e. `GETUID` or `WATERALERT`. In both states, the water sensor line is disconnected, so the machine will not start grinding even if the user tries. But if the user tries to "cheat" by first authenticating (machine unlocked), then pressing the coffee button (machine starts grinding), and then canceling the transaction (Arduino receives `CMD:BEG`), the machine will continue grinding and also making the coffee even though its water sensor line is open. This behavior could be remediated by not allowing the user to cancel the transaction after authenticating, but this solution would not be user-friendly. It is also impossible to reliably solve this by waiting for a specific time after the user cancels, since the grinding process may be prolonged when the bean container becomes empty. The chosen solution is to handle these cases indifferently on the Arduino by switching to the regular `COFFEEGROUND`

[1] see https://github.com/Seeed-Studio/PN532

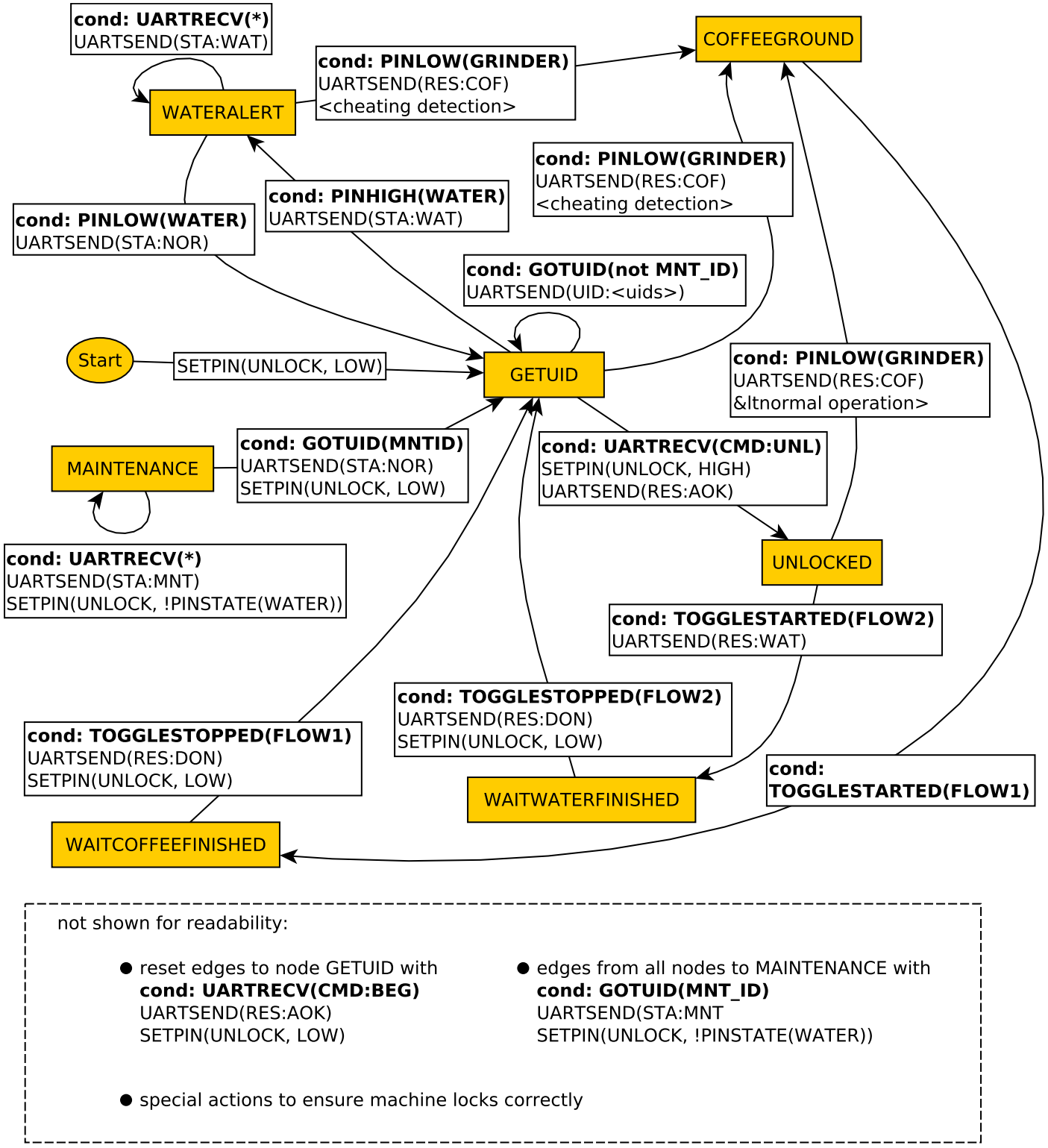

Figure 4.8: Mealy machine representing the software running on the Arduino

state and thus later on sending `CMD:COF` followed by `CMD:DON`. The Android application detects this and handles it as described in section 4.4.6.

4.4 Android Application

The Android application is written in Java using the Android framework. It is used as a so-called Home application which is automatically started after booting the tablet. When it is started, it immediately enables full screen mode. This removes the title bar, but unfortunately leaves the control buttons (Back/Home/Menu) available so the user could still exit the application. Android does not provide any API to remove these buttons. The application deals with this problem by stopping the corresponding process. For this procedure, it needs superuser permissions.

4.4.1 Class Overview

Figure 4.9 gives an overview of the classes used to implement the application.

Android applications need to implement an *Activity* by subclassing `android.app.Activity` to provide callbacks for handling the application's lifecycle. An Activity represents one screen of the user interface. For this application, a single Activity called `MainActivity` is implemented.

The following sections describe the individual classes.

4.4.2 User Interface

To implement the individual parts of the user interface, so-called *Fragments* are used. A Fragment represents a modular portion of the application's interface. For handling the actual access control functionality, there are three different Fragments: `StartDialogFragment`, `BuyWaitDialogFragment`, and `FinishDialogFragment`. Only one of them is displayed at the same time using a `FrameLayout` inside the activity layout. For switching between the Fragments, `FragmentTransaction.replace()` is used.

In the initial state (`StartDialogFragment`), a message prompting the user to put their card in front of the reader is displayed. After a successful authentication, this fragment is replaced by `BuyWaitDialogFragment`, displaying account information (name and balance) and a button for the user to choose whether they want to add milk. The coffee machine is now unlocked.

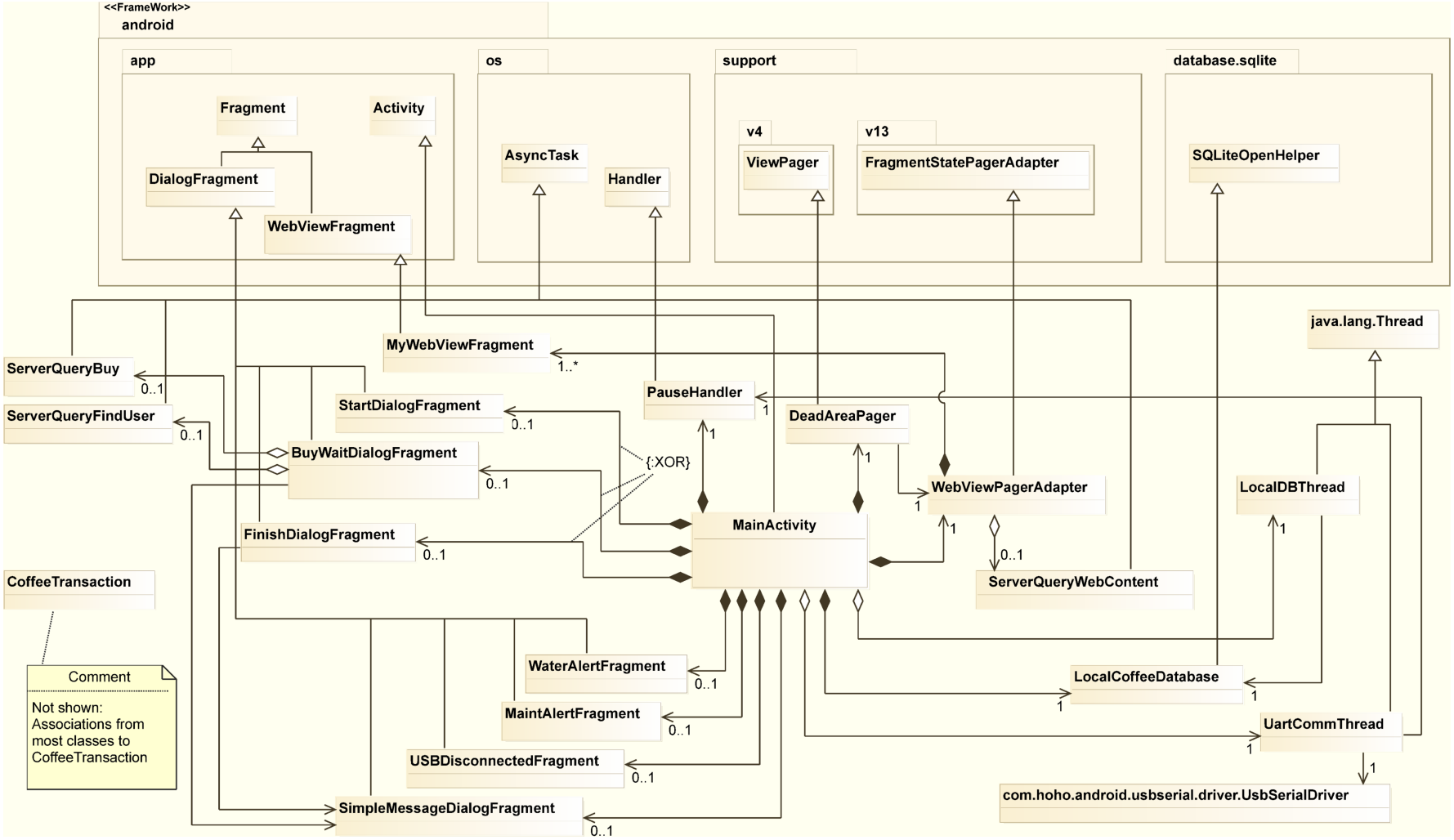

Figure 4.9: Class diagram of the Android application

Furthermore, the user can still cancel the transaction using the corresponding button. When the application finally receives `RES:COF` or `RES:WAT` from the Arduino (see section 4.4.3), `FinishDialogFragment` is displayed. It shows the new balance after charging the user's account. When `RES:DON` is received, it waits until a 10-second timeout has elapsed and then switches to the first fragment again. If a new user authenticates during this time, the second fragment is displayed immediately. Additionally, the user can press the button "Report error" during this time if anything went wrong during the transaction, i.e. the account is charged incorrectly. This will cause a message to be sent to the accounting system, which then logs the event. The user then has to visit the administrator who can review the logs.

Furthermore, there are Fragments for displaying various messages: `WaterAlertFragment` and `MaintAlertFragment` show a full-screen graphical alert, stating that either the water tank is empty or the Arduino is in maintenance mode. `USBDisconnectedFragment` displays a text alert when the Arduino is disconnected. These three messages cannot be dismissed directly, but only by eliminating the original cause. Apart from that, `SimpleMessageDialogFragment` is used to display simple text messages to the user. These include a message if an RFID token is unregistered, the user pressed the "Report error"-button, or if cheating is detected (see section 4.4.6).

The remaining Fragment, `MyWebViewFragment`, is used to display the HTML entertainment content to the user. It contains a `WebView` which can load a URL. For each content page (see section 4.4.7), the `WebViewPagerAdapter` instantiates one `MyWebViewFragment`. These are then displayed using the `DeadAreaPager` arranged below the `FrameLayout` described in the beginning of this section. The `DeadAreaPager` is used to provide a swipe interface for switching between the pages. It is a `ViewPager` slightly modified to ignore touch inputs in a certain area on the tablet, because the touch screen on the tablet used for this work is defective and generated erroneous touch events.

Figure 4.10 shows an example screenshot of the application after a user has successfully authenticated.

Screen Brightness

To ensure that the display and background illumination is always switched on, `android.os.PowerManager.WakeLock` is used. When the tablet is not used, only a `PowerManager.SCREEN_DIM_WAKE_LOCK` is held so the tablet may still dim the light to save energy and increase the life expectancy. During a transaction initiated by reading an RFID token, the application

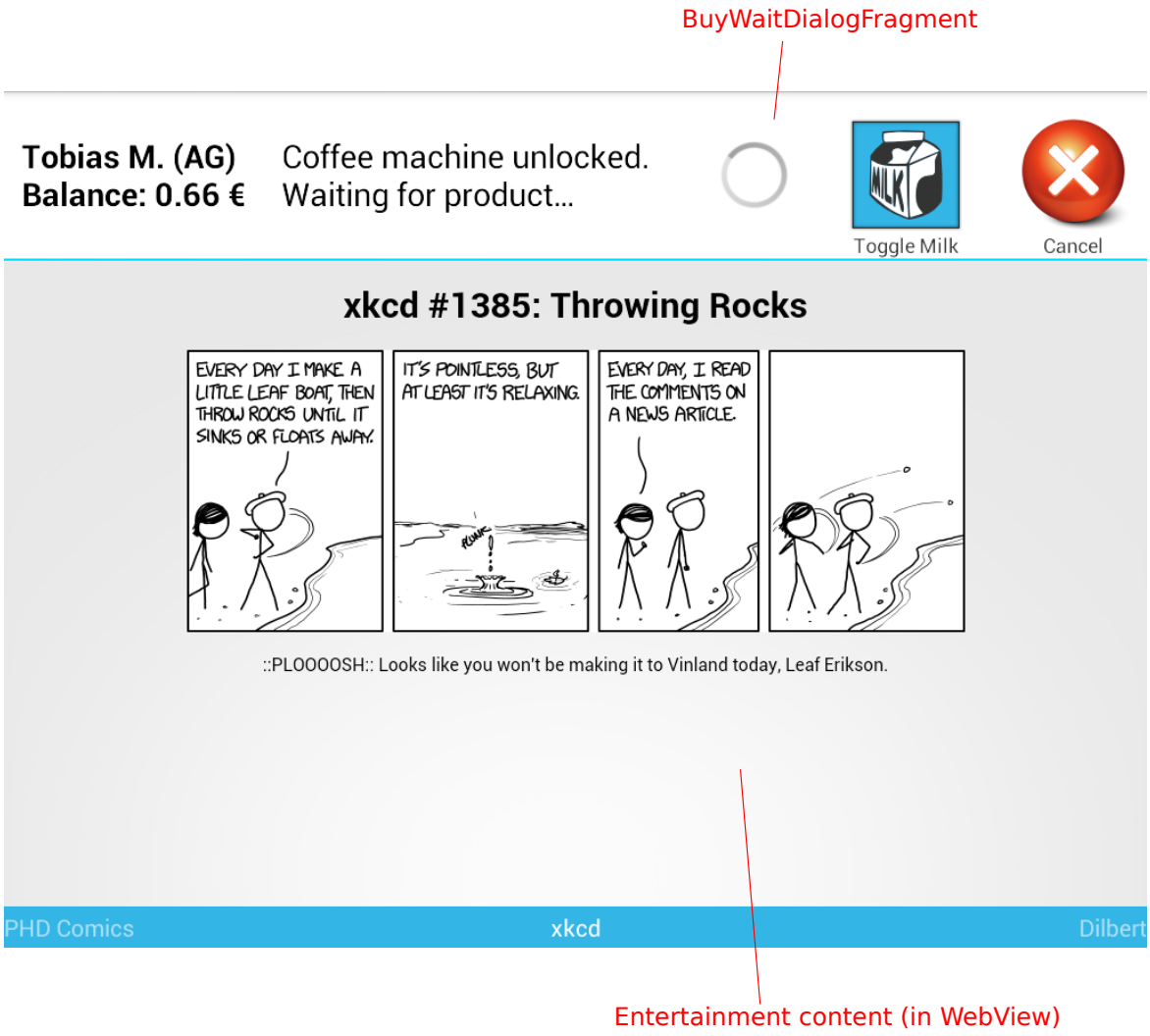

Figure 4.10: Screenshot of the Android application

additionally obtains `PowerManager.FULL_WAKE_LOCK` so the tablet does not dim down for the entire time span. At night-time (23:00 – 07:00), the application gives up the `PowerManager.SCREEN_DIM_WAKE_LOCK` and only holds a `PowerManager.PARTIAL_WAKE_LOCK`, allowing the tablet to disable the display and backlight completely but still keeping the CPU running so the application can react to RFID tokens being read. These time based changes are scheduled using a `android.os.Handler` and its `sendEmptyMessageDelayed()` function.

4.4.3 Communication with the Arduino Software

The application "speaks" to the Arduino using a virtual serial port over USB and the protocol described in section 3.3.3. The tablet acts as a USB host for this purpose. Android applications can access USB devices with their interfaces and endpoints using `android.hardware.usb.UsbManager`. For communication with the Arduino USB UART module, an open source third-party Java library is used (see https://github.com/mik3y/usb-serial-for-android). The application includes a separate background thread (`UartCommThread`) to operate this communication interface.

When transmitting messages, the main (UI) thread directly writes the data into a buffer which will then be handled by the communication thread.

Data reception is complicated by the fact that the application can be completely in the background, i.e. `onStop()` is called on the Activity, but the communication thread is still running and accepting messages that need to be handled. This situation arises when the screen is disabled at night-time or the Android framework decides to put the application to background for some other unspecified reason. In this case, the communication thread has to buffer the messages, wake up the Activity, and then deliver the messages when it is completely resumed. The problem is solved by introducing a subclass `PauseHandler` of `android.os.Handler` as inspired by http://stackoverflow.com/questions/8040280. The Activity instantiates it and pauses/resumes it when going to background or resuming. When the communication thread receives data, it creates a `android.os.Message` object and sends it to this Handler. Depending on its state (paused/resumed), this Handler either processes the message directly or pushes it to a stack and tries to wake up the Activity. When it has resumed, the messages are processed correctly.

4.4.4 Communication with the Accounting System

The existing accounting system offers a simple HTTP based interface using GET requests, implemented using PHP scripts. The application creates the corresponding requests when needed, sends them to the remote server over the network interface, and processes the response. This process is encapsulated into several `ServerQuery…` classes. These are subclasses of `android.os.AsyncTask`, creating short-running "one-shot" background threads not blocking the execution of the main UI thread. Each of these classes uses the static function `HTTPServerQuery.queryHTTP` which in turn uses `java.net.HttpURLConnection` to make the HTTP request and returns the response as a `java.util.List<String>`.

`BuyWaitDialogFragment` first uses `ServerQueryFindUser` to get user information for a given RFID UID. This launches a request to http://i80misc01.itec.kit.edu/coffee/getuser.php?rfid=<uid>, parses the response and creates an instance of `CoffeeTransaction` containing the user's name, account balance and last milk choice. If the server cannot be reached, the local offline database (see section 4.4.5) is queried.

When the machine reports the product chosen by the user, the `CoffeeTransaction` instance is updated with this information and used as a parameter for `ServerQueryBuy` which creates a request to http://i80misc01.itec.kit.edu/coffee/buy.php?rfid=<uid>&black=<bool>&water=<bool>. The two extra parameters specify whether the user just gets a hot water and whether they want milk or not. The response includes the updated account balance.

Another query class (`ServerQueryWebContent`) is used to get updated entertainment content. This process is described in section 4.4.7.

Network Setup

While the Linux kernel used by the Android system fully supports USB network interfaces, the Android user space does not provide any means to permanently configure user-added interfaces. Only the built-in wireless interface is supported.

For this reason, the application itself configures the USB interface by directly executing the (native, non-Java) `netcfg` utility to activate DHCP configuration and correctly apply DNS settings. For these operations, it needs superuser privileges.

4.4.5 Offline Mode

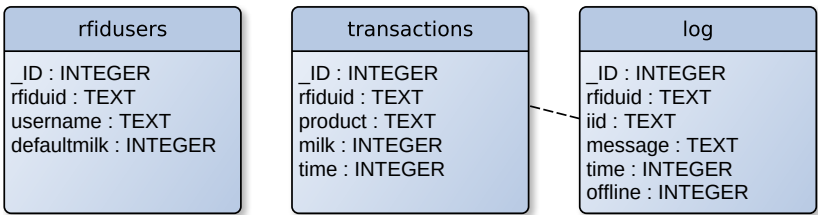

Figure 4.11: Tables used for saving offline information

To allow users to get coffee even when the accounting system cannot be reached, information about registered users is stored locally in a SQLite database. It is described and handled in `LocalCoffeeDatabase`. Figure 4.11 gives an overview over the used tables.

The table `rfidusers` is periodically synchronized with the remote database by `LocalDBThread`. It contains the user's UIDs, names, and milk choices. For updating, the script `getusers.php` is requested.

When a user gets a coffee in offline mode, the transaction is recorded in a second table called `transactions`. It is described by the user's UID, the bought product (coffee/water), whether milk was included and the time of the transaction. The database thread tries to apply the saved transactions periodically, using the same mechanism as `ServerQueryBuy`.

A third table called `log` is used to save log entries from `ServerQueryLog` that could not be submitted. For every log entry, the corresponding message and time is saved together with the user's UID (if relevant). The `iid` field may contain a reference to a transaction. If the `offline` field contains a `0`, `iid` describes a successful "online" transaction, specified by an ID returned from `ServerQueryBuy`. When the log entry can finally be sent to the server, this ID can directly be used. Otherwise (`offline=1`), `iid` contains an `_ID` referring to an entry in the transaction table. When this transaction is applied, the returned "online" ID is used to submit the log entry.

4.4.6 Cheating Detection

When the Arduino sends the command `RES:COF` or `RES:WAT` although no transaction is currently in progress (see section 4.3), cheating is detected. This can happen when the user presses the coffee button on the machine after it is correctly unlocked, but then cancels the transaction on the tablet before the grinder is finished.

To handle this case, the application always saves the last `CoffeeTransaction` containing information about the last user. When cheating is detected, a warning message is displayed and if the saved information is less than ten minutes old, the last user's account is charged.

Additionally, the incident is reported to the accounting system using `ServerQueryLog`, which creates an appropriate log entry using `errorlog.php`.

In case of an erroneous cheating accusation, the user can authenticate again, cancel the transaction, and use the "Report error"-button to report this.

4.4.7 Entertainment Content

As described in section 4.4.2, `DeadAreaPager` is used to display multiple `WebViews` showing various HTML pages containing e.g. comic strips for entertainment purposes. Exactly one page is visible at a time, selectable by the user by employing a swipe gesture.

The `WebViewPagerAdapter` responsible for instantiating the desired `MyWebViewFragments` includes an `updateContents()` function which uses `ServerQueryWebContent` to update the contents. It is called every time the local database is being synchronized. The content itself is located on the same web server that is used for the accounting system. This way, the displayed content can be changed without having to update the Android application itself.

`ServerQueryWebContent` sends an HTTP request to http://i80misc01.itec.kit.edu/coffee_entertain/linklist, which may contain one hyperlink per line, accompanied by a title. The first line has to read "linklist:". An example of this file looks as follows:

```
linklist:
PHD Comics; http://i80misc01.itec.kit.edu/coffee_entertain/phdcomics
xkcd; http://i80misc01.itec.kit.edu/coffee_entertain/xkcd
Dilbert; http://i80misc01.itec.kit.edu/coffee_entertain/dilbert
```

For every link, a `MyWebViewFragment` is instantiated, which uses the link as a parameter for `WebView.loadUrl()` to load the content.

Updating the Contents on the Remote Server

With the setup described above, the remote server (i80misc01) is responsible for updating the pages specified in the linklist file. For this purpose, a python

Component	Price
Arduino	21.42 €
Tablet	used; about 100.00 €
NFC shield	28.09 €
USB ethernet adapter	9.95 €
USB hub	11.95 €
Tablet enclosure (material)	about 15.00 €
Arduino enclosure	4.60 €
Adapter components	about 6.00 €
Total	197.01 €

Table 4.1: Breakdown of hardware cost (including taxes)

script is periodically executed as a cron job. This script individually checks if new content is available for the configured sources, downloads the content, and generates updated HTML files that are served to the Android application. It can easily be extended to allow the addition of new content.

4.5 Costs

The costs of this upgrade can be divided in recurring and initial non-recurring expenses. The non-recurring expenses are constituted by the acquisition cost of the hardware components, whereas the power consumption causes the recurring expenses.

The power consumption was measured over a time span of about 14 days using an off-the-shelf monitoring device inserted into the 230 V power line. The average consumption is 5.66 W. Considering the average electricity price of 28.84 ct/kWh (including taxes; for German private consumers in 2013 according to [BDE13]), this results in electricity costs of about 14.30 € per year.

The hardware costs are broken down in table 4.1. They accumulate to 197.01 €.

5 Conclusion

The coffee machine access control system designed in this work has been completely implemented and introduced to daily use at CES. Its use is mandatory, i.e. there is no other way to use the coffee machine without properly authenticating using a registered RFID token. It is working well so far and reduces coffee shrinkage to basically zero as no one can "forget" to record their coffee consumption.

The human interface is simple and easy to use, and the authentication/unlock process does not add a significant delay before the user can get coffee. Some employees already increased the comfort by putting sticky RFID tokens to the bottom of their cups, registering them to their account and using them instead of their KIT card.

The most failure-prone component of the upgraded system is the Android tablet. It has already caused issues during the implementation: The touch screen is partially defective, and the tablet needs to be reset regularly due to complete lock-ups. When it ceases to function, another tablet needs to be used to run the application. As this will most likely be a different model using a more recent Android version, software changes might be necessary. Additionally, the custom-made enclosure may need to be modified.

An interesting extension that could be implemented in the future is a closer control of the coffee machine. By intercepting/synthesizing more signals, especially the button signals, the machine could be completely operated from the outside. This way, users could get coffee automatically after authenticating without using the coffee machine's user interface at all. Furthermore, cheating could be more actively prevented instead of only detected.

Bibliography

[All93] Allegro MicroSystems, Inc. *3113, 3120, 3130, and 3140. Hall-Effect Switches*. 1993. URL: http://www.allegromicro.com/~/media/Files/Datasheets/UGN3113-20-30-40-Datasheet.ashx (visited on 05/27/2014).

[Ard12] Arduino SA. *Arduino Leonardo*. 2012. URL: http://arduino.cc/en/Main/ArduinoBoardLeonardo (visited on 06/01/2014).

[Ava10] Avago Technologies. *ACPL-227 / ACPL-247. DC Input Multi-Channel Half-Pitch Phototransistor Optocoupler*. July 20, 2010. URL: http://www.avagotech.com/docs/AV02-0752EN (visited on 05/27/2014).

[Ava11] Avago Technologies. *ACPL-217. DC Input , Half-Pitch Phototransistor Optocoupler*. Aug. 3, 2011. URL: http://www.avagotech.com/docs/AV02-0470EN (visited on 05/27/2014).

[BDE13] BDEW Bundesverband der Energie- und Wasserwirtschaft e.V. *BDEW-Strompreisanalyse November 2013*. German. Berlin, Nov. 20, 2013.

[Fri02] Andreas Fries. *AndyFAQ – FAQ zu Kaffeevollautomaten der Marke Saeco. Durchflussmesser / Impulsgeber*. German. 2002. URL: http://www.andyfaq.de/e_impulsgeber.html (visited on 05/27/2014).

[Fri07] Andreas Fries. *AndyFAQ – FAQ zu Kaffeevollautomaten der Marke Saeco. Anschlussliste Royal*. German. 2007. URL: http://www.andyfaq.de/e_anschlussliste_royal.html (visited on 05/27/2014).

[Mic10] Micronas GmbH. *HAL 5xy. Hall-Effect Sensor Family*. Apr. 15, 2010. URL: http://www.micronas.com/sites/default/files/downloads/files/HAL501___507,_508,_509,_HAL516___519,_523_Hall-Effect_Sensor_Family.pdf (visited on 05/27/2014).