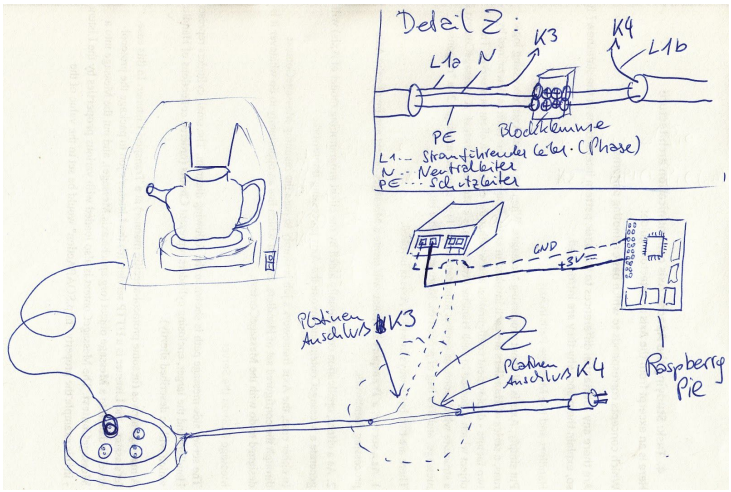
Implementations for Milestone 3

Concerning the objectives for milestone 3, only persistence was left for us to implement, as we already implemented message-passing in milestone 1. In our special scenario of a distributed IoT system with a non-fixed number of dynamic endpoints functioning offering different services, ActiveMQ makes little sense.
For persistence, we decided to use filestream to account for the limited RAM and memory resources of our Raspberry Pis. At first, we did look into storing all application data into the cloud (Google Cloud) using Objectify, but were struck down by Google's heavy paywall restrictions.
Currently, only the alarmClock class makes use of the persistence class; it makes no sense for the CoffeeMachine, as it only reacts situationally and has no data to store. For the final hand-in, we intend to make the list of subscribers persistent for all applications .

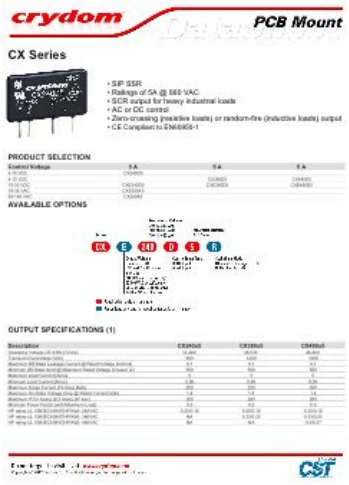
We also added a new application for our IoT network: CoffeeMachineService. It is a subscriber of the AlarmClock service and is currently programmed to serve coffee as soon as any alarm is sounded. One instance of our software can of course only run one application at a time, so it is necessary to determine before deployment which application is desired. At the moment, this can be modified via the web.xml. Thanks to our clean SOLID setup from Milestone 1, the same war file can be used for all applications. We are currently looking into a way to set the application from outside the program - preferably via tomcat, but this is not a top priority by now.

Coffee Machine Setup

The original idea was to replace the mechanical on/off switch of an old school filter coffee machine with a relay, which then can be controlled using raspberry pi's GPIO ports. But none of the team members had one of these old coffee devices. Instead we found someone to borrow such a machine, but we were not allowed making any modifications to the hardware.
That's why we decided for a workaround: Instead of modifying the maybe valuable coffee machine, we made a modification to an old power distribution cable and place the relay somewhere on the conductive conductor to control this device, where we connect the coffee machine (see figure below).

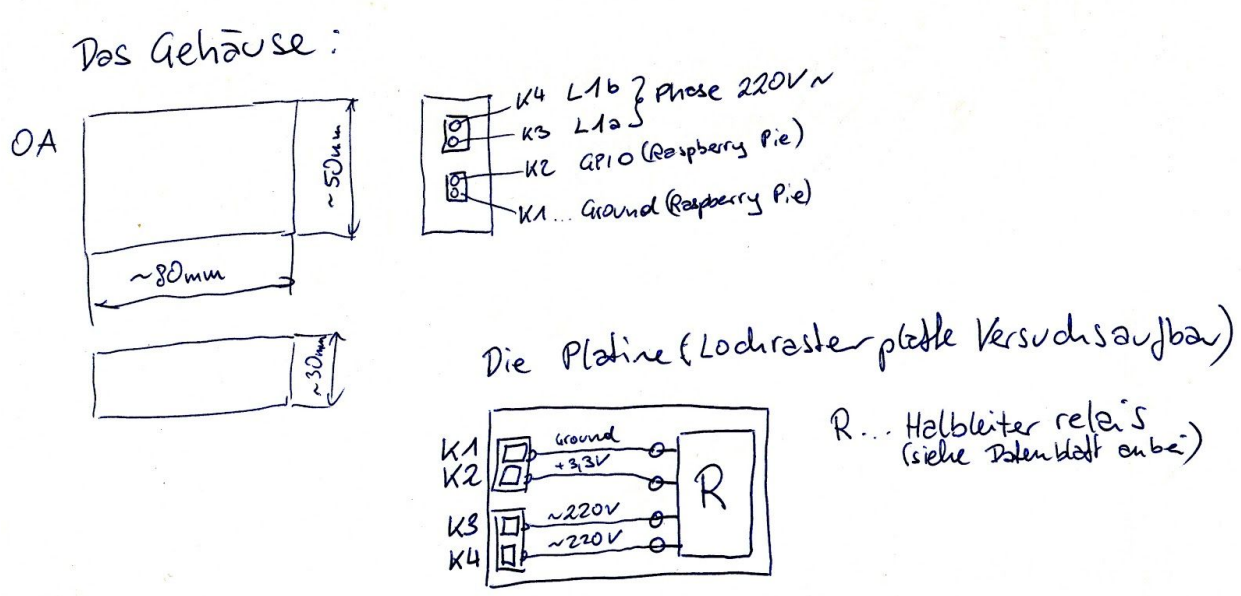


The relay we use to control the current supply (see also data sheet link below) can be used with a control voltage of 3-15V and on the output 240V AC with 5A maximum.



http://www.farnell.com/datasheets/2007466.pdf?_ga=1.121768904.2143082897.1483807560

These specifications made the Crydom CX240D5 semiconductor relay the perfect candidate for our project, since raspberry pi's GPIO voltage level is relative low with 3,3V. On the output side the coffee machine is operating with 220V and the current consumption will be definitely less than 5A. The price of this semiconductor relay is relatively high compared to a mechanical relay, but has the advantage to lower the circuit complexity since it can be used without amplifying the voltage to a level above 5V where cheap mechanic relay are starting to operate (as far we found out).
The test setup was done on a strip board, which we build into a housing, to not put our lives on risk (see figure below for housing and circuit details). Of course this apparatus is now way too big to be build into a coffee machine, but without the housing we could fit the circuit in almost any device.



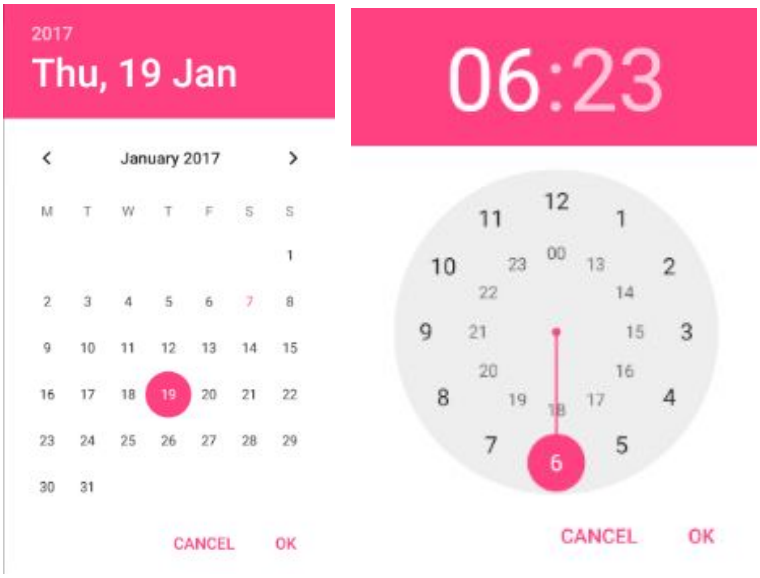
The cabling to operate our solution can be seen in the figures above:

- K1 must be connected to a GPIO ground port (0V).
- K2 must be connected to a GPIO port providing +3,3V.
- K3 & K4 are connected to the “hot” side of the power supply. Be sure power is not attached while connecting those wires.

New Android ACS Client

New in this release:

- For comfortable input of date and time and of course to avoid any confusion of data input format, we provide for user's convenience those awesome pink date and time pickers.



- In the previous version, the web server's IP address was hardcoded in the GUI. Nowadays, the app is listening to the subscription broadcasts arriving from an alarm device in order to detect its IP address. In case of any problem with IP detection, the hardcoded URL is still available as a fallback solution :)