TH06 Team 12

Technisch Verslag

Datum

19 januari 2016

Auteurs

Christiaan van den Berg	1660475
Aydin Biber	1666849
Martijn van Dijk	1660713
Chiel Douwes	1666311

Docenten

Wouter van Ooijen

Joost Schalken

Marten Wensink

Jan Zuurbier

Hogeschool Utrecht Faculteit Natuur en Techniek

Inhoudsopgave

1 Management Samenvatting

1.1 Inleiding

Dit hoofdstuk bevat de management samenvatting voor het technisch verslag. Hier staan een beknopte samenvatting van een aantal onderdelen van het technisch slag, met uitzondering van modellen.

De opdracht

Swirl ®heeft aan ons gevraagd een systeem te ontwikkelen wat van een afstand een wasmachine kan aansturen. Dit zal gaan door middel van een webinterface die met de wasmachine communiceert.

Het doel

Het doel is om een systeem te ontwikkelen wat in staat is om een wasmachine aan te sturen vanaf een computer in het huis. Om dit doel te bereiken zal gebruik gemaakt worden van een embedded systeem in de wasmachine wat de wasmachine aanstuurt.

Opgeloste problemen

1.2 Verzamelde informatie

Welke informatie is verzameld en hoe?

1.3 Motivering Solution Architecture

Welke keuzes zijn gemaakt bij het opstellen van de requirements en solution architecture.

1.4 Resultaten

Wat zijn de belangrijkste resultaten?

1.5 Conclusies en aanbevelingen

Wat zijn de belangrijkste conclusies en aanbevelingen?

2 Inleiding

2.1 Opening

Het bedrijf Swirl®heeft aangegeven te willen investeren in wasmachine's die op afstand kunnen worden aangezet. Dit zou gaan door middel van een website die verbonden is met de wasmachine van een klant. Via deze website kan de klant een wasprogramma starten en verschillende parameters van het wasprogramma aanpassen.

2.2 Doel

In dit verslag worden de technische specificaties van het systeem beschreven. Programmeur's en technici die werken voor Swirl®kunnen dit document gebruiken om het systeem zo gewenst aan te passen of problemen te verhelpen.

2.3 Vooruitblik

Eerst worden de onderzoeken die zijn uitgevoerd besproken. Hier geven wij aan welke zaken wij informatie over hebben verzameld en hoe dit is gebeurd. Tevens presenteren wij hier onze experimenten en de resultaten hiervan.

Na het onderzoek bespreken wij het Requirements Architecture. In dit onderdeel worden de belangrijkste aspecten van het Requirements Architecture besproken, evenals de keuzes die gemaakt zijn.

In het Solution Architecture bespreken wij de architectuur van het systeem. Dit wordt gedaan door een aantal modellen die in de loop van het project ontwikkeld zijn. Deze zijn het klassendiagram, concurrency model en het dynamic model. Tevens leveren wij motivaties voor de gebruikte communicatie protocollen en de taakstructurering.

Na het Solution Architecture wordt de realisatie besproken. Hier beschrijven wij de problemen die tijdens de ontwikkeling zijn opgetreden,

en oplossingen voor deze problemen. Hiernaast geven wij een gedetailleerde uitleg over de algoritmen in de code voor de onderdelen waar dit niet uit de code of commentaar duidelijk is.

Bij de evaluatie bespreken wij zaken die beter ontwikkeld konden worden en wat een mogelijke oplossing hiervoor zou zijn.

Tot slot maken we conclusies en geven hierbij aanbevelingen, gevolgd door bronvermeldingen.

3 Onderzoek

3.1 Inleiding

In dit hoofdstuk bespreken wij de informatie die verzameld is voor het project, en hoe wij aan deze informatie zijn gekomen. Tevens bespreken wij de onderzoeken en experimenten die zijn uitgevoerd tijdens de realisatie van het systeem.

3.2 Verzamelde informatie en bronnen

JSON in C++

De wasprogramma's worden opgeslagen in JSON formaat. Om uit deze JSON alleen de relevante informatie te halen en via de websocket naar de webinterface te sturen, moet de JSON aan de C++ kant verwerkt worden. Omdat dit met C++ alleen niet mogelijk is, wordt er gebruik gemaakt van een externe library zoals RapidJSON Yip, 2016a.

Er is voor RapidJSON gekozen vanwege zijn performance. Uit benchmark tests is gebleken dat RapidJSON vele malen sneller is dan andere populaire JSON parsers. Op een systeem wat gebruik maakt van de GCC 32-bit compiler zijn 1000 tests uitgevoerd waarvan de snelheid in milliseconden is opgenomen. Voor bijvoorbeeld het parsen van DOM had RapidJSON 796 milliseconden nodig, terwijl YAJL 6316 milliseconden nodig had en JsonCpp 6705 milliseconden Yip, 2016b.

Voor het toepassen van RapidJSON hebben wij geruik gemaakt van een online tutorial THL A29 Limited en Yip, 2016.

3.3 Experimenten en resultaten

RapidJSON

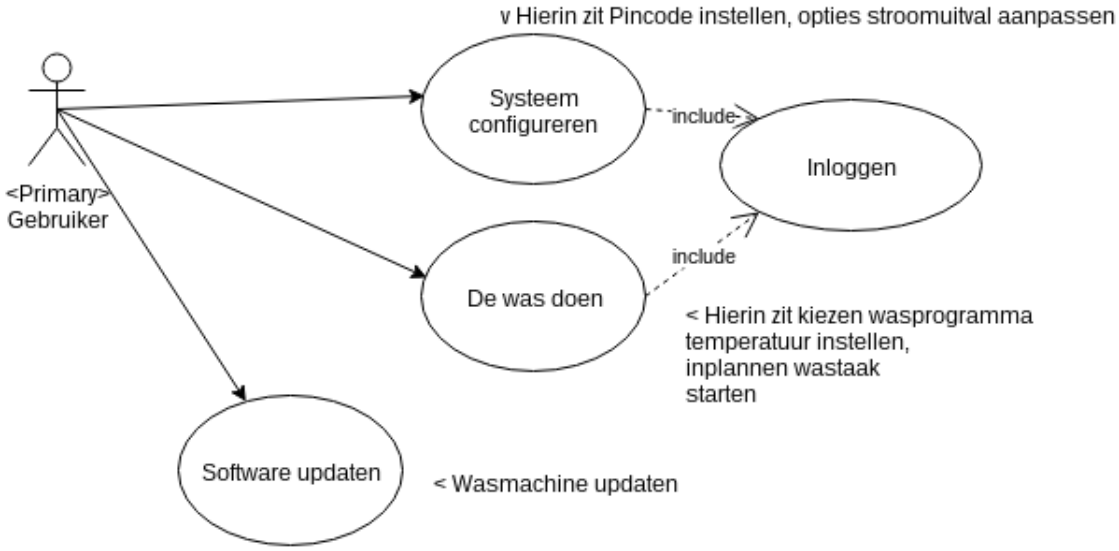
Zoals eerder vermeld maken wij gebruik van RapidJSON vanwege de hoge snelheid en compactheid van de library. Voor RapidJSON zijn een aantal

benchmarks uitgevoerd, maar omdat wij deze niet zelf hebben uitgevoerd zijn de resultaten vernoemd onder de kop ”Verzamelde informatie en bronnen”.

4 Requirements Architecture

4.1 Use-case diagram

In dit onderdeel zal het use-case diagram worden gepresenteert en de use-cases worden beschreven. Wanneer er wordt gerefereerd naar een ”gebruiker” dan wordt hiermee de klant bedoelt die de wasmachine gekocht heeft en deze gebruikt om de was mee te doen. Een ”beheerder” is iemand die werkt voor het bedrijf die wasmachine’s levert en beheert.



4.2 Use-case beschrijvingen

Systeem configureren

Doel	Het aanpassen van de pincode of veranderen wat er gebeurt na het uitvallen van de wasmachine.
Pre-condities	De gebruiker is ingelogd op de webinterface.
Post-condities	De aangepaste instellingen zijn opgeslagen.
Uitzonderingen	

De was doen

Doel	Zorgen dat de was gewassen wordt.
Pre-condities	De gebruiker is ingelogd op de webinterface.
Post-condities	De wasmachine is een wastaak aan het uitvoeren.
Uitzonderingen	

Authenticeren

Doel	Vaststellen dat degene die probeert toegang te krijgen tot de webinterface daadwerkelijk bevoegd is om de wasmachine te bedienen.
Pre-condities	Er is een webinterface-sessie gestart.
Post-condities	De gebruiker is geauthenticeerd.
Uitzonderingen	De gebruiker sluit de browser. De wasmachine gaat verder met taken indien deze voor het sluiten waren doorgegeven.

Software updaten

Doel	Het ontvangen van een update door deze te accepteren.
Pre-condities	De gebruiker is ingelogd op de webinterface.
Post-condities	Een nieuw wasprogramma is toegevoegd aan de lijst van wasprogramma's.
Uitzonderingen	

Log inzien

Doel	Het lezen van het logbestand om informatie te kunnen achterhalen over het systeem.
Pre-condities	
Post-condities	
Uitzonderingen	

Hervatten wasprogramma

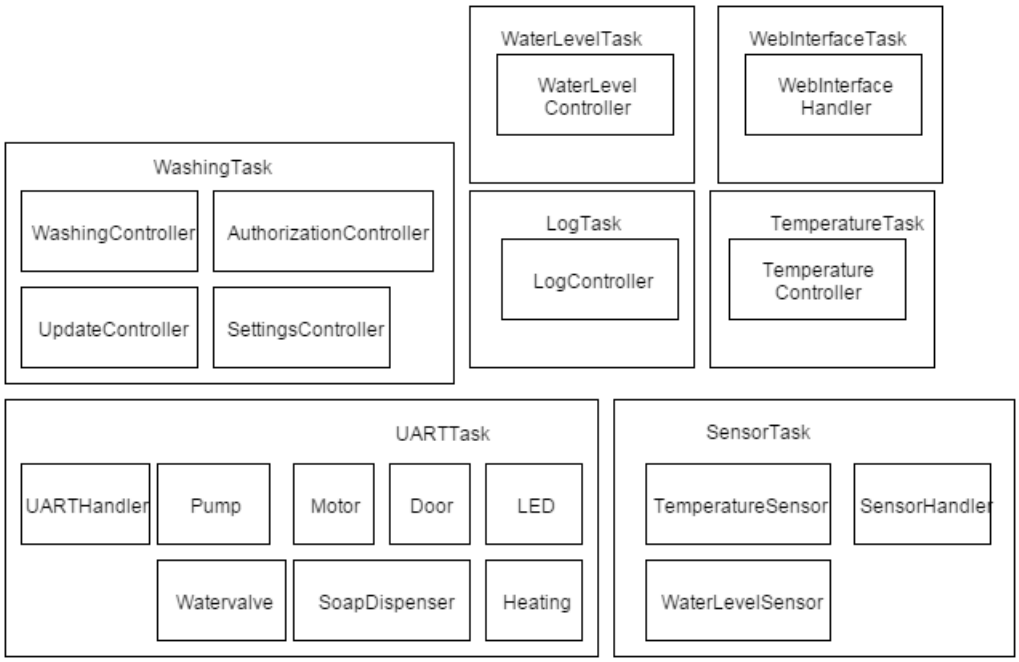
Doel	Het hervatten van het huidige wasprogramma na een stroomstoring.
Pre-condities	Er is een stroomstoring geweest waardoor het systeem tijdelijk niet in staat was om het wasprogramma uit te voeren.
Post-condities	Het systeem is bezit met het uitvoeren van het wasprogramma vanaf het laatst bereikte punt.
Uitzonderingen	De gebruiker heeft bij de instellingen aangegeven niet het programma te willen hervatten na een storing maar dat deze moet worden stopgezet. Het systeem pompt het water weg en ontgrendelt de deur.

5 Solution Architecture

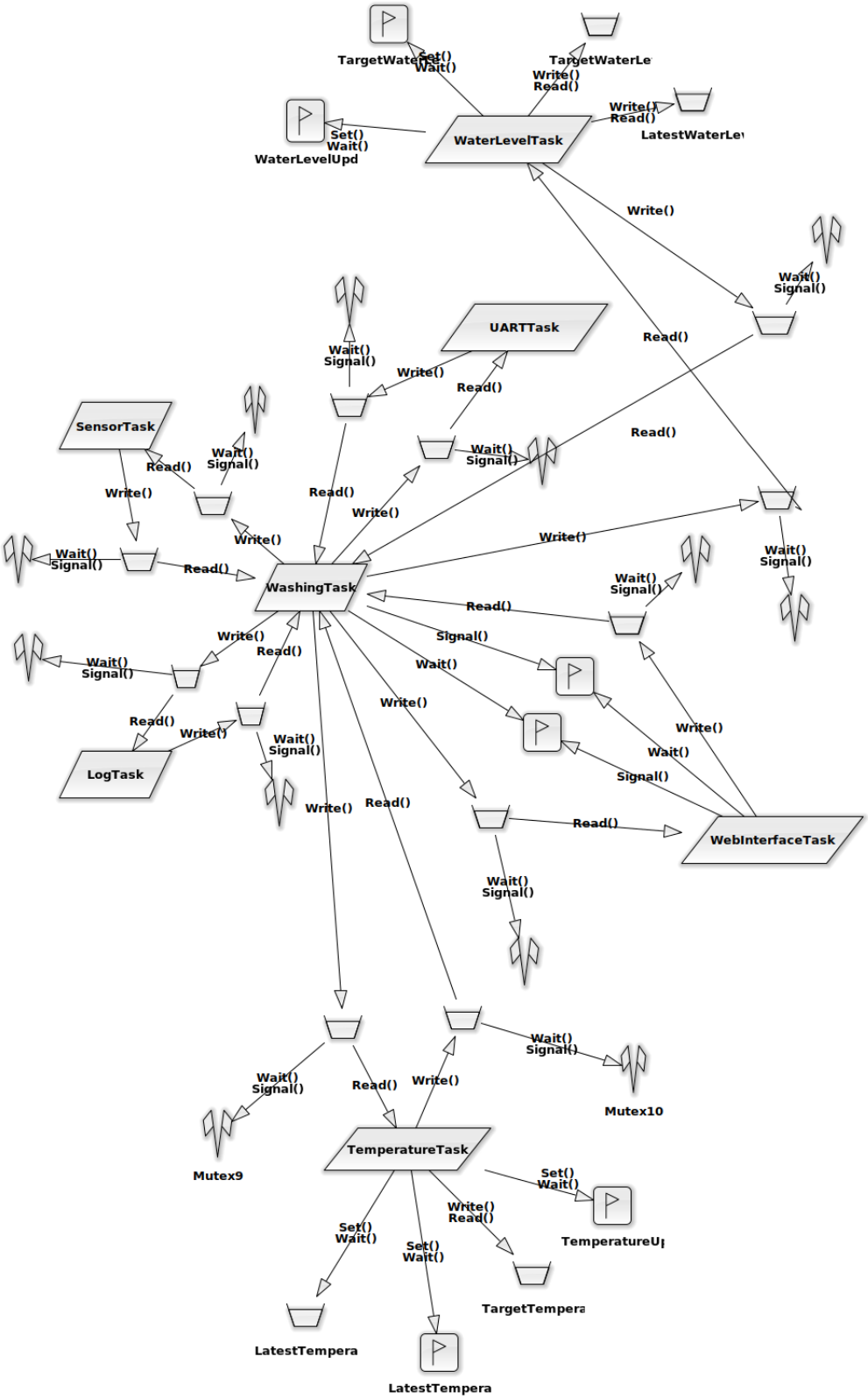
5.1 Klassendiagram

Het klassendiagram is helaas te groot geworden om af te drukken. Deze is als los bestand in de zip geplaatst. Het bestand heet klassendiagram.png.

5.2 Taakstructurering

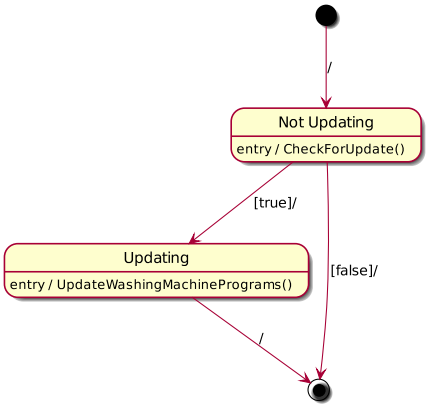


5.3 Concurrency Model

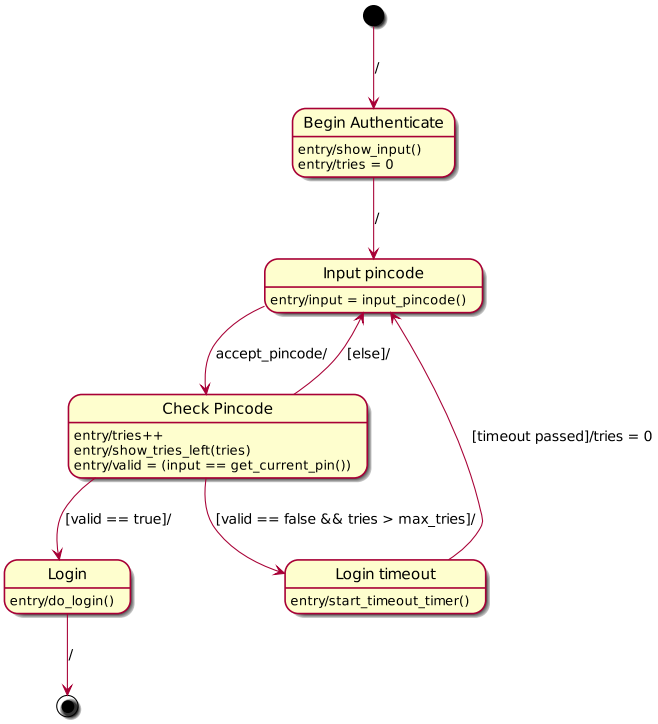


5.4 Dynamic model

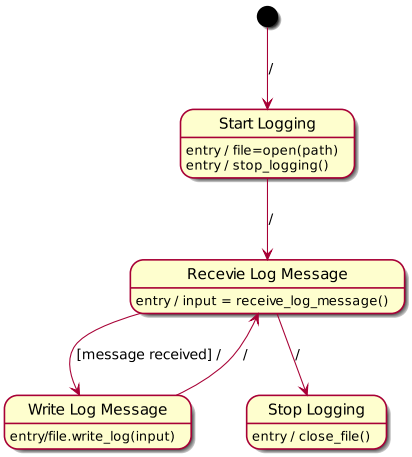
5.5 Update Controller



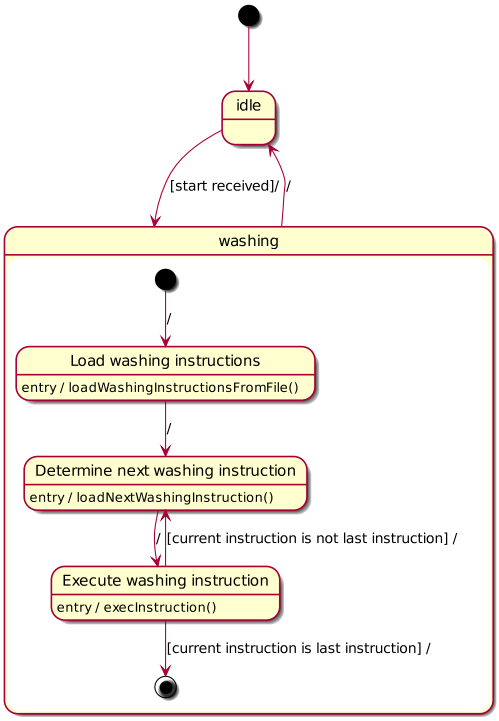
5.6 Authentication Controller



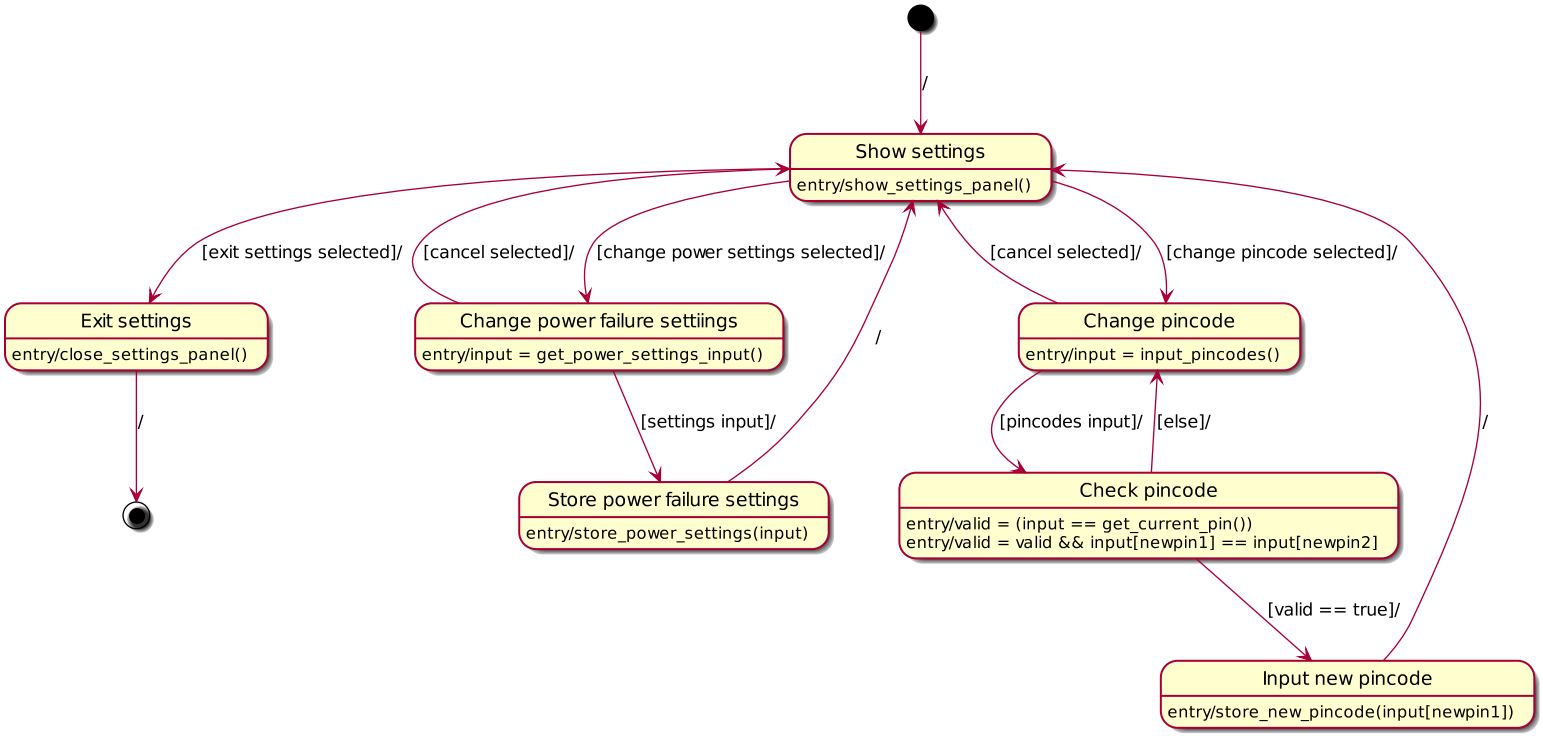
5.7 Log Controller



5.8 Washing Controller



5.9 Settings Controller



6 Realisatie

6.1 Opgetreden problemen en oplossingen

Schrijven naar bestand d.m.v. JavaScript

Om de gegevens van de gebruiker op te slaan (en op te halen) moesten wij deze ergens bewaren, bij voorkeur in een bestand. Tijdens het ontwikkelen van het systeem wat de gebruiker zijn gegevens laat op slaan, kwamen wij er achter dat JavaScript niet in staat is om bestanden aan de server-kant aan te passen. Dit was een groot probleem omdat wij geen gebruik maken van JavaScript libraries, welke mogelijk wel in staat zijn om bestanden aan te passen.

Uiteindelijk hebben wij voor dit probleem 2 oplossingen gevonden. De eerste oplossing was om uiteindelijk toch gebruik te maken van een JavaScript library, in dit geval JQuery. JQuery kan door middel van een Ajax call (in de vorm van $.ajax) externe en interne scripts aan roepen. Hiervoor hebben wij een kort PHP script geschreven wat het bestand aan past aan de hand van een binnen komende POST variabele.

De tweede oplossing was, alhoewel simpeler van aard, moeilijker toe te passen, en maakte de eerste oplossing nuttelloos. Tijdens de ontwikkeling van het systeem kwamen we tot de conclusie dat de bestanden met gebruikers gegevens aan de kant van de WebSocket geplaatst kunnen worden. Dit houdt in dat we door middel van c++ het bestand kunnen aanspreken, waardoor we geen externe library meer hoeven te gebruiken. Tevens hoeven we geen gebruik meer te maken van een PHP script.

Uiteindelijk hebben wij gebruik gemaakt van de tweede oplossing, omdat deze naast het probleem oplossing ook resulteerde in een betere architectuur.

Verbinden met websocket

Hoewel eerder de websocket prima verbinding maakte met de webinterface, lukte het sinds het implementeren van de websocket binnen het systeem opeens niet meer om een verbinding op te stellen. Uiteindelijk hebben we gebruik gemaakt van GDB, die gedetailleerder aantoont op welke regel de software misgaat. Hieruit bleek dat bij het verwerken van de binnenkomende requests, het woord request verkeerd gespeld was (Request was "Rqeuest").

RTOS CPU tijd

Tijdens de ontwikkeling van het systeem kwamen wij er achter dat het RTOS 100% van de CPU tijd gebruikt. Hierdoor gebruikte het systeem meer stroom, en konden andere onderdelen van het systeem niet goed functioneren.

Om dit op te lossen hebben wij het RTOS aangepast. Nu wordt er tijdens het aftellen van de timers gecontroleerd welke de kleinste tijds duur heeft. Indien er geen beschikbaar proces voor is dan wordt voor deze tijd sleep() gedaan. Hierdoor gebruiken wij ongeveer 5% CPU tijd i.p.v. 100%.

Fatale fout in websocket

Inkomende websocket verbindingen worden buiten het RTOS om afgehandeld. Tijdens de ontwikkeling bleek dat buiten het RTOS geen items toegevoegd konden worden aan channels. Dit hebben wij opgelost door een std::queue te gebruiken aan de kant van de websockets, en vanuit het RTOS een method aan te roepen om deze std::queue te pollen.

6.2 Onopgeloste problemen

Het systeem crasht soms als de websocket gesloten wordt vlak na het starten van een wasprogramma. Er wordt dan een exceptie opgegooid vanuit de websocket klasse, die niet afgehandeld wordt.

6.3 Gedetailleerde uitleg code

Boundary objecten

Communicatie tussen boundary objecten en de UART interface wordt afgehandeld door de UARTHandeler klasse. Deze wacht totdat er nieuwe commando's in een channel geplaatst worden, en pollt vervolgens voor inkomende reacties. Tijdens het wachten op een reactie van de emulator worden taken die commando's sturen stil gelegd.

Web interface

Communicatie tussen de webinterface en de rest van het systeem wordt uitgevoerd door twee hoofdonderdelen. De WebsocketHandler accepteert inkomende verbindingen. Een listener klasse wordt geabbonneerd op binnenkomende berichten op die verbindingen. Binnenkomende berichten worden in een std::queue gezet. Vanuit de WebinterfaceHandler wordt iedere 20 MS gepollt op deze queue. Vanuit de WebinterfaceHandler worden de binnenkomende berichten afgehandeld. Als er data opgevraagd wordt stuurt de WebinterfaceHandler de gevraagde data terug. Als er opdrachten gegeven worden zorgt de WebinterfaceHandler ervoor dat deze uitgevoerd worden.

Berichten over de websocket worden met JSON geformatteerd. Deze berichten worden geparsed met de rapidjson library. Ook op andere plekken binnen het programma wordt deze library gebruikt.

Wasprogramma's

Wasprogramma's worden opgeslagen in JSON geformatteerde bestanden. Zodra de webpagina wordt geladen worden deze bestanden ingelezen en geparsed. Het parsen JSON wordt gedaan met behulp van de rapidjson library.

Wasprogramma's bestaan uit een reeks instructies die door het systeem uitgevoerd worden. De motor laten draaien, water in de trommel pompen en de temperatuur instellen zijn enkele voorbeelden van instructies. Instructies worden vanuit het JSON bestand ingelezen en een voor een uitgevoerd door de WashingController.

Verdere toelichting

Verdere toelichting over de interne werking van de software is te vinden in de doxygen documentatie.

7 Evaluatie

7.1 Inleiding

In dit hoofdstuk bespreken wij de onderdelen van de realisatie die achteraf beter hadden kunnen zijn opgelost. Tevens bieden wij voor deze problemen beter oplossingen die zo gewenst in de toekomst alsnog toegepast kunnen worden.

7.2 Communicatie tussen RTOS en de rest van het systeem

Tijdens de ontwikkeling kwamen wij erachter dat communicatie tussen het RTOS en andere threads vrij omslachtig is. Mogelijk hadden wij dit beter kunnen oplossen door de websocket thread inkomende verbindingen te laten accepteren en de socket objecten vanuit het RTOS op te halen, en deze sockets binnen het RTOS verder af te handelen.

Bibliografie

THL A29 Limited, a. T. c. & Yip, M. (2016). Tutorial. http://rapidjson.org/md$_d$oc$_t$utorial.html.

Yip, M. (2016a). A fast json parser/generator for c++ with both sax/dom style api. https://github.com/miloyip/rapidjson.

Yip, M. (2016b). Performance. https://code.google.com/p/rapidjson/wiki/Performance.