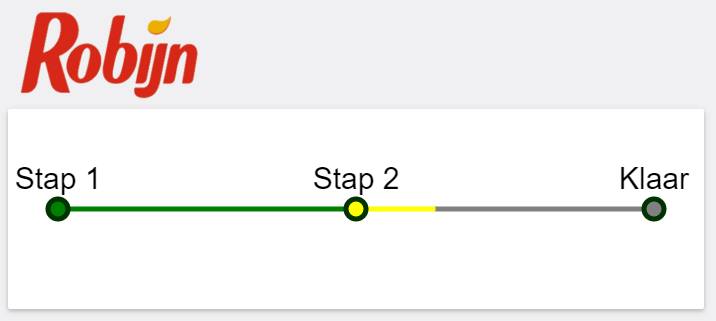
Technisch verslag

Wasmachines leven langer met Robijn



Teamnummer: 13

Teamnaam: Robijn

Klas: V2B

Studentnummer + Naam:

1615865 Waila Woe

1660936 Thijs Hendrickx

1665553 Mathijs van Bremen

1657667 Yorick Schellevis

1664653 Bouke Stam

Documentdatum: 18-1-2016

Versienummer: 1.0

Managementsamenvatting

**Probleem**

Wasmachines zijn tegenwoordig nog steeds hetzelfde als vroeger. Men kan zijn wasmachine instellen maar weet bijvoorbeeld niet wat precies in de achtergrond gebeurd. Men start zijn wasmachine met de instellingen die hij/zij wilt en haalt kleding uit de was wanneer de wasmachine gestopt is met draaien, maar weet niet de exacte tijdstip van wanneer het klaar is. Wat de wasmachine precies verbruikt is ook vaak onbekend. De opdrachtgever Swirl wil een wasmachine maken die te bedienen is op afstand en daarmee ook het genoemde probleem op te lossen. In dit kader is een onderzoek gedaan naar de mogelijkheden.

**Advies**

* inloggen op de browser
* een wasmachine kunnen starten en stoppen via de browser
* water temperatuur instellen van was
* de tijd kunnen instellen van de was
* De status kunnen zien van de huidige was
* mogelijk de Stroom verbruik

**Motief**

Voor het onderzoek is een interview gehouden met de opdrachtgever Jan Zuurbier. Hiervoor is een schets gemaakt van de browser om te verduidelijken hoe het ongeveer zou moeten werken. De beantwoorde vragen tijdens het interview zou hebben gezorgd voor extra duidelijkheid wat de opdrachtgever absoluut niet wilt of wat juist wel geïmplementeerd moet worden. Daaruit is gebleken dat veiligheid een belangrijk punt is en ook de status waar de wasmachine zich bevindt. Van de antwoorden is met behulp van de MoSCoW methode een lijst gemaakt, wat ervoor moet zorgen voor een duidelijk overzicht met prioriteiten. Door middel van de MoSCoW lijst is daarvan een use case gemaakt van alleen de belangrijkste taken die de wasmachine moet kunnen. En daarvan is er een activity diagram van gemaakt. Met deze use case, Activity Diagram en website schets is het Requirements Architecture document opgesteld.

Nadat alles duidelijk geworden is, is de Solution Architecture document geschreven. De Task Structurering diagram zal ervoor moeten zorgen dat het systeem optimaal functioneert. Taken zullen zodanig worden opgesplitst om dit te kunnen realiseren. In het dynamic model wordt vastgelegd hoe de objecten intern functioneren en reageren op bepaalde events. Verder is nog het klassen diagram van het uiteindelijke systeem waar alle code in geschreven wordt. Al deze diagrammen worden beschreven in het Solution Architecture document.

**Consequenties**

* Wasmachine besturen via browser betekend dat toegang tot het internet nodig is. Zonder internet is het niet mogelijk om toegang te krijgen tot de wasmachine

Inhoud

[1. ​Inleiding 5](#_Toc441242876)

[2. Onderzoek 6](#_Toc441242877)

[​2.1​ Raspberry PI 6](#_Toc441242878)

[​2.2​ Webpagina server 6](#_Toc441242879)

[​2.3​ Socket server 6](#_Toc441242880)

[2.4​ JavaScript server communicatie 7](#_Toc441242881)

[​2.5​ RTOS 7](#_Toc441242882)

[2.6​ Wasmachine emulator 8](#_Toc441242883)

[​2.7​ Wasprogramma 9](#_Toc441242884)

[3.​ Requirements architecture 10](#_Toc441242885)

[​3.1​ Use case 10](#_Toc441242886)

[​ Use case diagram: 10](#_Toc441242887)

[Use case beschrijving 11](#_Toc441242888)

[​3.2​ Activity diagram 12](#_Toc441242889)

[Starten van wasbeurt 12](#_Toc441242890)

[​Stoppen van wasbeurt 12](#_Toc441242891)

[Foutmelding tijdens wassen 13](#_Toc441242892)

[3.3​ Schets website 14](#_Toc441242893)

[Instel pagina 14](#_Toc441242894)

[Status pagina 15](#_Toc441242895)

[3.4​ Klassendiagram 16](#_Toc441242896)

[​Hardware 16](#_Toc441242897)

[UART 17](#_Toc441242898)

[Controllers 18](#_Toc441242899)

[Wasprogramma’s 19](#_Toc441242900)

[​Webinterface 20](#_Toc441242901)

[​3.5​ Taakstructurering 21](#_Toc441242902)

[​I/O taakstructurering 21](#_Toc441242903)

[Interne taakstructurering 22](#_Toc441242904)

[Taak structuur diagram 23](#_Toc441242905)

[3.6​ Concurrency model 24](#_Toc441242906)

[​WebInterfaceTask 25](#_Toc441242907)

[​UARTInterfaceTask 25](#_Toc441242908)

[3.7​ Dynamic model 26](#_Toc441242909)

[State Transition Diagram 26](#_Toc441242910)

[3.8​ Communicatie protocol 31](#_Toc441242911)

[​4​ Realisatie 34](#_Toc441242912)

[​4.1​ Hardware Problemen & Oplossingen 34](#_Toc441242913)

[​4.2​ Software Problemen & Oplossingen 34](#_Toc441242914)

[​Hardware Controllers en WashingProgramController 34](#_Toc441242915)

[WebInterfaceController 34](#_Toc441242916)

[WebServer, SocketServer en SocketMessage 35](#_Toc441242917)

[Algoritmes 35](#_Toc441242918)

[4.3​ Tests 36](#_Toc441242919)

[Systeem Test 36](#_Toc441242920)

[Security Test 38](#_Toc441242921)

[Stroom Uitval Test 39](#_Toc441242922)

[​5​ Evaluatie 41](#_Toc441242923)

[6​ Conclusies en aanbevelingen 42](#_Toc441242924)

[6.1​ Conclusies 42](#_Toc441242925)

[6.2​ Aanbevelingen 42](#_Toc441242926)

[7 Bronvermeldingen 43](#_Toc441242927)

[8​ Bijlagen 44](#_Toc441242928)

[Bijlage A: MoSCoW analyse 45](#_Toc441242929)

[Bijlage B: Interview vragen 46](#_Toc441242930)

# 1. ​Inleiding

In deze opdracht wordt de kennis van C++ in een wat meer systematische omgeving gebruikt. Het doel is een besturingssysteem te maken voor een wasmachine op het Internet of Things. Hierdoor kan de gebruiker de wasmachine via het internet aansturen met behulp van een website, app of programma.

We komen in een tijdperk waarin steeds meer systemen bediend kunnen worden over het internet. Consumenten kunnen lichten, verwarming, televisie en nog veel meer allemaal bedienen met één druk op een knop op hun smartphone. In deze opdracht ligt de focus bij een wasmachine. Van de watertoevoer tot de rotatiesnelheid van de trommel, alles moet door middel van een ingesteld wasprogramma bediend kunnen worden via een makkelijke gebruikersinterface op een website.

Dit verslag dient ervoor om uit te leggen hoe de architectuur van het besturingssysteem in elkaar zit, en hoe dit is gerealiseerd. Het verslag is bestemd voor toekomstige projectleden en mensen die geïnteresseerd zijn in de werking van het systeem. Het systeem is vooral gericht naar studenten die op zichzelf wonen en volwassenen die gebruik maken van een wasmachine.

In het begin van het verslag staat het onderzoek dat is gedaan naar de raspberry pi, RTOS en andere benodigde soft- en hardware. Daarna worden alle eisen aan het systeem beschreven, evenals een schets voor de layout van de website. In het midden van het verslag staat de architectuur van het systeem. Dit wordt weergegeven met behulp van een klassendiagram, taakstructurering, concurrency model en STD. Daar staat ook uitgelegd hoe het communicatie protocol werkt.

Aan het einde van het verslag wordt beschreven hoe dit project is gerealiseerd en welke problemen zijn voorgekomen. Daarna volgt een uitleg van welke tests zijn gedaan om de soft- en hardware te testen. Als laatste komt de evaluatie van het project en de getrokken conclusies en aanbevelingen voor volgende projecten.

# 2. Onderzoek

Voordat er begonnen kan worden met het bedenken van de oplossing moest eerst een onderzoek worden gedaan naar alle software en hardware die gebruikt gaat worden. Enkele zijn door school aangewezen en van andere moet worden uitgezocht welke het beste is. Hieronder wordt beschreven wat voor elk onderdeel moet worden uitgezocht.

## ​2.1​ Raspberry PI

De raspberry pi is een goedkope en kleine computer gemaakt voor educatieve en hobby doeleinden. De versie die gebruikt wordt is de Raspberry Pi Model B rev2. Deze bestaat uit de volgende onderdelen(Raspberry, 2015a):

* SD kaart lezer en schrijver
* HDMI output
* Composite video output
* 2 USB 2.0 poorten
* 26 GPIO pinnen
* 3.5mm audio aansluiting
* Camera interface poort
* LCD display interface poort
* MicroUSB voeding aansluiting (700mA, 3.5W, 5V)
* 100mb Ethernet port
* 512MB ram
* ARM 1176 700Mhz processor
* Er zijn meerdere operating systems beschikbaar voor de raspberry pi(Raspberry, 2015a). De populairste daarvan is Raspbian. Dit is een versie van Debian linux, omgevormd voor de raspberry pi. De nieuwste versie hiervan is “Jessie”, welke ook wordt aangeraden door school, vanwege de versie van gcc.

## ​2.2​ Webpagina server

* Wordt een bestaande server gebruikt of wordt één aangemaakt? Veel webservers zijn verkrijgbaar voor linux. De meest bekende is apache. Apache kan makkelijk worden geïnstalleerd op linux via het apt-get commando(Raspberry, 2015b). Alle bestanden in de /var/www/html folder zijn dan beschikbaar via het netwerk.

## ​2.3​ Socket server

* Hoe wordt de server socket gemaakt? De socket server die praat met de webpagina kan vrij simpel worden gemaakt in C++. Bij de cursus Operating Systems & Netwerk Programmeren is gebruik gemaakt van een kleine library die sockets implementeerd in C++(Donahoo & Calvert, 2001). Dit kan wederom worden gebruikt in dit project.

## 2.4​ JavaScript server communicatie

Om vanuit de browser te praten met de server zijn er twee mogelijke oplossingen. De eerste en meest bekende is AJAX. AJAX stuurt een http request naar de server zonder de pagina te vernieuwen. Het resultaat wordt dan teruggegeven via een callback methode.

De tweede optie is websockets. Sinds HTML5 wordt deze standaard ondersteund door de browser en is het vrij gemakkelijk geworden om dit te implementeren. Met een websocket maakt de browser een directe verbinding met de server, die open blijft zolang als het nodig is. Berichten kunnen via deze verbinding beide kanten op worden gestuurd.

Websockets zijn in ons geval de beste keus. Dit is omdat websockets veel sneller zijn waardoor vaker updates kunnen worden gestuurd naar de client. Ook hoeft de client niet voor elke update een request te sturen, maar kan de server dat doen wanneer hij nieuwe data heeft.

Javascript heeft goede ondersteuning voor websockets en heeft eenvoudige methodes om een socket verbinding te openen, berichten te ontvangen en te versturen. Hierdoor zal het niet veel tijd kosten om dit implementeren in de webpagina.

## ​2.5​ RTOS

RTOS is een library die synchronisatie objecten en een task scheduler biedt. Om dit toe te voegen moet de header files worden include en libraries worden gelinked. De libraries bestaat uit de volgende klassen:

* channel
* clock
* event
* flag
* mailbox
* mutex
* pool
* task
* timer
* waitable

RTOS draait in een realtime thread waardoor het de hoogste prioriteit heeft.

## 2.6​ Wasmachine emulator

Omdat het niet haalbaar is om echte wasmachines te gebruiken is er een wasmachine emulator gemaakt. Deze simuleert de hardware van de wasmachine. De emulator draait op de lpc1114 chip met de ARM Cortex-M0 processor(Wensink, 2015).

De wasmachine bestaat uit de volgende componenten:

* Wasmachine motor met instelbaar toerental van 0 t/m 1600 rpm
* Deurslot met vergrendeling
* Noodknop
* Verwarmingselement
* Kraan
* Pomp
* Zeeptoevoer
* Signaal led die brandt als het programma is afgelopen

De communicatie met de raspberry pi verloopt via een serieel interface, waarmee commando’s naar de emulator kunnen worden gestuurd. De emulator reageert vervolgens met een tegenbericht. Een commando bestaat uit twee bytes, een request en een command. Het request geeft aan om welk onderdeel het gaat en het command is vaak de actie.

De reactie van de emulator bestaat ook uit twee bytes, namelijk een response en een status. De response herhaalt de request byte, met het hoogste bit aangezet. De status geeft de status van het component aan(Wensink, 2015).

De emulator kan zich in 5 toestanden bevinden:

*Idle*

Dit is de begin toestand, de emulator reageert alleen het MACHINE\_REQ commando.

*Running*

De emulator reageert op alle commando’s en het indrukken van de knoppen.

*Stopped*

De deur is ontgrendeld en de watertemperatuur wordt niet meer gelijk gehouden. Het waterniveau blijft echter wel hetzelfde.

*Failed*

De fail knop is ingedrukt, het vullen, verwarmen en pompen stopt. Na nog een druk op de failknop verandert de toestand naar stopped. De failed toestand simuleert een stroomuitval.

*Halted*

De noodknop is ingedrukt, de emulator reageert alleen nog op het opvragen van de machine status.

## ​2.7​ Wasprogramma

Om een wasprogramma uit te voeren moeten de volgende stappen worden gedaan:

- Wacht tot deur is gesloten

- Vergrendel deur

- Start wasmachine

- Open zeepbak

- Vul water tot bepaald niveau

- Verhoog temperatuur tot bepaald niveau

- Langzaam ronddraaien afwisselend links en rechts voor lange tijd

- Voer al het water af

- Ga heel snel rond draaien voor korte tijd

- Stop wasmachine

- Ontgrendel deur

De variabelen in dit stappenplan zijn:

- Waterniveau

- Temperatuur

- Rotatie snelheid

- Tijd

- Rotatie wissel interval

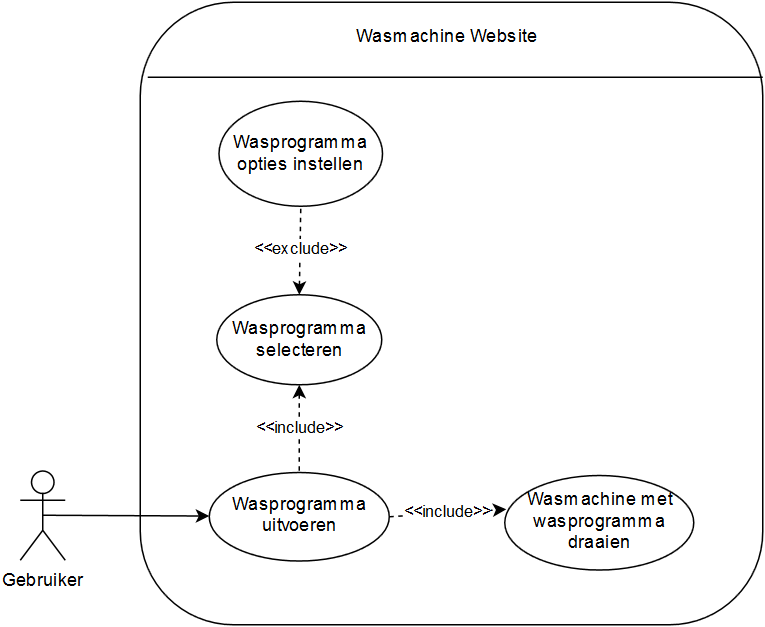
Variaties van deze variabelen leveren verschillende wasprogramma’s op. Bijvoorbeeld snel, eco, bonte was en nog meer(Greenem, 2015).

De wasprogramma’s kunnen het beste opgeslagen worden in een JSON formaat. Dit is makkelijk te lezen met Javascript en ook in c++ met behulp van een library. Ook is JSON makkelijk om te zetten naar een string waardoor het verstuurd kan worden met de websocket.

# 3.​ Requirements architecture

## ​3.1​ Use case

### ​ Use case diagram:



### Use case beschrijving

|  |  |
| --- | --- |
| **Use case naam** | **Wasprogramma uitvoeren** |
| Doel | Het starten van een wasprogramma. Wasprogramma wordt gestuurd naar de wasmachine zodat die zijn taak kan beginnen. |
| Pre-conditie | Wasprogramma geselecteerd of ingesteld. |
| Post-conditie | - |
| Uitzonderingen | - |

|  |  |
| --- | --- |
| **Use case naam** | **Wasprogramma opties instellen** |
| Doel | Het handmatig instellen van een wasprogramma of een standaard wasprogramma selecteren. |
| Pre-conditie | Er wordt geen wasprogramma uitgevoerd. |
| Post-conditie | Wasprogramma is gestart. |
| Uitzonderingen | 1. Als een gebruiker zelf de temperatuur van het water wil instellen, wordt de use-case Water temperatuur instellen uitgevoerd. 2. Als een gebruiker zelf de rotatiesnelheid wil instellen, wordt de use-case Rotatie snelheid instellen uitgevoerd. |

|  |  |
| --- | --- |
| **Use case naam** | **Water temperatuur instellen** |
| Doel | Het instellen van het water in de wasmachine in graden Celcius. Dit wordt ingesteld in een wasprogramma. |
| Pre-conditie | - |
| Post-conditie | - |
| Uitzonderingen | - |

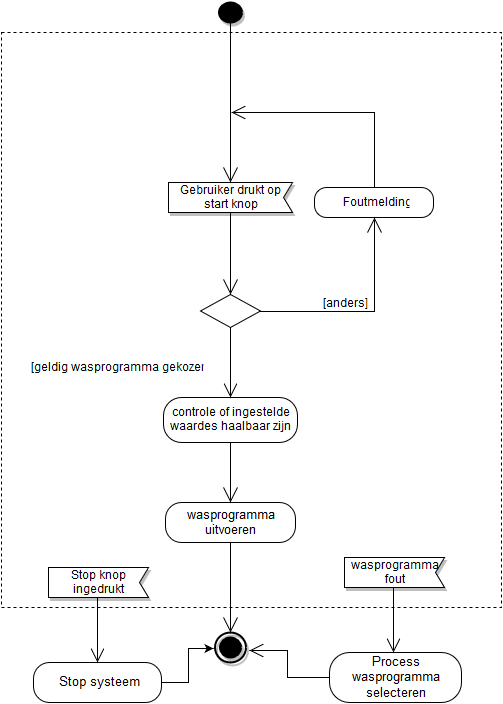
|  |  |
| --- | --- |
| **Use case naam** | **Rotatie snelheid instellen** |
| Doel | Het instellen van de snelheid in de wasmachine in omwenteling per minuut (RPM). Dit wordt ingesteld in een wasprogramma. |
| Pre-conditie | - |
| Post-conditie | - |
| Uitzonderingen | - |

|  |  |
| --- | --- |
| **Use case naam** | **Wasmachine met wasprogramma draaien** |
| Doel | Wasmachine draait een wasprogramma dat door de gebruiker ingesteld en uitgevoerd is. |
| Pre-conditie | Wasprogramma is ingesteld |
| Post-conditie | - |
| Uitzonderingen | Als de gebruiker het wasprogramma wil stoppen, wordt de use case Wasprogramma stoppen uitgevoerd. |

## ​3.2​ Activity diagram

Voor de use case “Wasprogramma uitvoeren” en de use case “Wasprogramma instellen” zijn activity diagrammen opgesteld. Deze zijn hieronder te vinden.

### Starten van wasbeurt

Het AD begint bij het event “Gebruiker drukt op start knop”. Dit event wacht (polling) tot iemand via de website op de start knop drukt. Na het drukken op een knop volgt er een decision node, wanneer een incorrect wasprogramma is gekozen gaat hij naar de actie “Foutmelding”. Na het uitvoeren van de “Foutmelding” actie gaat hij terug naar de beginstatus. Als er wel een correct wasprogramma is gekozen, zal hij door gaan naar de actie “Controle ingestelde waardes”. De actie “Controle ingestelde waardes” controleert of de ingestelde waardes wel haalbaar zijn en zal daarna naar de volgende actie “wasprogramma uitvoeren” gaan.

### ​Stoppen van wasbeurt

Tijdens een wasbeurt kan het misschien zo zijn dat de gebruiker iets vergeet. Een voorbeeld hiervan is bijvoorbeeld dat de gebruiker wasmiddel vergeet te gebruiken. De gebruiker kan in dat geval de wasmachine stoppen. Het event “stop knop ingedrukt” wacht tot dat de gebruiker de stop knop in drukt op de website. Wanneer dit zo is zal het systeem doorgaan naar de actie “Stop systeem”. De wasmachine zal dan stoppen met het opstarten/wassen.

### Foutmelding tijdens wassen

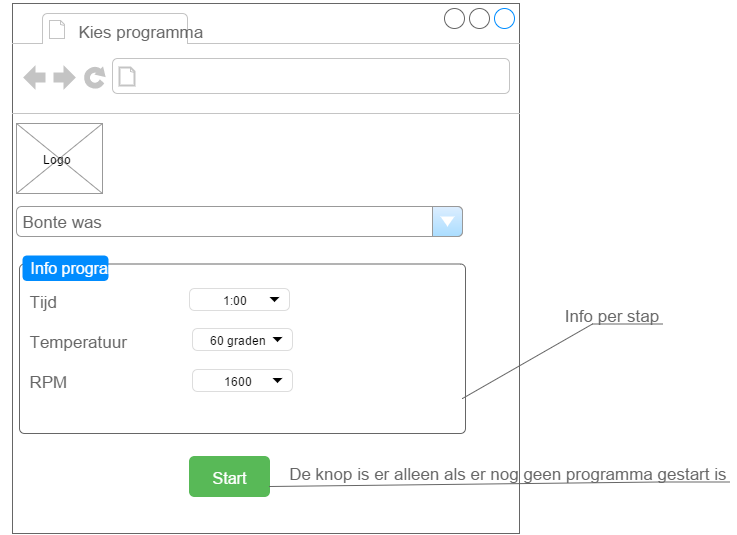
Tijdens wassen is het mogelijk dat het fout gaat. Wanneer er een fout optreedt zal dit het event “Wasprogramma fout” activeren. Dit event gaat door naar de actie “Process wasprogramma selecteren”.

## 3.3​ Schets website

Voor de website is een schets gemaakt dat laat zien hoe de website ongeveer uit komt te zien en de werking hiervan. Er zijn twee schetsen gemaakt die hieronder verder worden beschreven.

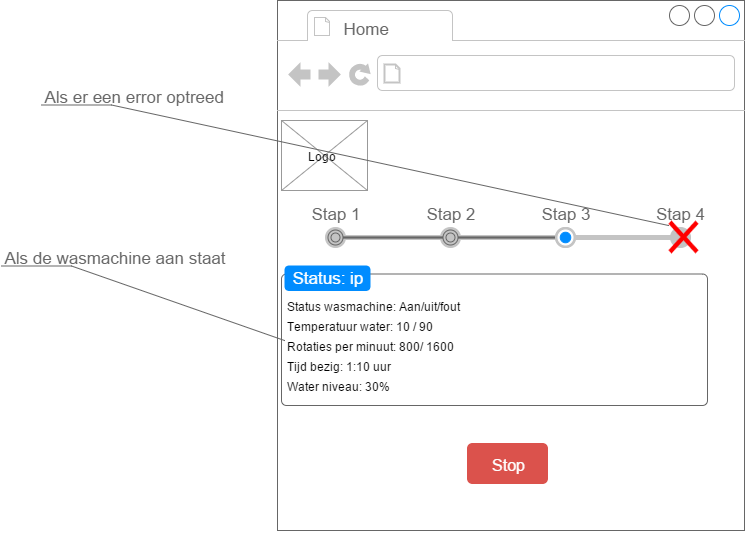
### Instel pagina

Op de eerste afbeelding is weergegeven wat de pagina laat zien als de wasmachine nog niet bezig is met het uitvoeren van het wasprogramma. De gebruiker kan hier een wasprogramma selecteren met behulp van een drop down list. Na het selecteren komt in het venster daaronder informatie te staan over het wasprogramma waarna ook de Start knop ingedrukt kan worden. De gebruiker kan eventueel zelf nog per stap aanpassingen doen aan het geselecteerde wasprogramma.



### Status pagina

Op de tweede afbeelding is weergegeven wat de pagina laat zien als de wasmachine al bezig is met het uitvoeren van het wasprogramma. Hier kan ook op “Stop” gedrukt worden om zo de wasmachine te stoppen. De gebruiker kan hier duidelijk zien met welke stap de wasmachine bezig is en wat de status van deze stap is.

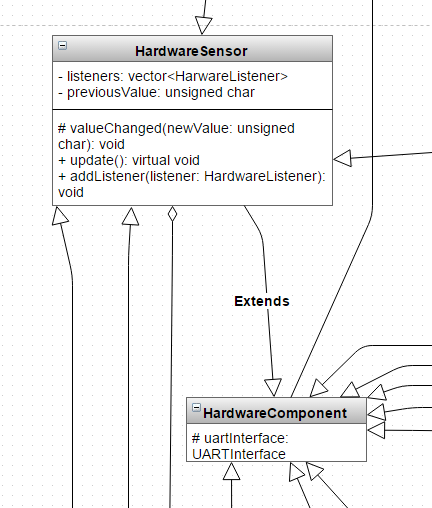
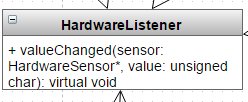


## 3.4​ Klassendiagram

### ​Hardware

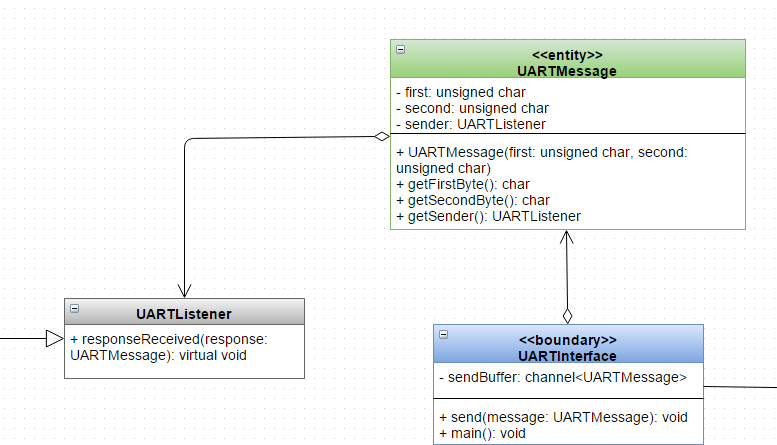
Er zijn twee soorten hardware: de actoren en de sensoren. De actoren vallen onder de HardwareComponent interface. De sensoren vallen onder de HardwareSensor interface, een sensor is ook een HardwareComponent.

Elk HardwareComponent heeft toegang tot het UARTInterface om berichten te kunnen versturen. De HardwareSensor houdt bij wie zijn listeners zijn en stuurt updates wanneer zijn waarde veranderd. Dit gebeurt via het HardwareListener interface. Elke class die het HardwareListener interface implementeerd kan zich aanmelden bij een HardwareSensor.



### UART

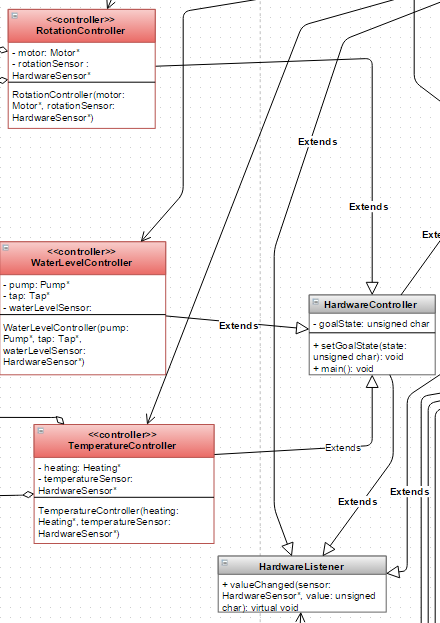
De UARTInterface is een RTOS::task en kan asynchrone berichten versturen via de UART verbinding. Een UARTMessage bestaat uit twee bytes en een sender. De sender is een UARTListener. Het UARTListener interface implementeerd de methode responseReceived, zo kan de UARTInterface het antwoord terug sturen naar degene die het verstuurd had wanneer het beschikbaar is.

De UARTInterface heeft een channel waar het alle berichten in op slaat, zo hoeft niemand te wachten tot het vorige bericht verzonden is.

### Controllers

De HarwareComponent worden gereguleerd door HardwareControllers, elke HardwareController heeft een goalState waarin die terecht wil komen. Iedere controller is een RTOS::task waardoor ze los van elkaar werken.

De HarwareControllers krijgen nieuwe waardes van de sensoren binnen via de HardwareListener interface.

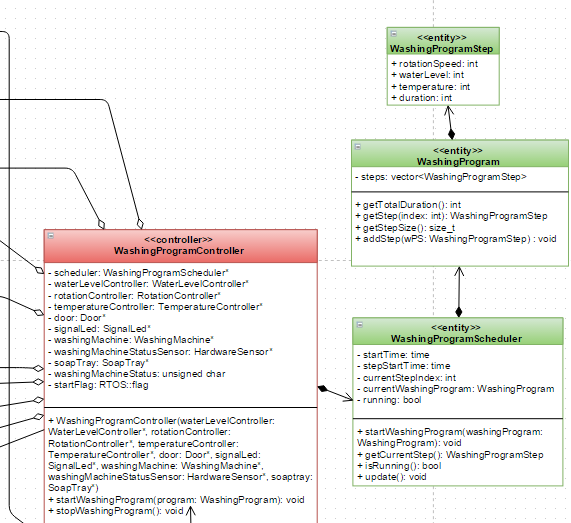


### Wasprogramma’s

De WashingProgramController heeft directe toegang tot de Door, SignalLed en SoapTray. Verder kent WashingProgramController alle andere hardware controllers. De taak van de WashingProgramController is om een WashingProgram uit te voeren door te zorgen dat alle hardware wordt ingesteld volgens het programma.

Een WashingProgram bestaat uit meerdere stappen. Elke stap heeft een rotatiesnelheid, waterniveau, temperatuur en tijd. De WashingProgramController heeft methodes om een WashingProgram te starten en te stoppen.

Verder kan de WashingProgramController de status van alle componenten opvragen en teruggeven via de methode getStatus(). Het geeft de status vervolgens terug als een WashingProgramStatus object, dat weer kan worden omgevormd tot SocketMessage om verzonden te worden naar de browser.

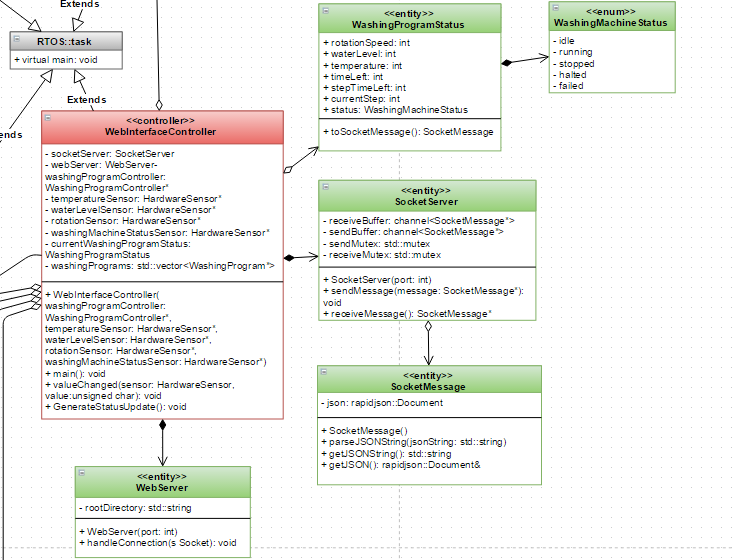
De WashingProgramController kent de UARTInterface waardoor het de status van de wasmachine kan opvragen. Ook kan het de wasmachine starten en stoppen.

### ​Webinterface

De Webinterface bestaat uit de Webserver die bestanden host en de SocketServer die de communicatie met de browser na het laden van de webpagina regelt. De SocketServer verstuurt en ontvangt SocketMessages. Een SocketMessage bestaat uit een JSON structuur en kan geparsed worden vanuit een string en geconverteerd worden naar een string.

De SocketServer heeft twee buffers waarin respectievelijk alle ontvangen en verzonden berichten worden opgeslagen. De WebinterfaceController hoeft hierdoor niet te wachten op de SocketServer.

De WebInterfaceController is geregistreerd als listener bij de sensoren. Deze waardes slaat die op in de WashingProgramStatus en geeft die door aan de SocketServer wanneer iets wordt veranderd.



## ​3.5​ Taakstructurering

### ​I/O taakstructurering

De UARTInterfaceTask verstuurt asynchroon berichten via de UART en geeft de antwoorden terug via een listener pattern. Dit kost 10 MS per bericht. Deze taak hoeft alleen actief te worden wanneer een bericht wordt verstuurd en is daarom niet periodiek.

De WashingProgramTask vraagt de status van de wasmachine op en start eventueel de wasmachine op. Ook kijkt WashingProgramTask of een nieuw wasprogramma moet worden opgestart.

De TemperatureTask stuurt de Heating aan en update de TemperatureSensor, dit gebeurt namelijk altijd tegelijkertijd. Het updaten van de sensor wordt asynchroon gedaan door de UART interface en kost daarom bijna geen tijd voor de TemperatureTask.

De MotorTask polled de RotationSensor en bepaald de te zetten snelheid van de Motor.

De WaterLevelTask update de WaterLevelSensor en stuurt aan de hand daarvan de Pump en Tap aan. In het ergste geval moet hij de Pump uitzetten en de Tap aanzetten.

De WebInterfaceTask stuurt elke minuut een update naar de browser met de resterende tijd van het wasprogramma. Ook stuurt het een bericht wanneer het een update krijgt van 1 van de sensoren via de listener interface.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Object | Type taak | Periode | Deadline | Prioriteit |
| Web interface | periodiek | 60000 MS | 100 MS | 6 |
| RotationSensor | periodiek | 500 MS | 10 MS | 0 |
| TemperatureSensor | periodiek | 500 MS | 10 MS | 1 |
| WaterLevelSensor | periodiek | 500 MS | 10 MS | 2 |
| Heating | asynchroon |  | 10 MS | 3 |
| Motor | asynchroon |  | 10 MS | 4 |
| Soap | asynchroon |  | 10 MS | 5 |
| UART interface | asynchroon |  | 10 MS | 6 |
| Signal led | geen |  |  |  |
| Door lock | geen |  |  |  |
| Pump | geen |  |  |  |
| Tap | geen |  |  |  |

### Interne taakstructurering

De samenvoeging van de taken van de vorige bladzijde leiden tot de volgende taken:

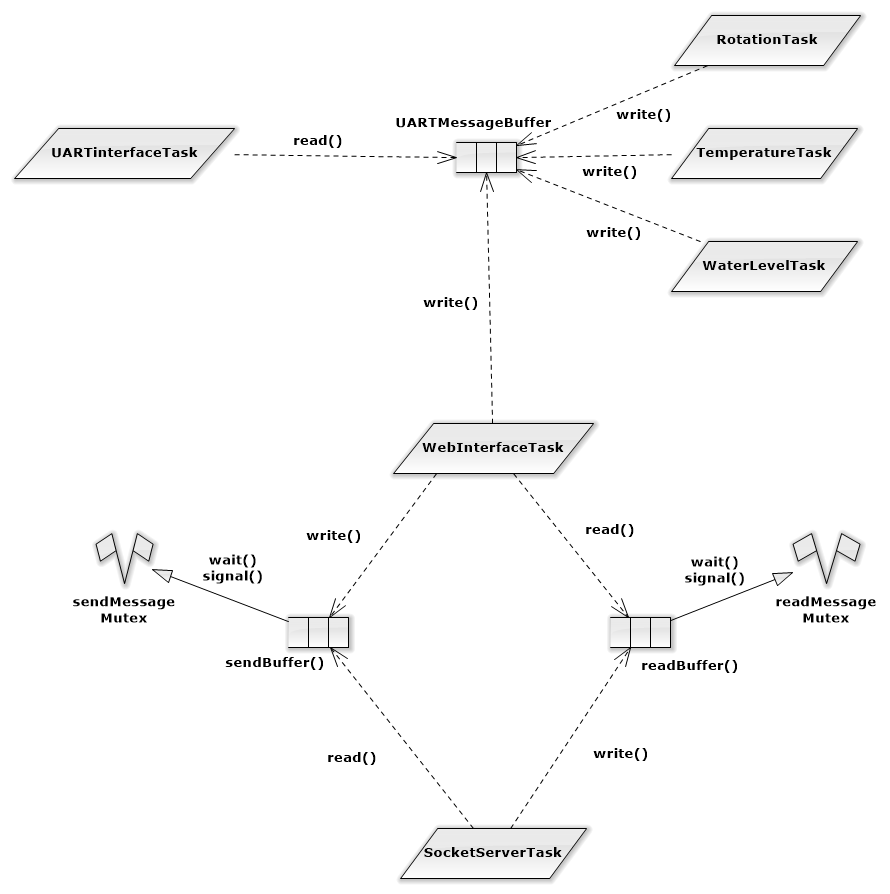
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Object | Type taak | Periode | Deadline | Prioriteit |
| UARTInterfaceTask | asynchroon |  | 10 MS | 0 |
| TemperatureTask | periodiek | 500 MS | 10 MS | 1 |
| MotorTask | periodiek | 500 MS | 10 MS | 2 |
| WashingProgramTask | periodiek | 1000 MS | 10 MS | 3 |
| WaterLevelTask | periodiek | 500 MS | 10 MS | 4 |
| WebInterfaceTask | periodiek | 60000 MS | 100 MS | 5 |

### Taak structuur diagram

Alle objecten zijn per taak ingedeeld in een taak structuur diagram. Hieronder is te zien hoe die taken zijn ingedeeld.



## 3.6​ Concurrency model



### ​WebInterfaceTask

Concurreert met:

SocketServerTask

Uitleg**:**

WebInterfaceTask en SocketServerTask schrijven en lezen allebei messages naar elkaar. Dit kan leiden tot ongeldige waardes als de een leest als de andere schrijft. Er kunnen ook meerdere messages tegelijk geschreven of gelezen worden.

Oplossing**:**

Twee channels voor beide richtingen van lezen en schrijven. Hier zit ook een mutex omheen om te voorkomen dat de een leest als nog niet alle berichten geschreven zijn door de andere.

Anders**:**

WebInterfaceTask heeft een klok om de mutex vrij te geven. Dit gebeurt voor 1 seconde. SocketServer heeft ook een klok om de mutex vrij te geven. Dit gebeurt voor 200 microseconden.

### ​UARTInterfaceTask

Concurreert met**:**

WaterLevelTask, HeatingTask, MotorTask, WebInterfaceTask

Uitleg**:**

WaterLevelTask, HeatingTask, MotorTask en WebInterfaceTask schrijven allemaal naar de UARTInterfaceTask. Deze moet alles goed lezen en zien wie wat heeft verstuurd. Zonder regeling kan er data verlies ontstaan.

Oplossing:

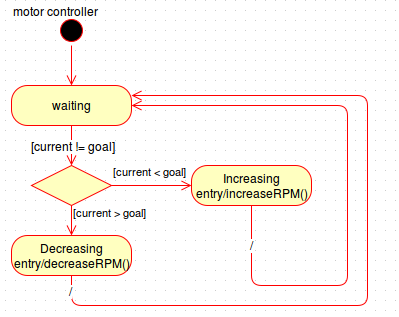
Channel. Zo wordt alle data van de taken gebufferd en kan de UARTInterfaceTask een voor een uit de buffer lezen.

## 3.7​ Dynamic model

### State Transition Diagram

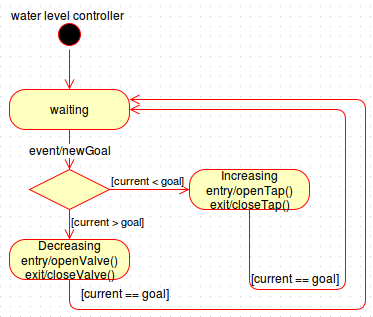
#### Motor Controller

De motor controller heeft 3 states. De ‘waiting’ state: daar wacht de controller zolang de huidige RPM (rotations per minute) niet gelijk is aan het doel. Als het niet gelijk is aan het doel dan gaat de controller naar de volgende state. Als de huidige RPM lager is dan het doel dan gaat de controller naar een ‘increasing’ state. Als de RPM al hoger is dan gaat die naar ‘decreasing’ state. Hij blijft deze cyclus maken tot ‘current’ gelijk is aan het doel. Dan komt die weer in een wacht staat terecht tot het doel aangepast word.



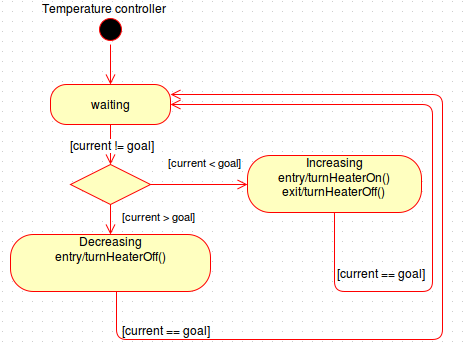
#### ​Water level controller

De water level controller heeft 3 states. Hij begint in de waiting state en wacht op een nieuw doel. Als het doel veranderd dan gaat de controller naar een nieuwe state afhankelijk of het doel lager of hoger is dan het huidige water niveau. Als het huidige water niveau lager is dan gaat de controller naar de increasing state. In die state zet die de kraan open, als ‘current’ gelijk is aan het doel dan wordt de kraan dicht gedaan. Als het doel lager is dan het huidige water niveau dan wordt de pomp aangezet. Die gaat uit zodra het water op het goede niveau zit.



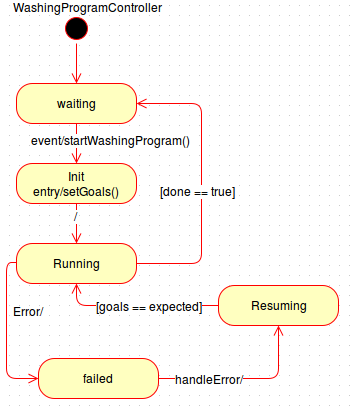
#### Temperature controller

De temperatuur controller heet ook 3 states. De wait state, daarin wacht de controller op een nieuw doel. Als die een nieuw doel krijgt dan gaat die naar de Increasing of Decreasing state afhankelijk of huidige temperatuur lager of hoger ligt dan het doel. Als de temperatuur hoger ligt dan het doel dan wordt de verwarming uitgezet. Hierna zal gewacht moeten worden tot het water afkoelt. Als de temperatuur lager ligt dan het doel, dan wordt de verwarming aan gezet tot het doel gehaald is. Als het doel behaald is wordt de verwarming uitgezet en keert de controller weer terug naar de waiting stand.



#### Washing Program Controller

Washing program controller begint in de waiting state. Daar blijft die in staan tot een start wasprogramma event ontvangen word. Dan in de init state worden alle doelen aan hun eigen controller doorgegeven. Daarna komt die in de running state. Wanneer in de running state een error voorkomt dan gaat die naar de failed state. Als de error afgehandeld wordt dan gaat de controller in de resuming state. Daar wordt alles weer naar het verwachte peil gebracht. Daarna gaat de controller verder in de running state. Als het programma klaar is dan gaat de controller terug naar de waiting state.



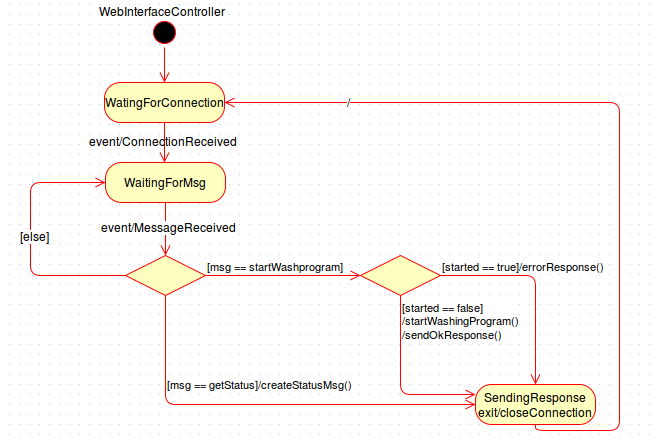
#### Webinterface Controller

De webinterface controller begint met het wachten op een verbinding. Zodra een verbinding is gemaakt, zal de controller wachten op berichten van degene die verbonden heeft. De mogelijke berichten die verzonden kunnen worden zijn te vinden onder het kopje “Communicatie protocol.” Hieronder worden er twee berichten besproken:

1. Het bericht dat het wasprogramma zal starten
2. Het bericht dat de status van de wasmachine opvraagt

Bij het eerste bericht wordt gekeken of het wasprogramma al gestart is of niet. Als het programma al gestart is, dan wordt er niks met het bericht gedaan. Wanneer nog geen programma gestart is, dan wordt het wasprogramma gestart en zal er een status update terug naar de client worden gestuurd.

Bij het tweede bericht wordt er een status update gemaakt. Die wordt doorgestuurd naar de client waarna de client bepaald wat er op de website te zien is.



## 3.8​ Communicatie protocol

De communicatie tussen de browser en het besturingsprogramma zal met JSON werken. Dit wil zeggen dat data in een JSON string wordt gestopt. Deze string kan bij de browser met behulp van Javascript verwerkt worden. Bij het besturingsprogramma zal deze JSON string met een JSON parser/generator verwerkt moeten worden.

Om berichten te kunnen sturen moet de gebruiker wel eerst inloggen. Hiervoor is een systeem bedacht dat het inloggen op een veilige manier afhandelt. De gebruiker moet op de website een naam en een wachtwoord invullen. Als de gebruiker vervolgens probeert in te loggen wordt het wachtwoord gehashed zodat het niet als tekst bij de server aankomt. Een bericht dat naar de server gaat ziet als volgt uit:

{

"**event**":"verify",

"**name**":"admin",

"**password**":45321329

}

Als de bovenstaande gegevens geldig zijn dan zal de server het onderstaande bericht terugsturen. Hierin staat of de verificatie gelukt is en als dat het geval is zal hij een hash meesturen die gebaseerd is op de tijd dat het bericht verzonden werd. Deze hash wordt bij elk ander bericht gebruikt om te kijken of het bericht wel verwerkt moet worden.

{

"**event**":"verify",

"**ok**":true,

"**hash**":998081953

}

Als het inloggen niet gelukt is zal de website het onderstaande bericht terugsturen:

{

"**event**":"verify",

"**ok**":false

}

Nadat de gebruiker succesvol heeft ingelogd stuurt de browser de onderstaande JSON string naar de websocket:

{"**event**":"statusUpdate", "**hash**": 998081953}

Na het ontvangen van de bovenstaande JSON string zal de websocket een status update terug sturen. Dit bericht zal later aan bod komen. Als uit de status update blijkt dat de wasmachine nog niet bezig is zal de browser “{"**event**":"getWashingPrograms", "**hash**": 998081953}” sturen. Na het ontvangen hiervan zal de websocket de wasprogramma’s terugsturen. Bepaalde opties zullen standaardwaarden zijn die door de gebruiker aangepast kunnen worden. Een voorbeeld van hoe dit bericht uit kan zien staat op de volgende pagina.

{

"**event**": "washingProgramList",

"**washingPrograms**": [

{

"**description**": "Washing program 1",

"**steps**": [

{

"**degrees**": 60,

"**rpm**": 1600,

"**water**": 60,

"**time**": 90,

"**rotationInterval**": 30

},

{

"**degrees**": 50,

"**rpm**": 1800,

"**water**": 60,

"**time**": 60,

"**rotationInterval**": 300

}

]

},

{

"description": "Washing program 2",

"steps": [

{

"**degrees**": 60,

"**rpm**": 1600,

"**water**": 60,

"**time**": 90,

"**rotationInterval**": 30

},

{

"**degrees**": 70,

"**rpm**": 1500,

"**water**": 70,

"**time**": 80,

"**rotationInterval**": 300

}

]

}

]

}

Op dezelfde manier als hierboven kan het gekozen wasprogramma naar het besturingsprogramma gestuurd worden. Dit bericht bevat, naast het startcommando, alle opties die bij het wasprogramma horen. Het bericht om het wasprogramma te starten is op de volgende pagina te vinden.

{

"**event**": "startWashingProgram",

"**hash**": 998081953,

"**washingProgram**": {

"**steps**": [

{

"**degrees**": 60,

"**rpm**": 1600,

"**water**": 60,

"**time**": 90,

"**rotationInterval**": 30

},

{

"**degrees**": 50,

"**rpm**": 1800,

"**water**": 60,

"**time**": 60,

"**rotationInterval**": 300

}

]

}

}

Om het wasprogramma te stoppen wordt het volgende bericht vanaf de browser naar de server gestuurd:  
{"**event**":"stopWashingProgram", "**hash**": 998081953}

Als de wasmachine al bezig is met het wasprogramma moet de website bepaalde data laten zien aan de gebruiker (een status update). De websocket stuurt het onderstaande bericht wanneer nieuwe data beschikbaar is. Dit bericht kan als volgt uit zien:

{

"**event**": "statusUpdate",

"**washingProgram**": {

"**description**": "Washing Program 1",

"**currentStep**": 1,

"**totalSteps**": 5,

"**data**": {

"**status**": 4,

"**currentDegrees**": 60,

"**currentRpm**": 1500,

"**currentWaterLevel**": 60,

"**timeLeft**": 1200

}

}

}

# ​4​ Realisatie

## ​4.1​ Hardware Problemen & Oplossingen

Bij het testen van de uart kwamen er meerdere problemen tevoorschijn. De uart gaf geen enkele reactie en er was geen duidelijke foutmelding. Door verschillende testen uit te voeren bleek dat het

probleem lag aan het DB103 bordje. Na het getest te hebben met een bordje van een ander team werkte het om onverklaarbare reden opeens. Het lag uiteindelijk dus niet aan de software maar aan de hardware.

## ​4.2​ Software Problemen & Oplossingen

### ​Hardware Controllers en WashingProgramController

Het grootste probleem dat zich optrad bij de WashingProgramController was dat deze eerst was geschreven met het idee dat de Start en Stop methoden vanuit de Controller zelf aangeroepen zouden worden. Hierdoor zaten bijvoorbeeld timer/wait methoden in die private waren van de klas zelf. Aangezien de Start methode wordt aangeroepen door WebInterfaceController leverde dit problemen op.

De oplossing was om de Start methode een flag te laten zetten en alle code in de main te hebben na een wait voor die flag in een while lus. Zo kan de taak zichzelf niet blokkeren maar heeft het hetzelfde effect als toen de code in de Start stond. De Stop methode roept alleen de stop methode aan van de scheduler, de afsluitende code staat aan het einde van de while lus in de main.

### WebInterfaceController

De webinterface controller had problemen met de hoeveelheid informatie die hij kenden en terug moest sturen. Zoals de tijd dat het wasprogramma bezig is. Een aantal implementaties die uiteindelijk veranderd zijn, is dat de controle de looptijd kan opvragen bij de WashingProgramController. Uiteindelijk zou een betere oplossing zijn als de washing program controller alle informatie over het huidige wasprogramma vast houd en dat de web interface controller die opvraagt om te sturen.

### WebServer, SocketServer en SocketMessage

Bij de drie onderdelen WebServer, SocketServer en SocketMessage waren een paar kleine problemen bij de SocketServer, die met een paar kleine aanpassingen snel verholpen werden, en één zeer groot probleem, wat later uitgebreid aan bod zal komen. Bij de SocketServer was één van de kleinere problemen dat op hetzelfde moment in een buffer geschreven en gelezen kon worden.

Om het probleem met de buffers op te lossen is er gebruik gemaakt van twee mutexen: één voor de sendBuffer en één voor de receiveBuffer. Zo zullen nooit twee threads op hetzelfde moment de buffer gebruiken.

Het zeer grote probleem dat al eerder werd vermeld was dat de Socket die verantwoordelijk is voor het sturen van berichten de error “corrupted double linked list” veroorzaakte. Dit probleem heeft lang geduurd en gebeurde elke keer op een willekeurig moment wanneer de multicaster berichten wilde versturen. Dit gebeurde echter niet bij alle berichten: soms was het bij het eerste bericht en soms was het pas bij het achtste bericht dat dit gebeurde. Om dit op te lossen is een programma geschreven wat achter elke ‘;’ een lijn print met de filenaam en het lijnnummer. Hierdoor was goed te zien op welk punt in het programma deze error optrad. Helaas hielp dit niet op het probleem op te lossen, aangezien hieruit bleek dat het programma crashte nadat hij een functie had uitgevoerd.

Uiteindelijk is het probleem (gedeeltelijk) opgelost door pointers naar SocketMessage objecten te vervangen met SocketMessage objecten die door middel van een kopie meegegeven worden aan andere functies. Nadat er een kopie gebruikt werd in plaats van een pointer deed de foutmelding zich slechts een enkele keer voor.

​

### Algoritmes

Authenticatie:

Bij de authenticatie wordt er vanaf de site een username en een wachtwoord verwacht. Het wachtwoord wordt met JavaScript gehashed in een numerieke code. De code wordt gegenereerd met het wachtwoord. Het algoritme loopt door alle karakters in de string en van die karakters pakt die het karakternummer. Na de hash 5 bits op te schuiven wordt de vorige hash er van afgetrokken en het karakternummer er bij opgeteld. Als laatste wordt de huidige hash met bitwise or 0 gedaan. Dit herhaalt zicht voor het aantal karakters in de string. Hierdoor kan het wachtwoord verstuurd worden zonder dat het direct als tekst over het internet wordt gestuurd.

Aan de kant van de server wordt gekeken of de user met het gehashte wachtwoord bestaat en klopt. De server stuurt dan een nieuwe hash terug die gebaseerd is op een const string “wakkaUbuntu9NEIN”. De nieuwe hash om mee te starten word gebaseerd op de huidige tijd van de server. Dan wordt er voor ieder character in " wakkaUbuntu9NEIN" over de hash gegaan. De hash word vermenigvuldigd met 101 en dan word het character waar die op dat moment is bij opgeteld. Die hash word in een verify bericht terug gestuurd en bij de server word de hash opgeslagen. De nieuwe hash word door de client gebruikt bij alle requests die wat met de wasmachine doen, zoals het starten en het stoppen van de wasmachine. De hash is niet nodig voor het opvragen van de status van de wasmachine omdat dat geen interactie heeft met de echte wasmachine alleen met de class die de status bijhoudt.

## 4.3​ Tests

### Systeem Test

Type:

Functionele Test

Doelvraag:

Werkt het sturen en handelen van ingegeven informatie op de website naar de wasmachine, en draait het wasprogramma helemaal daarmee?

Vereisten:

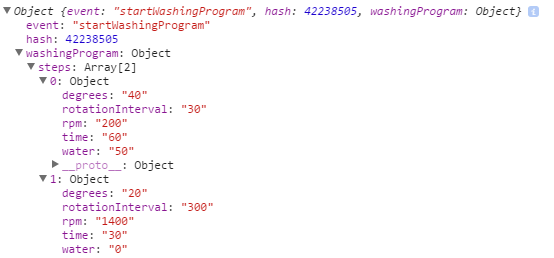
Werkende webserver, socketserver en wasmachine emulator

Uitvoering:

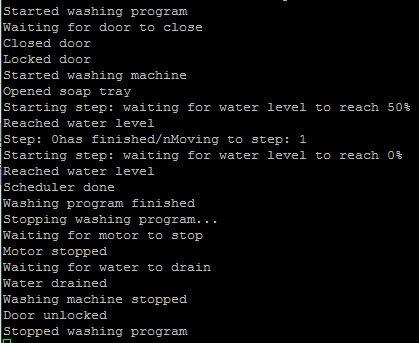
1. Voer informatie in op de website en start het wasprogramma
2. Bekijk binnenkomende messages op socketserver output
3. Bekijk resultaat op wasmachine emulator
4. Wacht tot wasprogramma klaar is
5. Bekijk binnenkomende messages op socketserver output
6. Bekijk resultaat op wasmachine emulator

Resultaat:

Ingegeven informatie wordt door gegeven aan de socketserver en wasprogramma wordt goed gestart en gestopt.







Conclusie:

Na het opgeven van de informatie wordt alles opgeslagen in een wasprogramma object. Deze wordt uitgelezen door de WashingProgramController die via een WashingProgramScheduler bepaald welke goalStates er op dat moment gezet moeten worden. De WashingProgramController geeft de goalStates aan de hardware controllers zodat de wasmachine onderdelen hun states updaten. Zo blijft de hardware het wasprogramma volgen. De WebInterfaceController update de webserver met de states van de hardware zodat de gebruiker kan zien hoe ver het programma is. Nadat het wasprogramma klaar is stopt de WashingProgramController de wasmachine en zal hij wachten op een nieuw programma.

### Security Test

Type:

Functionele Test

Doelvragen:

Werkt het security systeem op de website?

Vereisten:

Werkende webserver en socketserver

Uitvoering:

Scenario 1

1. Voer goede username en goede password in en log in
2. Bekijk de output op socketserver
3. Verifieer dat met elke request een hash wordt mee gestuurd, en deze wordt afgehandeld

Scenario 2

1. Voer niks in en probeer in te loggen
2. Kijk of er een error optreedt die aangeeft dat username en password verkeerd of niet ingevuld zijn

Resultaat:

Scenario 1:

Er komt een console output dat aangeeft dat inloggen succesvol is (ok = true). Elke request stuurt een hash mee.





Scenario 2:

Er komt een console output dat aangeeft dat inloggen niet succesvol was (ok = false).



Conclusie:

Door een username/password systeem op de website is deze beveiligd. Gebruikers kunnen op dit moment geen eigen username en password aan maken maar de hardcoded admin user werkt. Elke request verstuurt een hash die de socketserver checked om te zien of de connectie geldig is. Dus het security systeem werkt naar behoren.

### Stroom Uitval Test

Type:

Functionele Test

Doel:

Handelt het programma een (gesimuleerde) stroomuitval goed af?

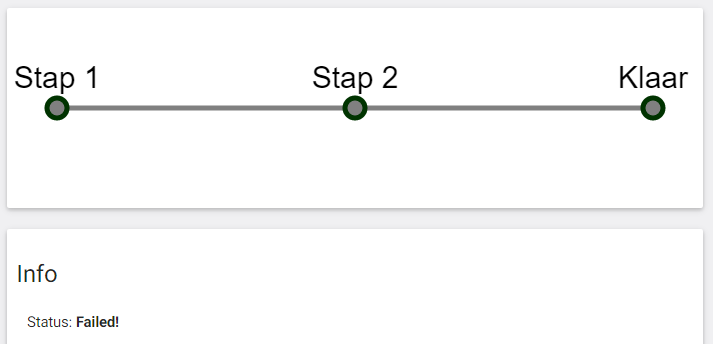
Vereisten:

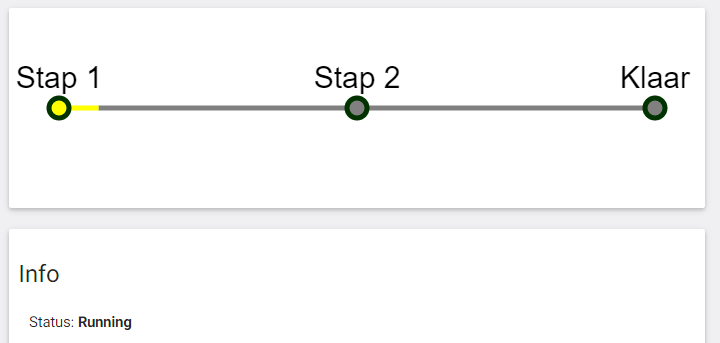
Werkende wasmachine emulator

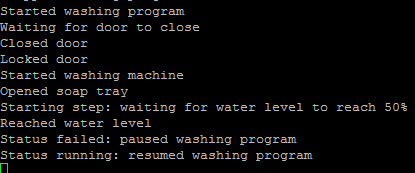
Uitvoering:

1. Start een wasprogramma
2. Druk terwijl het wasprogramma draait op de stroom uitval simulatie knop
3. Verifieer of het wasprogramma na de gesimuleerde stroom uitval verder gaat

Resultaat:

Het wasprogramma gaat verder na de gesimuleerde stroom uitval. 





Conclusie:

Door een pause in de WashingProgramScheduler aan te roepen zodra de Wasmachine status FAILED is wordt bijgehouden hoe ver een wasprogramma was voordat de stroomuitval gebeurde. Nadat stroom terug is en de wasmachine status terug naar RUNNING gaat unpaused WashingProgramController de scheduler en worden gelijk de goalStates geupdate naar waar de scheduler was voor de stroomuitval.

# ​5​ Evaluatie

In dit project zijn een aantal punten naar voren gekomen die bij nader inzien anders gedaan hadden kunnen worden. Hieronder staan een aantal punten met een eventuele oplossing voor dat punt.

**Klassen**

Niet alle klassen stonden goed in het klassendiagram. Zo waren er nog een aantal punten die niet in het diagram stonden maar die wel noodzakelijk waren voor het eindproduct. Tijdens het bespreken van de klassen was dit helaas niet naar voren gekomen. Tijdens de implementatie kon het niet meer zo makkelijk aangepast worden. Dit kan wellicht beter door tijdens het ontwerpproces de bedachte structuren al vast te testen.

**Programmeren**

Het programmeren ging wat moeizaam omdat er maar één week was ingepland hiervoor. De rest van de weken is meer tijd besteed aan de documenten die nog gemaakt moesten worden. Het is daarom beter voor in de toekomst om eerder te beginnen met het testen van bepaalde ideeën voordat ze in de diagrammen worden gezet.

**Documenten**

Omdat alles van te voren werd uitgedacht kwamen er een aantal ontwerpfouten naar voren. Deze fouten hebben veel tijd gekost tijdens het corrigeren ervan. Veel functionaliteiten konden hierdoor niet getest worden. Tijdens de uitvoering moest uiteindelijk nog veel aangepast worden in de documentatie. Voor in de toekomst is het verstandiger om eerder hieraan te beginnen zodat er meer tijd overblijft voor het testen.

**Samenwerking**

De samenwerking verliep vrij goed, alleen soms wel erg zelfstandig. Sommigen zijn veel bezig met hun eigen deel, waardoor zij niet precies weten hoe ver de andere delen zijn. Het is daarom verstandiger om voor de volgende keer Trello intensiever te gebruiken voor heel het project.

**Hardware**

Het compileren op de Raspberry Pi ging nogal langzaam. Dit werd opgelost door een tool te gebruiken die alle dependencies in de Makefile zet(Brunhof, 2015). Verder werd veel tijd gespendeerd aan het uitvogelen waarom de uart steeds verkeerde berichten terug gaf als antwoord.

Dit bleek door een defecte microcontroller te komen. Hierdoor is veel tijd verspild die beter aan de “Should Haves” van de MoSCoW besteed kon worden.

# 6​ Conclusies en aanbevelingen

## 6.1​ Conclusies

Het project verliep vlot. Alle Must Haves in de MoSCoW zijn gehaald met nog een redelijke time frame over. Door een aantal problemen was er geen tijd meer over voor Should Haves. Veel van deze problemen hadden te maken met de hardware die niet volledig werkte. Het vinden dat de oorzaak de hardware was koste veel tijd. Ook bleek dat er van de originele modellen af werd gestreden door verschillende situaties waar geen rekening mee was gehouden tijdens het modelleren. Dit zorgde voor veel veranderingen in de modellen gedurende het project.

## 6.2​ Aanbevelingen

Mocht het project worden voortgezet of herhaald worden in de toekomst, hier zijn een paar aanbevelingen om het project beter te laten verlopen.

1. Test de hardware voor het beschikbaar stellen aan project leden. Er was veel tijd verloren gegaan door debuggen op een kapotte microcontroller totdat het aan het licht kwam.
2. Geef een briefing over de software voordat alle modellen gemaakt worden. Het is handig delen van de gegeven software te testen om te helpen bij het modelleren.

# 7 Bronvermeldingen

Wensink, M. (2015). Beschrijving wasmachine-emulator. Geraadpleegd op 3 december 2015 op

<https://cursussen.sharepoint.hu.nl/fnt/35/TCTI-V2THO6-14/Studiemateriaal/Beschrijving%20wasmachine-emulator.pdf>

Dijk, S. van. (2015). Handig! alle wasprogramma’s voor een wasmachine bij elkaar. Geraadpleegd op

<http://www.greenem.nl/wasprogrammas-voor-een-wasmachine/#1440591205639-fb4d4437-9ec5>

Donahoo, M., & Calvert, K. (2001) TCP/IP Sockets in C: Practical Guide for Programmers. Geraadpleegd op 3 december 2015 op

<http://cs.ecs.baylor.edu/~donahoo/practical/CSockets/practical/>

Raspberry. (2014a). *Raspberry Pi Model Specifications.* Geraadpleegd op <https://www.raspberrypi.org/documentation/hardware/raspberrypi/models/specs.md>

Raspberry. (2014b). Setting up an apache web server on a raspberry pi*.* Geraadpleegd op <https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md>

Brunhof, T. (2015). Makedepend(1) - Linux man page geraadpleegd op 15 januari 2016 op

<http://linux.die.net/man/1/makedepend>

# 8​ Bijlagen

* Bijlage A: MoSCoW analyse - 1 pagina
* Bijlage B: Interview vragen - 1 pagina

## Bijlage A: MoSCoW analyse

**Must have**

**Documentatie:**

- Technisch verslag

**Operating system:**

- Wasprogramma’s starten en stoppen

- Wasprogramma’s in stappen uitvoeren

- Wasprogramma’s opslaan in bestanden

- Bij stroomuitval het wasprogramma hervatten wanneer de stroom weer terug is

**Websocket server:**

- Verbinden met de browser

- Login met naam en wachtwoord

- Communicatie protocol

- Berichten ontvangen en versturen

**Webserver:**

- Bestanden hosten

**Webpagina:**

- Verbind met de websocket server via Javascript websockets

- IP adres van websocket server handmatig instelbaar

- Start en stop knop

- Wasprogramma uit lijst selecteren

- Handmatig wasprogramma aanpassen

- Status van het draaiende wasprogramma laten zien

- Status constant updaten via websocket

**Modellen:**

- Use case diagram

- Activity diagram

- Klassendiagram

- Taakstructurering

- STD van de controllers

- Beschrijving van communicatie protocol tussen browser en operating system

**Should have**

- Websocket server laat alleen verbindingen binnen hetzelfde netwerk toe

- Geluid afspelen als het wasprogramma klaar is

**Could have**

-

**Would have**

-

## Bijlage B: Interview vragen

1. Welke informatie moet de gebruiker allemaal kunnen zien op de website?
2. Welke acties moet de gebruiker allemaal kunnen uitvoeren vanaf de website?
3. Hoeveel gebruikers kunnen tegelijk de wasmachine besturen?
   1. Wordt een gebruiker geblokkeerd als iemand al bezig is de wasmachine te besturen?
4. Welke gegevens moeten getoond worden op het lcd-scherm van de wasmachine?
5. Wat zijn de belangrijkste punten waar de lay-out van de website aan moet voldoen?
6. Moet de website de mogelijkheid hebben verschillende wasmachines aan te sturen als je de adressen mee geeft?
7. Wordt er van uit gegaan dat er altijd hetzelfde wasmiddel wordt gebruikt of is dit ook een kiesbare optie?
8. Moet de site zichzelf updaten of wordt er verwacht dat de gebruiker zelf herlaad?
9. Moet de verbinding met de website beveiligd worden?
10. Wat moet er gebeuren als de stroom uitvalt? (opvangen met accu?)
11. Wat moet er gebeuren als een deel van de wasmachine niet meer werkt?
12. Kan de gebruiker alleen kiezen uit vooraf gedefinieerde wasprogramma's of ook alles apart?
13. Heeft de gebruiker de mogelijkheid om zelf een wasprogramma in te stellen?
14. Welke temperaturen moet de wasmachine ondersteunen?
15. Welke ronddraai snelheden moet de wasmachine aankunnen?