Struers Remote Connection - Functional Prototype

Thomas M. Mortensen

Kongens Lyngby 2016

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstrakt

I dette projekt vil det blive forsøgt at lave en ny remote løsning til Struers udstyr, der kan bruges til eksempelvis: remote service, logning af data, firmware opdatering og styring af udstyret. Der vil primært blive kigget på en autonom funktionalitet til firmware opdatering, da det ville give mulighed for at lave ny funktionalitet til eksisterende udstyr. Dette vil blive gjort ved hjælp af en embedded computer, der kobles til Struers udstyr. Denne opkobling vil ske gennem en seriel forbindelse og gøre brug af en protokol Struers har udviklet til kommunikation gennem netop denne forbindelse. Herfra vil den embedded computer kunne sende og modtage data gennem et tilkoblet 3G modul, der kommunikerer med en cloud tjeneste. Denne tjeneste vil så kunne sende og modtage data fra et iOS device. Det vil betyde at slutbrugeren til denne løsning vil sende og modtage al kommunikation igennem en dedikeret Struers Service applikation til iOS.

Dette gøres for at få et samlet overblik over, hvordan en løsning kunne se ud for Struers fremtidige planer om connectivity i deres udstyr, men også om man kunne lave noget på eksisterende udstyr. Samtidig vil man også gerne se på, hvad der er af fordele og ulemper i forhold til de forskellige dele, der indgår i systemet

Abstract (English)

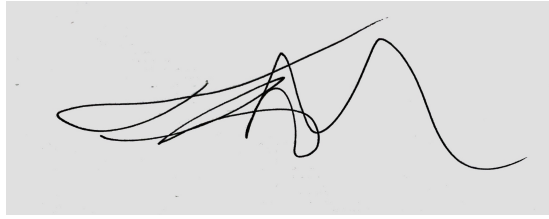
In this project it will be attempted to create a remote solution for Struers equipment that might provide opportunities like: remote service, logging of data, firmware update and controlling the equipment. The focus will primarily be on an autonomous function for firmware updating. This would give the opportunity to create new functionality for the old equipment. This is done with help from an embedded computer connected to Struers equipment. This connection will be made with a serial connection and take advantage of a Struers developed communication protocol meant for communication through this particular connection. From here on the embedded computer will be able to send/receive through a connected 3G connection that communicates with a cloud service. This service should be able to send/receive data from an iOS device. This means the end user should be able to send and receive data through a dedicated Struers Service application for iOS.

This is done to get a collective overview of how a solution could look like for Struers' future plans on connectivity in there equipment, but also to see if something could be implemented on existing equipment. At the same time the company wanna know what advantages and disadvantages relating to the different components in the system.

Introduktion

Dette er et afsluttende diplomingeniørprojekt lavet i samarbejde mellem DTU Compute og Struers ApS, og forsøger at opfylde kravene for at opnå en B.Eng. Formålet med projektet er at undersøge hvad der skal til for at implementere en Internet of Things (IoT) løsning på Struers udstyr. Dette set i forhold til både, nuværende situation, fremtid, hardware, software, etik og juridiske aspekter. Udstyret/softwaren brugt til opgaven er leveret af Struers ApS. Opgaven består af denne rapport inklusive bilag plus kode, som er afleveret elektronisk sammen med opgaven beskrevet i bilag C på side 59.

Lyngby, 13-Juni-2016



Thomas M. Mortensen

Anerkendelser

Jeg vil gerne takke min vejleder Ken Carlsen hos Struers, for både at hjælpe eller henvise til den rigtige hjælp når dette har været nødvendigt. Samtidig har jeg også fået frihed til selv at løse diverse problemstillinger og strukturere min hverdag.

Meget af omtalte hjælp har jeg måttet hente i Software teamet hos Struers, og her fik jeg nærmest en vejleder mere i Niklas Bo Greve. Samtidig har både Erik Madsen og Jan Madsen været behjælpelig med en del af analysen på lidt ældre Stuers udstyr, og kommunikationen med dette. Samtidig har jeg trukket en del på Marek Paucek i forbindelse med firmware opdatering. Marek Hluchnik har været en stor hjælp i forhold til udstyrets bootloader funktionalitet. Den sidste mand i teamet Jiri Sedlacek har givet gode inputs og nye perspektiver, når vi har gennemgået status på projektet.

Faktisk er jeg nødt til at takke alle i DE hos Struers for at vise interesse og komme med input til forskellige aspekter af projektet.

Min vejleder Stig Høgh fortjener også tak. Han har givet rigtig gode indspark til projektet både i startfasen og i slutfasen, men stadig givet mig frihed til selv at komme med tanker og idéer.

Jeg vil i samme omgang takke mine forældre for at støtte mig i min beslutning om at påbegynde en helt ny uddannelse på dette tidspunkt i mit liv. Uden den opbakning er jeg ikke sikker på, at jeg havde taget springet ud i det.

Indhold

Kapitel 1

Indledning

Denne rapport indeholder dokumentationen, der er udført i forbindelse med mit afsluttende diplomingeniørprojekt. Dokumentationen består af en projektplan, og efterfølgende en beskrivelse af alle delelementer, der har været undersøgt, og/eller implementeret i forbindelse med projektet. Forventningen er sat til at få en autonom firmware opdatering til at fungere. De resterende ting for at lave et system med remote connection ser man gerne belyst/implementeret, men ikke som en nødvendighed for opgaven.

1.1 Problemformulering

Jeg har inden igangsættelsen af projektet forsøgt at finde ud af hvilke områder Både virksomheden og jeg selv, syntes kunne være interessant at få belyst i forhold til en fungerende prototype med remote connection. Efter megen overvejelse kom jeg frem til disse 3 overordnede spørgsmål:

- Hvordan skal et eventuelt system se ud for at opfylde opgaven?
- Hvilke aspekter skal undersøges/overvejes i forhold til de ting, der skal indgå i systemet?
- Hvor moden er virksomheden i forhold til remote connection?

1.2 Projekt plan

Til at starte med, har jeg løseligt lavet en projekt plan, der er forsøgt overholdt så vidt muligt. Dog er testfaserne ikke kørt så stramt, men mere som en del af selve implementeringsprocessen.

- 2. februar - analyse
- 15. februar - design
- 1. marts - implementering af firmware update
- 1. april - test af firmware update
- 15. april - implemetering af cloud på prototype
- 1. maj - test af cloud
- 10. maj - simpel frontend implementering
- 20. maj - test af simpel frontend
- 23. maj - 13 juni - dokumentation

Kapitel 2

Remote Connection (IoT)

Vi fandt hurtigt ud af, at vores remote connection egentlig nok var bedre betegnet med et andet ord - nemlig Internet of things (IoT). IoT er ved at være et varmt emne i mange virksomheder i dag, og det forventes at 212 milliarder enheder vil være forbundet inden 2020 [1]. Muligheden for at få et data feed fra enheder giver virkelig mange muligheder, såsom: fjernkontrol, maskiner der lærer fra hinanden, produktudvikling baseret på dataopsamling osv. Dog virker service, efter at have læst en del artikler, til at være den vigtigste parameter.

2.1 IoT løsning

IoT er ikke en ny term i Struers. Det første projekt man har haft oppe at vende, har været omkring Machine 2 Machine (M2M). Det virker som om at det grundliggende koncept egentlig var det samme dengang. Dog har en del tekniske termer fra den tid ændret sig, og nogle teknologier, som til gengæld er mere aktuelle i dag, fandtes ikke dengang. Projektet blev dengang nedlagt i koncept fasen, men nu virker det som om, at mange større industrier kigger hen mod denne forbundne verden på forskellig vis. Dette har bevirket, at man i Struers synes det er værd at kaste et blik på teknologien bag, for at finde ud af om man kan bruge det i virksomhedens strategi.

Da det i dag ikke kun er omtale af maskiner, men i realiteten alt udstyr med en internet forbindelse, hører man i dag mest om IoT kontra M2M. Jeg vil derfor prøve at belyse de fordele og ulemper, der er i at forbinde sit udstyr med skyen og lave IoT udstyr.

Udnyttelse af IoT

IoT bliver tit beskrevet som en sammensmeltning af den digitale og den fysiske verden, og dette ændrer den måde man driver forretning på i mange forskellige virksomeder på det globale plan. Der er en ting, disse virksomheder har til fælles. De har indset, at IoT ikke handler om udstyr. Det handler om service.

Det handler derfor om at forvandle sin produkt baserede forretning til en serviceforretning. Det handler om at ændre fokus fra udelukkende produktudvikling, til at have mere fokus på den service produkterne kan give.

Salg af "engangs produkter"bliver hurtigt forældede, når man har internetforbundet udstyr, samtidig med at kunderne forventer virksomhederne, kontinuert tillægger ny værdi og leverer nye oplevelser over tid ved hjælp af forbundne enheder. Samtidig forventer kunder, at du ved hvordan og hvorfor man fikser opståede problemer med det samme [2]. De største operationelle spørgsmål i forbindelse med disse altid furbundne produkter er:

- Hvad skal være strategien i forhold til on-board på nye enheder, levering af netværkstjeneste og brugspolitikker?
- Hvordan vil du monitorere og optimere service performance?
- Hvordan skal man kunne diagnosticere og løse service problemer remote?
- Hvordan skal skalerbarheden være? Skal det være 10.000 kunder eller skal det være hele verden?

For at få en succesfuld IoT forretning er der fire hovedkrav, der skal overholdes:

- Udstyr med indlejrede tilslutningsmuligheder.
- Netværk til at forbinde disse enheder.
- Applikationer og IT-systemer til at styre operationerne.
- En IoT service platform, der gør dig i stand til at udnytte real-tids interaktion med kunderne.

Big data

I forbindelse med IoT snakker man meget om begrebet Big data. Big data er samtidig blevet synonym med "big business"ifølge en række undersøgelser. Disse konkluderer at datadrevne virksomheder i gennemsnit er 5 pct. mere produktive, og 6 pct mere profitable end virksomheder der slet ikke anvender Big data.

Denne forskel kan realiseres i alle led i en produktionsvirksomheds værdikæde. Det kan være i alt fra design af nye og bedre produkter ved hjælp af kunde og brugerdata over supply-chain management i forsyningskæden til forbedret performance og risikostyring i produktionen. Samtidig får man et kæmpe potentiale i forhold til marketing og salg af virksomhedens produkter.

Studier viser at små og mellemstore virksomheder tager Big data værktøjer i brug, når de er modnede og tilgængelige i virksomhedens udstyr. Samtidig er virksomhederne tilbageholdende, når det kommer til implementering af løsninger, der kan give et samlet overblik over virksomhedens samlede datastrøm. Dette forventer man dog vil aftage med tid, i takt med at nye og billigere systemer til analyse af datastrøm kommer til [3].

IoT standarder

En stor forhindring for udviklingen af IoT er manglen på fælles protokoller og ikke-proprietære standarder. På nuværende tidspunkt viser listen af netværkstandarder som: ZigBee, Z-Wave EnOcean, Bluetooth LE eller SigFox og Lora, at der er alt for mange standarder, som ikke kan køre sammen.

Det virker dog som om at udviklingen er ved at vende med ZigBee 3.0, der nærmer sig Googles Thread, selvom disse to stadig bliver udfordret af Bluetoothkonsortiet, som arbejder på Bluetooth mesh standarden. Også Wi-Fi Alliancen arbejder på en standart, de kalder for Halow [4].

En anden artikel fra version2, går så vidt som at påstå, vi skal glemme IoT standarder til industrien [5]. Dette er under betragtning af at industrierne typisk opererer med utrolig lange livscyklusser. Samtidig kan den teknologiske udvikling gøre at standarderne hurtigt bliver forældede eller utilstrækkelige. Derfor er det nemmere bare at finde den ønskede hardware, og efterfølgende tilpasse den til systemet.

Al den usikkerhed og uenighed om standarder, gjorde også at vi valgte genbrug af SCP, som vores kommunikationsform kombineret med API til den cloud løsning,

som vi nu kom til at benytte os af.

2.2 Fordele ved IoT

Der er rigtig mange fordele ved IoT, hvilket måske også er et problem. Dette vil jeg komme nærmere ind på i afsnittet om ulemper 2.3 på side 8. De følgende undersektioner vil dog beskrive nogle af de fordele, jeg har fundet frem til.

Udvikling af nye muligheder til kunderne

Her har jeg listet et par af disse nye muligheder kraftigt inspireret af Cisco's whitepaper [2].

- **Tilbyd nye services**, som GM har gjort med ders biler. De har i dag fået underholdningstilbud som Netflix og Spotify. I Struers regi kunne man måske forestille sig at kunderne kunne bestille forbrugsvarer direkte på udstyret, eller at udstyret selv kunne bestille det.

- **Udvid mulighederne på eksisterende produkter**. Dette kunne f.eks. være en kundeportal til kunder med meget Struers udstyr, der kan give dem overblik over realtids drift og historik over brugen af udstyret.

- **Find helt nye måder at lave forretning på**. En mulighed kunne være at kunderne kunne lease deres udstyr, og så betalte for det aktuelle forbrug på udstyret.

Continuois Engineering

Continous engineering giver mulighed for konstant at tilpasse sine produkter ud fra præmisser som kundeindsigt i forhold til brug, købsvaner med mere og forbedring af innovationshastighed og udviklingseffektiviteten baseret på data-opsamling [1].

Forbedret service

I Cisco's artikel [2] foreslår de at automatisere processer og signifikant reducere respons tid med følgende punkter:

- Kør remote diagnostik på udstyret, da det vil tillade øjeblikkelig respondering på hændelser.
- Sørg for at have en proaktiv service. Jo mere udstyr, der bliver forbundet, jo mere kan der analyseres på data, som i sidste ende kan give virksomheden advarsler om mulige hændelser før de sker. Dermed kan man mindske nedetiden hos kunden.

Lær at forstå din forretning

Samme artikel forsøger at give et bud på, hvordan man lærer at forstå sin forretning i realtid, og dermed lave præcise estimater af, hvor den er på vej hen, og her er specielt to punkter relevante for Struers:

- Skræddersy dine produkter så kunden får en mere personlig oplevelse ved hjælp af dataanalyse om forbrug med mere.
- Forstå hvordan dine kunder bruger udstyret. Dette kunne være helt ned til overvågning af hvilke procedure kunden kører mest eller hvilke menuer kunden bruger mest.

Mange flere

Jeg har kun nævnt et par af de mange muligheder IoT, og mange flere har vi garanteret slet ikke udtænkt på nuværende tidspunkt. Bare tænk over hvad internettet har gjort for PC'en og for vores smartphone. Denne connectivity har givet os muligheder vi aldrig havde drømt om, så det er egentlig kun fantasien, der sætter grænsen.

Herudover har jeg med vilje ikke beskrevet hverken muligheden for cloud storage eller Firmware Over The Air (FOTA), da disse punkter er så essentiel en del af systemet, at de hver har fået et kapitel til mere detaljeret beskrivelse.

2.3 Ulemper ved IoT

Den største ulempe er nok, at der faktisk ikke er en egentlig begrænsning på systemet. Man kan altid tilføje en ny sensor i hardwaren eller ændre logikken i softwaren, og nye idéer til sammenføring af data kan nemt tilføjes.

Alle disse muligheder har gjort, at det er svært at danne en overordnet strategi. Vi har i udviklingsafdelingen fremlagt idéen for de andre afdelinger i virksomheden. Dog har vi haft utrolig svært ved at få feedback på, hvilke muligheder de forskellige afdelinger egentlig kunne tænke sig i sådan et system. Det har virket som om at den endeløse strøm af muligheder, egentlig har bevirket at man som afdeling har haft svært ved at forholde sig til, hvad man egentlig ville prioritere højest.

Man er derfor nok nødt til at sætte sig ned i de forskellige afdelinger og lave en prioriteret liste af muligheder, og så må den ansvarlige afdeling for udvikling så samle alle inputs og lave en overordnet strategi for den endelige prioritering. Dog skal man være opmærksom på at denne bør kunne modificeres undervejs, hvis man pludselig kommer i tanke om en idé, som bør have høj prioritering.

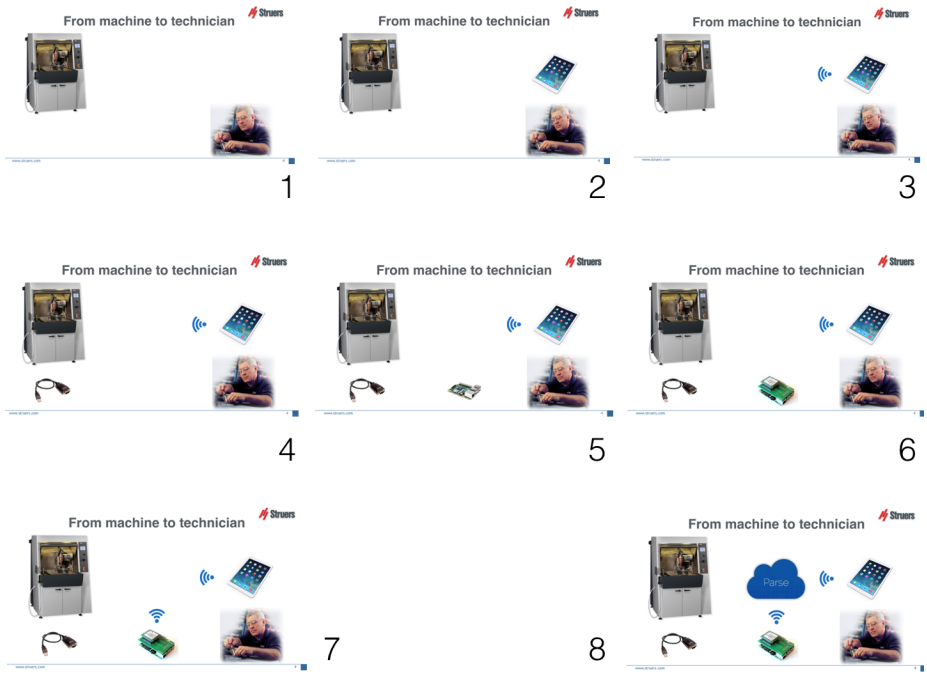
Generelt set burde virksomheden inddrage alle afdelinger i startfasen til et sådant projekt, så man får alle muligheder belyst på en sådan måde at man kan forberede både hardware og software på en ordentlig måde. På den måde kan man forhåbentlig slippe for at skulle ændre for meget på platformen senere hen, da man har forsøgt at være lidt forudseende. Dette kan ikke klares af udviklingsafdelingen alene, da den i sig selv ikke har dyb indsigt i hvordan f.eks. forbrugsvarer bliver brugt, eller hvad den afdeling kunne tænke sig af data.

2.4 System arkitektur for Prototypen

For at give et indblik i systemets arkitektur, vil jeg i det følgende forklare om opbygningen i den valgte løsning. Herunder vil jeg forsøge at belyse hvilke fordele og ulemper, der er i de foretagede valg omkring løsningen. [6]

Hardware opbygning

Allerede under mit praktikophold i virksomheden havde vi lavet et Proof Of Concept (POC). Vi valgte at forsøge os med den samme arkitektur, da vi for-



Figur 2.1: Systemet vist i 8 trin

holdsvis hurtigt ville kunne genbruge dele af denne. På længere sigt ville man nok undersøge muligheden for, at få hardwaren indbygget som en del af selve Struers udstyret. I figur 2.1 er systemet præsenteret i 8 trin. Disse trin giver det overblik, som er er beskrevet i detaljer i [7]. Jeg har dog beskrevet alle trin kort i det følgende og indikeret dette med tilhørende nummer på figuren.

1. Dette trin viser det overordnede problem. Hvordan kobler man en service teknikker og Struers udstyr sammen.

2. Vi starter med en frontend app til service teknikkeren. I dette tilfælde en tablet.

3. Denne tablet har inbygget Wi-Fi og 4G/LTE forbindelse. Dermed kan vi sende data Over The Air (OTA).

4. Struers udstyret skal ud med sin opsamlede data, og her har vi sat en seriel forbindelse på, da PCB'et inden i ikke har hukommelse eller hardware til at forbinde direkte til tabletten. Derfor er dette et nødvendigt onde.

5. Fordi vi netop ikke kan ændre så meget på det eksisterende PCB, valgte vi at forbinde Struers udstyret med en Raspberry Pi (vores prototype) gennem den serielle forbindelse. Prototypen kunne så sende og modtage telegrammer til og fra maskinen.

6. Vi fandt et kit, som simpelthen er et skjold, der kan monteres et 3G modem på og tilkobles en antenne. Skjoldet var lavet til at nemt montere på enten en Arduino eller en Raspberry Pi.

7. Dette kit havde kode og en applikation til en ppp forbindelse. Koden skulle dog modificeres, men gjorde det muligt at sende/modtage data Over The Air (OTA).

8. Den sidste ting we havde brug for var en cloud baseret løsning, der kunne kontrollere og gemme den data, der skulle sendes mellem tabletten og Struers udstyr.

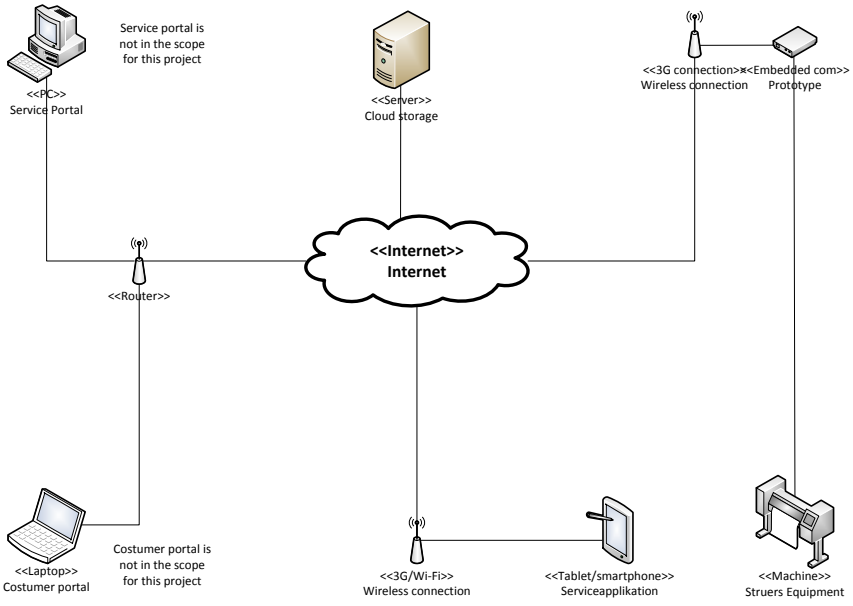
Ud over opbygningen i trin har vi forsøgt at lave et lidt mere komplet overblik over systemet struktur i et netværks diagram 2.2 på næste side. I diagrammet ses det, hvordan skyen er tiltænkt som det centrale omdrejningspunkt i systemet. Her kobler vi så prototypen til og vores Frontend service applikation. Vi har også prøvet at tænke systemet en msule videre, med en service og kundeportal, som dog ikke er tiltænkt som en del af dette projekt. Vi lavede i en tidlig fase et håndtegnet udkast til systemet, der kun koncentrerede sig om selve opgaven, dette ses i B.2 på side 55.

Sikkerhed i OTA

Sikkerheden i Over The Air (OTA) forbindelser er tit håndteret på forhånd. I vores prototype har vi brugt en 3G forbindelse, og her er det normal praksis at bruge en Authentication Key Agreement (AKA) protocol [8]. Denne protokol skal sikre en sikker dataoverførsel med blandt andet en kraftig hashfunktion.

Operativsystem og software

Det grundliggende koncept havde vi allerede fra min praktikperiode, men denne løsning var utilfredsstillende specielt i forhold til firmware opdateringen. Selve opdateringen skulle manuelt startes af en service teknikker. Samtidig tog det flere minutter blot at tjekke firmwaren før vi kunne starte en meget langsommelig opdateringsproces.



Figur 2.2: Netværksdiagram, der giver overblik over strukturen

Dette fik os til at kigge på hvilke forbedringer, vi kunne lave for at forbedre systemet. Operativsystemet vi før havde benyttet, var Raspian, som er en Linux distribution, der er lavet ret nem at installere på en Raspberry Pi. Derfor valgte vi at fortsætte med dette. Det var godt at have en nem installation, hvis prototypen blev en succes, og man ville forsøge sig med flere prototyper.

Den software vi havde udviklet var dog ikke særlig tilfredsstillende. Den var udviklet i python, og ulempen i dette er at python er et fortolket sprog. Et fortolket sprog oversætter koden ved runtime, og eksekverer så det oversatte [9]. Det betød at vores performance blev alt for dårlig. Hver gang en firmwarefil skulle køres igennem en checksum tog det omkring 3 minutter, hvilket var uacceptabelt.

Vi besluttede derfor at finde et sprog, der havde en bedre performance, men samtidig også kunne køre på en Windows Embedded platform, hvis man nu gerne ville skifte operativ system. Her har virksomheden nemlig ingen fast strategi, men bruger Windows Embedded til en del ting i forvejen, så det kunne være en mulighed.

Valget blev derfor C#, hvilket kan køre på både Windows og Linux, hvis man bruger Mono, der er en software platform designet til at lave cross platform applikationer [10], hvilket også gav os mulighed for at bruge en del af .NET's inbyggede funktioner.

2.5 Køb eller gør det selv

Ved udviklingen af en computerplatform til en embedded applikation, kan man vælge at designe den helt fra bunden, eller man kan bygge løsningen op omkring et funktions- og applikationsoptimeretbaseboard sammen med et "plug-in"computermodul.

Dette har været et stort spørgsmål under hele forløbet, og derfor har vi undersøgt mulighederne lidt nærmere. Dels for at se hvad mulighederne var, og dels for at se, hvad virksomheden måtte finde interessant. Mange af de ting vi er kommet frem til, er de samme som nævnes i en artikel netop om dette emne [11].

Figur 2.3: Et eksempel på et modulbaseret design fra Compulab

Skræddersyede designs

Et skræddersyet design giver bedst mulighed for at vælge de bedste stiktyper og tilpasset positionering. Denne type af computerboards forventes typisk kun at virke til en applikation, der giver mulighed for kostoptimering i en tidlig fase. Til gengæld er udviklingen af disse en tidskrævende proces, der kræver meget know-how. En vigtig faktor er at have bevågenhed omkring tilgængeligheden af de nøglekomponenter, man benytter i designet. Man skal hurtigt kunne finde en løsning, hvis et komponent går ud af produktion.

Det kan også typisk kræve adgang til avanceret PCB designsoftware, og hvis applikationen kræver det, måske også adgang til specifikke testinstrumenter. Skal boardet også produceres in-house, vil man også typisk skulle bruge montageudstyr.

Derfor må man konkludere at det bedst kan betale sig med denne type design, hvis der er tale om høje produktionstal, eller meget specialiserede eller dyre slutprodukter.

Modulbaserede designs

Disse designs som i figur 2.3, har som regel kortere "time-to-market", og produktionslevetiden garanteres som regel helt mellem 7 og 10 år. Man har også mulighed for at opgradere modulerne. Det kunne f.eks. være ved at udskifte processortypen til en hurtigere ved at udskifte modulet, som normalt kan gøres enten helt uden software ændringer eller med meget få.

Man kan sige at det er et koncept, der giver en høj grad af fleksibilitet, men samtidig har et stort potentiale for at adapteres til den specifikke applikation.

Figur 2.4: Et eksempel på et proprietært modul

Mange designmæssige krav skal der endog ikke tages stilling til, da løsninger som f.eks. høj-hastighedsbusser og processorafhængige interfaces er fuldt implementerede på modulerne, så udviklerne slipper for at bruge tid på disse ting. Almindelige interfaces som Ethernet, USB, med videre er typisk også lette at implementere, men kræver dog en høj viden om håndtering af operativsystemer og binære drivere.

Proprietære ARM-moduler

Disse moduler som i figur 2.4, giver en høj grad af samme fleksibilitet som de modulbaserede designs, men ulempen er her, at produkter fra en producent i mange tilfælde ikke er kompatibel med et produkt fra en anden producent. Det gør sig gældende både i forhold til hardware og software.

Hvad bør virksomheden gøre

Da virksomheden på nuværende tidspunkt har et meget skræddersyet design og et relativt lavt produktionsantal, må man nok sige at det modulbaserede design virker som et oplagt valg i forhold til fremtidige maskiner.

Vi valgte dog en proprietær løsning til denne prototype, da vi var afhængige af at den kunne kommunikere med allerede eksisterende udstyr. Samtidig kunne vi indkøbe denne relativt billigt, og få de nødvendige dele til præcis det virksomheden gerne ville vise.

2.6 Konklusion på system arkitektur

System arkitekturen virker ved første øjekast ikke så kompliceret, men den kan hurtigt blive det. Da jeg har valgt at Cloud løsningen egentlig er den centrale del i systemet, kan man udvide med hvad man næsten har lyst til. Det kunne være man ville have en PC applikation, eller en app til Android senere, og vores cloud kan implementeres som en del af begge disse tilvalg. Samtidig kan man bruge al Struers udstyr så længe det overholder Struers Communications Protokol (SCP), og samtidig har en seriel forbindelse.

En mere detaljeret samlet konklusion kommer til sidst, når alle dele af systemet er beskrevet.

Kapitel 3

Firmware opdatering

For at få Struers udstyr klart til en verden med IoT fandt vi ud af at en essentiel del, måtte være at kunne opdatere udstyrets Firmware Over The Air (FOTA). Udstyret ville få brug for en masse ny funktionalitet, og ændringer i SCP, så derfor fandt vi det logisk at prototypen skulle være i stand til autonomt at downloade en firmware fra skyen og efterfølgende installere den på maskinen. Dette kapitel vil derfor belyse de aspekter, der er ved en sådan FOTA løsning.

3.1 Aspekter i FOTA

En af de store fordele ved FOTA er at man kan gøre sit udstyr mere stabilt, eller udvide med funktionalitet, som udstyret ikke har "out of the box"[12]. Dette var netop grunden til at vi valgte dette som hovedfokus i vores samlede løsning.

Derudover har vi kigget lidt på hvordan man tidligere har brugt FOTA i bilindustrien [13]. Her skævede man til telekom industrien, som havde gjort et stort stykke arbejde for at implementere FOTA. I dag kan derfor stort set alle enheds fabrikanter tilbyde en FOTA service. Nogle fabrikanter har allerede lavet millioner af firmware opdateringer i skyen, der har reduceret omkostninger, undgået tilbagekaldelser og øget kundetilfredsheden.

Hos BMW har man opgjort at 50-70% af udviklingsomkostningerne til en ECU er software relateret og at 40% af en bils pris udgøres af elektronik og software.

Kigger man på hele bilbranchen, har man opgjort at garantiomkostninger ligger på $14 milliarder, hvoraf priserne per bil ligger på alt fra $265 til over $1.000 for et garanti krav. Som skrækeksempel var Toyota i 2005 nødt til at hjemkalde 75.000 Prius hybridbiler på grund af en software fejl.

Da mængden af software i biler er stigende er FOTA derfor en oplagt mulighed for at reducere sine omkostninger, få mere tilfredse kunder, reducere antallet af fejl og forbedre kvaliteten.

Et andet argument er at mange virksomheder kæmper for at få en hurtigere produktudvikling, som kan resultere i meget lidt tid til at lave detaljerede tests. Samtidig vokser den mobile internet dækning med hastige skridt, og gør det derfor forholdsvis billigt at koble de rette komponenter på sit udstyr. Dette bevirker at man kan lave et system, der er selv-vedligeholdende uanset hvilken lokation udstyret har [14].

En ren manuel proces kan være en dyr fornøjelse. En bruger kan nemlig vælge at ignorere en advarsel om ikke at må slukke for strømmen, for at lave en genstart midt i opdateringssekvensen, og herefter vil man stå med en måske ubrugelig maskine.

For at undgå dette vil man typisk i følge William Gatliff lave en micro programmer. En sådan programmer vil typisk sætte det indlejrede system i en tilstand, hvor det er forventet at download proceduren bør starte. Dette vil typisk ske på foranledning af en bruger der har trykket på en update knap, men dette mente vi var mere optimalt at gøre autonomt den første gang prototypen tilkobles i vores tilfælde. Dette skulle gerne få target udstyret til at bringe alle kontrollerede processor til et sikkert stadie.

Dette har været en del af den manuelle opdatering i virksomheden længe. Det har derfor ikke været nødvendigt at implementere ny funktionalitet på Struers udstyret. Det har udelukkende været et spørgsmål om at lave den del af micro programmeren, der skulle sætte prototypen i det rigtige stadie i forhold til kommunikationen med udstyret, når en overførsel af firmware skulle ske.

Ulempen i en micro programmer er at håndteringen af processen kommer til at kræve flere udvikler ressourcer, specielt hvis disse systemer er dårligt koblet sammen [15]. Her har jeg dog haft en egenudviklet kommunikationsprotokol, der har gjort denne kobling lidt lettere. Denne protokol vil blive beskrevet senere.

Man kan i Struers godt opdatere firmware på nuværende tidspunkt, men det

fungerer via. en kablet forbindelse til en bærbar computer, der skal have downloadet firmwaren, inden en service teknikker kan starte opdateringssekvensen.

Det vil helt klart lette brugervenligheden i forhold til en serviceteknikker, hvis opdateringen kunne gøres med FOTA, og vil helt sikkert også gøre det nemmere løbende at forbedre stabiliteten af udstyret løbende.

Dog er der den ulempe at det vil kræve en hård disciplin i forhold til test/afprøvning af micro programmeren, da vi har valgt den skal fungere autonomt.

3.2 Design og implementation

Da vi havde beslutte at firmware opdateringen var en essentiel del af opgaven, kigegde vi samtidig på muligheden for roll back, fordi det ville give mulighed for at signalere til alle online maskiner og konvertere tilbage til en fra fabrikken ren version af firmware [16]. Det ville gøre hele systemet hurtigt genopretteligt, og sikrer at slutbrugeren ville få minimal nedetid i forbindelse med en kritisk firmware fejl.

Det viste sig dog at det nuværende setup ikke helt ville kunne klare det, da det kræver en stærk disciplin i sin versionering af firmware, og ditto med serienumre. Derfor er denne funktionalitet fravalgt på nuværende stadie.

I stedet vil jeg fokusere på hvordan opbygningen af kommunikationen med Struers udstyr er implementeret, da det samtidig afleder hvordan implementeringen af firmware opdateringen foregår, som også beskrives i det efterfølgende.

3.2.0.1 Struers Communication Protocol

For at sende beskeder mellem prototypen og Struers udstyr har jeg benyttet mig af en protokol udviklet af Struers. Denne protokol er en standard for hvordan man sender et byte array mellem en tilkoblet enhed til Struers udstyr. Et sendt eller modtaget byte array skal overholde protokollen, som ses i figur 3.1 på næste side.

Som det ses i figuren beskriver de første 16 bits typen af telegram nummeret, og herefter følger en mængde databytes fra Data 1 til Data X. Efter dette puttes værdien fra et Cyclic Redundancy Check (CRC) i de næste 16 bits, og til sidst en End Of Transmission (EOT). Denne er vedtaget altid at være `x04` i hexadecimal.

Type	Data 1	Data 2	Data 3	Data 4	¼¼¼	Data X	CRC	EOT
16	8	8	8	8		8	16	8

Figur 3.1: Figur der viser byte strukturen i et telegram i SCP

Value	Coding
0x00 ~ 0x03	No coding
0x04 (EOT)	Send as 0x1B + 0xFC, i.e. double byte
0x05 ~ 0x1A	No coding
0x1B (ESC)	Send as 0x1B + 0xE5, i.e. double byte
0x1C ~ 0xFF	No coding

Figur 3.2: Oversigt over hvilke byte værdier, der skal escapes

Da man nu har valgt en fast EOT, er man nødt til at tage højde for at denne værdi kan sendes med som data i telegrammet. Her har man så valgt en måde at escape ulovlige bytes. I figur 3.2 ses hvordan man har valgt at håndtere denne escape sekvens.

3.2.0.2 Cyclic Redundancy Check

Et cyclic redundancy check (CRC) er en fejlfindings algoritme, som normalt bruges i digitale netværks og hukommelses enheder til at detektere tilfældige fejl i rå data. Man putter en kort tjek værdi på de blokke af data, der sendes gennem systemet. Dette er baseret på en polynomisk division af indholdet. Ved hentning bliver kalkulationen gentaget, og hvis en værdi ikke passer, kan der fortages fejlretning mod korrumperet data.

CRC er populært fordi det er simpelt at implementere i binær hardware, let at analysere matematisk og er især godt til at opdage normale fejl forsaget af støj i transmission. Fordi tjek værdien har en fast længde, kan funktionen der generere den også bruges som hash funktion ind imellem [17].

Måden et check fungerer på hos Struers ses i 3.3 på næste side. Denne stump pseudo kode var allerede givet, så denne skulle blot implementeres i det ønskede sprog. Egentlig er selve funktionaliteten underordnet, da dette allerede var måden at gøre det på i udstyret. Derfor skal enhver kommunikerende part benytte sig af samme metode.

```
CrcSum = CrcSum ^ (DataByte * 256);

for (Count = 0; Count <= 7; Count++)
{
  if (CrcSum > 0x7FFF)
    CrcSum = (CrcSum * 2) ^ 0x8005;
  else
    CrcSum = CrcSum * 2;
}
```

Figur 3.3: Struers implementering af CRC

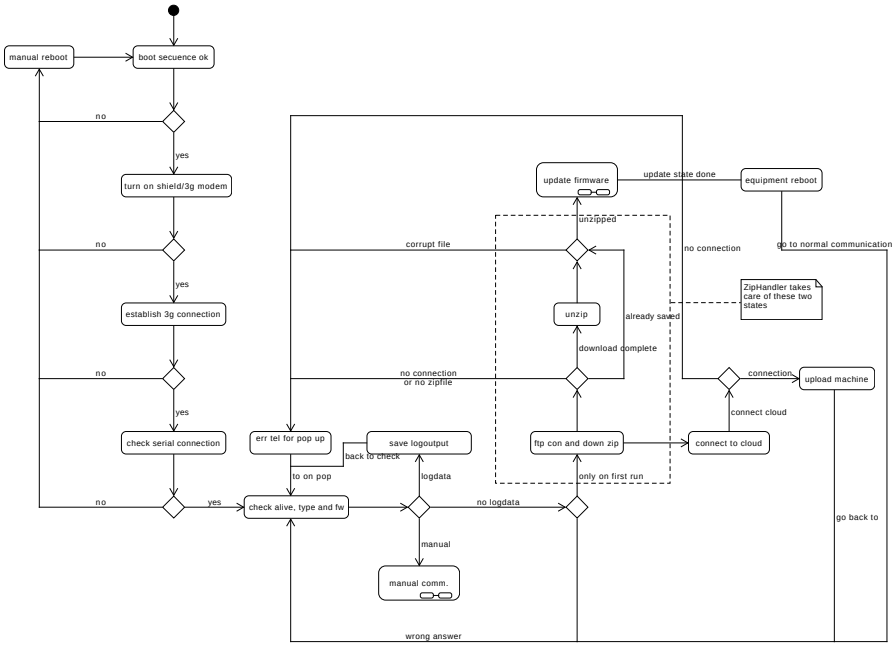
3.2.0.3 Tilstandsmaskine for download

Før vi kan overføre firmware til Struers udstyr er vi nødt til først at downloade den til vores prototype. Da dette kræver forskellige tilstande med forskellig funktionalitet, der kan lede til nye tilstande eller tilbage til tidligere tilstande, var det oplagt at implementere det som en tilstandsmaskine 3.4 på den følgende side. Denne tilstandsmaskine bruges hver gang man tilslutter prototypen til Struers udstyr og tilslutter strøm. Hvis der er en nyere firmware tilgængelig, vil vi springe til en ny tilstandsmaskine, som forklares senere. Hvis ikke går vi til en tilstandsmaskine, der håndterer den normale kommunikation. Denne tilstandsmaskine er væsentlig mere simpel, og består nærmest bare af et sendt telegram og efterfølgende et modtaget eller tomt svar.

Med udgangspunkt i beskrivelser for tilstandsmaskiner i Larmans bog[18], har jeg forsøgt at skabe oveblik over de forskellige tilstande. De første tilstande, der ses er manuelle, og ikke en del af koden, men nødvendige tilstande, som styresystemet er sat til at tage sig af, når en serviceteknikker sætter strøm til prototypen. De mange første tilstande kan ikke rigtigt kontrolleres remote, så derfor går alle disse tilstande til en fejltilstand, hvor det er nødvendigt at lave en manuel genstart.

Uden fejl vil prototypen boote, hvorefter et script vil starte 3G modemmet på prototypen, lykkes dette vil den forsøge at skabe en ppp forbindelse. Herefter vil seriel forbindesen blive forsøgt åbnet. Hvis seriel kablet ikke er sat på eller på anden vis fejler, skal man også her manuelt genstarte. Meget af alt dette kan undgås, hvis visionen er at få prototypen ind som en del af selve Struers udstyret på et tidspunkt. Dette aspekt er et tilbagevendende punkt igennem hele rapporten.

En tilstand, der er ret væsentlig i hele opbygningen af prototypen er at vi med et bestemt telegram fra SCP kan spørge om maskinen er i live, og hvilken type maskine vi snakker med. Egentlig skulle vi også kunne tjekke firmware version,



Figur 3.4: Initial state machine that downloads and unzip firmware

men der er en detalje, der gør at man ikke kan dette alligevel. Denne detalje beskriver jeg i kapitlet omkring modenhed.

Fra dette stadie kan der ske tre ting. Det første er at gemme logdata fra Struers udstyret. Dette gøres ved at tjekke formatet på det svar, der kommer fra maskinen. Hvis det ikke er standardsvaret fra SCP er der en bestemt måde man har formateret sine log strenge på. Er det en af disse gemmes strengen i en tekstfil.

Har vi allerede opdateret firmwaren fra opstart vil der være sat en `boolean`, der fortæller at vi er klar til manuel kommunikation. Dette er håndteret i en separat tilstandsmaskine.

Hvis der ikke var logdata har vi forhåbentlig modtaget stadardsvaret fra SCP. Hvis ikke sender vi vores telegram igen. Er det istedet det rigtige svar, tjekker vi serienummeret på maskinen. Her kommer vi igen til et punkt, der skal tages i betragtning i forhold til modenhed. Det er ikke alt Struers udstyr, der har serienummer på nuværende tidspunkt.

Hvis serie nummeret findes tjekker vi den tilhørende firmware igennem vores cloud løsning. Her får vi også besked på hvilken placering på Struers FTP server, den er placeret som zip-fil. Da projektet startede havde den valgte cloud løsning nemlig ikke filhåndtering. Dette er dog netop kommet, og bør derfor overvejes som et alternativ. Dette bliver beskret nærmere i afsnittet omkring cloud løsningen.

Der er lavet en klasse der kun har til formål at håndtere zipfilen med firmware, så disse to tilstande klares af denne klasse, og burde måke have været ført over i sin egen tilstandsmaskine, men på den anden side er der meget begrænset logik i de to tilstande, så det gav meget god mening at beholde dem i samme tilstandsmaskine, da det giver et godt indtryk af helheden. Tilstandende består simpelt i at hente firmwaren som zip-fil fra en Struers FTP server, for så efterfølgende at pakke den ud.

Samtidig skaber vi forbindelse til vores cloud, og ligger den information vi har om vores prototype og udstyr op. Kan vi ikke få forbindelse er får vi smidt en fejlmeddelelse af sted.

Når alt dette er klaret går vi i en tilstand, som har sin egen tilstandsmaskine. Her klares selve opdaterings sekvensen. Denne tilstandsmaskine bliver beskrevet i 3.2.0.4 på næste side.

Til sidst skal jeg nævne at tilstanden til fejlmeddelelser (error pop up) ikke findes på nuværende tidspunkt, men det var et ønske fra virksomheden at fejlkommunikation blev meddelt på en måde, og da prototypen ikke har noget display,

blev løsningen at man ville lave nogle nye telegrammer i SCP, der kunne sende fejlmeddelelserne til maskinen. Her ville man så være nødt til at opdatere firmwaren for at kunne vise de nye skærmbilleder der skulle til. Dette ville dog betyde at første gang man firmware opdaterer noget Struers udstyr er det lidt i blinde, da man så ikke ville kunne se fejlmeddelelser. Derfor er dette heller ikke implementeret i skrivende stund, men da det er vedtaget at sende beskeder gennem SCP, kan dette meget hurtigt klares.

3.2.0.4 Tilstandsmaskine for update sekvensen

Denne maskine er det man ville kalde for microprogrammeren. Denne er lavet som en tilstandsmaskine, der håndterer alle skridt i selve opdateringssekvensen uden indblanding fra brugeren 3.5 på side 26. Selve tilstandende har jeg været nødt til at aflure fra et gammelt opdateringsprogram fra Struers, men princippet er det samme på nær at det hele kører autonomt her. Med det gamle program skulle brugeren tage stilling til hvilken firmware, der skulle på maskinen og selv genstarte til bootloaderen, for så at kunne starte resten af opdateringssekvensen. Selve telegrammerne fra SCP ligner derfor sig selv og mange af tilstandene er de samme, men tilpasset brugen i prototypen.

Måden jeg har implementeret det på, er ved at spørge om vi er klar til opdatering, som er en standard besked valgt i SCP, som giver et forventet svar. Hvis svaret ikke passer på grund af enten en fejlet CRC eller at udstyret sender en log besked, går vi tilbage til samme state og spørger igen. Hvis svaret er korrekt genstarter vi til bootloader.

Dette er en grim løsning, da jeg har været nødt til at få tråden til at sove i en tidsperiode, der svarer til at udstyret kan nå at genstarte. Når denne tidsperiode er overstået spørges der om vi er kommet i bootloader.

Man kunne implementere noget logik, der blev ved med at spørge, da det en sjælden gang imellem måske fejler at gå i bootloader, og derfor er interessant at vide om vi stadig er i applikations mode. Dette er dog besværligt at implementere, da vi også er nødt til at ændre konfigurationen på vores serielport, da den ændrer sig afhængig af hvilken mode vi er i. Man burde nok lave et telegram i SCP, der gav et rigtigt svar om det er applikationen eller bootlaoderen, der er i brug, men det vil kræve at firmwaren på det tiltænkte udstyr bliver opdateret først.

Den tidligere løsning man har valgt, var simpelt at lave en knap til at lave genstarten. Denne løsning er dog ikke specielt brugervenlig, og må derfor betragtes som ikke ønskelig i en autonom løsning.

Tilbage til vores spørgsmål om vi er kommet i bootloader, kan vi konstatere, at hvis svaret er forkert, må vi spørge igen eller genstarte til applikationen. Det forkerte svar opstår, hvis maskinen stadig tror den er i applikation, og vi sender en besked med seriel indstillinger matchende bootloader. Jeg har ikke undersøgt præcis hvilket telegram, den tror det er, men har taget højde for at svaret er forkert. Forkerte svar burde kun fremkomme hvis tidsperioden er sat for lavt, så normalt vil man forvente, at vi går videre til næste tilstand.

I denne tilstand spørger vi om udvidet info på maskinen, som giver os den præcise version af firmwaren. Efter at have spurgt om firmware versionen, går ind og sammenligner det med den firmware, der blev hentet i skyen.

Hvis firmwaren bliver godkendt, sender vi et telegram om at udstyret skal gemme sin gamle firmware på sit Struers Memory Module (SMM). Det er dog ikke alle maskiner, der har SMM indbygget og derfor kan prototypen bare gå til næste tilstand, hvis dette er tilfældet.

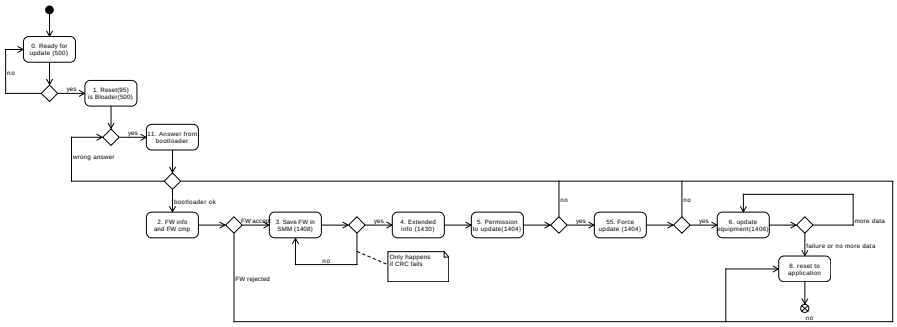
Tilstanden vi nu kommer til spørger lidt indtil hvor stor SMM'et er, og om hukommelses størelsen på PCB'et. I teorien kunne dette step undværes, da vi allerede har fået bekræftet vores firmwares validitet tidligere, men da tilstanden bruges i det gamle opdateringsprogram, kunne man godt risikere at det skulle bruges senere. Hvis udstyret forventer telegrammet fra denne tilstand, så er det bedre at have det med, end senere at få fejl fordi vi ikke har det med.

Næste tilstand sørger for, at spørge bootloaderen om en opdatering er tilladt. Hvis ikke genstarter vi til applikationen. Hvis det er tilladt hopper vi til en tilstand, der fortæller udstyret at en opdatering starter.

Herefter har vi en overførselstilstand. Denne tilstand kører i en løkke, indtil udstyret giver besked om at overførslen er færdig. Skulle det ske at prototypen sender den samme besked mange gange, er der lavet en timeout funktion, der går til genstarts tilstanden. Det samme sker hvis udstyres giver en fejlmeddelelse. Når udstyret giver besked om at overførslen er færdig, går man også til genstarts tilstanden.

Man skulle tro at en afbrudt overførsel ville ødelægge maskinen, men her bliver man reddet af SMM'et. Det sørger for at overføre den gamle firmware tilbage til udstyret ved næste genstart. Derfor har vi ikke behøvet at tage højde for dette scenario på prototypen.

Når den sidste tilstand er udført, går vi tilbage til den første tilstandsmaskine 3.4 på side 22. Denne vil så opdage, at vi er klar til manuel kommunikation, og derfor springe over i denne tilstandsmaskine i stedet.



Figur 3.5: State machine for firmware update sekvens

3.3 Konklusion på firmware update

Løsningen fungerer fint til et eksempel på en prototype, og da der har været nogle Struers normer for dele af implementeringen, har det gjort processen nemmere. Samtidig er der dog ting, der bør implementeres fuldt ud, før en eventuel prototype tages i brug. Jeg mener at rollback er så vigtig en funktion, at den ikke kan undværes, hvis man skal have en eller anden form for kontrol over en fejlbehæftet release, men det har vist sig at versionsstyringen, og måden at tjekke firmware versionen på ikke har været skarp nok.

Det kan godt lade sig gøre at få lavet lappeløsninger til mange af disse ting, og man kunne få en hæderlig firmware opdatering til at køre. F.eks. kunne en serviceteknikker få lov at taste serienummer på de apparater, hvor dette kunne lade sig gøre via frontend applikationen. Der er dog også undtagelser, hvor udstyr ikke har implementeret muligheden for tildelingen af et serie nummer i firmwaren, men kun har et fysisk klistret på siden. Begge dele kunne bare tastes, men sandsynligheden for fejl i indtastninger, kunne resultere i to ens nøgler i databasen, hvor man gerne så serienummeret som en unik nøgle. Derfor må det nok anbefales ikke at inkludere de typer udstyr, og måske i stedet kun kigge fremadrettet, hvor man kan have prototypen indbygget og sammenkoblet i selve udstyret, eller som minimum kun på udstyr der har fået indtastet serienummer.

Kapitel 4

Cloud Computing

Cloud computing er den hurtigst voksende teknologi inden for IT ifølge Chhabra [19]. Man bruger typisk skyen til at tildele resurser. Disse resurser kan være servere, lagerplads, programmer og netværk. At holde disse resurser i skyen, kan være medvirkende til at spare virksomheden for at holde styr på infrastrukturen med mere. Cloud computing er en løsning på at kunne tilgå data hvor som helst og når som helst igennem netværk.

At sende og modtage information gennem skyen, sætter selvfølgelig høje krav i forhold til valg af en Cloud Provider. Specielt fordi sikkerheden i forhold til følsomme data skal være i orden.

4.1 Design og implementation

Idéen var som tidligere nævnt at bruge vores cloud løsning, som den centrale del i systemet. Her vil al hardware så koble sig på ved hjælp af en eller anden form for internet forbindelse.

Valg af cloud

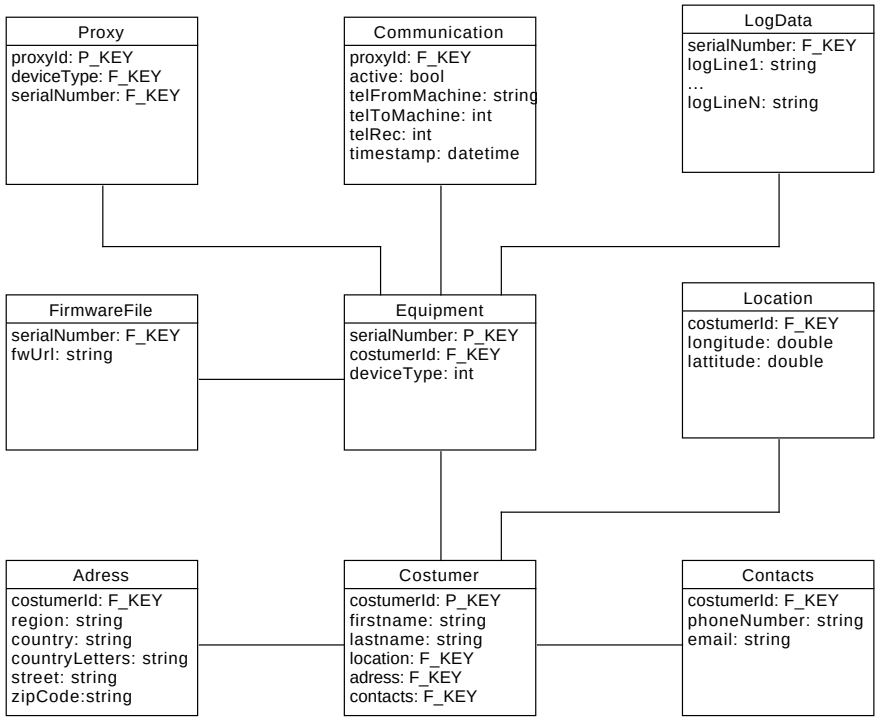
Der er snart et hav af forskellige cloudløsninger, så vi har valgt ud fra prototype aspektet. Nemlig at det skulle være nemt, hurtigt, billigt og understøtte flere platforme (eksempelvis med et REST API).

Vi valgte at gå med Firebase til prototypen. Dette er udelukkende på grund af at vi hurtigt og billigt kunne komme i gang, samtidig med at udvikle på flere platforme, da Firebase har API til både iOS, Android og et REST API til stort set alt andet. Dette gjorde det muligt at få vores prototype i Mono (C#) til at kommunikere direkte med vores frontend, som er en iPhone applikation (Swift). Dette fungerer egentlig ret godt på trods af at apllikationerne er af forskellig type og skrevet i forskellige sprog. Til gengæld er den træ struktur Firebase bygger på ikke god, hvis vi skal op i meget stor skala på et tidspunkt.

Inden vi lavede implementeringen på de forskellige dele af systemet, satte vi os, og lavede et overblik over, hvordan databasen skulle struktureres med et ER diagram inspireret af [20]

I diagrammet kan man se, at vi har oprettet vores prototype som en proxy til vores udstyr. Dette udstyr har så en tilhørende firmwarefil, indeholder noget kommunikation, tilhører en kunde og opsamler noget logdata. Den tilhørende kunde har vi ikke implementeret på nuværende tidspunkt, da vi kun har et par stykker udstyr i databasen. Det eneste der er rigtig spændende i databasen, er at vi har en bool der indikerer om udstyret er aktiv. For at finde ud af dette sætter vores prototype denne bool til sand hver gang prototypen sender til skyen. Vi sender samtidig et tidsstempel med. Dette bruger vi i vores frontend til at bestemme om maskinen stadig er aktiv. Vi kan jo ikek bare tjekke vores bool. Hvis udstyret bliver afbrudt vil vores bool ikke kunne blive sat til falsk. Derfor ligger ansvaret på nuværende tidspunkt i vores frontend. Her bruger vi tidstemplet, som vi tjekker to gange efter et predefineret interval. Hvis tiden har ændret sig, er udstyret stadig aktivt, men hvis ikke sætter vi vores bool til falsk, og dermed er udstyret inaktivt.

På længere sigt vil det være mere optimalt at lægge ansvaret på en form for midter lag, der kun tager sig af logik til skyen. Noget tyder nemlig på at det ikke er det eneste logik, som hverken frontend eller prototype bør have ansvar for. Specielt ikke når man begynder at ville analysere på det data man samler op.



Figur 4.1: ER diagram over databasen

Sikkerhed i cloud

Det aspekt der har givet størst anledning til bekymring er sikkerhed. Det kan jo være meget personfølsomme data, der kan hives ud af udstyret, og Struers' kunder ønsker måske ikke at disse oplysninger falder i de forkerte hænder.

Skulle vi have gået mere grundigt til værks, kunne vi igen kigge på de råd, der gives af Chhabra [19] i forhold til sikkerhed. Her bliver det blandt andet belyst, at man skal være forsigtig i forhold til valg af Cloud provider. Det anbefales at mødes med de forskellige providers for at få et overblik over hvad de tilbyder i forhold til sikkerhed, infrastruktur, kvalitet og teknisk formåen.

Det anses som vigtigt at der tilbydes en sikker data transmission. Det vil sige at data skal som minimum kunne krypteres plus andre sikkerhedstiltag som sikker browsing med HTTPS.

Data integritet skal sikres ved at databasen kun kan ændres af autoriserede transaktioner. Data bør samtidig være tilgængelig altid. Der bør derfor være en form for backup løsning i forbindelse med en server/system fejl.

Derfor virker vores valg af cloud løsning ikke særlig indlysende, og bør helt sikkert være et aspekt, der undersøges endnu nærmere før man endeligt kaster sig ud i projektet.

4.2 Konklusion på Cloud løsning

Til en fungerende prototype fungerer vores sky løsning egentlig fint nok. Vi kan dog godt se, at sikkerhed og skalerbarhed ikke er god nok i den valgte løsning. Skal vi til at tage os af følsom data fra kunder er det et meget vigtigt aspekt, at alting er i orden her. Sikkerheden i Firebase er ikke dårlig, men uden en indsats fra udviklerne, vil databasen egentlig være helt åben, hvis man bare opsnapper den rigtige URL et sted. Meget af denne sikkerhed er delvist givet i andre løsninger, og samtidig har virksomheden allerede valgt en cloududbyder til andre services. Derfor ville det være oplagt at integrere denne løsning til dette system.

Kapitel 5

Frontend

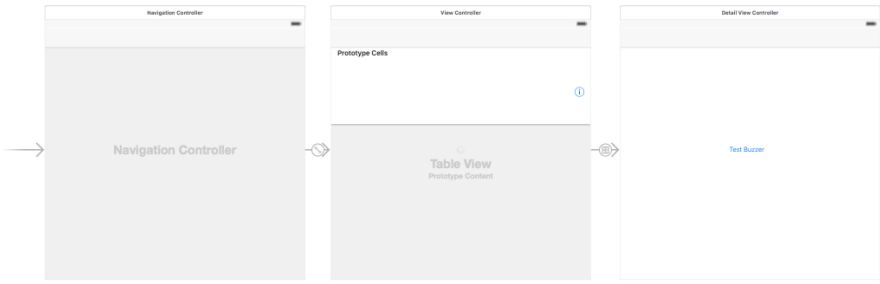
Da vi også skulle bruge en måde at interagere med Struers udstyr på, har vi kigget lidt på at lave en frontend applikation til dette. Der har været mange overvejelser inde over dette, som jeg vil belyse i dette kapitel.

5.1 Valg af platform til frontend

Vi startede meget simpelt med at finde ud af, hvilken type af frontend applikation vi synes virksomheden havde mest brug for. Valget faldt ret hurtigt på en service applikation, som en serviceteknikker kunne bruge. Dette er på baggrund af de konklusioner vi kom frem til ved at undersøge hele IoT fænomenet i kapitel 2 på side 3.

Her har jeg tidligere været med til at lave en prototype til en iPad app, for at undersøge hvilke fordele en serviceteknikker kunne få ud af et mobilt device. Vi valgte at gå af samme vej, da det ville være forholdsvis let at lave en frontend til dette, da jeg allerede var bekendt med Apple's Cocoa Touch framework og principperne i iOS udvikling.

Vi kunne også have valgt at lave noget multiplatform i HTML5 eller Javascript, men efter at have læst en artikel [21], der konkluderer at det sjældent giver



Figur 5.1: Frontend GUI udkast

gode resultater at lave multi platformsudvikling. Artiklen tager godt nok kun fat i Titanium, men da den er læst som en del af et kursus i mobil applikationsudvikling og vores underviser har gjort sig en del erfaringer på det område, blev konklusionen den samme for lignende tiltag. Det nærmeste jeg kom frem til som en løsning var Xamarin, der faktisk er det samme som Mono, vi har brugt til vores prototype. Men også her er der begrænsninger, som begrænsede UI muligheder og apps der fylder utrolig meget sammenlignet med native apps [22].

Dog er planen senere at udvide med flere applikationer til flere platforme, men den præcise strategi for dette er ikke vedtaget på nuværende tidspunkt.

5.2 Funktionalitet

Da man allerede havde en prototype fra tidligere, der havde vist vejen for en service app, valgte man at fokusere på det kun absolut mest nødvendige for app'en. Derfor lavede vi en simpel GUI, som man kan se et udkast til i 5.1. Her kan man se at brugeren vil blive præsenteret for en liste af aktuelt udstyr. Det som ikke ses på figuren er at et billede af udstyret, nøglen fra en tilkoblet proxy og udstyrstypen, bliver vist.

Hvis man vælger noget udstyr, bliver man ført ind til et detalje view. Her er det meningen al info om udstyret bør komme såsom den eksisterende firmware version, seneste logdata, fejlhistorik osv. Dette kan sagtens hentes fra databasen og vises i nogle passende tekstfelter eller lignende. Det er dog ikke noget, der er forsøgt brugt for meget tid på, da der ikke har været nogen helt klar plan for hvad man egentlig ønskede at få vist.

Det eneste sjove i detalje viewet lige nu er, at vi har lavet en knap til at aktivere et indbygget horn i maskinen. Det er ikke en nødvendig funktion, men beviser at der er hul hele vejen igennem, så man faktisk kan styre udstyret remote.

5.3 Konklusion på frontend

Med det formål at bevise, der kan kommunikeres fra en frontend applikation hele vejen til udstyret, er vi vel nået i mål. Kigger man lidt mere overordnet på vores frontend, er det ærgerligt at der ikke var lagt en lidt mere langsigtet plan, og at vi ikke nåede at lave andre typer af applikationer, som f.eks. en PC applikation, der kunne agere kundeportal eller lignende.

På den anden side set har vi fået belyst at løsningen fungerer til iOS som minimum. Da der er et REST og et Android API til databasen også, burde det være en smal sag at udvide til andre platforme. Prototypen kommunikerer ved hjælp af REST API'et, så det ved vi allerede virker.

Kapitel 6

Open Source vs. Closed Source

6.1 Fordele og ulemper

Som Erik J. Martins artikel siger [23], er de bedste ting i livet er gratis, og dette kan måske også overføres til software. Bare se på populariteten af de mange Open Source Software (OSS) som for eksempel: Apache HTTP Server, Linux, Android operativsystemer eller internet browsere som Mozilla Firefox.

Faktisk har man i en undersøgelse fundet ud af, at halvdelen af de adspurgte virksomheder har adopteret OSS løsninger på en eller anden måde i forbindelse med virksomhedens IT strategi, og selvom artiklens hovedfokus er på online publicering, kan grundelementerne i diskussionen sagtens genkendes i Struers' situation.

Fordele

I denne sektion vil jeg liste nogle af de fordele som artiklen nævner, og som vi er kommet frem til i virksomheden.

- Ingen licens betaling.
- God udviklingsfællesskabs-support, da der altid er ildsjæle, der prøver at forbedre platformen.
- Personalisering. Da koden er åben, har man større frihed til at tilpasse løsningen til ens eget brug.
- Hurtigere time-to-market (Ny funktionalitet bliver hurtigere delt med udviklingsfællesskabet).
- Ingen betaling på udviklingsværktøjer
- Man er ikke fastlåst til en leverandør
- Fleksibilitet (Virksomheden kan integrere let med andre løsninger uden at være fastlåst af den leverede software fra leverandøren).

Udfordringer

I denne sektion vil jeg liste nogle af de ulemper som artiklen nævner, og som vi er kommet frem til i virksomheden.

- Support usikkerhed: En stor risiko, er at support fælleskabet er utrolig småt - dette kan være medvirkende til at softwaren bliver forladt eller at der er meget få firmaer/udviklere der vedligeholder, tilbyder assistance og/eller resurser.
- Fremtidssikring: Man er nødt til at have det som en del af strategien, at man skal bruge resurser på at implementere patches, opdateringer og forbedringer til ens kerne version af platformen.
- Træning: Eftersom at løsningen kan være specielt bygget til virksomheden, kan et vist niveau af uddannelse være nødvendig

6.2 Konklusion på OSS

Ud fra disse aspekter synes jeg det er svært at konkludere hvad der er bedst for Struers. Det virker som om at OSS kan være en rigtig god løsning og spare virksomheden for penge i sidste ende. Samtidig kræver det en omkostning for virksomheden, da de vil være nødt til at afsætte de nødvendige resurser. Da jeg nu har været i virksomheden i et godt stykke tid, kan jeg konstatere at netop frigivelse af resurser vil være et problem medmindre virksomheden ændrer deres software strategi.

Til gengæld har vi set at en modulær løsning på hardware siden, ville være en god ting for Struers, og her virker OSS til at have en klar fordel. Så hvis man får lavet grundlaget i orden for OSS, er det helt klart vejen at gå.

Kapitel 7

Modenhed

Undervejs i projektet er vi støt på nogle ting, der gør at modenheden i forhold til en prototype, som vi har sat i værk, ikke kan resultere i en udgivelse af produktet. Jeg har undervejs i projektet samlet disse punkter sammen, og inddelt dem i et teknisk aspekt, hvad man som virksomheden bør overveje og et juridisk aspekt. Disse har været samlet i en log på virksomhedens sharepoint site B.1, for at kunne have disse overvejelser in mente, hvis man vælger at indføre projektet i virksomhedens strategi.

7.1 Teknisk (Application & bootloader)

Vi konstaterede meget hurtigt, at vi ikke havde et serienummer på alle maskiner. Dette var ellers noget vi ret hurtigt blev enige om skulle være den unikke nøgle til alt udstyr i systemet. Noget udstyr kunne man bare tage PCB serienummeret fra, men da det kan blive defekt og skal udskiftes, ville det være en dårlig løsning i længden.

Samtidig kunne vi kun se den aktuelle firmware version ved at genstarte maskinen til bootloader, fordi man kun kan aflæse et modenheds-bogstav i bootloader. Det er ikke særligt hensigtsmæssigt for kunder, at vi genstarter deres udstyr i tide og utide, for at tjekke hvilken firmware der ligger på deres udstyr.

Samtidig havde vi ingen måde at fortælle om fejlkommunikation på, så en vigtig parameter må være at finde en løsning til dette, før man går videre. Jeg har foreslået at man bruger det eksisterende display på udstyret, men det vil kræve en opdatering af firmwaren på alt udstyr, der skal kunne kommunikere med prototypen.

7.2 Som virksomhed

Den måde man i dag bruger modenheds-bogstavet er ikke optimal, da man faktisk ikke har styr på kombinationen af firmware versioner og costum versioner. Dette kunne man få styr på ved at lave al custom firmware som et nyt device. Så i stedet for kun at have en Axitom som device, også har en Axitom med clamping tool, og måske en tredje med luftkøling i stedet for vandkøling. Hvis den er sin egen device, kan man pludselig også regne med firmware versionen. På nuværende tidspunkt bruges modenheds bogstavet til costum made på alt Axitom, og man ved derfor ikke hvilken type af custom made produkt man står med, blot at det er costum made.

7.3 Juridisk

Der er på nuværende tidspunkt ingen, der har overblik over hvor meget data man må opsamle i de lande man sælger Struers udstyr til, og om man overhovedet må opsamle data.

Et andet aspekt er, om Struers' kunder uanset lovgivning ønsker at tillade dataopsamling. Samtidig vil det være interessant at vide om lovgivningen og interessen hos kunder ændrer sig, hvis man deler data med 3. part, som f.eks. leverandører af skæreskiver til udstyret.

7.4 Konklusion på modenhed

Jeg må konstatere at virksomheden ikke er moden til et IoT system på nuværende tidspunkt, men man kan jo snige det lidt ind af bagvejen. Med det mener jeg at man allerede nu gerne vil bruge datalogningsdelen, og da koden har en fornuftig opdeling, kan dette nemt klares. Senere kan man så udvide funktionaliteten.

Hvis Struers skal blive klar til IoT, tror jeg det er lettere at lave udstyret klar til connectivity i forbindelse med at man skifter til en ny hardware platform, da det vil kræve en enorm mængde resurser i udviklingsafdelingen alene, for at ændre eller lappe på al den legacy, der efterhånden er i forbindelse med den nuværende platform. Samtidig har de andre afdelinger på nuværende tidspunkt ingen idé om hvilke muligheder/konsekvenser en IoT løsning vil få for deres afdeling. Derfor vil det nok være nemmere at sørge for en simpel platform til at starte med. Denne vil så kunne udvides på sigt.

Ulempen i denne fremgangsmåde er at man er nødt til at gætte lidt på hvordan platformen skal struktureres, hvilket kan besværliggøre eller begrænse mulighederene for ønsker andre afdelinger måtte have til systemet.

Fordelen er at det kan sætte deres tanker i gang, og på den måde få sat ord på hvordan netop deres afdeling kan drage fordel af et IoT system.

Hvis ikke man ikke kan vente med at rulle løsningen ud, bør man i hvert fald kigge på de ting, der er showstopper for en fuldstændig løsning. Man kan sagtens patche på det allerede eksisterende, men det er nok nødvendigt at lave en overordnet strategi, for hvad man egentlig vil, inden man sætter i gang.

Kapitel 8

Test

Test har været en vigtig parameter i forhold til selve kommunikationen mellem Struers udstyr og prototypen. I de følgende sektioner vil jeg beskrive hvordan disse tests har været udført.

8.1 Reverse engineering

Jeg har brugt en del tid på at finde ud af, hvordan hele kommunikationsformen i en download procedure foregik i det gamle Struers program til opdatering, for så at sammenligne med min egen procedure, da der ikke var nogen dokumentation af dette.

Det har dog resulteret i en god forståelse af selve opdateringsproceduren og har derfor kunnet implementere en tilstand af gangen i vores prototype. Jeg vil dog mene, at arbejdet med blandt andet frontend kunne have været lang mere detaljeret, hvis ikke så meget tid havde været brugt på reverse engineering.

8.2 Black box testing

Formålet med black-box tests er at sikre ens program løser de problemer, det er sat til at løse. Derfor er det også nødvendigt med en nogenlunde præcis idé om, hvad det er for et problem, programmet skal løse. I princippet behøves programmets kode ikke i testøjemed. Man tester i stedet over typiske, såvel som ekstreme input værdier og inputs der er beskrevet som undtagelser eller ulovlige [24].

Dette princip kunne vi benytte i forhold til at teste SCP funktionaliteten i applikatione, som også var det bærende element for at resten kunne fungere.

SCP test

black box test princippet har vi brugt, til at teste om vores kommunikation ind i udstyret gav det forventede svar, eller om vi havde sendt det rigtige telegram. Da vi havde SCP som en tabel B.4 på side 57, var det egentlig bare at teste med typiske og ekstreme værdier, for at se om output kom ud som forventet.

8.3 Sniffe værktøj

For at teste om input var sendt korrekt og om output var korrekt mellem Struers udstyr og prototypen, valgte vi at lave en sniffer, der kunne opfange data sendt eller modtaget mellem de to enheder. Snifferen 8.1 på næste side består meget simpelt af 3 rs232 stik, som er forbundet i en samlemuffe. Her er de forbundet ved hjælp af et oversigtsdiagram over rs232 ledningsforbindelser, som angiver hvilke ledninger der er transmitter eller modtager. Den ene ende kobles til prototypen, den anden til udstyret og den sidste kobles på en PC.

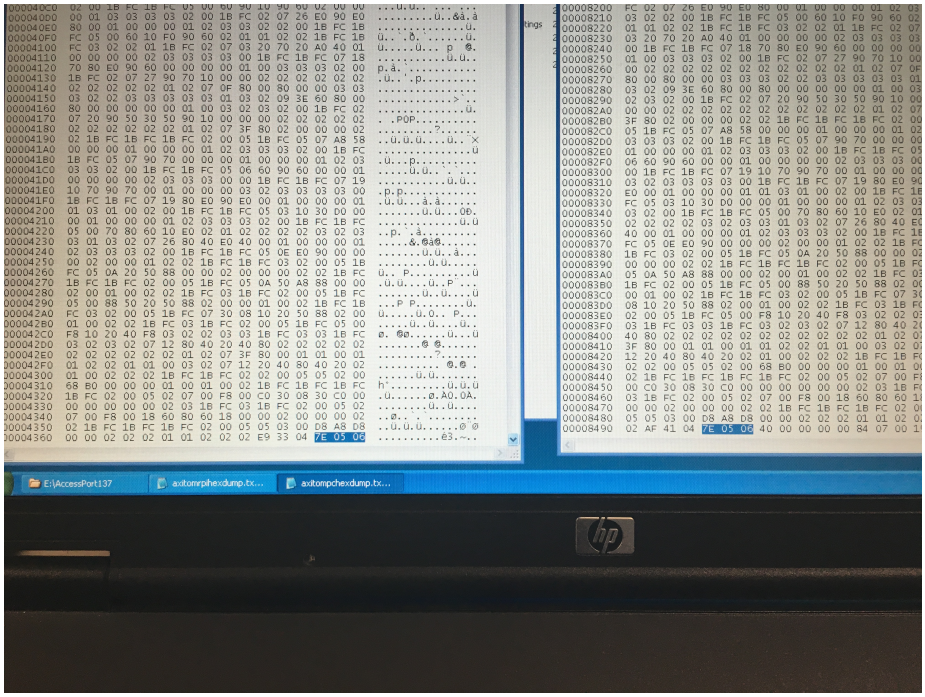
Hertil downloadede jeg et stykke software, der kunne vise hexa decimal værdierne for de beskeder, der blev sendt og modtaget. Herefter kunne jeg så sende beskeder fra en tidligere applikation til kommunikation. Dette kunne jeg så sammeligne direkte med kommunikationen fra min applikation ved aflæsning af begge svar på PC'en 8.2 på side 46.

På den måde kunne jeg undersøge, om de svar der kom fra udstyret var som forventet, eller om jeg havde fejl i min implementering. Man kunne have lavet nogle tabeller med de forskellige inputs og forventede outputs, for mere præ-

Figur 8.1: Selve snifferen

cist at følge black box princippet, men da først CRC funktionen fungerede var det som regel mest et spørgsmål om at fange de små forskelle, der var i SCP, afhængigt af om du snakkede med bootloader eller applikationsdelen i udstyret.

Resten af applikationen er testet uden en decideret strategi, men mere for hver gang en ny funktion er implementeret, som også har fanget en del fejl, men flere kunne nok fanges ved en bedre og mere grundig teststrategi.



Figur 8.2: Et eksempel på output fra sniffe værktøjet fra både prototype og andet udstyr

8.4 Konklusion på test

Vi har valgt mest at fokusere på test af selve kommunikationen mellem prototype og udstyr, og dette er også det vigtigste i forhold til at få en fungerende prototype. Jeg må samtidig erkende at det resterende nok ikke er testet helt grudigt nok. Skal man have prototypen ud i marken, skal man nok overveje om ikke, der skal afsættes lidt resurser til at teste det resterende system ordenligt igennem.

Kapitel 9

Konklusion

De tre overordnede spørgsmål er blevet besvaret på bedst mulige måde. Jeg har fået undersøgt hvordan et system, der kan opfylde opgaven bør se ud. Der er også lavet en del undersøgelser af de delelementer, der udgør den samlede løsning. Vi har samtidig også set på om Struers er opgaven moden på nuværende tidspunkt.

Det første jeg fik belyst var hvilke elementer, der bør indgå i systemet, ved at undersøge hvad et IoT system egentlig indeholder, og giver af muligheder for så at lave en system arkitektur tilpasset en Struers løsning.

Vores fokus på autonom firmware opdatering virkede ret naturlig, hvis man ville benytte eksisterende udstyr i systemet. Dog er der nogle mangler i forhold til versionsstyring, der gør at en vigtig funktionalitet som roll back, ikke er mulig endnu. Hvilket er kritisk hvis man får udgivet en dårlig firmware version.

Vores cloud løsning er ikke klar til produktion i stor skala, men fungerer fint til nogle få prototyper i brug. Her vil man nok skulle ind og kigge på, hvad virksomhedens strategi for cloud computing i det hele taget er, inden man kaster sig ud i det helt store.

Om man skal vælge OSS eller lade være, kan jeg nok ikke konkludere ud fra dette projekt alene. Man er nok nødt til at finde ud af, hvad virksomheden vil på lang sigt i forhold til software udvikling og hardwareplatform.

Test af kommunikationen mellem prototype og udstyr har fungeret godt, men resten af systemet kunne godt have været testet mere systematisk.

Alle aspekter leder frem til modenheden hos Struers. Her må jeg konstatere, at virksomheden har nogle mangler i forhold til at få en fuld fungerende prototype ud nu. Men hvis man forsigtigt starter med at gemme udstyrets datalog i skyen, kan man måske snige en IoT løsning ind af bagvejen på eksisterende udstyr.

Systemet har under hele forløbet (både praktik og projekt) vagt opsigt. Specielt når vi inviterede til præsentationer, hvor vi havde lavet en teaser med de indgående elementer i projektet B.3 på side 56. Men nysgerrigheden har ikke åbnet op for et endeligt projekt, da netop modenheden i virksomheden ikke helt er på plads endnu. Derofor må det nok konstateres, at hvis Struers skal have Remote Connection skal man tage det i små etaper med f.eks. data log i skyen som første skridt. Eller simpelthen vente til man alligevel skal skifte hele den eksisterende platform ud, og så planlægge connectivity med i udviklingsfasen.

Bilag A

Forkotelser

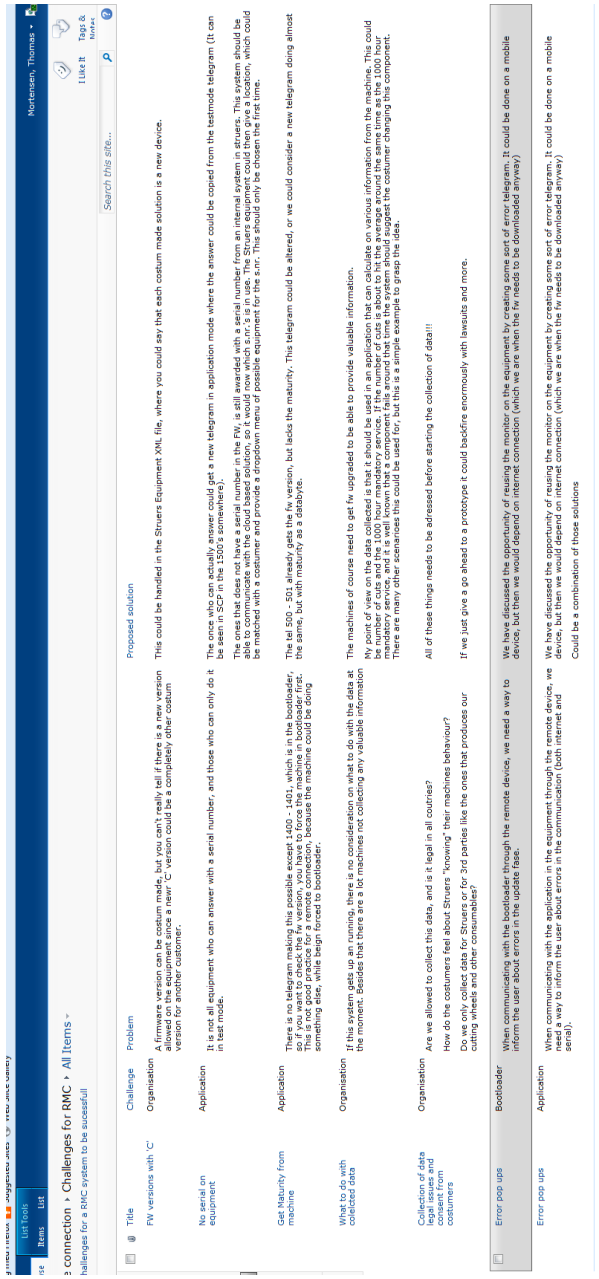
Dette er en liste over relevante forkortelser brugt i teksten. Nogle af de mest gængse termer som PC mm. har jeg valgt ikke at tage med, da det må forventes at de fleste mennesker allerede er bekendte med disse udtryk:

- **AKA** Authentication an Key Agreement
- **FOTA** Firmware Over The Air
- **IoT** Internet of Things
- **M2M** Machine 2 Machine
- **OSS** Open Source Software
- **OTA** Over The Air
- **POC** Proof Of Concept
- **SCP** Struers Communication Protocol
- **SMM** Struers Memory Module
- **3G** Third generation of mobile telecommunications technology
- **4G LTE** Fourth generation Long-Term Evolution

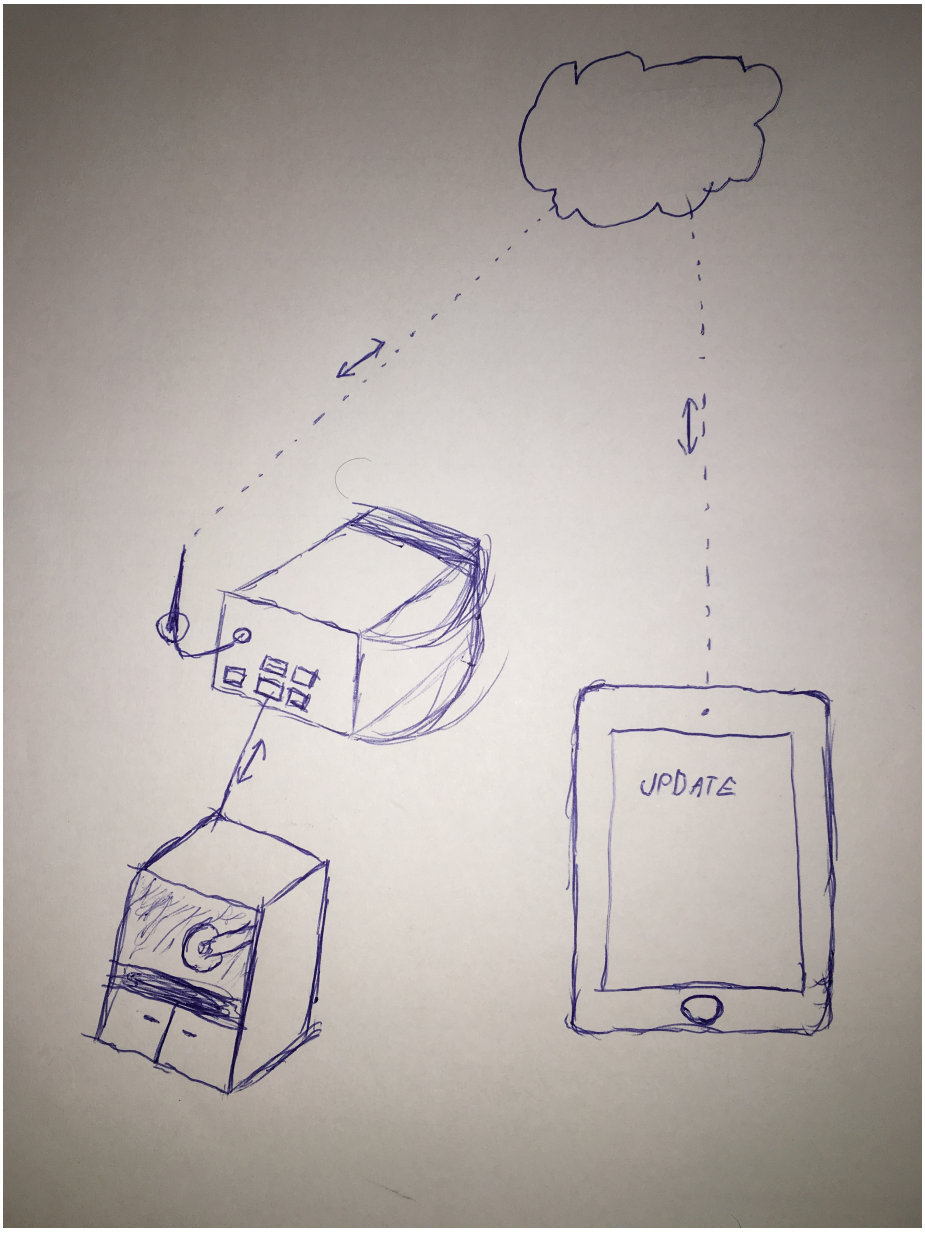
Bilag B

Figurer

I dette bilag, har jeg samlet alle figurer, der ikke har behøvet at indgå direkte i rapporten, men som stadig er vigtige for forståelsen af det figuren hører til.



Figur B.1: Liste på Struers Intranet, der beskriver tiltag for at virksomheden kan sætte en prototype i verden



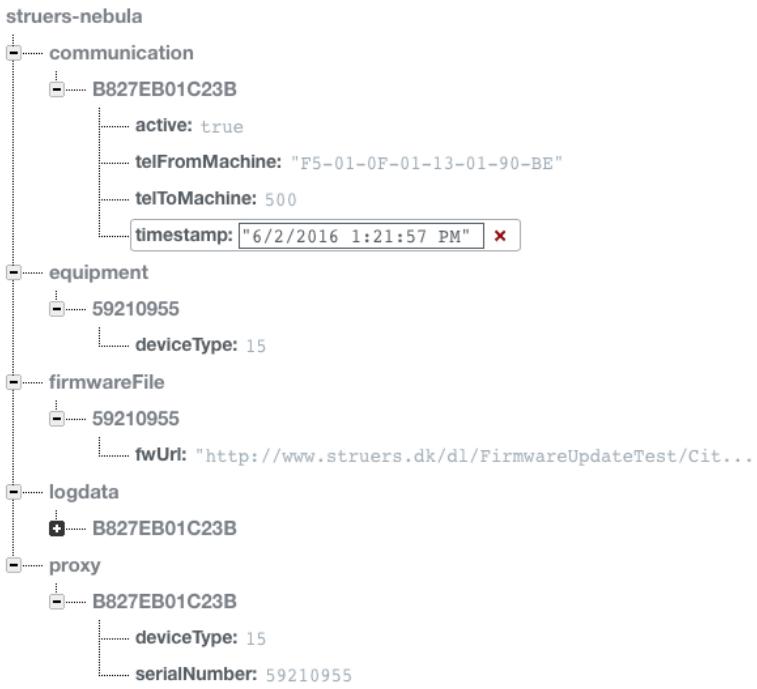
Figur B.2: Tidlig idé til koncept

Figur B.3: Illustration af systemopbygningen

Dir. = '®': Telegram requesting or giving information
Dir. = '¬': Telegram reply with requested information

Nr.	Type	Dir.	Data bytes
1	ID Req.	®	None
2	ID	¬	Data 1: Device Type: 1: TegraPol-11 2: TegraPol-15 3: TegraPol-21 4: TegraPol-25 5: TegraPol-31 6: TegraPol-35 10: TegraForce-1 11: TegraForce-3 12: TegraForce-5 19: TegraForce-1 or 3 (unknown at start up) 20: TegraDoser 5 30: TargetMaster 31: TargetForce (PCB in TargetMaster) 32: TargetDoser 33: TargetX 40: ViaSampler 50: RecSystem 60: Bootloader Data 2+3: Software version No.: 0-99: Pre-release versions >100: Official release versions Data 4+5: Database version No. / Hardware version No.
90	Select service mode	®	None Will only have effect during start up phase

Figur B.4: eksempel på en telegram beskrivelse i SCP



Figur B.5: Udsnit af Firebase Dashboard

Bilag C

Kode

Koden er afleveret med som 2 zipfiler på campusnet. Den ene `proj.zip` indeholder kode til prototypen, der skal installeres på en Raspberry Pi med Raspian. Dette er både kode udviklet i C# og i python. Disse programmer skal ligges i en mappe i roden på Raspian for at køre ved boot.

Dette er der ikke lavet en vejledning til endnu, men projektet skal overleveres til Struers, så de kan viderudvikle på det, og her vil det være en ting, der skal laves.

Den anden fil `Struers Nebual.zip` indeholder frontend programmet, som er en iOS app, og kræver derfor X-code for at kunne eksekveres.

Generelt fungerer ingen af delene, hvis der ikke er noget Struers udstyr koblet på, så koden er kun til gennemsyn for de nysgerrige.

Litteratur

[1] I. Corporation, "The impact of the internet of things on product development," pp. 1–11, 2015.

[2] Cisco, "Capitalizing on the internet og things," *Cisco Jasper white paper*, 2016.

[3] T. Institut, "Anvendelse af big data i produktionsvirksomheder," *Analyse og Erhvervsfremme*, 2014.

[4] Version2. Myriader af standarder hæmmer udviklingen af internet of things. [Online]. Available: https://www.version2.dk/artikel/myriader-af-standarder-haemmer-udviklingen-af-internet-things-625419

[5] ——. Softwarehus: Glem standarder til iot i industrien. [Online]. Available: https://www.version2.dk/artikel/softwarehus-glem-standarder-til-iot-i-industrien-732298

[6] A. S. Tanenbaum and M. Van Steen, *Distributed systems : principles and paradigms*. Pearson, Prentice Hall, 2007.

[7] T. Mortensen, "Internship at struers," pp. 1–15, 2016.

[8] Wikipedia. Authentication and key agreement (protocol). [Online]. Available: https://en.wikipedia.org/wiki/Authentication_and_Key_Agreement_(protocol)

[9] stackoverflow. Why are interpreted languages slow? [Online]. Available: http://stackoverflow.com/questions/1694402/why-are-interpreted-languages-slow

[10] Mono. Mono. [Online]. Available: http://www.mono-project.com

[11] T. Zilly, "Skal man købe eller udvikle selv?" *Nyhedsmagasinet Elektronik & data*, 2016.

[12] D. Murphy, "Give hardware new life with firmware updates." *PC World*, vol. 27, no. 9, pp. 88–90, 2009.

[13] M. Shavit, A. Gryc, and R. Miucic, "Firmware update over the air (fota) for automotive industry," *Sae Technical Papers, Sae Techni. Paper*, 2007.

[14] G. Jurkovic and V. Sruk, "Remote firmware update for constrained embedded systems," *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (mipro)*, pp. 1019–1023, 2014.

[15] W. Gatliff, "Implementing downloadable firmware with flash memory. a microprogrammer-based approach to firmware updates," *Firmware Handbook, Firmware Handb*, pp. 285–297, 2004.

[16] L. Katzir and I. Schwartzman, "Secure firmware updates for smart grid devices," p. 3, 2011.

[17] Wikipedia. Cyclic redundancy check. [Online]. Available: https://en.wikipedia.org/wiki/Cyclic_redundancy_check

[18] C. Larman, *Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development.* Prentice Hall, 2005.

[19] S. Chhabra and V. S. Dixit, "Cloud computing," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 2, pp. 1–11, 2015.

[20] T. Connolly and C. Begg, *Database systems : A practical approach to design, implementation, and management.* Addison-Wesley, 2001.

[21] L. . Bendsen. Udvikling i titanium - fordele, ulemper og faldgruber. [Online]. Available: https://www.lundogbendsen.dk/artikel/27621/Udvikling_i_Titanium_fordele_ulemper_faldgruber

[22] Nexgendesign. 7 reasons xamarin can be a trouble. [Online]. Available: http://www.nexgendesign.com/7-reasons-xamarin-can-be-a-trouble

[23] E. J. Martin, "Open source." *EContent*, vol. 36, no. 1, p. 14, 2013.

[24] P. Sestoft, "Systematic software testing," p. 17, 2008.