Implementation of modern communication standard on SGU

Christian Mühlendorph

Kongens Lyngby 2013
IMM-B.Eng-2013-13

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-B.Eng-2013-13

Summary (English)

This paper explores the possibilities in and execution of an upgrade to an electrical measurement instrument, a Smart Grid Unit (SGU), to communicate via a modern standard protocol in the industry. The standards in question are the existing protocol, IEC 60870-5-104 and the future, IEC 61850, which covers the design of substations in the electricity supply. IEC 61850 is designed with the notion to automate the surveillance of and reporting from the increasing number of new devices i the electric network. The new devices shall, therefore, be designed to work together and govern themselves intelligently. The project will thoroughly examine the existing possibilities in the new standard protocol and a prototype will be constructed to test the standards functionality. In addition, a study of the hardware changes that should be made to the SGU'en to make it compatible with the standard will be conducted. In addition to the IEC 61850 standard, the project will examine safety hazards, such a system may be exposed to, and what steps you can take to protect it.

The report was written under the assumption that the reader has a background in IT and electrical industry, and possess a deeper understanding of the teknologies in this field.

Summary (Danish)

Denne afhandling omhandler planlægningen og gennemførelsen af en opgradering af et elektrisk måleinstrument, kaldet en Smart Grid Unit(SGU) til at kunne kommunikere via en nyere standard inden for industrien. Standarden, der skal opgraderes fra er IEC 60870-5-104, og opgraderingen, der skal opgraderes til, er IEC 61850, der dækker over udformningen af substationer på el-forsyningen. IEC 61850 er designet med baggrund i behovet for at automatisere overvågning af og rapportering fra den stigende mængde af nye enheder på el-nettet. De skal derfor designes til at kunne arbejde sammen og styre sig selv intelligent. Projektet vil grundigt undersøge mulighederne, der findes i standarden, og der vil blev konstrueret en prototype, der skal bruges til at teste standardens funktionalitet. Dertil kommer en undersøgelse af, hvilke hardwareændringer, der skal fortages på SGU'en for at gøre den kompatibel med standarden. Udover IEC 61850 standarden vil projektet undersøge de sikkerhedsrisici, et sådan system kan blive udsat for, og hvilken skridt man kan tage for at imødegå dem.

Rapporten er skrevet med udgangspunkt i, at læseren har baggrund i IT og el-industrien, og besidder en dybere forståelse af emnerne inden for dette felt.

Forord

Denne Diplom bachelor opgave er udarbejdet i perioden 26. februar til 01. juli 2013 på Danmarks Tekniske Universistet (DTU) ved Institut for Matematisk Modellering og informatik(IMM). Den markerer afslutningen på Diplom-IT uddannelsen og er en afspejling af de gennem uddannelsesforløbet opnåede kompetencer.

Projektet er lavet i samarbejde med elektronikfirmaet Thiim A/S, Transformervej 31, 2730 Herlev.

En stor tak skal gå til Søren Thiim & Paul Styrbæk, der var mine samarbejdspartnere og vejledere i firmaet, hvor både praktik og dette projekt er gennemført. Også et stort tak til min DTU-IMM vejleder Bjarne Poulsen, der har været med til at forme projektets udfald og som - ikke mindst - har været inspirerende, motiverende og hjælpsom igennem hele forløbet.

Lyngby, 01-July-2013

Christian Mühlendorph

Indholdsfortegnelse

Figurer

List of Tables

Kapitel 1

Indledning

Dette kapitel beskriver, hvad der ligger til grund for projektets dannelse, hvem aktørerne er, hvad de håber og ønsker, resultatet af denne afhandling bliver. Dertil kommer en gennemgang af udviklingsmetoden, der bruges igennem projektet, og hvordan denne metode bliver indført i en overordnet projekt plan. Tilsidst vil der være en lille gennemgang af resten af rapportens opbygning.

1.1 Baggrund

Nogle år tilbage udviklede firmaet Thiim A/S i samarbejde med de belgiske jernbaner et system af måleenheder til at overvåge kortslutninger på banestrækningerne. Udbyttet af dette samarbejde fik firmaet til at begynde udviklingen af en Smart Grid Unit (SGU), som bruger een af de patenterede metoder, der blev udviklet til jernbane projektet til måling af overgange i højspændings kabler. SGU'en har fået interesse fra store internationale firmaer og organisationer, selvom det stadig er et ungt produkt. Thiim A/S har derfor en stor interesse i, at deres SGU følger udviklingen i industrien og ønsker derfor, at den bliver opgraderet til den nyere kommunikations standard (IEC 61850), for dels at kunne inddrage nye features i produktet fremtidige udvikling, udvide markedsgrundlaget og at kunne appellere til den størst mulige kundemasse. Forfatteren af dette projekt er

blevet, tildelt denne opgave som en tilføjelse til, at der i hans forgående praktik periode hos Thiim A/S blev arbejdede med at implementere IEC 60870-5-104 på en mikrokontroller i SGU'en.

1.2 Thiim A/S

Thiim A/S blev grundlagt helt tilbage i 1940 af den nuværende ejer Søren Thiims far. Søren selv overtog firmaet i 1971. Firmaet har igennem årene udviklet sig med tiderne og har derfor skiftet branche flere gange. I dag er firmaets primære eksistensgrundlag salg og dybdegående service i industriel IT og specialelektroniske komponenter, som firmaet selv producerer. Det industrielle IT, de sælger, er typisk løsninger med computeren indbygget i touch paneler, i alle størrelser og i design fra hverdagsbrug til hård industribrug hvor komponenterne skal kunne tåle vand, varme og støv. De komponenter, som firmaet selv laver, er typisk intelligente relæ af forskellige typer til f.eks batteri sikring af redningsbåde på skibe. Der laves også specialomskiftere efter kundernes ønsker, som tit kunne være til 20-30 år gammel maskiner. Det høje serviceniveau i firmaet er klart det, der har gjort, at firmaet har kunne holde succesen og kunderne i næsten 75 år, og det er der, de skiller sig ud fra konkurrenterne. Imellem de to hovedområder i firmaet er en mindre udviklingsafdeling, der udover at yde support til de to afdelinger arbejder på nye versioner eller helt nye produkter. Det er denne afdeling, der udvikler SGU'en. Udover erfaringen fra sammenarbejdet med de belgiske jernbaner, har Dong energi også bidraget til udviklingen, idet de har lavet kravspecifikationen og ikke mindst haft flere enheder til test både i deres laboratorium og i felten.

Figur 1.1: Thiim A/S logo.

Website: www.Thiim.com

1.3 Smart Grid Unit - SGU

I et Smart Grid Netværk er der behov for at vide, om netværket er i orden, hvad strømstyrken er, og hvor strømmen bruges. SGU enheden er designet til at give denne information på real tids-basis. Dens form faktor er designet til at overvåge elforsyninger i en transformer/sikringsboks, da den kan monteres på boksens busbar og trække sin forsyningsstrøm derfra.

SGU'en kan overvåge op til 6 x 3 faser, hvor der for hver fase måles spænding, strøm, aktiv og reaktiv effekt og retningen af strøm i hver af de 18 kabler. Derudover måles frekvensen og temperaturen i den centrale enhed via dens egen forsyningsstrøm. Af sikkerhedsmæssige årsager, og for at undgå elektriske farer, er kommunikationen mellem sensorerne og hovedenheden udført ved hjælp af optiske fibre, der nemt kan skæres i den ønskede længde. Denne optiske real-tids kommunikation er patenteret, og da kommunikationen er digital og med automatisk styrkekontrol er kalibrering ikke nødvendig. De målte og beregnede værdier overføres til en central computer/SCADA for yderligere evaluering i Smart Grid Netværket. Kommunikationstilstanden kan enten være Ethernet eller GSM/GPRS-netværk. De nuværende kommunikations protokoller er SMS, modbus(TCP) eller IEC 60870-5-104. SGU'en kan - alt efter model - have 3-18 eksterne sensorer til at kvalificere og udvide de målinger, der tages i selve boksen. Det betyder cirka 90 mål punkter på den fulde model. Det er disse måleværdier, der skulle kunne tilgås via IEC 61850 standarden, der arbejdes på at implementere i denne afhandling og denne SGU, dette projekt vil har sit fokus på. Nærmere bestemt er det opgraderingen til den nyere system kommunikations standard IEC 61850, som der undersøges mulighederne for. Igennem projektet vil en mulig hardwareopgradering blive undersøgt. Denne HW-opgradering vil kunne give SGU'en mere regnekraft til at kunne håndtere den nye standard.

Figur 1.2: SGU Boks

1.4 Konkurrenter til SGU

Den primære konkurent til Thiim A/S's Smart Grid Unit er Powersense's DISCOS® system, hvori det er den del, der hedder SMARTCOM RTU, der kan sammenlignes med det, der arbejdes med i dette projekt. Powersense RTU'en minder på mange punkter om en RaspberryPi[Fou13b],den er også baseret på en ARM® CPU hvorpå der køres en linux kernel og har mange af de samme I/O tilslutningsmuligheder. Den store forskel på SMARTCOM RTU, og Thiim SGU'en er, at Powersense allerede har IEC 61850 standarden implementeret, plus mange af de andre fordele der ligger i at bruge et linux system, som SSH, FTP, USB mm., frem for en programmeret microcontroller som SGU'en bruger nu, hvor funktionerne er sparsomme og svære at implementere. Der findes en række firmaer, så som Siemens og AAB der sælger enkeltstående servere, hvortil der kan tilsluttes eksterne målere, men hvor de så ikke på sammen måde er et komplet system. Det er her, Thiim's SGU skiller sig ud ved, at den er en komplet løsning, hvor server, beregningsenheder og sensorer, er designet samlet, og let kan installeres i f.eks en sikringsboks.

1.5 Motivation

I de fleste el-net rundt i verden, er der meget få målepunkter i nettet, hvilket gør det problematisk for el-selskaberne at have et fuldt overblik over, hvad der sker på nettet. Det er et udtalt ønske fra mange af elselskaberne at få bedre mulighed for at overvåge, hvordan strømmen bevæger i nettet. Faktorer som, at der er begyndt at blive tilsluttet flere privatejede micro-kraftværker til deres net i form af solceller og vindmøller mm. betyder, at en mere systematisk overvågning vil kunne give dette et højere udbytte, og ikke mindst er nødvendigt for at sikre, at nogle områder ikke bliver overbelastet. En anden faktor er, at flere el-selskaber har kabler i deres netværk, der er langt over deres forventede levealder, hvilket hæver risikoen for fejl i nettet. Det vil derfor også være strategisk klogt at overvåge de områder, hvor de kabler er.

Den primære motivation bag dette projekt ligger i at få en dybere indsigt i IEC 61850 standarden, da den forventeligt vil blive markedsstandard inden for kort tid. Ydermere er det ønsket at få større indsigt i spændingsfeltet mellem implementering af hardware komponenter og computere og hvordan udviklingen i dette område foregår.

Motivationen for firmaet Thiim A/S i dette projekt ligger i at udbygge deres produkts funktionalitet, og dermed få større succes med produktet, da det ville

kunne sælges til en større kundegruppe og ikke mindst tillade deres nuværende kunder at opgradere deres nuværende system.

Dong Energi som er en af de store kunder til SGU'en, har efterspurgt at der skal være øget sikker på enheden, da der i stigende grad er en tendens til at vitale systemer i infrastrukturen bliver hacket, hvilket kan have katastrofale følger.

1.6 Vision

Visionen er at udvikle en prototype IEC 61850 server, der kan tilsluttes Thiim's Smart Grid Unit. Denne prototype skal være designet med henblik på, at skulle kunne indarbejdes i den allerede eksisterende SGU boks. Jeg forventer ikke at kunne nå at implementere 100% af IEC 61850 standarden, men at kunne få lavet en server, hvorfra målinger kan hentes, og hvis det er muligt også logging. Jeg forventer dog, at selvom der er tale om en prototype at den skal designes med henblik på at, køre helt autonomt, og at al servicering kan ske via fjernkontrol (også kaldet remote management). Produktet af dette projekt skulle gerne blive et proof of concept, hvor de vigtigste dele er lavet, og mindre vigtige er undersøgt og beskrevet. Som Dong efterspørger vil der blev sat fokus på sikring og beskyttelse af enheden, og muligheden for at kryptere kommunikationen vil blive gjort muligt.

1.7 Problemformulering

På baggrund af ønsket om at Thiim's SGU, skal kunne konkurrere med Powersense SMARTCOM RTU, og ellers implementer en moderne kommunikations standard,er der udarbejdet følgende problemformulering.

- Hvad er fordelene ved IEC 61850 standarden over den gamle 60870.
- Hvad består den nuværende hardware af.
- Hvilken hardware udfordringer er der i opgraderingen.
  - Er det muligt at opgradere SGU'en til IEC 61850, uden det kræver en opgradering af hardware.
  - Hvordan kan hardwaren opgraderes.
  - Hvilke opgraderinger vil fremtidssikre SGU'en bedst.

  - Hvilke ekstra funktioner vil en opgradering give.

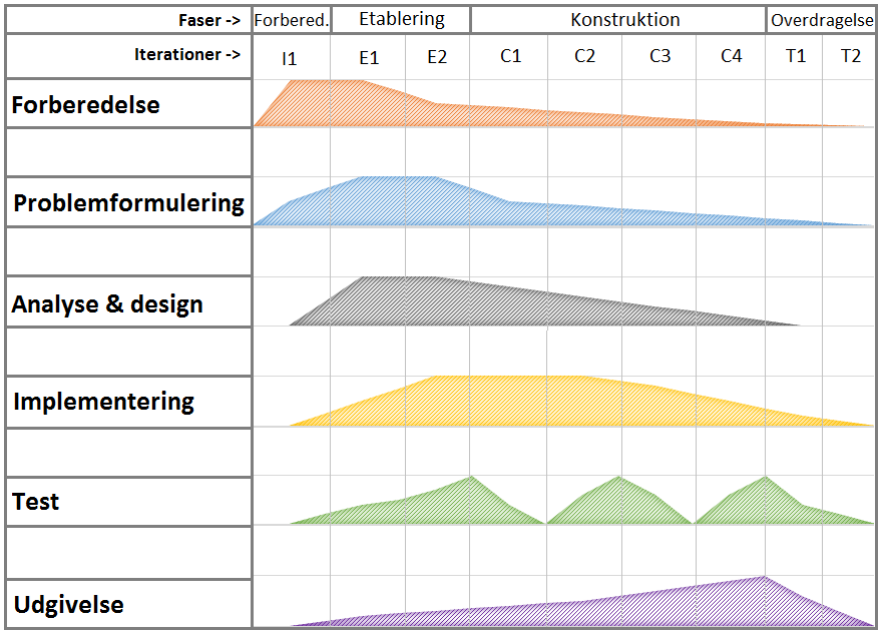
- Hvilken dele af IEC 61850 er relevante at implementere.
  - Er lokal logging på SGU'en muligt.
  - Hvilken sikkerhed er der i IEC 61850.
  - Hvilken kommunikationsform skal der bruges.
- Er det muligt at sikre SGU'en, imod ondsindet angreb.
- Hvilken type angreb kan der ske og hvad er faerne i dem.
- Hvilken ønsker har SGU'ens brugere til denne opgradering.

1.7.1 IEC 61850 Standarden

IEC 61850[IEC13] er den nye standard indenfor netværks kommunikation for enheder i forbindelse med el-nettet, dvs. transformerstationer, relæ- og sikringsbokse, mikro-elproducenter mm. Denne standard tager et stor skridt fra de foregående standarder, da den er tiltænkt et bedre samarbejde enhederne imellem, men også fordi den bygger på brug af eksisterende teknologier som Ethernet netværk frem for serial og teledata kommunikation. Det gør IEC 61850 billigere at installere, konfigurere og vedligeholde. I IEC 61850 behandles alle data som objekter, hvilket giver store muligheder for at implementere funktioner, det ellers ville være meget upraktiske at realisere. Der vil blev taget et dybere kik på IEC 61850 i sektion 2.1 af denne rapport.

1.8 Udviklingsmetode - Unified process

Som udviklingsmetode til systemmet er der valgt at bruges systemudviklingsprocesserne fra Unified Process[Lar01] der også populært kaldes UP. UP er en iterativ udviklings metode som udmærker sig ved at have fokus på objektorienteret udvikling, hvilket er passende, da IEC 61850 også har fokus på at være objektorienteret. UP bemærker sig ved, at den er grundlægende opdelt i 4 arbejdesfaser, hvor der i hver af dem er et primært arbejdesområde, men at der også arbejdes på de andre områder, på sådan en måde at der er en sammenhængende flow i arbejdsforløbet.



Figur 1.3: Diagram over aktiviteten i Unified Process.

1.8.1 Forberedelse

I forberedelses fasen arbejdes der først og fremmest på at indsamle informationer og på at tilegne sig viden inden for emnet. Da der her er tale om standard, vil den primære kilde til information være de officielle dokumenter fra Cern. Dernæst tages der et kik på hvad der findes af allerede eksisterende løsninger, og om disse ville kunne bruges helt eller delvis i sammenhæng med Thiim's SGU. Andre ting, der også undersøges i denne sektion, er hvilke værktøjer der vil give det bedste endelige produkt, og vurderingerne af, om de bedste løsninger også giver mest mening ud fra både et design synspunkt og et produktionsøkonomisk synspunkt.

1.8.2 Etablering

I denne sektion tages de konklusioner, der blev lavet i forgående sektion og sættes sammen i et overordnet design, hvorfra konstruktionen af systemet kan tage udgangspunkt. Den største del af denne sektion er, at der opstilles krav til

systemet, hvor nogle af dem skal være opfyldt, mens andre er ønskede krav.

1.8.3 Konstruktion

Konstruktions fasen er meget selvsigende, idet det er her, selve koden i produktet skrive. Det er også i denne fase, at systematiske test udføres for at sikre, at systemets dele fungerer som designet.

1.8.4 Overdragelse

I denne fase arbejdes der på at tage det nyudviklede produkt og forberede det på at blive sat i produktion/funktion. Der vil i denne fase foregå en række praktiske test, og ikke mindst vil der blive lavet ændringer, der ikke blev forudset, da systemet blev designet.

1.8.5 Projektplanlægning

Projektplanen vil tildeles blive bygget på de principper, der er gennemgået i foregående afsnit om UP. Dog er der på grund af forgående viden om størrelsen og omfanget af IEC 61850 givet ekstra meget til forberedelsesfasen, og selve planlægningen vil så først rigtig begynde når den fornødne viden er indsamlet. En detaljeret projektplan vil blive opstillet i sektion. 2.6 efter analysen er gennemført. Men en meget detaljeret tids plan vil ikke blive lavet da der kun er en person der arbejder på dette projekt og der derfor ikke er brug for, at have aftaler om hvornår dele skal være færdige til sammen bygning.

1.9 Rapportens opbygning

Denne Diplom bachelor Rapport er skrevet i LaTeX, og benytter sig af den opsætning der af IMM,DTU er givet[Chr13]. Opsætningen IMM har givet er baseret på LukeThesis, med nogle ændringer laver af IMM, og nogle små ændringer er blevet lavet af forfatteren.

Rapporten er opsat så den er delt op i nogenlunde de sammen sektioner som Unified Process, for at give en sammenhæng imellem arbejdet der blev udført.

Hvert kapitel vil have en delkonklusion, der opsummere kapitlets hovedpunkter og hvordan de vil blive brugt i det efterfølgende arbejde.

For at undgå misforståelser i oversættelsen fra den på engelske skrevne standard, vil de engelske navne og udtryk blive brugt, selvom der skrives på dansk.

Rapporten vil indeholde nogle kode eksempler, men hele koden vil ikke blive vedhæftet, da det vil være meget uoverskueligt i printet form. Derfor refereres der til den med projektet afleverede zip-fil, hvori den komplette kode kan findes.

Kapitel 2

Analyse

I dette kapitel vil de forberedende trin nævnt i unified process blive gennemgået for at kunne danne et kendskab til de teknologier og metoder der omhandler opgraderingen til IEC 61850 og skabe det nødvendige overblik. Jeg vil overordnet gennemgå IEC 61850's hovedpunkter, og de dele af den, der relaterer specifikt til den type enhed, Thiim's SGU er.

2.1 IEC 61850

Lige siden de første el-net blev sat op, har der været brug for at overvåge nettene, og ikke mindst have muligheden for at kontrollere, hvad der sker i dem. Dette har før i tiden være gjort med, for i dag meget primitive metoder, der enten byggede på brug af almindelige telefonlinjer eller analog, serielle forbindelser. Disse forbindelser egner sig rigtigt dårligt til større anlæg, da de ikke skaleres godt, idet de kræver en konstant forbindelse. Med dette problem for øje, begyndte arbejdet på IEC 61850 [IEC13], som bygger på moderne digitale teknologier både i kommunikationen, men også på mange andre punkter. Det største punkt her er, at systemer begynder at have intelligente elektroniske enheder kaldet IED'er. Disse skal have muligheden for at kunne arbejde sammen og danne et samlet netværk, der er lettere at overvåge og kontrollere.

Som det kan ses ud fra det nedenstående overblik af standarden, er IEC 61850 en meget stor standard. Den er derfor opdelt i adskillige, mindre dele. Dette projekt vil derfor ikke kunne nå at have det hele med, og derfor ligger det primære fokus på 61850-7-10 delen, men andre område af standarden vil blive berørt.

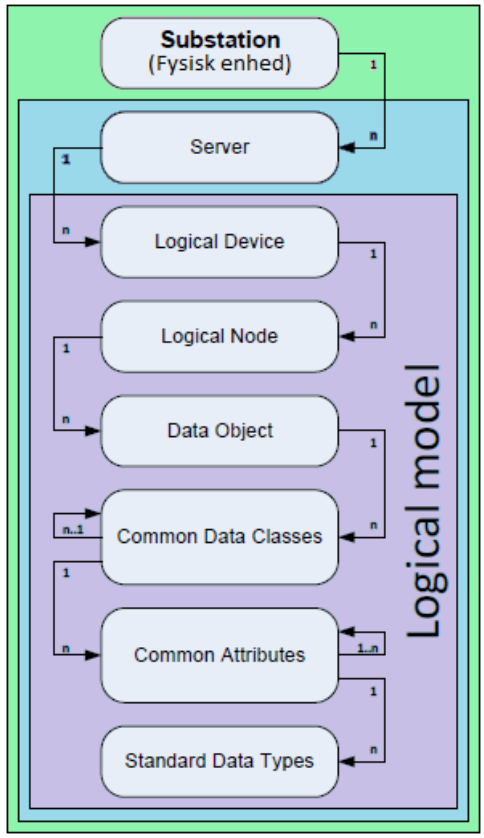
2.1.1 Overblik

- IEC 61850-1: Introduction and overview
- IEC 61850-2: Glossary
- IEC 61850-3: General requirements
- IEC 61850-4: System and project management
- IEC 61850-5: Communication requirements for functions and device models
- IEC 61850-6: Configuration language for communication in electrical substations related to IEDs
- IEC 61850-7: Basic communication structure for substation and feeder equipment
  - IEC 61850-7-1: Principles and models
  - IEC 61850-7-2: Abstract communication service interface (ACSI)
  - IEC 61850-7-3: Common Data Classes
  - IEC 61850-7-4: Compatible logical node classes and data classes
  - IEC 61850-7-10: Communication networks and systems in power utility automation - Requirements for web-based and structured access to the IEC 61850 information models
- IEC 61850-8: Specific communication service mapping (SCSM)
  - IEC 61850-8-1: Mappings to MMS (ISO/IEC9506-1 and ISO/IEC 9506-2)
- IEC 61850-9: Specific communication service mapping (SCSM)
  - IEC 61850-9-1: Sampled values over serial unidirectional multidrop point to point link
  - IEC 61850-9-2: Sampled values over ISO/IEC 8802-3
- IEC 61850-10: Conformance testing

IEC 61850 er resultatet af mange års erfaringer inden for felten og danner et solidt grundlag for opbygningen af et intelligent netværk. En af de store dele af IEC 61850 er, at de forskellige enheder skal kunne kommunikere effektivt sammen. Dette samarbejde imellem de forskellige enheder er vigtigt, da elnettet er blevet så stort, at det er umuligt for et menneske at overvåge det, men også fordi der er stor mulighed for at kun spare meget energi. Disse besparelser kunne f.eks være et intelligent solcelle anlæg, der fortæller husets elbil, når solen skinner, så bilen kan planlægge, hvornår det er billigst at lade op. Dette samarbejde gøres muligt ved, at standarden har en objektorienteret datamodel, som er ens for alle enheder, og hvor objekterne er selvbeskrivende.

2.1.2 Datamodel

Datamodellen, som er en af de grundlæggende punkter i IEC 61850 standarden, og som er illustreret i figur. 2.1, bygger på en træstruktur, hvori alle ting som enheden dækker over, systematisk placeres i de rette kategorier. Denne virtuelle inddeling, giver således muligheden for at hele substationer kan visualiseres til en enkelt enhed, via de før nævnte enhed til enhed samarbejde.



Figur 2.1: IEC 61850 data model.

2.1.2.1 Server

Overordnet har alle enheder en server, hvortil der kan tilsluttes. Det er også muligt for en enhed at have flere servere, der så tilbyder forskellig funktionalitet.

En server konfigureres til at kunne tilbyde en række af de services, der er defineret i IEC 61850. Der er her tale om funktioner, som at kunne hente/sende objekter, alarmer, log mm. Ligesom en enhed kan have flere servere, kan en server også repræsentere flere fysiske enheder, kaldet Logical Devices (LD).

2.1.2.2 Logical device - LD

Et logical device er en virtuel repræsentation af en fysisk enhed. Dette objekt over en fysisk enhed, kan på samme måde som en server indeholde flere logical nodes, der beskriver den fysiske enheds funktionalitet. Fælles for alle LD er at de har to obligatoriske logical nodes, LLN0 som indeholder information om den virtuel enhed som LD'en repræsentere, og LPHD indholder informationer om den fysiske hardware enhed, LD'en repræsentere.

2.1.2.3 Logical node - LN

En logical node beskriver en funktionalitet i en enhed. Der er fra IEC 61850 standardens side givet en bestemt inddeling af, hvilke typer der findes i denne kategori. Dette er lavet for at give overskuelighed over, hvilken type af funktion der er tale om. Der er overordnet 13 hovedkategorier, hvori der i hver findes adskillige underkategorier. De 13 hovedkategorier ses i nedenstående tabel:

Logical node funktionaliteter	
A	Automatic Control
C	Supervisory Control
G	Generic Function References
I	Interfacing and Archiving
L	System Logic Nodes
M	Metering and Measurements
P	Protection Functions
R	Protection Related Functions
S	Sensors, Monitoring
T	Instrument Transformer
X	Switchgear
Y	Power Transformer and Related Functions
Z	Further (power system) Equipment

Table 2.1: Logical node funktionaliteter

Da SGU'en er et måleinstrument, vil det være 'M' funktionaliteten, der primært vil blive benyttet i dette projekt. Dog vil der også ses de obligatoriske 'L' noder. Kikker man i IEC 61850-7-4, vil man kunne finde den præcise type, der skal bruges til målinger på el-nettet. Der er her tale om MMXU klassen som der ses i figur. 2.2, hvori de præcise elementer (data class), der passer på SGU'ens målinger, er markeret.

MMXU class				
Attribute Name	**Attr. Type**	**Explanation**	**T**	**M/O**
LNName		Shall be inherited from Logical-Node Class (see IEC 61850-7-2)		
Data				
Common Logical Node Information				
		LN shall inherit all Mandatory Data from Common Logical Node Class		M
EEHealth	INS	External equipment health (external sensor)		O
Measured values				
TotW	MV	Total Active Power (Total P)	O	
TotVAr	MV	Total Reactive Power (Total Q)	O	
TotVA	MV	Total Apparent Power (Total S)	O	
TotPF	MV	Average Power factor (Total PF)	O	
Hz	MV	Frequency	O	
PPV	DEL	Phase to phase voltages (VL1VL2, ...)	O	
PhV	WYE	Phase to ground voltages (VL1ER, ...)	O	
A	WYE	Phase currents (IL1, IL2, IL3)	O	
W	WYE	Phase active power (P)	O	
VAr	WYE	Phase reactive power (Q)	O	
VA	WYE	Phase apparent power (S)	O	
PF	WYE	Phase power factor	O	
Z	WYE	Phase Impedance	O	

Figur 2.2: LN MMXU klasse tabel, fra IEC 61850-7-4.

2.1.2.4 Data class

Data class beskriver et informations element af en bestemt type. Det kan f.eks være den målte spænding på en fase. En data class kan både indholde flere data attributes og flere data classes, hvilket betyder, at der her kan findes flere niveauer i træstrukturen, som hele modelen er bygget op på.

2.1.2.5 Data attribute

En data attribute er det sidste objekt i modeltræet, og det vil være her, de faktiske måleværdier vil være at finde. De er i dette objekt let genkendelige, da der både er definineret navn, type og evt. beskrivelse, men også dertil giver et kik tilbage i modeltræet deres præcise placering.

2.1.3 Kommunikation

IEC 61850 dikterer ikke en specifik kommunikations protokol, da standarden ønskes holdt uafhængig af teknologiske ændringerne og udviklingen i kommunikationstyper. Den beskriver i stedet en abstrakt tilgang til, hvordan kommunikationen imellem enheder finder sted, som kaldes ACSI (se nedenfor). Den nævner dog stadig MMS som en mulig kandidat til kommunikationsstandarden.

2.1.3.1 Abstract Communication Service Interface (ACSI)

ACSI er den abstrakte kommunikationsmodel, der er beskrevet i IEC 61850-7-2, der specificerer, hvordan tilgangen til den førnævnte datamodel foregår. Den dækker over lang række af forskellige funktioner, der er tæt bundet til datamodellen og giver muligheden for at hente og skrive data i den. Der er i den specificeret, hvorledes tilgangen til de enkelte elementer og deres evt. log bør foregå, og hvordan resultaterne skal formidles. Der er i den bla. funktioner til at hente nogle overordnede modeler af enheden, hvori de forskellige logical devices kan lokaliseres. Der er selvfølgelige også muligheden for at hente enkelte elementers nuværende værdier. En sammensætning af flere kald vil kunne give specielle samlinger, hvilket giver en klient muligheden for at designe kald, der er tilpasset et bestemt formål.

2.1.3.2 Generic substation events

Generic substation events(GSE), er en funktion i IEC 61850, der tillader, at en IED kan sende en besked ud til alle enheder på substationen. Denne besked kan enten være nogle værdier i en model, der kaldes Generic Object Oriented Substation event(GOOSE), hvor andre IED eller lign. kan bruge dem til at udføre en handling. Den anden beskedtype er Generic Substation State Event(GSSE), der fortæller, at der er tale om, at en IED skifter stadie, og at andre skal skifte i henhold til det, er sket. Det kunne f.eks være en hel transformer station, der

går fra frakoblet til online. Disse er ikke funktioner, det er tiltænkt, at SGU'en skulle kunne i denne fase, men i fremtiden kunne det f.eks bruges til, hvis der er overgang i netværket, at sende en besked ud om nogle relæer skal slukkes.

2.1.3.3 Manufacturing Message Specification

Manufacturing Message Specification(MMS) er en standard for kommunikation tiltænk brug på fabrikker. IEC 61850 forslår, at ACSI kan realiseres med denne protokol. Denne specifikation er designet til at have en realtids kommunikation imellem enheder på en fabrik og er ikke tiltænkt til at skulle bruges over internettet, selvom det dog er muligt via tunnelforbindelser.

2.1.4 IEC 61850-7-10 Requirements for web-based

Er en ny udvidelse til standarden, hvor der arbejdes på at opsætte en standard for kommunikation over internettet via de allerede gennemtestede protokoller HTTPS 2.3.4 og TCP, der er fundamentet i World Wide Web. Denne standard er stadig på 'Draft'-stadiet, så den er ikke komplet og vedtaget endnu. Dog er dette en vigtig udvidelse, idet den egner sig til det formål, SGU'en har brug for, nemlig kommunikation via internettet, der er sikret, men ikke nødvendigvis skal være realtids kommunikation med millisekund præcision. Den beskriver en metode, hvor ACSI funktioner håndteres i HTTPS url på sammen måde som i REST 2.3.5, og dataobjekter sendes/modtages som XML 2.3.6, JSON eller lign. på sammen måde som HTML sendes i WWW. XML er dog at fortrække, da den deler samme struktur som konfigurationsfilen til IEC 61850, der beskrives i sektion 2.1.6.

2.1.5 Logging og Rapportering

IEC 61850 specificere muligheden for en enhed kan opsamle data lokalt og gemme det til senere afhentning. Det løsner op for, at firmaer skal have servere, der poll'er data konstant og derfor kan køre f.eks på dag-, uge-, eller månedligt skema for datahentning. Nogle enheder kan dog have brug for at kunne give øjeblikkeligt besked om en event. Til dette har IEC 61850 muligheden for, at en enhed kan sende en alarm både til andre enheder på substationen, men også til tilsluttede klienter, der abonnerer på disse.

2.1.6 Konfiguration

En anden stor del af ICE 61850 standarden er specificeringen af et fælles konfigureringsprog, der generaliserer, hvordan en enhed er sat op og hvilke funktioner, den tilbyder. Dette er lavet for at gøre samarbejdet på tværs af enheder lettere at opnå. Konfigureringen forgår via en Subsation Configuration Language(SCL) fil, der er baseret på XML syntaksen, hvori der er defineret elementer til de forskellige opsætningsmuligheder. Ved en kort gennemgang af, hvad en SCL fil indholder, findes der i starten konfigureringsmuligheder for, hvordan der tilsluttes til serveren, hvad det er for en enhed og hvilken funktioner, den tilbyder. Dertil er der en komplet datamodel over, hvilke logical devices serveren indholder og videre ned i deres modeltræ med LN, DO, DA osv. Så der i SCL filen er mulighed for at få overblikket over hele enheden. Det er dog muligt at udlade at lave de komplette objekter i datamodel, såfremt de laves i data template delen af SCL filen er defineret og der refereres dertil, dette er især smart hvis man har flere objekter, der er ens.

2.1.7 Opsummering af IEC 61850

Det er tydeligt, efter at have sat sig ind IEC 61850 standarden, at den er en yderst omfattende, og at den er tiltænkt mange forskellige typer af IED'er mm. Der er derfor kun tale om at Thiim's SGU skal have implementeret en brøkdel af funktionerne, da den i sin nuværende stadie kun er tiltænkt at være en måleenhed, der samler data. Det er planen at ACSI ikke skal udføres med MMS, men med en HTTP service, som det er beskrevet i IEC 61850-7-10, da det egner sig bedre til kommunikation over store netværk og internettet.

2.2 Ondsindet angreb

En af de store bekymringer indenfor alle industrier, der bruger informationsteknologier, er chancen for ondsindet angreb på IT-infrastrukturen og farerne dette bringer, det er derfor vigtig at gøre sig overvejelser om disse, og hvordan man beskytter sig imod dem. Disse angreb kan komme fra alt lige fra konkurrerende firmaer, aktivistgrupper og ligefrem fremmede regeringer, og skaderne, de kan forårsage, kan ikke bare koste penge at udbedre, men kan i værste tilfælde koste liv. Men der findes mange forskellige typer, og der komme hele tiden nye typer af angreb til, så en 100% sikker løsning findes næppe. Men med fokus på vigtigheden af, at der laves en vurdering af, hvilke typer angreb der kunne

ske, og hvilke skridt man kan tage for at beskytte enheden imod dem, kan det lykkes at komme tæt på de 100% sikkerhed.

2.2.1 Brutforce angreb & Distributed Denial of Service

Brute force og DDoS er en type angreb, hvori der bruges stor computerkraft til at overbelaste en enhed, hvor der, nærmere bestemt i Brute force, prøves at få adgang til et system ved at prøve alle tænkelige kodeord hurtigt efter hinanden, og hvori DDoS der blokeres adgang til enheden ved konstant at sende den mere data, end den kan håndtere, og muligvis kombinere det med en exploit, når den er overbelastet. Begge disse kan der forholdsvis let sikres imod, ved at have en firewall, der blokerer for uønsket trafik og ikke har nogle serverporte der kan angribes, og en simpel blokerings algoritme der blokere efter for mange adgangskode forsøg.

2.2.2 Man in the middle

Dette er en teknik, hvor en person aflytter forbindelsen mellem klient og server og opsnapper koder mm.. Også selvom koderne er krypterede, er det muligt at lave playback af dem og ligeledes få adgang. Det er også muligt at beskytte enheder imod dette ved at begge parter har verificerede certifikater (se sektion 2.3.8, så de via handshake med disse certifikater ved, at de ikke snakker med en imellem dem stående person. Efterfølgende opsættes der så en krypteret forbindelse til kommunikationen, så den heller ikke kan lures. Dette er abstrakt, hvad der sker i både HTTPS og netværks tunneller.

2.2.3 Virusser & Trojaner

Disse vil have meget lille chance for kunne ramme dette system da de, pr. design kræver, at brugeren af systemet vil være årsagen til, at de kan fremkomme, og systemet ikke har en bruger, der besøger hjemmesider eller modtager e-mails. Ydermere er det primært noget, der rammer brugerbaserede styresystemer som Windows og Mac OS. Der er registreret virusser, der har angrebet Linux, men dette er et meget lille antal og har slet ikke ramt mikrokontrollere, der kun kører en minimal Linux kernel. Så denne type angreb er der beskyttet imod i selve systemmets design. Dog kan det ikke udelukkes med 100% sikkerhed, at der kan laves en der kan få adgang, men meget usandsynlig, hvis tilslutning til enheden er sikret som nævnt i forrige sektion.

2.2.4 Exploits (0-day)

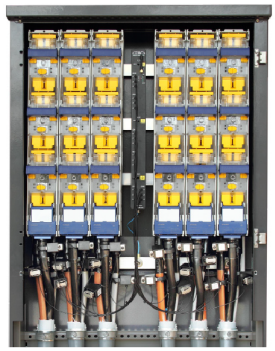
Dette er den sværeste type angreb at beskytte sig imod. Den bygger på, at en fejl i en enhed udnyttes til at overtage kontrollen af den. Det bedste, man kan gøre for at beskytte imod disse, er at sikre, at det slet ikke er mulig at få adgang til enheden, ved enten ovennævnte, sikrede forbindelse eller en helt privat linje, der ikke fysisk er adgang til udefra.

2.3 Teknologier

Der vil i denne sektion gennemgås, hvilke teknologier det er relevant at have kendskab til, før der kan tages stilling til, hvordan opgraderingen af SGU'en udføres bedst muligt. Og det vil være en dybdegående gennemgang af Thiim's SGU enhedens funktioner og opbygning.

2.3.1 SGU Hardware beskrivelse

SGU'en er som tidligere nævnt blevet udviklet i samarbejde med Dong Energy, og derfor har designet af den fysiske boks, taget form således, at den passer til montering i en standard vejside sikringsboks se figur. 2.3.



Figur 2.3: SGU Monteret i sikringsskab.

Det fysiske design kan dog let ændres da hardwaren indeni er modul baseret og derfor kan omrokeres til at give en anden formfaktor. Modulerne i boksen er: strømforsyning, spændings-board, main-board og 1 -6 stk sensor-boards alt efter konfiguration. Strømforsyningen og spændings-boardet er tæt forbundet, sådan at spændings-boardet kan måle spændingen på de tre faser og deres frekvens. Spændings- og frekvensmålingerne styres af en lille mikrokontroller på 16 bit, som der så sender dem videre til main-board via en I^2C-bus. Main-boardet består af en central 32 bit mikrokontroller, der har et par eksterne F-ram, temperatursensorer

og slave mikrokontroller. Main-boardet har også - alt efter konfiguration - enten et GPRS/3G modem, der kan sende målingerne via sms/Modbus-TCP, eller et Ethernet server board, der håndterer IEC 60870-5-104 kommunikation. Det sidste printboard, man finder i SGU'en er sensor-boardet. Dette består af 3 ens kredsløb, et for hver fase. Hver af disse kredse består af en optisk læser tilsluttet en 16 bit mikrokontroller, der dekoder signalet modtaget fra den tilsluttede sensor, og laver de nødvendige beregninger, så de faktiske målinger kan videregives i systemmet. De tre kredse på sensor boardet virker uafhængigt af hinanden, så skulle en fase blive frakoblet, vil målinger på de andre forsætte. Hver sensor-kreds beregner den gennemsnitlige og højeste strøm imellem hver polling, main-boardet laver, samt den aktive og reaktive power og deres retning. Alle sensor-boards målinger hentes igen via en I^2C-bus, som styres af Main-boardet. Målingerne kan ske helt ned til en gang i sekundet. Selve sensormålingerne foregår med 5KHz. Den sidste del af systemmet er selve sensoren, der sættes på kablet, der ønskes målinger på. Denne sensor kan ses på figur. **??**. Sensoren er den del, der gør Thiim's unik sammenlignet med andre selskabers målere, da den bliver strømforsynet via det kabel, den måler på. Det gør den ved, at den bøjle, der sidder rundt om kablet, opfanger det elektromagnetiske felt og leder det igennem et sæt af spoler, hvori det omdannes til strøm via induktion. At sensoren er selvforsynet, betyder at der ikke skal bruges nogen form for strømledende kabel imellem den og SGU boksen, hvilket er en vigtig sikkerheds foranstaltning imod kortslutning, men også at installationen forenkles, da de optiske kabler der bruges i stedet let kan klippes til i korrekt længde. Det er denne del, der også bruges i det førnævnte system lavet til de belgiske jernbaner. Forskellen er dog, at de kun har brug for at vide, om der er kortslutning på nettet, så den hurtigt kan lokaliseres og udbedres. Dog skal det sige, at sensoren kun kan bruges på kabler med strømme over 2,5A, da der skal være lidt liv i linjen, før der dannes nok strøm i sensoren. Som sensoren er i dag kan den måle op til 500A før metallet i bøjlen er mættet, og højere værdier kan derfor ikke måles korrekt, men Thiim arbejder i øjeblikket på en ny version af sensoren der kan gå helt op til 800A. Ved opgradering af systemet skal der tages højde for, at målingerne skal kunne hentes via den førnævnte I^2C-bus, da det er via den alle systemdele er forbundet.

Figur 2.4: SGU Sensor enhed.

2.3.2 Hardware muligheder

Da det nuværende system ikke vurderes egnet til at blive opgraderet til en så omfattende standard, er det nødvendigt at lave en analyse af, hvilke muligheder der findes for at opgradere SGU'en.

Figur 2.5: SGU Vægmonteret på transformer station.

2.3.2.1 Mikrokontroller

Systemmet er allerede designet omkring en styrende mikrokontroller, så der vil hardware - fra et hardware synspunkt - være det letteste at implementere det på. Dog er her en række softwaremæssige udfordringer, der nedvejer denne løsning. For det første er IEC 61850 en objekt- og modelbaseret standard, hvilket ikke implementeres let på en chip, der ikke har stor hukommelse. For det andet er de nuværende mikrokontrollere kun 80 mHz i den hurtigste, hvilket kan give problemer i hastigheden på svartider, i den større standard. Ønsket om at implementere sikkerhedsprotokoller taler heller ikke godt for brugen at denne type implementation. Generelt give manglen på et styresystem, der kan håndtere mere end en ting af gangen, en stor udfordring for udvikling af IEC 61850 til denne metode. Der er mulighed for at køre et minimalt realtime operating system(RTOS), men igen dets mangel på de grundlæggende protokoller i IEC 61850 som f.eks https, gør denne løsning meget tidskrævende.

2.3.2.2 Field-Programmable Gate Array (FPGA)

En FPGA er en utrolig smart type af integreret elektronik, da en FPGA chip kan programmeres til at udfylde flere forskellige funktioner, som både en mikrokonkontroller, hukommelse og logisk elektronik. Ikke mindst kan den omprogrammeres igen og igen, hvilket giver den en stor fleksibilitet mht. fremtidig opgradering. Disse funktioner er mulige, da en FPGA er et kæmpe array af forskellige typer af programmerbare kredse, der kan tilsluttes hinanden efter behov. FPGA'ens

svaghed ligger i, at det er et utroligt dyrt stykke værktøj - og kompliceret at udvikle.

2.3.2.3 Integreret computer

Muligheden for at sætte en hel computer ind i SGU boksen kan muligvis lyde skræmmende, men udviklingen inden for dette område har taget enorme skridt de sidste par år, hvilket betyder, at man kan få komplette computere som system on a chip(SoC). Det er derfor muligt at få en chip for mindre end 4cm^2, der både har en kraftig cpu, men også har ram og netværk. Et andet stort skridt inden for denne felt er, at udviklingen inden for bla. smartphones har betydet, at disse SoC'er ikke har brug for hverken passiv eller aktiv køling, hvilket er en vigtig faktor i et system, der skal kunne monteres i små skabe, og bør køre i årevis dér uden opsyn. Det, der taler for denne løsning, er, at det vil være muligt at køre et helt styresystem som Linux eller Windows i boksen, hvor der ikke blot er mulighed for at køre flere samtidige processer, men også er et rigt udvalgt af ekstra funktionalitet som fjerntilgang, webserver, databaser mm.

2.3.3 I^{2}C-bus

Al kommunikation på SGU'en kører som nævnt via I^{2}C, der er en bredt adopteret metode inden for digital elektronik, der bla. bruges at store internationale firmaer som Intel, Motorola, Siemens og Texas Instruments. I^{2}C stammer helt tilbage fra starten af 80'erne, og er igennem årene blevet udviklet og opdateret flere gange, så der er understøttelse for moderne hastigheder. I^{2}C virker ved, at der i systemmet er en masterkontroller, (mikroprocessorer, fpga el.lign) der er tilsluttet alle underkomponenter(slaves) via 2 tovejs data linjer, Serial DAta(SDA) og Serial CLock(SCL). Masteren kan så via de to forbindelser kalde funktioner på alle slaverne ved at kalde deres unikke adresse og et funktions nummer. For nogle funktioner vil det være masteren, der skriver til slaven og andre omvendt - alt efter komponent. Dette er markeret i et read/write bit i adresse kaldet. Den store fordel ved at bruge I^{2}C, ud fra et hardware synspunkt er, at der kun bruges to linjer til at forbinde op til 1008 komponenter, hvilket gør design af print dertil medgørligt. Ud fra et softwaresynspunkt er det en fordel, at der er en protokol, der kan være fælles for alle underkomponenter, masterkontrolleren er tilsluttet.

2.3.4 HTTP(S) - HyperText Transfer Protocol

HTTP er standardprotokollen, som World Wide Web bygger på, så det må siges at være en meget udbredt og gennemtestet standard. Starter man med at se på, hvilke teknologier HTTP er baseret på, finder man det der hedder Open Systems Interconnection Reference Model(OSI-Modellen) 2.2, hvor det ses, at HTTP er det øverste lag. I de underliggende 6 lag finder man sikkerhedsprotokollerne, der gør HTTP til HTTPS. Disse bruges derfor ikke altid. I det 5. lag styres TCP sessionen, dvs. at der ikke ved hvert kommunikationsforsøg skal oprettes en ny forbindelse med handshakes og evt. krypteringsnøgler. Kun ved den første tilslutning, og/eller hvis sessionen er udløbet. Den store grundsten i HTTP standarden er baseret på Transmission Control Protocol(TCP)i det 4 lag, da den sikrer, at alle pakker kommer frem til modtageren i den rækkefølge, de er blevet sendt og ikke mindst styrer den, at pakker, der går tabt, bliver gensendt automatisk. Dette betyder, at der ikke skal indføres sekvensnummerering eller modtagelsesbekræftelse, når man implementere ACSI via HTTP. De sidste underlæggende lag er netværks- og hardwarelagene, som er kendt fra internettet, hvori der er mulighed for at kommunikationen kan foretages via kabler, fiber og trådløs.

OSI-modellen		
Lag	Navn	HTTP komponenter
7	Applikationslaget	HTTP/HTTPS
6	Præsentationslaget	SSL/TLS
5	Sessionslaget	TCP-session
4	Transportlaget	TCP
3	Netværkslaget	IP
2	Data Link-laget	wifi/ethernet
1	Det fysiske lag	kabler/radio

Table 2.2: Illustration af OSI modellen

HTTPS er den sikrede version af HTTP, hvor HTTP kommunikationen bliver ført igennem en sikret protokol enten SSL eller TLS. Dette sikrer forbindelsen imod at blive aflyttet, og giver en stærkere autentificering. TLS er den nyeste standard af de to og har en anelse bedre sikkerhed, men primært er forskellen, at TLS er designet til brug i webkommunikation, hvor SSL kun er tilpasset brug dertil. Fælles for begge er, at de bruger asymmetrisk kryptering, hvor en public krypteringsnøgle 2.3.8 deles imellem både server og klient, hvor data krypteret med den kun kan læses, hvis man har den private del af nøglen. HTTPS er så en passende metode at bruge, hvis der ikke ønskes at bruges VPN eller SSH tunnel.

2.3.5 REST

REpresentational State Transfer (REST) webservice er en arkitektur udtænkt af Roy Fielding. Den er som sådan ikke er en decideret standard, men mere en grundlæggende metode ti,l hvordan man kan lave webbaseret kommunikation. REST udmærker sig ved at være en simpel metode til at bruge den gennemtestede HTTP protokols funktioner til at håndtere kommunikationen. Det overordnede princip i REST er, at alle funktioner skal tilknyttes en HTTP URL, og at hver af disse skal være sekvensløse (stateless). REST benytter sig så af HTTP protokollens 4 metoder GET, POST, PUT og DELETE, til henholdsvis hente, opdatere, uploade og slette dataobjekter. Data i REST kan sendes på flere måder: de to mest populære er XML og JSON, hvor de så udgør HTTP svaret i stedet for et HTML dokument. ACSI objekt referencen minder tildels om den måde URL opbygges til REST og vil derfor kunne bindes således, at URL adressen vil svare til det ACSI objekt, der ønskes. Der er også muligt i REST at overføre argumenter i URL adressen ved at man tilføjer et '?' tegn i slutningen af adressen og derefter indsætter et argument. Dette kan ligeledes bindes til ACSI typens argumenter ved de tilføjes her. Den sidste del, der mangler af ACSI at skulle bindes til REST er, om det er en hent, lav, opdater eller slet kommando, det vælges så her, at det bindes til HTTP protokollens 4 metoder GET, PUT, POST og DELETE. Det er derfor konklusionen, at ACSI så let kan bindes til REST, at REST vælges som kommunikationsmetode.

2.3.6 Extensible Markup Language - XML

Som nævnt i forrige sektion kan REST bruges med flere objektsprog, hvor XML[W308] og JSON er de mest populære. Dog giver det hurtigt mening at fravælge JSON over XML da IEC 61850 standarden allerede har flere dele, der optegnes i XML f.eks SCL konfigurationsfilen. XML syntaksen er en strengoptegning baseret på HTML, dvs. at alle tags skal afsluttes, hvilket gør det let at tolke hierarkiet i strukturen og opbygge objekter, der indholder objekter. XML struktuen definere et sæt af tags således

```
<tagnavn1>
    <tagnavn2 attribut=""> værdi </tagnavn2>
    <tagnavn3 />
</tagnavn1>
```

hvor et tag har et navn imellem krokodillenæbbene, og hvor der i den første halvdel af tag'et også kan være et antal af attributter. Det er ikke tilladt at

have attributter i det afsluttende tag. Et tag kan også afslutte sig selv, som det ses i tagnavn3, eller det kan indeholde andre tags. Det er i Java muligt at serialisere et dataobjekt til og fra XML via JAXB API'et, så det ikke skal gøres manuelt med String sammensætning eller Dom objekter(Document Object Model).

2.3.7 MySQL

For at kunne udføre logningen, der beskrives i IEC 61850, er der brug for en smart metode til at gemme de mange datasæt, så de også er let tilgængelige på et senere tidspunkt. Der er derfor nødvendigt med en form for styret database, og af den slags findes der MySQL, MsSQL, MariaDb, Sqllite og mange flere. Det er valgt at bruge MySQL, da den er opensource og kan installeres på stort set alle typer af styresystem. MySQL databasen gemmer lokalt på den enhed, hvor den er opsat, i en database, man selv designer af et antal af tabeller, hvori hver tabel kan have flere forskellige attributter af forskellige typer og ikke mindst have automatisk tidsstempling. Der skal så designes en database, der passer til de værdier, der hentes ud af SGU'en, og IEC 61850 programmet skal så - ved hver måling - gemme værdien i MySQL databasen. Al tilgang til databasen sker via SQL sproget, hvor der findes et stort udvalg af muligheder f. eks kan man hente specifikke data ud fra kriterier. Men mest at alt bruges funktionerne til at indsætte data, hvor man for at beskytte MySQL databasen imod, at man ved fejl skriver forkerte værdier, kan der oprettes det, der kaldes 'stored procedures' (SP), hvor man forudprogrammerer metoder, der kan teste, at de skriver værdier korrekt til databasen. SP kan også bruges som en adgangssikkerhed i, at man kun giver nogle brugere ret til bruge disse, så de ikke kan lave andre typer af kald. I det tilfælde at man ønsker at flytte al data fra en database til en anden - f.eks ved udskiftning af SGU - kan man i MySQL lave et komplet dump af hele databasen, som så kan blive indlæst på den nye med en enkelt metode.

2.3.8 Sikkerheds metoder

Den tilligere nævnte metode HTTPS er smart, hvis SGU'en skal sidde på et lokalt netværk og ikke skal tilgås over nettet, men i det tilfælde, den skal det, eller at den f.eks er tilsluttet nettet via en mobildataforbindelse, vil HTTPS være problematisk, da IP adressen kan ændre sig eller at SGU'en kan være bag en firewall. Det er derfor nødvendigt også at tænke på andre muligheder for at sikre kontakten til enheden.

Den første mulighed er at sætte en Secure socket shell(ssh) tunnel op. Den kan

initialiseres både af server og klient og giver en direkte krypteret forbindelse imellem parterne. Forbindelsen er kryptere på sammen måde som HTTPS, men tillader så altså, at SGU enheden kan være den, der oprette den krypterede forbindelse.

En anden mulighed er at bruge Virtuel Privat Netværk(VPN), der som navnet siger laver et virtuelt, krypteret netværk imellem de to parter. Denne metode har mange fordele i, at der kan tilsluttes adskillige servere og klienter i en og samme VPN, og de alle vil kunne kommunikere krypteret og sikkert. Sættes VPN op med en central netværkskontrolenhed, kan enheder også tilslutte og frakoble, uden at de påvirker forbindelsen imellem de andre på nettet. Der findes flere forskellige måder, dette kan opsættes på - både via software og hardware, men den umiddelbart smarteste løsning er, at bruge den software, der hedder Hamachi, lavet at firmaet LogMein. Hamachi giver administratoren af VPN mulighed for at styre et netværket og bestemme, hvilke enheder der har adgang. På enhederne skal der kun installeres et lille program, som automatisk tilslutter, logger ind og krypterer forbindelsen ved hver systemstart. Så med Hamachi er det ligemeget, om en enhed er bag firewall, bliver genstartet, eller IP'en skifter, den vil altid have den sammen adresse på det VPN, Hamachi har sat op.

2.3.8.1 Sikkerheds Nøgler/Keys

Fælles for alle de typer af sikkerheds metoder inkl. HTTPS er, at de er baseret på, at de to enhede,r der skal kommunikere, udveksler det, der kaldes en public key. En public key er en del af et asymmetrisk krypteringsalgoritme, hvor man ud fra et generering program får en privat og public nøgle. Når to enheder så deler de to public nøgler, vil de hver især have et nøglesæt bestående af deres egen private og deres partners public key. De kan så kryptere beskeden i det af protokollens tildelte program med deres nøglesæt, hvor efter beskeden kun vil kunne blive læst af en person, der har de modsatte nøgle sæt, så det vil sige, at selvom datapakkerne bliver opsnappet af ikke ønskede modtagere, vil de ikke kunne læses. Ydermere bliver pakkerne tidsstemplet inden kryptering, så det er heller ikke muligt bare at optage dele af data transmissionen og gensende den sendere.

I disse keys er det også muligt at tilføje indentitetscertifikater, hvis man ønsker ultimativ sikkerhed. Det foregår ved, at en certifikatudsteder(CA) inkluderes i starten af kommunikationen, og den så også har et nøglesæt med CA-servere, hvor serveren skal svare fra en bestemt IP og have det korrekte certifikat. Dette bruges dog mest af banker, så kunderne ved, at det ikke er en falsk hjemmeside, de er kommet ind på.

2.3.9 Opensource licenser

Projektet vil altså være baseret på brugen af en række af Opensource baserede løsninger. Det er derfor værd lige at nævne, hvad dette handler om. Opensource kan falde under en række forskellige licenstyper, der giver brugerne forskellige rettigheder, men fælles for dem alle er, at selve koden er tilgængelig for alle og kan hentes uden omkostning. Der er dog nogen af licenserne, der har regler om, hvorvidt det er tilladt at bruge produktet, de dækker over kommercielt, men da der her er tale om et non profit forsøg, er det ikke et problem, før der er tale om, at produktet skal sælges, og selv der kan det i de fleste tilfælde gøres lovligt, så længe man distribuerer licenserne med produktet, og alle ændringer man har fortaget også gives ud til Opensouce fællesskabet.

2.3.10 Git versions styring

Der vil under udviklingen af det nye system til SGU'en være brug for løbende backup og versionsstyring. Til dette benyttes en Git server. En Git server tager backup, hver gang en ændring gemmes til den, så det også er muligt at gå tilbage til en tidligere version, hvis noget udvikling skulle slå fejl. Dette gøres for at mindske risikoen for datatab, men også fordi det er en hjælp, når koden skal flyttes fra udviklingsmaskinen til den fysiske enhed for test. Fremadrettet vil det også kunne bruges, når firmwaren på solgte SGU'er skal opdateres, ved man simpelthen frigiver opdateringer via dette.

2.4 Delkonklusion del.1

Det er meget hurtig muligt ud fra analysen, at tage nogle overordnede valgt om, hvordan denne prototype skal udformes: Det første er, at en integreret computer vil være den bedste løsning at implementere IEC 61850 på, da standarden er yderst omfattende, og forhøjet regnekraft er nødvendigt. En anden årsag er, at det vil være lettere at foretage opgradering og udbygning af funktioner i fremtiden på denne platform, da systemet er meget fleksibelt. Valget om at bruge en REST service, der kører på HTTP protokollen giver en stor fleksibilitet i og med at kommunikationen med enheden kan foretages over internettet eller på et lokalnetværk, og - ikke mindst - er det muligt at gøre det via trådløse teknologier. Det er en stor fordel, da det er de færreste vejside skabe, der har internet. Når man ser på sikkerheden, giver en integreret computer også mulighed for at opsætte et VPN system til at håndtere kommunikationen helt

krypteret og verifiiceret. Det giver en høj sikkerhed imod angreb, selvom der benyttes et offentligt netværk til styringen.

Der er - på baggrund af denne delkonklusion - udarbejdet følgende Kravspecifikation & projektplan.

2.5 Kravspecifikation

En stor del af kravene til denne prototype kommer fra IEC 61850 standarden selv, og de regler den opstiller, men dertil er der opstillet følgende lister af funktionelle og ikke-funktionelle krav.

2.5.1 Funktionelle krav

- ACSI skal udføres vha. HTTP og rest.
- Mållinger skal kunne hentes enkeltvis.
- En måling skal kunne hentes for 3 faser.
- Mållinger skal kunne hentes i alle typer samlet fra en linje(3 sensorer).
- Mållinger skal kunne hentes i en komplet model over hele SGU.
- Mållinger skal udformes efter IED 61850 datamodellen.
- En konfigurationsfil skal laves i henhold til IEC 61850
- SGU skal logge alle målinger lokalt og gemme dem i 30 dage.
- Loggede målinger skal kunne hentes enkeltvis og i sæt af intervaller.

2.5.2 Ikke-funktionelle krav

- Al kommunikation skal kunne sikres.
- Opgraderingen af SGU skal tage højde for økonomiske forhold.
- Opgraderingen skal passe ind i nuværende formfaktor af SGU boks.
- Opgraderingen skal kunne tåle 24 timers drift og vekselende temperaturer.
- Skal kunne genstarte automatisk efter strømudfald.
- Fremtidig opgradering skal være lettest muligt.

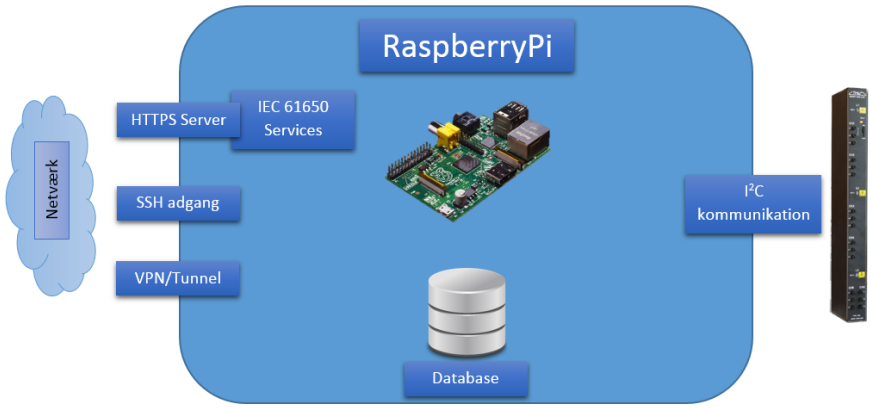
2.6 Projektplan

Der er udtænkt følgende projektplan over, hvordan tidsforbruget skal fordeles på de enkelte opgaver, hvor og hvornår, de forventes færdige. Der er allerede taget højde for, at analysen af dette emne har taget ekstra tid pga. størrelsen af IEC 61850, og at den spreder sig over en del andre teknologier - ikke mindst - hardware implementering.

Figur 2.6: Diagram over den foreløbige tidsplan.

2.6.1 Konceptuel design

Overordnet set er der udarbejdet følgende konceptuelle design over protypen 2.7. Den tager grundstenen i en RaspberryPi mini computer, hvorpå der køres Linux. Denne minicomputer skal således stå for at hente alle datamålingerne ud af SGU'ens mange hardwaresensorer via en I^{2}C-bus. Det er så tanken, at minicomputeren selv logger al data lokalt i sit eget filsystem eller på en ekstern usb-disk, så det ikke er nødvendig for kommende brugere at have en konstant forbindelse. Der vi så blive udviklet en prototypeversion af en IEC 61850 server med begrænset funktionalitet, som kan køres på denne lille enhed, for at få en idé om mulighederne, dette giver. Adgangen til minicomputeren kan så sker via flere forbindelser. Primært vil selvfølgelig være via HTTPS til den IEC 61850 server, der er bundet til HTTPS via REST. For at kunne administrere enheden, opsættes der en sikret SSH adgang, hvor det er muligt at logge på enheden. Som en ekstra sikkerhed vil enheden også blive sat op til at kunne køre via en VPN/tunnel, hvor al data er krypteret og med de øvrige fordele, der er nævnt i afsnittet herom.



Figur 2.7: Overordnet koncept til opgradering.

2.6.2 Afgrænsning af projekt

Da der kun er tale om en prototype, vil fokus i projektet være at overføre SGU'ens nuværende funktioner, og ikke at give den intelligente funktioner som IEC 61850 ellers giver rig mulighed for. Det vil kun kort blive beskrevet i

sektion 6.3, hvilke muligheder der kunne være for brugen heraf. Dvs. SGU'ens funktion til at alle mållinger kan hentes over internettet, dog denne gang i IEC 61850 datamodellen og via en forbindelse, der er sikret mod angreb.

2.7 Delkonklusion del.2

Med konklusioner fra Delkonklusion del.1, er der blevet opstillet en række af krav til hvad der forventes at denne prototype skal kunne realisere, og designet af systemet vil så have fokusset på disse. Der er blevet opstillet en projektplan som der skal hjælpe mig med at få et overblik over hvad der skal laves og hvornår det skal være færdigt for at der kan gives tid til alle aspekter af rapporten bliver realiseret. Dertil har jeg besluttet i det overordnet koncept design at det skal laves på en RaspberryPi mini computer da den vil være tæt på hvad der rent faktisk kan indbygges i SGU ved en fremtidig opgradering.

Kapitel 3

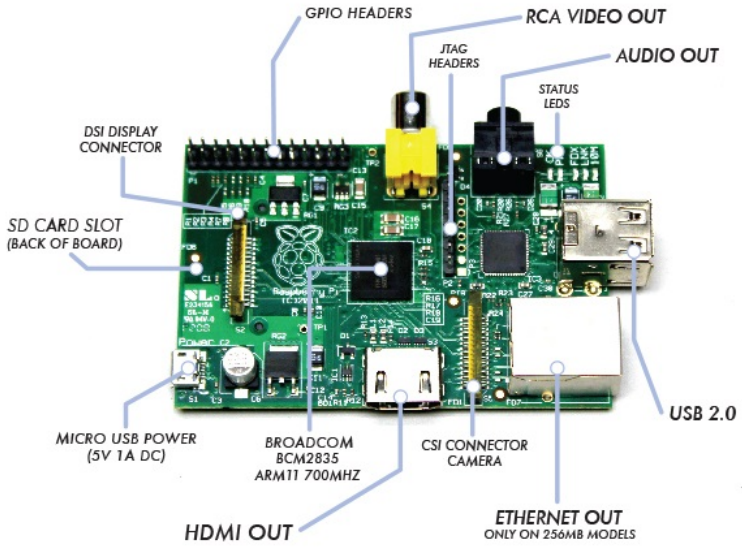
Design

Der vil i dette kapitel gennemgås de overordnet design principper, og udlægges en design plan for hvordan systemmet skal udformes. Der vil i dette kapitel også tages overordnet valg om hvordan programmeringen udføres, i form af hvordan de forskellige dele af systemmet hænger sammen og hvilken ansvars område den enkelte system dele vil blive tildelt.

3.1 Hardware design - RaspberryPi

Som det blev konkluderet i analysekapitlet, var integration af en minipc i SGU'en den bedste løsning, men da en fuld hardwareimplementering af en minipc tager en del tid, er det valgt, at prototypen skal laves på en RaspberryPi(RPi). RPi er en billig, lille, integreret computer, der er designet med henblik på, at man let kan lave prototyper og forsøg i undervisningsmiljører. Den passer derfor rigtigt godt til dette formål. Alt ved RPi er åbent, så design, print & kode, kan findes og bruges ved mulig fremtidig hardwareimplementering i selve SGU'en. RPi er designet til at køre Linux, der er en fantastisk platform, når det kommer til softwaremuligheder, idet der næsten ingen begrænsninger er, og der allerede findes et stort udvalg af softwarekomponenter, der er frit tilgængelige. Den specifikke hardware RPi'en består af en 'System on a chip' som nævnt i section. 2.3.2.3

hvor den centrale processeringsenhed(CPU) er fra ARM11[ARM13] processer familien og kører på en klok frekvens på 700 mhz. Også i SoC'en finder man en grafik processor af typen VideoCore IV og 512MB ram. RPi har en række kommunikations muligheder, men vigtigst af alt har den I^2C, så den let kan tilsluttes den nuværende SGU. Det eneste, der skal gøres, er fysisk at forbinde SCL, SDA og strømmen og gøre den nuværende master chip passiv. RPi har så selv alt, hvad der skal bruges i hardware, indbygget ram, lager på SD-kort, Ethernet netværk og mulighed for WIFI/3G/4G netværk via usb. RPi giver også muligheden for både skærm, mus og keyboard, hvilket selvfølgelig ikke er nødvendigt at have i SGU'en, men for prototypen giver det en række fordele i opsætningen.



Figur 3.1: RaspberryPi.

Ser man lidt fremadrettet, vil det faktisk ikke være utænkeligt, at vælge at indbygge en RPi i SGU boksen frem for at designe et specifikt modul med lignende egenskaber. RPi'en har en lille nok formfaktor til at kunne passe ind i SGU'en, og de dele, der ikke bruges, vil kunne afloddes. En undersøgelse af RPi'ens brugsforhold gav konfliktende kilder, men en temperatur påvirkning 0 - 70 °C burde den - ifølge specifikationerne - kunne holde til. Konflikten gik så på, hvorvidt den kunne holde til minus grader, hvor det efter sigende skulle være dens Netværkskontroller[Fou13a], der ikke tåler det.

En ting, der er sparet væk på RPi'en, er et hardwarebaseret realtids ur(RTC), der kan holde klokken og datoen, når strømmen er gået eller systemmet genstarter, sådan som en BIOS chip gør i en normal PC. Den har i stedet et softwarebaseret ur, der får klokken og datoen via Network Time Protocol(NTP), men dette kræver dog, at den har forbindelse til internettet, hvor disse findes. Hvis man ønsker at være helt kritisk mht. sikkerheden, vil NTP let kunne blive saboteret til at give en forkert tid og derfor ugyldiggøre datamålingerne. Det er derfor heldigt at SGU'en allerede har et hardwarebaseret RTC med batteri backup, så den kan holde tiden, selvom den er uden strøm i længere tid. SGU'ens RTC sidder på samme I^2C-bus, som resten af komponenterne sider på, så klokken vil kunne hentes derfra på sammen måde som resten af målingerne.

3.2 Styresystem

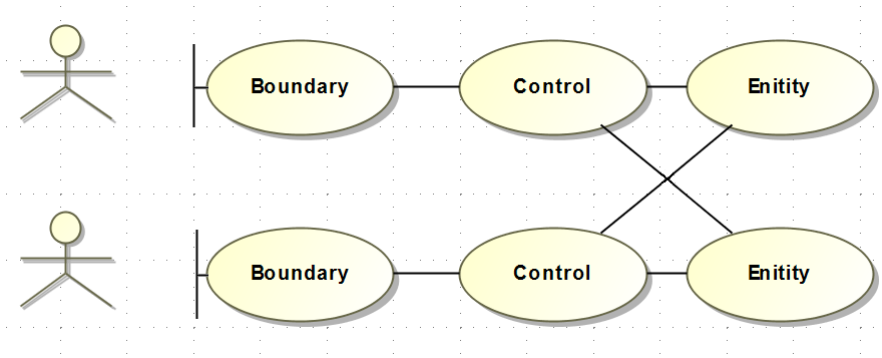
Der vil på RaspberryPi'en blive installeret Raspian styresystemet[Fou13b], og intet i dettes design vil blive ændret, udover at der vil blive konfigureret og installeret nye softwaremoduler. Det vil dog blive gjort rigeligt brug af de i systemmets allerede tilgængelige funktioner, der bla. tæller netværksstyringen, herunder både tilslutning og kryptering. Det antages, at dette er en computer, og der vil derfor ikke blive beskrevet videre om designet af dette, og i de efterfølgende sektioner vil der kun være tale om designet af selv program koden, der eksekveres på dette styresystem.

3.3 Programmeringssprog

Der findes mange muligheder for, hvilket programmeringssprog systemmet kan laves i, og en række af argumenter for, hvorfor de forskellige har mest relevans. Da systemmet er tæt bundet med hardware komponenter, har C programmering er stor relevans, da dets styrke ligger i dette område. Dog er C ikke et objektorienteret sprog, hvilket ikke passer ind i principperne i IEC 61850, der er dybt hæftet i objekter. Så valget falder på de to store objekorienterede sprog Java og C#, to sprog der på mange punkter er meget ens, men alligevel er meget langt fra hinanden. Af similariteter er, at begge sporg kan kompileres og køres på både Windows og Linux platforme, selvom C# primært er udviklet af Microsoft til Windows, findes der også Opensouce alternativet Mono, der giver muligheden for at køre C# under Linux. Men det endelige valg falder på Java, fordi det allerede er besluttet at bruge en RaspberryPi som prototype platform, og der ikke findes biblioteker til at benytte Rpi'ens I^2C bus i C#, og at skulle udvikle disse selv vil være for tidskrævende.

3.4 Designmodel - Entity-Control-Boundary

ECB er designmodellen, systemmet vil blive designet efter, da det er en data-centrisk model, og systemmet, der skal laves, bygger på sammen princip om, at det er en server der skal levere data til en række af forskellige klienter. ECB er en variation af Model-View-Controller designmodellen, hvor strukturen er mere lineær, og views ikke kan kommunikere direkte med datamodellen. ECB kan på dansk forstås som dataenhed, styring og adgangsbrugerflade, hvor det skal forstås som, at adgangsbrugerfladen er der, hvor klienter tilsluttes til, og de kun vil være bekendte med de funktioner, denne tilbyder. Hele kontrollen ligger så i styringsdelen, der behandler funktioner som adgangsbrugerfladen efterspørger. Control kan dog ikke selv generere data og får det derfor fra Entity, som kan deles ud i flere data enheder som databaser, filer, eller hukommelse.



Figur 3.2: BCE system struktur.

3.4.1 Boundary

Overordnet i dette system vil Boundary kun bestå af HTTP(S) serveren. Den har som ansvar at tillade klienter at kunne tilslutte systemmet. Alt efter hvordan systemmet konfigureres skal der enten være en direkte adgang til REST funktionerne på HTTP, eller hvis det er sat op til at køre HTTPS skal den håndtere hele proceduren, der ligger i dette - med udveksling af krypterings nøgler, forhandling af krypterings standard mm. Som det dikteres i REST skal alt dette laves 'stateless'. Der må derfor ikke være krav om en bestemt url, der skal startes på for at opnå HTTPS. Dog bliver en HTTPS session normalt aktiv, hvis der ikke går længe imellem flere kald, og stopper først ved time-out, men det har ikke anden betydning end, at der ikke skal udveksles nøgler mm. ved

hvert kald. Boundary skal så stå for at behandle HTTP(S) kaldet og formulere det til et ACSI kald, som er Boundary's tilgang til Control. Dette ACSI kald skal bestå af 4 argumenter:

1. Funktions type. [GET, PUT, POST eller DELETE]
2. Funktions navn og argument. [GetVal/GetLog&from=dato1,to=dato2]
3. Objekt reference funktionen kaldes på. [SGU/BUSBAR/MMXU1/Hz/]
4. Indhold af kaldet der bruges, hvis en funktion skal skrive data til SGU'en.

Det er ikke tænkt, at der vil blive lavet funktioner, der bruger det 4. argument, men det tages med til fremtidig reference. Fra ACSI kaldet vil der blev returneret et objekt, svarende til hvad der skal gives til klienten. det objekt vil så, inden det gives til klienten, blive omdannet til XML.

3.4.2 Control

Control vil, som navnet siger, være den styrende del af systemmet, hvor størstedelen af logikken udføres. Den primære funktion vil så være at behandle det førnævnte ACSI kald, der er indgangen til Control fra Boundary, hvor argumenterne skal bindes til specifikke metoder, der udfører de tilhørende funktioner. ACSI kaldet vil - som standard - skulle danne et IEC 61850 objekt, der indholder alle de informationer og værdier, der er efterspurgt i argumenterne. De korrekte værdier findes ved at lave en forspørgsel til Entity på enten de nyeste værdier eller loggede værdier fra databasen. Et andet ansvar af denne del af koden er, at fortælle de to nedenstående entitys, hvor ofte de skal hente målinger og gemme dem i databasen. Som nævnt er det kun tiltænkt, at denne prototype skal kunne udføre de allerede eksisterende funktioner af SGU'en, altså at man kun kan hente målinger. Der vil dog også arbejdes på, at den selv skal logge målingerne, og man skal kunne hente disse via ACSI funktionen. Det vil såvidt muligt blive implementeret et skelet, der giver en idé om, hvor i koden de andre funktioner, der ikke bliver lavet, ville ligge.

3.4.3 Entity

Entity vil bestå af to dele, en del der håndtere alt kommunikation med SGU hardwaren, der henter alle øjebliks målingerne og hele tiden giver det nyeste dataset til Control. Og den anden del der står for kommunikationen med database server. Disse to dele er per definition 'dumme' funktioner, der ikke gør noget med mindre Control kalder dem. Datasikkerheden i denne del er såvidt mulig sikret, mod brugerfejl ved, at brugeren ikke har direkte tilgang til

dem, men skal bruge funktionerne tilgængelige i Boundary delen, som der så udføres af Control delen.

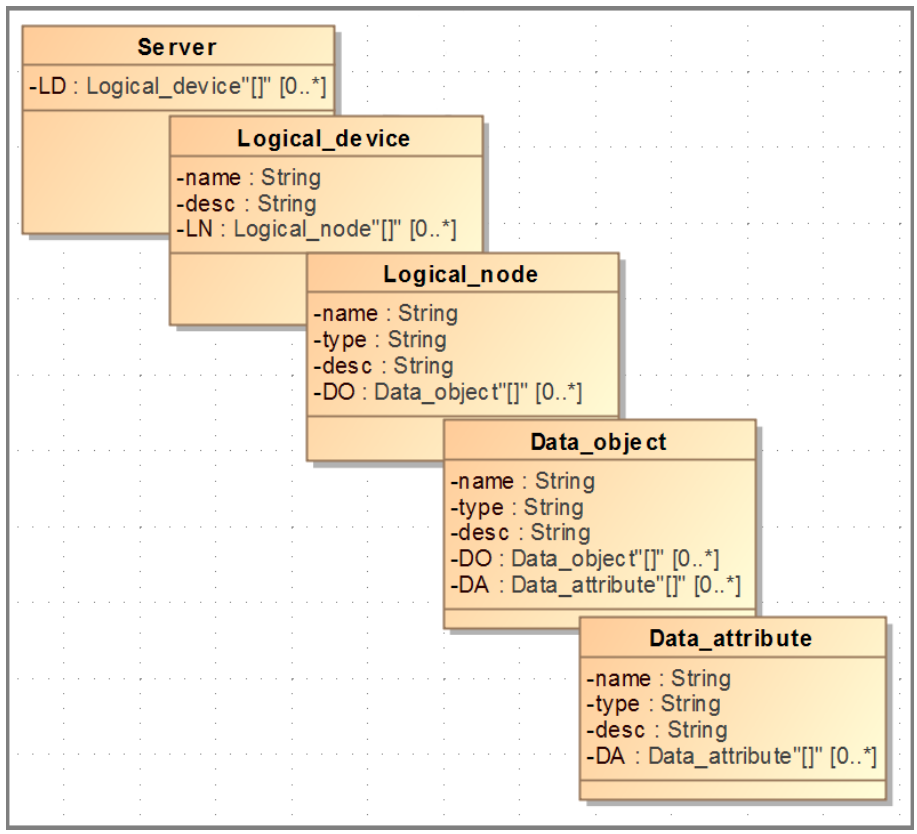
3.5 Systemdesign

3.5.1 Dataobjekter

Der er i dette system brug for en række forskellige datamodeller til de forskellige dele. Det er desværre ikke muligt at designe systemmet efter en enkelt datamodel, selvom det ville gøre det hele meget lettere, da IEC 61850 dikterer en datamodel, der ikke egner sig til at blive gemt i en database, og datamodellen fra hardwaren ikke kan ændres. I stedet vælges det, at de forskellige modeller bruges til hver deres specifikke område, og at de så konverteres, når de bliver flyttet fra en del til en anden.

3.5.1.1 IEC 61850 datamodel

Den første model er IEC 61850 datamodellen, som den er beskrevet i analysen og kan ses i figur. 3.3. Det er den, der beskriver, hvordan de XML objekter, der genereres, skal udformes. Det er også den, som skal indgå som en del i SCL konfigurations filen, som IEC 61850 beskriver. Når et objekt skal returneres, skal det så lægges ind i denne model. I den specifikke type, der efterspørgers, vil der så i nogle tilfælde kun være tale om dele af modellen, der skal returneres. F.eks efterspørges en enkelt måleværdi, vil det kun være Data_attribute klassen, der skal udformes til XML. Denne model er blive opsat i et UML program Magicdraw[mi13], hvori den efterfølgende kan autogenereres til tilsvarende Java klasser, som der kan arbejdes videre med i implementeringen.

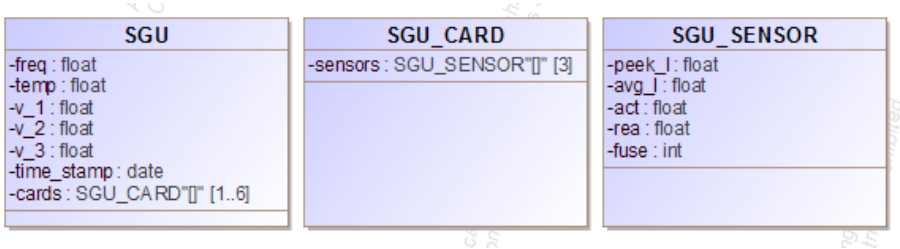


Figur 3.3: IEC 61850 datamodel.

3.5.1.2 SGU datamodel

SGU datamodellen, der ses i figur. 3.4 er dannet udfra de informationer, der hentes i hardwaremålingerne, så det logisk hvilke værdier, der er hentet ud. Der er overordnet nogle SGU værdier, som findes i samtlige versioner af SGU'en, om den så har 1 eller 6 sensor kort. De overordnede værdier er samlet i SGU objektklassen, selvom de hardwaremæssigt kommer fra tre forskellige komponenter, fordi det virker mere logisk at samle dem eet sted. SGU objektklassen har så for hvert installeret sensor kort reference til et SGU_CARD klasse objekt, der i sig selv kun indholder reference til dets 3 SGU_SENSOR klasseobejekter. SGU_CARD klassen kunne godt have været udeladt, men det giver en god overskuelighed af systemmets opbygning. SGU_SENSOR indholder attributter

for de data, der er målt/udregnet på en enkelt strøm linje/fase. Det kan ses i table 3.1, hvad de forskellige attributter dækker over.



Figur 3.4: SGU datamodel.

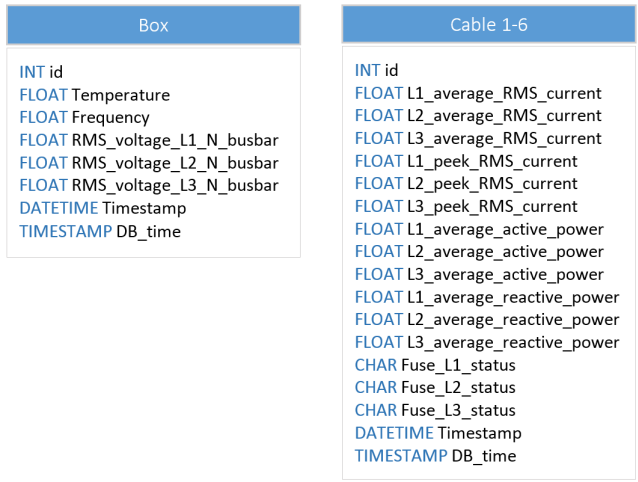
Navn	Beskrivelse
freq	Frekvensen på netværket
temp	Temperatur i SGU boks
v_1	Spænding på fase 1
v_2	Spænding på fase 2
v_3	Spænding på fase 3
timestamp	Tidstempling af måling
peek_i	Højeste strøm
avg_i	Gennemsnitlig strøm
act	Aktiv Power på linjen
rea	Reaktiv Power på linjen
fuse	Status på sikring i sensor

Table 3.1: Attribut beskrivelse.

3.5.1.3 Database datamodel

Datamodellen, som set i figur. 3.5, tager udgangspunkt i SGU datamodellen, men er blevet ændret lidt for at passe på det karakteristiske layout af en MySQL database. Der er blandt andet valgt, at hver sensor ikke har sin egen table, men at de i stedet er samlet i en enkelt table, der dækker over et helt SGU sensor kort. Dette er valgt, fordi målingerne hænger sammen rent logisk, og det så er lettere at hente dem tilbage fra databasen frem for, hvis de lå i forskellige tabeller og var forbundet via table keys eller id'er. Tabellen for SGU kortene er så oprettet 6 gange, så der er en til hvert unikke kort. Ellers er det de sammen attributter fra SGU datamodellen. Den eneste tilføjelse, der er kommet, er, at der, når værdierne bliver gemt i databasen, også bliver tidsstemplet af

databasen. Der er således 2 forskellige tidsstemplinger pr. måling, hvor den ene er tiden i hardwareuret, og den anden er NTP tiden. Det er valgt at inkludere begge stemplinger, da valget om, hvilken der er smartes at bruge, kommer an på, hvordan netværket, SGU'en er tilsluttet, er udformet. Tidsstemplingen er inkluderet i alle tabeller for at undgå, at de er bundet eksplicit sammen, og data så ikke kan redigeres uden at påvirker hinanden. Der kunne laves et argument for, man kunne binde alt sammen med tabel keys, og på den måde spare pladsen på det lille SD kort. Dette kan vurderes, når man arbejder videre med prototypen mod udgivelse.



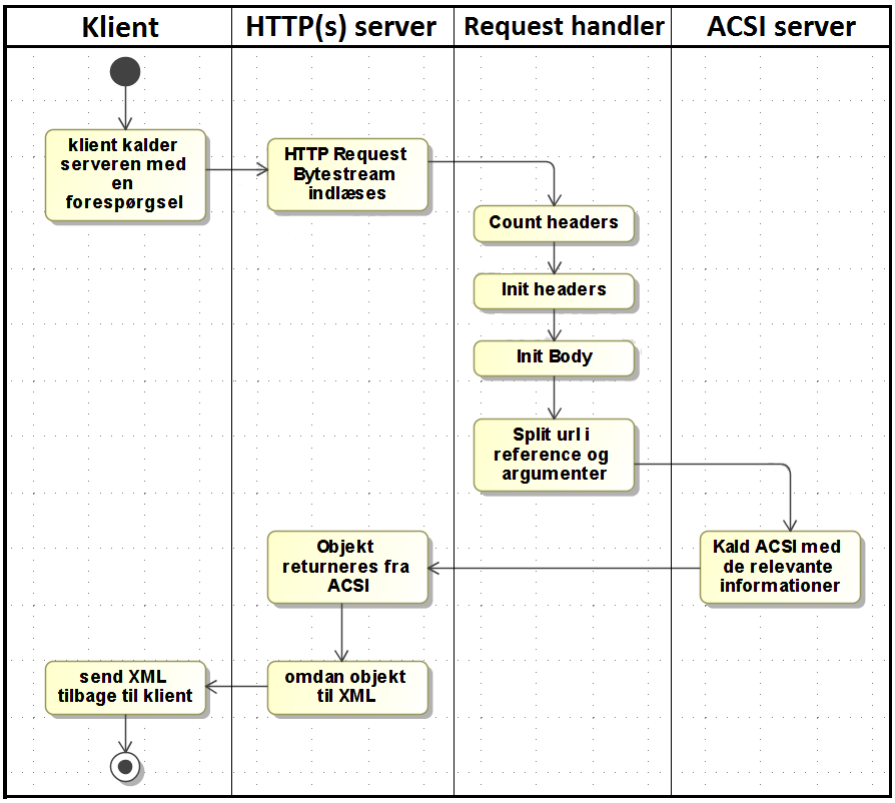
Figur 3.5: Database datamodel.

3.5.2 HTTP(S) server - Boundary

Når det kommer til designet af HTTP serveren, er det åbenlyst ikke at vælge at starte fra bunden og lave sin helt egen. Der findes ufatteligt mange allerede udviklede og gennemtestede Opensource servere, der kan egne sig til dette formål. Der blev lavet en undersøgelse af, hvilke der kunne passe til formålet, hvor favoritterne lå på Apache og Moongoose. I sidste ende faldt valget på at bruge Java's indbyggede (SSL)server socket API, da den let passer sammen med resten af koden, der er i Java og kræver minimal konfigurering. Der er fundet et eksempel [Dha10] på en Java HTTP(S) server, der bruger server socket API'erne, som der er valgt at arbejde ud fra, og modificere til dette formål.

3.5.2.1 Aktivitetsdiagram - HTTPS Request

På figur. 5.9 aktiviteten af en klient laves en forespørgsels til HTTP(s) serveren. Serveren vil modtage denne forespørgselse i form af en Bytestream, der så skal behandles til noget, der kan forstås. Det valgte Java server eksempel har allerede nogle af metoderne til at få dette sorteret ud, men skal lige tilpasses. Efter at HTTP headeren og bodyet er blevet behandlet, kan det formuleres om til et ACSI kald med argumenterne til, hvilket objekt, der ønskes, funktions type, funktions argumenter og data. ACSI serveren skal så returnere et IEC 61850 datamodelobjekt, der skal omdannes til XML strukturen, og udskrives tilsidst til klienten.



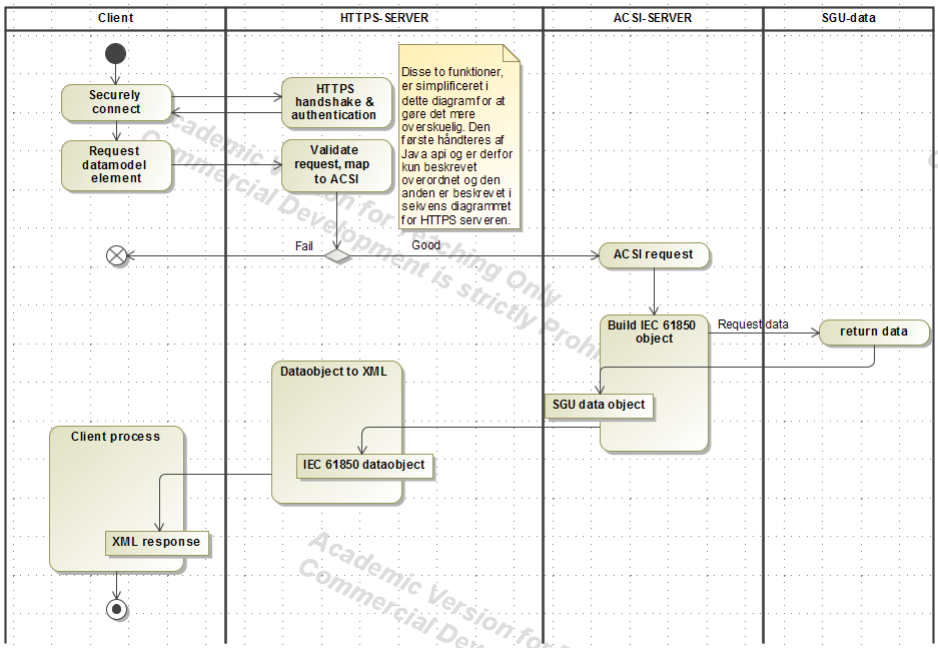
Figur 3.6: Behandling af HTTP forespørgsel.

3.5.3 IEC 61850 server - Control

Det overordnede arbejde, der udføres i Control er som nævn styringen af de andre dele af systemmet, hvilket i dette system primært vil bestå i datahåndtering og opbygning af datamodeller til brug i systemet. Den vil så ikke have et udpræget intelligent system, da det som nævnt kun er idéen at implementere de nuværende funktioner af SGU'en med en tilføjelse af lokal logning af målinger. Men det vælges at designe denne del af koden med tanke på, at der i fremtiden vil kunne lægges mere styring ind.

3.5.3.1 Aktivitetsdiagram - HTTPS kald

Som en viderebygning på aktivitetsdiagram i figur. 5.9, er der i figur. 3.7 et lidt større overblik af, hvordan controlsektionen i koden planlægges at skulle laves. Der er her tydeligt at se, hvordan koden bliver opdelt i de tre Entity-Control-Boundary dele, og hvordan de arbejder sammen.

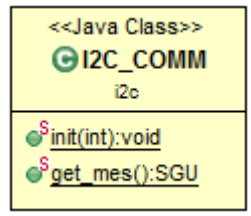


Figur 3.7: Stor version er vedlagt i appendiks.

3.5.4 Data - Entity

3.5.4.1 Hardware måling

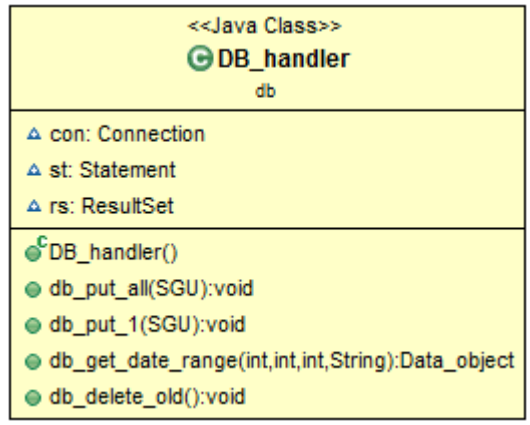
Hardwaremåligerne skal som nævn hentes via en I^2C forbindelse imellem SGU hardwaren og RPi'en. Dette kan man i Java via 'Pi for Java' (Pi4J) [Pi4]. API'en, som er en tilføjelse til Java API, specielt udviklet til brugen af RPi'ens hardware udgange. Der vil ud fra brugen af Pi4J dannes en klasse, der håndterer alt vedrørende I^2C forbindelsen. De kan ses i figur. 3.8. Det ønskes, at det er muligt at hente alle målinger samtidligt, da nogle af dem er en kalkulation baseret på målingerne siden sidste hentning, og de så vil give skæve resultater, hvis disse kalkulationer ikke dækker over de samme perioder, når de sammenlignes senere. Når denne klasse initialiseres, vil den også skulle udføre nogle små kommandoer, for at hardware kører korrekt bla. skal temperatursensoren initialiseres.



Figur 3.8: I2C_COMM klasse.

3.5.4.2 Database

Database Entity'en bliver en enkelt klasse, der opretter forbindelsen til MySQL serveren og har indbyggede metoder til at læse og skrive data i den. Der vil - så vidt muligt - blive designet stored procedures til alle metoder, hvor der skrives til database for at undgå fejlskrivning af data. Den mest problematiske del af denne sektion er, at det skal være muligt at hente mållinger fra et bestemt tidsinterval, og det vil derfor kræve lidt eksperimentering for at få denne metode optimal. På figur. 3.9 ses klassens design, og de metoder den skal indeholde. Der laves dog kun funktioner til håndtering af een SGU enhed med 1 målekort og 6 målekort, da de mellemliggende modeller kun vil tage ekstra tid at lave, men ikke bidrager med ekstra vi-

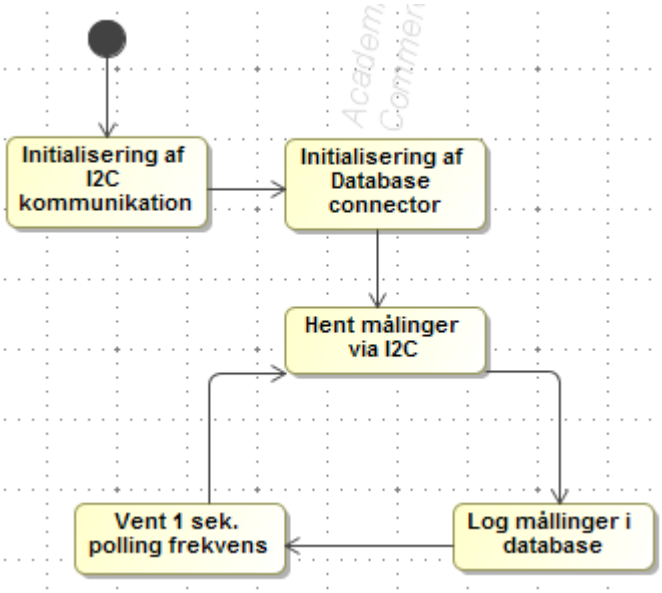


Figur 3.9: DB_handler klasse.

den til projektet.

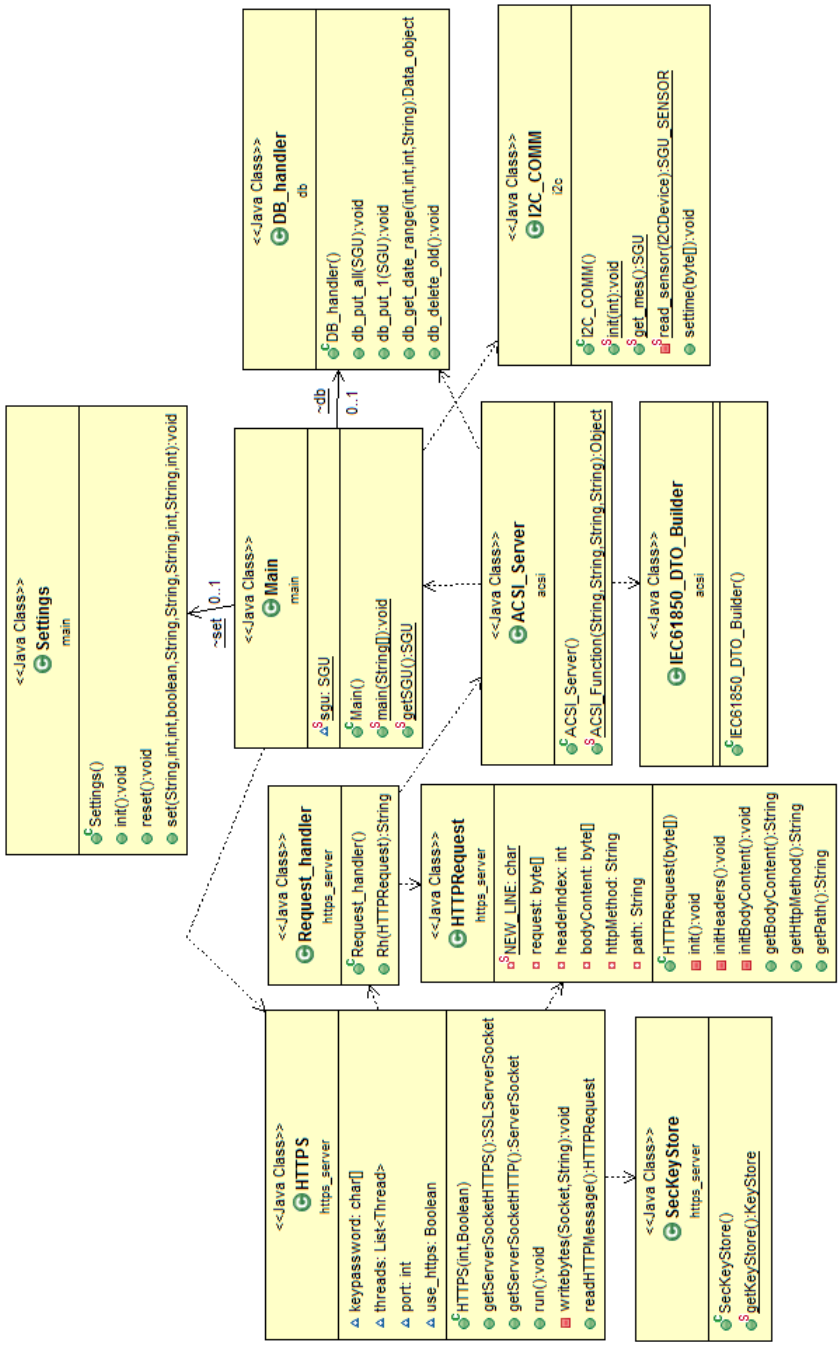
3.5.4.3 Aktivitetsdiagram - Logging

Der er udtænkt følgende aktivitetsdiagram for dataloggingstråden, som grafisk kan ses i figur. 3.11. Den går i al sin enkelhed ud på, at den ved start initialisere I^2C-bussen via de inkluderede API dertil. Dernæst gøres det samme for MySQL forbindelsen til databasen. Med begge dele initialiseret, begynder en uendelig løkke, der henter målinger fra hardwaren via I^2C-bussen, skriver dem til databasen og så ellers venter en prædefineret sampling tid f.eks hvert sekund, og så starter forfra med at hente målinger igen.



Figur 3.10: aktivitetsdiagram over logning.

3.5.5 Design klasse diagram



Figur 3.11: Design klasse diagram.

3.6 Delkonklusion

Der er nu udformet et design af systemmet, der kan bruges til implementeringen af prototypen, som så efterfølgende kan bruges til teste mulighederne for en mulig, fremtidig opgradering er Thiim's SGU. Det er selvfølgelig at forvente, at ikke alt er muligt at implementere, som det er tiltænkt her i designfasen, og at der vil komme ting op, der ikke var forudset. Men den grundlæggende struktur i systemmet er ganske simpel og drager nytte af mange både hardware- og softwaretekniske løsninger, der allerede eksisterer, så det er en bredere funktionalitet uden ekstra arbejde, der er lig med ekstra omkostninger for firmaet Thiim.

Kapitel 4

Implementering

Projektet er nu kommet til det stadie, hvor al forberedelse er gennemgået, og arbejdet med konstruktionen af softwaren kan begynde. Dette kapitel vil så svare til konstruktionstrinnet nævnt i "Unified "Proces", og som det ses på figur. 1.3, vil der løbende i denne fase også foregå lidt analyse og design, som ikke var forudset, og - ikke mindst - vil der blive testet løbende. Test af selve systemet og dets dele vil blive udførlig gennemgået i næste kapitel, men dele af aftestningen vil også blive nævnt i dette kapitel. Selve implementeringen vil forgå på den måde, at de forskellige moduler af koden vil blive udviklet og testet separat og efterfølgende blive sat sammen til et komplet system, der så også vil blive grundigt testet.

4.1 Integrated development environment (IDE)

Det er valgt at bruge IDE'et Eclipse Java EE udviklingsmiljøet til at arbejde med kodeudviklingen, da det giver en række ekstra fordele frem for kun at arbejde i tekstbehandlingssoftware. Først og fremmest laver Eclipse konstant test af, hvad der skrives af programmeringskoder, dette betyder at stave- og semantiske fejl fanges lige der, hvor de bliver lavet og man så ikke senere skal rette dem, når complileren fejler. Dernæst letter Eclipse meget af skrivningen,

idet den kan autofuldføre og generere koden, hvilket igen sparer tid. Det sidste, jeg vil nævne, er det miljø, der er rundt om selve kodedelen. Der giver en - som programmør - det store overblik og kontrol over koden, idet man med enkelte tryk kan eksekvere, debugge eller backuppe koden, for nu bare at nævne nogle af de mange fordele. Dette IDE bruges på en desktop pc frem for RaspberryPi'en, da den ikke lige leverer computerkraft nok til at gøre brugen af dette til en tilstrækkelig stor fordel og generelt er lidt for langsom til at være udviklings pc'en. Det er dog ikke helt uundgåeligt at bruge RPi'en, da Pi4J API'erne ikke virker på andre maskiner.

4.2 Opsætning af RaspberryPi

RPi'en kommer - som standard - uden nogen form for styresystem, men har et anbefalet styresystem "Raspian", der kan hentes fra fabrikantens hjemmeside [Fou13b] Det er baseret på den meget populære Linux distribution Debian. Da det er Linux, er det muligt at designe sit eget styresystem med kun de ønskede funktionaliteter, men da det er en meget omfattende opgave, vil dette blive undladt til fordel for brugen af Raspian.

4.2.1 Installation

Selve installationen af Raspian kan gøres på flere måder, hvor den letteste er at købe et SD-kort, hvor load-image allerede er installeret, men da jeg allerede havde et SD-kort valgte jeg at bruge BerryBoot[ber13]. Ved BerryBoot metoden bruges en anden computer til at downloade og placere en lille bootloader på et SD-kort, og når man så indsætter dette i RPi'en, får man en liste af valgmuligheder for det OS, den skal installere via internettet. Der blev så her valgt Raspian, og 10 min efter var systemet klar til brug.

4.2.2 Sikkerhed

Første del af opsætningen var at forbedrede systemet til fjernadgang, hvilket først betyder at standardkoden til systembrugeren skulle ændres, efterfulgt af, at SSH blev sat op til at tillade udefrakommende forbindelser. Når man tillader SSH adgang til sit system, anbefales det, at man bruger et public/private key-set med adgangskodebeskyttelse, men da her kun er tale om en prototype, der bruges bag firewall, undlod jeg at sætte dette op og kun bruge adgangskodetjek

for at udgå senere komplikationer. Der er dog stadig tale om en rimelig sikkerhed, idet at 3 forkerte adgangskode forsøg vil udløse blacklistning af IP'en, der forsøger adgang. Derefter blev VPN softwaren Hamachi [Log13] opsat, så al kommunikation var sikret og krypteret. Dertil kom også fordelen, at det så var muligt at kommunikere med prototypen, selvom den var bag firewall hos firmaet Thiim, og jeg sad hjemme og arbejdede.

4.2.3 Java

Som standard kommer Raspian ikke med Java installeret, men det kan installeres via et Advanced Packaging Tool kald, der via en indbygget softwareliste kan finde frem til, hvor softwaren kan hentes, og hvordan den skal installeres. Et eksempel på dette kald vises nedefor:

```
> sudo apt-get install openjdk-7-jdk
```

Den skarpe læser vil bemærke, at det er Open Java developer kit, der installeres og ikke Oracle JDK. Dette skyldes, at Oracle ikke har frigivet en færdig version af JDK til ARM baserede systemer. Der findes dog en beta, som jeg har testet, og som ikke gav nogen mærkbar forskel.

For at Java koden kan kommunikere med RPi'ens hardwareudgange, skal der også opsættes Java biblioteker. Denne funktionalitet er stadig under udvikling, men heldigvis er I^2C kommunikation allerede understøttet i det projekt, der hedder Pi4J [Pi4]. Guiden på Pi4J's hjemmeside følges, og det eneste, der ellers skal gøres for at det virker, er at inkludere bibliotekerne i ens Java projekt.

4.2.4 MySQL

Da der lokalt på RPi'en skal gennems datalogning, er det en fordel at benytte en SQL database til at holde disse informationer, frem for at gemme dem i variabler i Java koden eller lignende, da de i en SQL database er beskyttet i det tilfælde, at strømmen skulle gå. Samtidig er det i en SQL database lettere at finde de præcise data set, der ønskes. Der findes flere forskellige slags SQL database software, hvor MySQL er den mest udbredte og bedst understøttede inden for Linux, men alternativer som SQLlite og MariaDB vil også kunne bruges. MySQL serveren installeres på RPi'en på sammen måde som Java ved at lave et apt-kald til at udføre installationen. Under installationen guider den automatisk igennem opsætning af root brugeren af databasen og de nødvendige

kodeord, hvor det så senere er muligt at oprette flere konti med varierende adgangsrestriktioner.

```
> sudo apt-get install mysql-server-5.5
> ...
> Enter MySQL root password: cm2013dtu
```

Da der her er tale om en prototype, vælger jeg at åbne databasen til fjernadgang, så deter muligt for mig at se præcis, hvad der sker i den. Dette er dog sikkerhedsmæssigt ikke klogt, da det giver muligheden for at databasen kan blive hacket. Det vil derfor blive undladt ved en fremtidige produktionsversion, hvor det kun vil gøres muligt at forbinde fra localhost(127.0.0.1). Det er dog værd at have en lille bemærkning her, til at det SD kort, RPi'ens systemet kører på, kan have en begrænset levetid ved konstant logning, men ved køb af kort fra den bedre ende af skalaen burde det ikke være et problem, da de tåler 100.000 skrivninger det sammen sted på disken.

4.3 Database design implementering

Til opsætning af databasens design bruges der udviklerværktøjet HeidiSQL [Bec13], der er Opensource og gratis at bruge. Med HeidiSQL gives der en grafisk brugergrænseflade, der er fuld af funktioner, der giver overblik og letter arbejdet for udvikleren. Både store og små ændringer kan fortages i GUI'et, som så automatisk oversætter det til et MySQL kald, der udføres på databasen. Det sparer rigtigt meget tid, da man ikke skal bruge tid på at skrive MySQL kald manuelt.

4.3.1 MySQL tabeller

Layoutet af tabellerne, der skal oprettes i databasen, er som givet i designkapitlet. De oprettes i HeidiSQL med den dertilhørende funktion, hvor attributter tilføjes til tabellerne, og defineres med type mm.. Hver måling tildeles et id, der automatisk inkrementere med 1 for hver instans, der tilføjes. Som tidligere nævnt vil der være to tidsstemplinger, der bliver gemt i databasen, en fra hardwaren og en fra RPi'ens software. Den første gemmes i databasen ligesom andre værdier, mens den anden er databasen sat op til at udfylde med det tidspunkt, hvor instansen blive tilføjet til tabellen.

4.3.2 MySQL stored procedures

Til al data skrivning/sletning på databasen er det tænkt, at der skal udføres stored procedures(SP), som et en form for metode, man kalder på databasen med data som argumenter. Brugen af disse sikrer, at alle elementer i de forskellige tabeller bliver oprettet samtidligt. Der oprettes så først en 6_cables_insert SP til at gemme et komplet SGU data-set med alle 96 attributter opdateres i et enkelt kald. Parametrene til SP'en bliver så de 96 attributter, hvor SP rutinen kender deres korrekte plads i databasen. Rutinen indsætter efterfølgende hele data-settet i databasen. En funktion - næsten magen til den forrige - oprettes til at udføre det sam som forrige, men dog kun for et SGU data-set fra en SGU med et sensor kort. Der kunne også oprettes SP for alle mellemliggende antal sensor kort, men det er ikke nødvendigt i denne prototype. Til testning har jeg også lavet en SP der renser hele databasen i et kald, så den hurtig kan resettes.

En lidt mere spændende SP er den for automatisk sletning af gamle data, som MySQL koden til ses nedefor. Når denne SP kaldes, går den ind i samtlige tabeller og undersøger alle data instansers database timestamp for, om de er ældre end 30 dage, og hvis de er det, vil den slette dem, og derved skabe mere plads til nye målinger. Det er selvfølgelig kun som et eksempel at 30 dage er valgt. Man kunne måske have overvejet, at SP'en som parameter tog en værdi af, hvad alders grænsen skulle være.

```
CREATE DEFINER=`root`@`%` PROCEDURE `delete_old`()
       LANGUAGE SQL
       NOT DETERMINISTIC
       CONTAINS SQL
       SQL SECURITY DEFINER
       COMMENT ''
BEGIN
DELETE FROM box WHERE DATEDIFF(CURDATE(), box.DB_time) > 30;
DELETE FROM cable_1 WHERE DATEDIFF(CURDATE(), cable_1.DB_time) > 30;
DELETE FROM cable_2 WHERE DATEDIFF(CURDATE(), cable_2.DB_time) > 30;
DELETE FROM cable_3 WHERE DATEDIFF(CURDATE(), cable_3.DB_time) > 30;
DELETE FROM cable_4 WHERE DATEDIFF(CURDATE(), cable_4.DB_time) > 30;
DELETE FROM cable_5 WHERE DATEDIFF(CURDATE(), cable_5.DB_time) > 30;
DELETE FROM cable_6 WHERE DATEDIFF(CURDATE(), cable_6.DB_time) > 30;
END
```

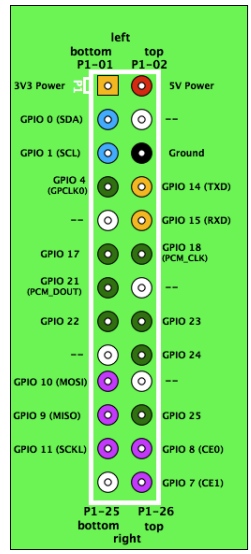
4.4 Java kodning

Selve IEC 61850 serveren vil, som tidligere nævnt, blive implementeret i programmeringssproget Java. Det vil blive udnyttet, at Java er et sprog, der kommer med et stort sæt af API'er, og hvor der let kan tilføjes specielle. Blandet de mange API'er, der bruges i projekte,t er det værd at nævne et par af de vigtige. JAXB er et API i Java, der bruges til at binde objektklasser til XML. Dette vil have en vigtig rolle i REST kommunikationen, hvor dataobjekter skal sendes som XML. JBDC API'et er en specifik driver, der bruges til MySQL forbindelsen. Den er designet til den type af SQL server, der er valgt i systemet. Pi4J er et API, der er specielt lavet til RPi, og som gør det muligt at bruge RPi'ens GPIO udgange i Java.

4.4.1 I^2C kommunikation

4.4.1.1 Hardware tilslutning

Første skridt i at få I^2C kommunikationen til at køre, er at forbinde RPi'en til SGU'en. Med hjælp fra Thiim's elektro specialist, Paul Styrbæk, fik vi tilsluttet 3 kabler til SGU'ens hardware, således at vi havde SDA, SCL og jord. Da elektriske kredsløb kan være bygget til at arbejde på forskellige spændinger, blev det også lige sikret at de to hardware dele var kompatible, hvilket de heldigvis var, da begge bruger en spænding på 3,3v. Men før det var muligt at tilslutte Rpi'en, skulle den nuværende hovedkontroller deaktiveres, da der kun kan være én master på en I^2C-bus. Dette blev gjort ved at omprogrammere den med et tomt program, så den ikke havde nogen funktioner. SGU'en kunne så tilsluttes RPi'en med de tre kabler, der blev tilkoblet Rpi'ens GPIO, på pin 3,5,6 som ifølge af figur.4.1.



Figur 4.1: Pinout af RPi. kilde: [MS13]

4.4.1.2 Hardware kommunikation

For at kunne kommunikere med de forskellige
komponenter i SGU hardwaren, skal man sætte sig ind i de I^{2}C funktioner, de forskellige komponenter har. Fælles for dem alle er, at det er RPi'en, der kører som master i I^{2}C protokollen, mens de andre er slaver. Starter vi med at se på temperatur sensoren(MCP9800), skal vi først vide, hvilken adresse på I^{2}C-bussen den svarer på. Dette kan man finde ud af ved at slå det op i det af producenten ginve faneblad [mic13], hvor det så vil vise sig at være nr. 72 eller i hex 0x48. I tabel. 4.1 er der så givet et eksempel på, hvordan man læser temperaturen, ved at man i en I^{2}C kommando kalder adressen af komponenten, hvilket ses i de første 7 bit af den første Byte. Den overskydende bit bruges så til af fortælle, om det er en read- eller writefunktion, der kaldes. Dernæst sendes en Byte nr. 2, der beskriver, hvilket register/funktion kaldet skal udføres på. I dette tilfælde er det register 0, der får temperatur sensoren til at returnere temperaturen i form af 2 Bytes. Hele denne transaktion håndteres af Pi4J api'et, så man som udvikler ikke skal tænke på, at der i I^{2}C protokollen også skal sendes start, stop, og acknowledge bit samt en clock, der skal styres.

Byte 1								Byte 2							
b8	b7	b6	b5	b4	b3	b2	b1	b8	b7	b6	b5	b4	b3	b2	b1
A7	A6	A5	A4	A3	A2	A1	R/W	R7	R6	R5	R4	R3	R2	R1	R0
Hent temperatur eksempel:															
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0x48							R	0x00							

Table 4.1: I^{2}C master kommando.

For at finde adressen på de resterende komponenter, brugte jeg en indbygget funktion i RPi'en til at finde alle I^{2}C komponenter på bussen, da dokumentationen af de specielle komponenter var stærkt utilstrækkelig.

```
pi@raspberrypi:/opt/pi4j/examples$ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- 0a 0b 0c -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- 32 -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figur 4.2: I^{2}C detect funktion.

Med adresserne kendt, kunne man metodisk gennemgå den givne c fil og matche adresserne til komponenterne og danne listen i tabel.4.2 som der kunne bruges til at danne et overblik af, hvordan målingerne skal hentes. Det interne ur og temperatursensoren skiller sig ud fra de andre ved, at de skal initialiseres for at fungere korrekt. Temperatursensoren skal opsættes til at give en højere opløsning på dens måling, og det interne ur skal indstilles til den nuværende tid. Det interne ur skiller sig yderligere ud ved, at det ikke returnerer alle Bytes ved en read forespørgsel på register nr. 0, men skal have en forespørgsels på hvert register fra nr.0 op til nr.6 og for hvert kun giver en enkelt Byte retur.

I^2C adresse	Enhed	Bytes svar	Bemærkning
0x32	voltage board	9	Byte 6, kasseres dummy temp.
0x48	temperatur sensor	2	Skal initialiseres til 12 bit mode.
0x68	Intern ur	7	Skal hentes fra flere registre.
0x0a	card1_sensor1	9	4 målinger og 1 fuse status.
0x0b	card1_sensor2	9	4 målinger og 1 fuse status.
0x0c	card1_sensor3	9	4 målinger og 1 fuse status.
++	...	9	Ens for alle kort
0x1b	card6_sensor3	9	4 målinger og 1 fuse status.

Table 4.2: I^2C komponenter i SGU.

4.4.1.3 I^2C Java kodning

I^2C kodningen findes i i2c pakken som en enkelt klasse, hvor alt relateret til I^2C kommunikationen håndteres. Klassen er lavet static, så der kun findes en instans, da mere end én vil give et kaos på bussen. Som det første skal klassen initialiseres, før den kan bruges. Under denne initialisering bliver I^2C api'et, der styrer bussen sat op samt alle enhederne, den kommunikerer med i SGU hardwaren. Initialiseringen er lavet fleksibel, så den tager højde for, hvor mange sensorkort, der er i sat SGU'en. Temperatursensoren skal ved initialiseringen ændres til at køre med en højere opløsning i dens målinger, hvor ADC RESOLUTION, som i configregistret er bit 5 og 6. Begge sættes høje for at skifte til 12 bit mode. Et eksempel på dette, og hvordan temperatur målingen hentes og beregnes, er vist i nedenstående kodeeksempel. En ting, der ikke var husket i designet af denne klasse, var, at det også skulle være muligt at indstille tiden på det interne ur i SGU'en. Denne funktion er så blevet tilføjet og kørt en enkelt gang for at sætte klokken korrekt, og er nu udkommenteret da uret kun har brug for at blive sat, hvis det ikke går korrekt.

```
public static void example() {
	byte[] buf = new byte[2];
	try {
		// init i2c bus
		I2CBus bus = I2CFactory.getInstance(I2CBus.BUS_1);

		// init enhed
		I2CDevice temperature = bus.getDevice(0x48);


		// {config register, ADC RESOLUTION bit 5-6}
		// '01100000' = hex '0x60'
	    buf = new byte[] { 0x01, 0x60 };

		// gaa i config mode og set ADC RESOLUTION
		temperature.write(buf, 0, 2);

		// tilbage til normal mode
		temperature.write((byte) 0x0);
		// laes temperatur
		temperature.read(buf, 0, 2);

	} catch (IOException e) {
		e.printStackTrace();
	}
	// udregn korrekt temperatur ud fra de 2 bytes
	int t1 = buf[0];
	int t2 = buf[1];
	double res = 0;
	res = ((((t1 * 256) + t2) >> 4) * 6.25) / 100;
	System.out.println("B0:"+buf[0]+" B1:"+buf[1]+" temp: "+res);
}
```

Selve programmeringer af afhentningen af målinger fra de forskellige komponenter var så kun et spørgsmål om at opsætte systematisk i de forskellige typer. Udfordringen var så, at værdierne modtages i Bytes og skal samles for at give den korrekte værdi. Ydermere antager Java alle variabler som typen 'signed', og nogle af måleværdierne modtages som 'unsigned'. Disse to problemer er så blevet løst ved, at der udføres bit shifting og at værdier AND'es med '1111 1111' for hver Byte. Fordi alle værdierne bliver sendt fra hardwaren som int, hvor decimaler ikke er muligt er de blevet ganget op med håndholdsvis 10 og 1000, der giver et og tre decimals præcision. Dette skal så divideres ned igen, hvorefter værdierne blive gemt i en float, der godt kan håndtere decimaler. Alle værdierne bliver så lagt ind i et SGU Dataobjekt, der bliver returneret til control

systemet.

4.4.2 SGU Dataobjekter

SGU dataobjekterne blev som nævnt autogenereret i UML programmet MagicDraw. Dog kan denne process ikke tage højde for specielle konfigurationsønsker, og de skal derfor tilføjes manuelt efterfølgende. Der er derfor efterfølgende blevet tilføjet getter og setter metoder, der adresserer enkelte kort/sensorer i stedet for hele arrayet. MagicDraw opretter også kun en tom constructor. Der er her blevet lavet en ekstra constructor i alle filerne, der hvor der initialiseres værdier. For SGU klassen er der også blevet lavet en test constructor, hvor der sættes nogle standardværdier ind, så det var muligt at teste uden databasen/I^2C forbindelse. Den vil selvfølgelig skulle fjernes på et tidspunkt.

4.4.3 MySQL handler

Med MySQL serveren installeret og konfigureret og tabellerne oprettet, kan arbejdet med at få Java koden til at kommunikere med den påbegyndes. Det første skridt her er at inkludere JDBC driveren i Java kodens build path og definere, at det er den, der bruges i driver URL'en, som også er den, der indholder IP adressen, porten og navnet på databasen, der forbindes til. Når MySQL driveren så forbinder til MySQL serveren, bliver den bedt om brugernavn og adgangskode, før den vil blive tilladt adgang, men med dem inkluderet er alt klar til at oprette forbindelsen. Når forbindelsen er oprettet gemmes den samt dens tilhørende statement i globale variabler, så klassens andre metoder kan bruge dem. Bemærk dog, at de her ikke laves statiske som i I^2C forbindelsen, da det er muligt at have flere samtidige forbindelser til databasen, og at det bliver brugt, når loggeren kører, og der samtidige skal hentes logget data ud til en klient på HTTP serveren. Den ovennævnte opstartssekvens er lavet i klassens contructor, så det bliver udført, når objektet oprettes. Til de to SP, der er lavet på databasen til at indsætte målinger, laves der til hver en metod,e der udfører dette kald. Metoderne tager som argument et SGU dataobjekt, der indholder de målinger, der skal indsættes. Ligeledes er der lavet en metode til at kalde SP'en, der sletter gamle data, den tager dog ikke nogen argumenter. Hvad der er mere spændende i denne klasse er metoden til at hente logning fra et bestemt dato interval. Som argumenter modtager den værdierne på, hvilken sensor den skal hente målingerne på, og en String med dato intervallet. Stringen er endnu ikke blevet testet eller opdelt, og skal derfor have hentet de to datoer ud. For at udtrække datoerne fra String'en bruges der RegEx pattern match, der kun accepterr det korrekte format af dato strengen. Man kunne også vælge

at RegEx kommandoen skulle teste, om datoen er gyldig, men det blive meget kompliceret, især hvis man også vil have skudår med, så det er lettere bare at lade MySQL server teste det, hvor den så giver fejl, hvis datoen ikke findes. Ud fra de to datoer samt hvilken sensor, der skal hentes, dannes en MySQL statement, som MySQL serveren (hvis der findes data i intervallet) returnerer alle instanser af mållinger der opfylder. Disse mållinger bliver så først gemt i et SGU dataobjekt. Fordi der hentes log, er det besluttet, at der kun hentes en værdi ad gangen (avg/peek strøm hænger sammen i dette tilfælde) og der er valgt en hurtig løsning med, at alle værdier i et SGU dataobjekt sættes til samme værdi, i stedet for at lægge det i den specifikke enkeltplacering, når nu IEC 61850 dataobjektbuilderen ved, hvilken er den korrekte. SGU objektet bliver som nævnt omdannet til et IEC 61850 dataobjekt, som efterfølgende tilføjes til et array af disse. Det sammen bliver gjort for alle instanser i resultatet fra MySQL serveren, hvor der til sidst så vil være en stor samling af IEC 61850 dataobjekter i arrayet. Arrayet blive indsat i et omsluttende IEC 61850 dataobjekt, for det giver fejl senere, når det bliver behandlet at XML marshalingen. Det resulterende IEC 61850 dataobjekt, returneres af metoden til videre behandling.

4.4.4 IEC 61850 Dataobjekter og builder

IEC 61850 Dataobjekterne blev ligesom SGU objekterne autogenereret af MagicDraw, uden behov for de store tilføjelser. Det komplicerede her var så, at danne objekterne, så de passer på IEC 61850 datamodellen, og fleksibelt at kunne lave objekter. Det er præcis, hvad forespørgsler skal bruge. Det er her valgt at se på objekt referencen i ACSI kaldet, som en trinvis trappe, hvor det første 'level' som jeg har valgt at kalde det, er svarende til /SGU. Det andet er /SGU/CH1,og det tredje er /SGU/CH1/MMXU1 og så videre som vist i tabel. 4.3. Der er så opstillet en metode til generering af et IEC 61850 dataobjekt, passende til hver level, og hvor de næsten rekursivt kalder de underliggende metoder og får dem til at danne de fornødne objekter. Målingerne, der skal indsættes i IEC 61850 dataobjektet, gives til metoden som et SGU dataobjekt, hvorfra den så henter de målinger ud, der passer i objektet. LLN0 og LPHD Noderne fra IEC 61850 standarden, har desværre måtte falde for afgrænsningen, selvom det er et krav, at de inkluderes, da de ikke var kritiske for at der kunne sendes måle objekter. Derimod er både quality- og timestampobjekterne inkluderet, og hvor quality bruges som status indikator for sikringerne i sensoren. Et valg, jeg tog her i udformningen, som jeg senere har fundet ud af, var forkert, men ikke har fået rettet er, at jeg skulle have samlet alle elementerne i en enkelt Logical device, og ikke give hvert sensorkort sit egen.

4.4.5 IEC 61850 objekt til XML

For at kunne få et IEC 61850 objekt omdannet til XML, er det valgt at bruge JAXB, der blev også forsøg med at manuelt at danne DOM objekter[Wik13], men dette blev hurtigt til en meget rodet metode og derfor droppet. Idéen med JAXB er, at der i de klasser, der ønskes omdannet (kaldet marshalling i denne sammenhæng) tilføjes nogle @XML tags, der beskriver, hvordan de forskellige attributter skal håndteres i XML, og som JAXB API'et så kan konvertere automatisk. Det var idéen - men det gik ikke helt uden problemer, da JAXB ikke kunne håndtere Data_attribute objektet fra IEC 61850 datamodellen. Data_attribute bryd en regel i JAXB om at et objekt ikke både kan have en værdi og under objekter. For at løse dette, var jeg nødt til at oprette nogle forskellige klasser for Data_attribute, hvilket programringsmæssigt ikke er så pænt, men det betød, at jeg kunne få JAXB konverteringen til at virke.

4.4.6 HTTP(S) server

Som nævnt er det som HTTP(S) server valgt, at der bruges de indbyggede API'er i Java til at danne ramme om denne. For at denne server kan køre samtidig med resten af systemet, bliver den lavet sådan, at i dens contructor oprettes en ny tråd til at køre sig selv i, og derved kan lade tråden, der kaldte contructor'en køre videre. Klassen er lavet til at tagge to argumenter i constructor'en, for hvilken port, der skal lyttes på, og om det skal være en HTTP eller HTTPS server. I tilfældet at der bruges HTTPS, vil serversocket'en selv stå for at forhandle om der bruges SSL eller TLS, alt efter, hvilken klienten understøtter, men hvor TLS er at fortrække. Men for at der kan oprettes en krypteret forbindelse, skal der først genereres en krypteringsnøgle(set), der kan inkluderes i Java koden. Denne nøgle kan laves via et lille program, der følger med i Java JDK, der hedder keytool[ORA13]. Med keytool kan der så generes en nøgle ud fra nogle forskellige argumenter, men til dette formål var en standardnøgle tilstrækkelig. Det er for nu udeladt at implementere bruger login eller anden type autentifikation, og at serveren kører multithreaded til flere klienter, da det vil komplicere systemtest med den begrænsede tid til projektet.

Når en klient kalder serveren, modtager serveren en lang Byte stream, som så skal behandles, for det er lettere at behandle kaldet. Der tages derfor en række skridt for at verificere, at kaldet er gyldigt og frasortere de dele af kaldet, der skal bruges i videre behandling. Hele denne behandling laves i HTTPRequest objektet, når det bare bliver initialiseret med et Byte array som argument, hvor man så efterfølgende kan kalde den information, man ønsker. Når det indkommende kald så er blevet lavet om til et HTTPRequest objekt, kan det

videregives til metoden Request_handler, der skal håndtere kaldet, omformulere det til en ACSI metode (ACSI metoden gennemgås senere i dette kapitel). Fra ACSI metoden returneres et IEC 61850 objekt. Dette er, hvad klienten har efterspurgt i det indledende kald, så før det kan returneres til klientten, skal det omdannes til XML via JAXB som beskrevet i et tidligere afsnit. Tilsidst bliver den genererede XML kode tilføjet en HTTP header med HTTP kode 200 OK og indhold type XML. Skulle en del af valideringen slå fejl, vil HTTP koden blive 400 istedet for Bad request(forkert forespørgsel).

4.4.7 ACSI server via REST

Som nævnt mappes HTTP URL'en til IEC 61850 datamodellen samt ACSI funktionerne. Der er i denne prototype lavet funktioner til at hente målingerne på det forskellige levels i objektet, samt muligheden for at hente loggen af en måling i et givet interval. Denne controller tager så URL'en og omdanner den til en række talværdier, der kan bruges til logisk at komme frem til det element, URL'en beskriver. Det alt efter, hvilken funktion, der er kaldt, henter de tilsvarende data via datalaget og får IEC61850_DTO_Builder til at danne et objekt ud fra dataerne. Den returnerer så dette objekt til HTTP server, der så står for konverteringen til XML og leveringen til klienten. Der er dannet nedestående tabel over funktionerne på de forskellige levels, og hvad deres tilsvarende ACSI funktion hedder. Sensor-kortene i systemet kaldes i denne sammen hæng for CH1-6 og de værdier der tages i selv SGU boksen kaldes for BUSBAR.

Url adresse: IP:port/SGU +	Argument	ACSI tilsvarende	Level
/	?GetAllDataValues	*1	1
/CH1	?GetAllDataValues	GetAllDataValues	2
/CH1/MMXU1	?GetAllDataValues	GetDataValues	3
/CH1/MMXU1/A	?GetAllDataValues	GetDataValues	4
/CH1/MMXU1/A/phsA	?GetAllDataValues ?GetLog *2	GetDataValuesla ListOfLogEntries	5
/CH1/MMXU1/A/phsA/magavg	?GetAllDataValues	GetDataValues	6
/CH1/MMXU1/A/phsA/magavg/f	?GetAllDataValues	GetDataValues	7

Table 4.3: ACSI funktioner implementeret i REST.

(*1) Det er ikke tilladt/muligt ifølge standarden at lave et GetAllDataValues kald på en hel server, men pga. fejlen, jeg nævnte, jeg havde lavet i 4.4.4, har

jeg valgt at tillade det, som om , det var blevet rettet, og alt var rykket et trin ned i datamodellen, så det vil svare til et kald på et logical device, som er tilladt. (*2) Der skal her også gives datoer med i formatet "&from=2013-06-21T04:59:54,to=2013-06-21T05:07:01", men der var ikke plads i tabellen.

Der er i appendiks D, E og F vedhæftet et par eksempler på svarene, der gives til klienter ved tre forskellige kald.

4.4.8 samlet system controller logic

Main klassen er den hvor det hele er samlet og den, der starter alle tråde og logning. Ved at det er samlet her, kan de forskellige moduler let ind- og udkommenteres under test og implementation, men bør nok i en fremtidig version bygges ind i ASCI controller klassen, så alt styring sker der. Det er denne klasse, der står for at initialisere settings og uddeler alle indstillinger til de andre klasser.

4.5 SCL konfigurations fil

Som IEC 61850-6 standarden dikterer, skal der laves en SCL fil, der beskriver enhedens funktioner, moduler og layout struktur. Selv om der kun her er tale om en prototype, og en konfigurations fil derfor ikke vil være nødvendig, vil det være smart at lave en alligevel, da det giver en idé om systemets korrekthed. Som en hjælp til udviklingen af denne fil er der fundet programmet OpenSCLConfigurator [Ort13], hvor man grafisk kan vælge de elementer og måletyper, en enhed udbyder, og den så hjælper med at generere SCL filen. Det er dog med en hvis kritisk tilgang at programmet bruges, da programmet ikke er færdigudviklet, og er blevet minimalt opdateret siden 2009.

Et udsnit af SCL filen ses nedefor. Den fulde version er vedhæftet som appendiks A, og findes i zip filen med resten af projektes kode.

```
1 <IED name="SGU0001" type="SMART GRID UNIT" manufacturer="Thiim A/S"
      configVersion="1">
2   <AccessPoint name="AccessPoint1" router="false" clock="false">
3     <Server timeout="30">
4       <Authentication none="false" password="false" weak="false" strong="false" certificate
            ="true" />
5       <LDevice inst="CH1">
6         <LN0 lnType="LLN01" lnClass="LLN0" inst="1" />
7         <LN lnType="LPHD1" lnClass="LPHD" inst="1" prefix="" />
8         <LN lnType="MMXU1" lnClass="MMXU" inst="1" prefix="" />
```

```
9         </LDevice>
```

Den kritiske holdning til OpenSCLConfigurator programmet viste sig at være korrekt, da det ikke fik oprettet målingerne Det betød, at et sæt af 3 faser ikke blev oprette korrekt, og disse så måtte manuelt tilføjes. Et problem, der dog opstod, var, hvordan sensorenes sikrings status skulle håndteres. Der kunne ikke findes en specifik objektklasse i IEC 61850-7-4. Hvorledes sikringerne så skal håndteres, vurderes til 3 forskellige metoder- Første er at 'Quality' målingerne, sensoren tager, sættes til Bad, når sikringen er gået. Anden er, at logical node: LPHD.health sættes til Bad, når sikringen er gået. Den sidste mulighed er, at der oprettes en Generic process I/O logical node, hvor det håndteres. Der er valgt at bruge metode nr. 2 hvor det er 'Quality' målingerne, der viser om sikringen er gået, da det den simpleste og som nævnt i 4.4.4 er LPHD delen faldet for afgræsningen af projektet.

4.6 Delkonklusion

IEC 61850 standarden har som forventet været udtrolig svær at implementer, og det er ikke fordi koden er svær, men fordi standarden strækker sig så bedt som den gør, og der derfor skal gennemlæses utroligt meget materiale for rent faktisk at finde de informationer man søger, så men den meget korte tid der er givet til sådan et projekt er det forventet at jeg nok har misset et par ting. Men med det sagt er de vitale dele som der blev stillet som krav til, i den første del af rapporten, blevet implementeret i sin helhed så system både tager rigtige målinger, behandler dem, og levere dem via den opsatte protokol. Det er også lykkes at implementer det hele på en lille computer svarende til hvad der er muligt at opgrader SGU til at blive kontrolleret af.

Kapitel 5

Test

I dette kapitel testes de forskellige dele af prototype både hver for sig og som et komplet system, for at se om systemmet lever op til de krav der er stillet det både af forfatter og den standard den implementere. Største delen af testene der bliver udført vil være af typen Greybox-testing, hvilket ikke er optimalt fordi man ikke kan udlukke alle fejl ved den teknik. Grunden til der så bruges Greybox er at der ikke findes nogle specifikke programmer som ville kunne bruges til, at teste denne nye standard jeg har implementeret. Men hvor det er muligt vil jeg brugt test værktøjer til at checke de enkelte elementer af koden. Under implementationen blev browseren fra Google, Chrome[Goo13] brugt som klient til at lave forspørgelserne, den vil også blive brugt her under testen, da den har en række af udvikler værktøjer der kan bruges.

5.1 XML

Til test af det XML svar som serveren sender til klienten, bruges Chrome's indbygget XML phaser(dekoder). Når der i Chrome besøges en server, hvor svaret enten er sat som værende XML i HTTP headeren eller hvor der returneres en fil af typen .xml, automatisk aktiveres Chrome's XML dekoder. Dekoderen vil så når man besøger HTTP serveren på prototypen, prøve at dekode XML dataerne i det return objekt der modtages og give en fejl hvis det ikke overholder

XML standarden. Der har dog kun været fejl af denne type i det meget tidlige stadige af udviklingen, hvor der blev eksperimenteret med at manuelt at skrive XML'en som string. Det er også at forvente at der ikke sker fejl her, da det laves med JAXB API'et, men med denne test kan det så bekræftes.

5.2 Dataset test

Da det er forholdsvis upraktisk at lave en decideret systematisk test, af om datasettene det modtages i XML er opbygget korrekt der denne del gjort manuelt, hvilket ikke er den mest pålidelige metode, men da der ikke findes nogen form for software til at tjekke at alle elementer og attributter er inkluderet korrekt, var dette den eneste mulighed, ud over selv at producere test softwaren. Men en grundig gennemgang af forfatter gav ingen manglende elementer, udover dem som var fravalgt i afgrænsningen. Den logiske opdeling af elementer, blev også fundet til at stemme overens med den der er lavet i SCL filen.

5.3 HTTP(S) server test

Tiden det tager for at få svar på en kald til HTTPS server er fundet ved hjælp af Chrome's udvikler værktøj, der både viser (se figur. 5.3) den faktiske tid kaldet tog, og latency'en på serveren. Det må her konkluderes at et en svar tid på 0,8-1,5 sek. er acceptablet, men bør sænkes til under 0,5 sek. så der kan hentes live målinger hvert sekund uden problemer. Nærmere undersøgelser med tidtagning i Java klasserne har vidst(se figur. 5.2) det er JAXB der tager langtid at køre på RPi'ens lille CPU, det vil være et punkt der så skal optimeres ved fremtidig videre udvikling hvis det skal være mulig at hente objekter hvert sekund. Der blev også lavet test med HTTP uden sikkerhed som var noget hurtigere til at forbinde ved første kald men ved de efterfølgende var den kun marginalt hurtigere.

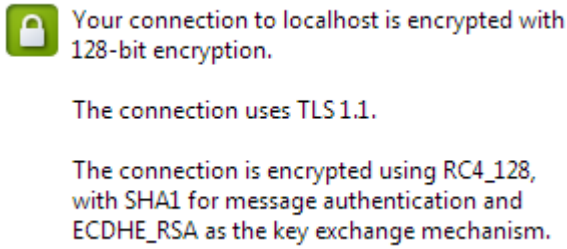
Name Path	Method	Status Text	Size Content	Time Latency
?GetAllDataValues /SGU	GET	200 OK	8.0 KB 7.9 KB	887 ms 879 ms

Figur 5.1: Tids test over internet via VPN.

```
   HTTPS thread Started.
1. GetAllDataValues
   JAXB tid: 1938
2. GetAllDataValues
   JAXB tid: 247
3. GetAllDataValues
   JAXB tid: 256
4. GetAllDataValues
   JAXB tid: 891
5. GetAllDataValues
   JAXB tid: 428
6. GetAllDataValues
   JAXB tid: 689
```

Figur 5.2: Undersøgelse af JAXB tider(millisekunder).

Når serveren besøges med Chrome vises det at forbindelsen forbindelsen som forventet køre krypteret og at de to parter har forhandlet sig til de bruger en TLS kryptering (se figur. 5.3).



Figur 5.3: Chrome viser forbindelsen er krypteret.

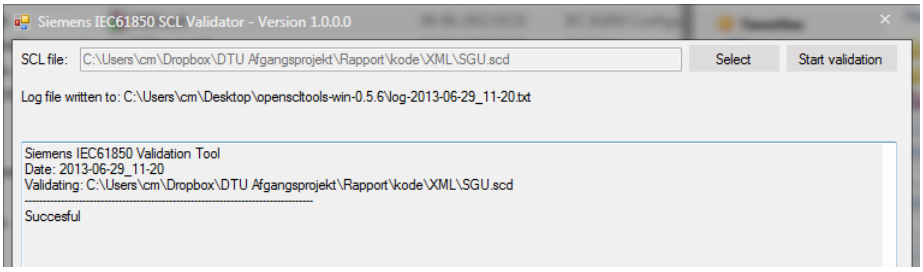
5.4 Database Test

Der blev for databasen udført test af stored procedures, ved der i HeidiSQL blev kaldt metoderne med forskellige data i alle fletter af argumentet efterfulgt af en visuel inspektion, af de var blevet tilføjet i de korrekte tabeller. Og omvendt for de SP'er der hentet data blev der tjekket om de hentet stemte overens med dem der fandtes i tabellerne. Hvis der var oprettet funktioner der matchede hinanden som getter og setter metoder kunne man have opstillet test kode til validering men da de ikke matchede kunne denne metode ikke bruges. Med logning hvert

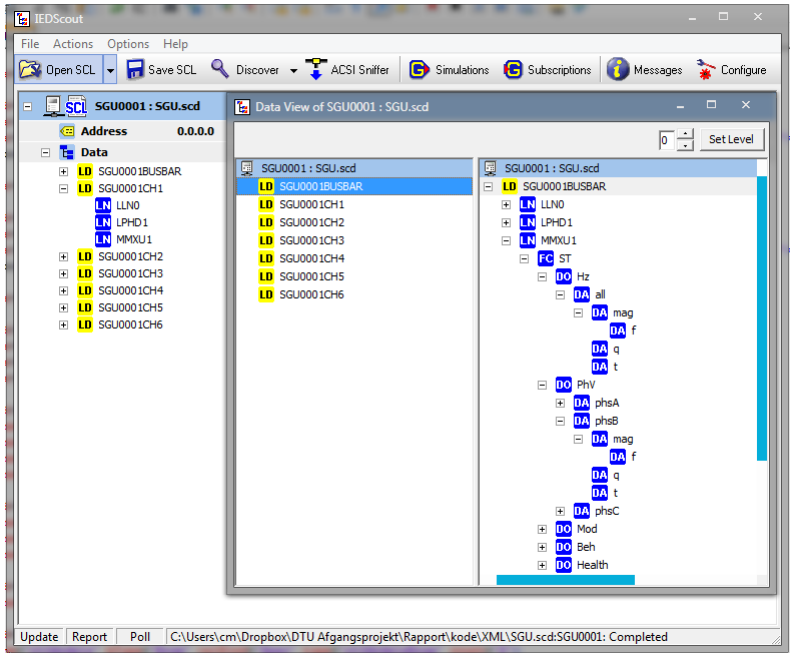
1 sek, blev det testet og beregnet til at give cirka 100 MB data om dagen for en SGU med alle 6 sensor kort. Med 100 MB om dagen vil det give cirka 3 GB om måneden hvilket er en størrelse der vil kunne findes plads til på et SD kort, så SGU'en vil kunne køre et måned uden at have forbindelse

5.5 SCL Validering

Som den eneste del der kan testes i relevans til IEC 61850 standarden, findes der et par værktøjer udover det der blev brugt til at danne SCL filen, som kan bruges til at teste om den er lavet korrekt. Det første der testes med er Siemens SCL Validator[Sie13], hvor en succesfuld validering er vist i figur. 5.4. Det andet program der blev brugt til at teste SCL filen var OMIcron IEDScout, som ikke bare validere men også giver en grafisk fremstilling af hvad filen siger i firgur. 5.5. Som nævnt tidligere i implementationen er der nogle objekter der ikke nåede at komme med i protypen, men de findes så i denne SCL fil for ellers ville den give fejl når den testes.



Figur 5.4: Test med Siemens SCL validator.

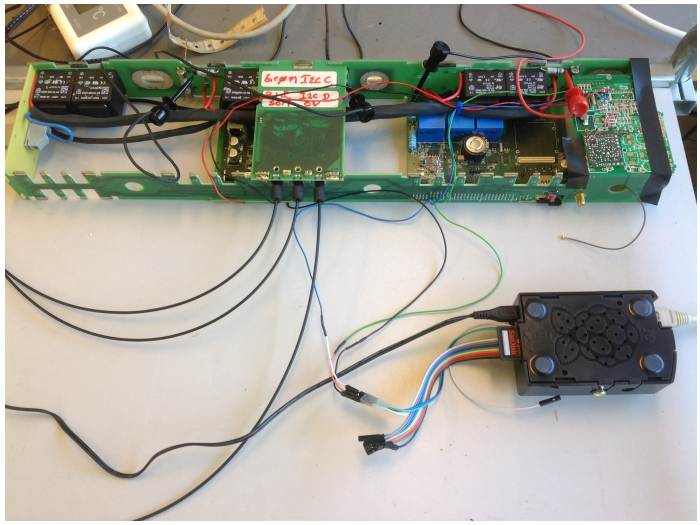


Figur 5.5: Test med OMIcron IEDScout.

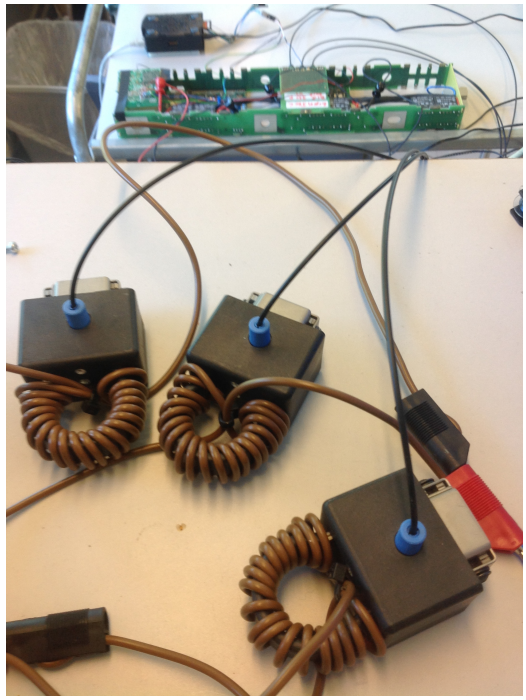
5.6 Drift test

Den mest omfattende test der blev udført var der i Thiim's el værksted blev opsat et komplet test scenarie, hvor der var rigtige strømme og spændinger i systemmet, som blev styret i skiftende mønster, via et computerstyret Watt-skab. I denne test kunne man se hvordan de skiftende mållinger blev behandlet korrekt også selvom der blev skiftet imellem positive og negative værdier for de af målingerne der var signed.

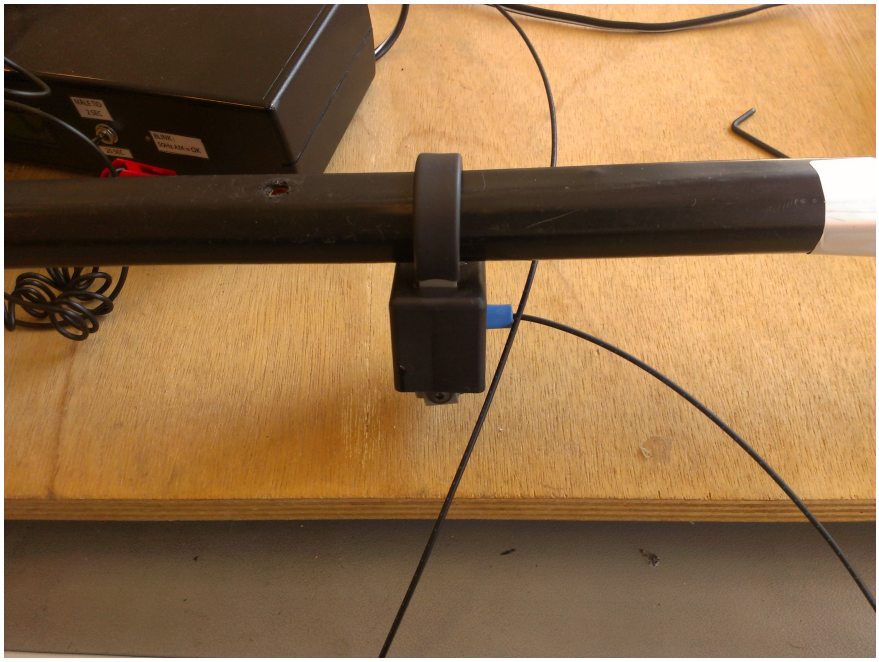
For at teste holdbarheden i kodens design, blev der også udført en weekend test, hvor den fik lov at stå og tage målinger 3 dage i træk, uden der opstod nogen problemer. RPi'en har praktisk talt faktisk stået tændt de sidste 2 måneder uden nogen problemer eller genstart, så det må siges at være et stabilt stykke hardware.



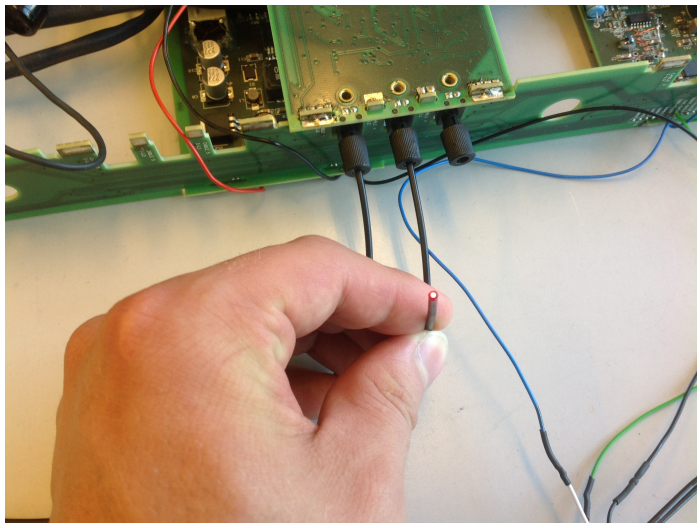
Figur 5.6: RPi'en tilsluttet SGU'en via I^2C.



Figur 5.7: 3 sensorer til test af funktionen af et helt SGU kort.



Figur 5.8: Test setup med variable strømkilde.



Figur 5.9: Eksempel på aktiv fiber forbindelse.

5.7 Delkonklusion

Testen af systemmet afslørede et problem i at JAXB API'et er for tungt til RPi'en kan køre med det hvis der skal kunne leveres eet liv signal med en frekvens på 1 sek. Dette er en ret central del af koden, og kan derfor ikke nå at blive udskiftet på dette tidspunkt i projektet. Men ellers kørte alle delene som de var designet til hvilket må konkluderes at være optimalt, specielt den lokale logning i databasen, og hentning af målings intervaller virkede over forventning. Men som nævn var det største problem ved at teste dette system at der ikke findes en rigtig REST-HTTP-IEC 61850 kompatible klient software af der kunne tilsluttes systemmet. Der er derfor min plan frem imod forsvaret af dette projekt at arbejde på at lave en minimal version af sådan en klient, som jeg så kan demonstrere til forsvaret.

Kapitel 6

Konklusion

6.1 Opsummering

Det er igennem projektet blevet skabt indsigt i IEC 61850 standarden, og lavet en dybdegående undersøgelse af mulighederne som en opgradering dertil kan bibringe Thiim's Smart Grid Unit. Det er blevet undersøgt, hvilken hardware opgradering, der skal til for at kunne opsætte SGU'en til være en IEC 61850 kompatibel server, hvilket inkluderer muligheden for, at enheden selvstædigt kan indsamle data lokalt. Der er blevet lavet en vurdering af, hvilken type kommunikation der skal implementere IEC 61850's ACSI kommunikationen, og designet en prototype der håndtere denne. Prototypen er blevet udsat for en række af testscenarier, der har vist, hvordan det kan forventes, at en fremtidige implementering vil kunne fungere. Der er blevet gennemgået en opgradering af systemsoftwaren til at være Linux baseret, og der er blevet fremlagt en række eksempler på, hvilken ekstra værdi dette vil bringe SGU'en. Igennem hele projektet har der været fokus på at sikre SGU'en. Der er blevet opsat et sikret system, der også tilbyder muligheden for flere typer af sikret kommunikation. Der er generelt blevet taget er stor skridt i undersøgelsen, af alle aspekter der er inkluderet i at skulle opgradere Thiim's SGU til en moderne enhed i THE SMART GRID.

6.2 Projekt konklusion

Ser man på de kriterier, der blev stillet i problemformuleringen i kapitel 1 og kravspecifikationen i kapitel 2 har projektet dækket langt de fleste og endt ud med en prototype langt over forventningerne. Der er selvfølgelig dele af prototypen, som stadig kan forbedres, og funktioner man godt kunne have ønsket sig tid til at nå at implementere. Men ser man på det som en helhed, er det faktisk lykkes at gå fra Byte registre i SGU'ens hardware til avancerede dataobjekter leveret til en klient i form at XML via en krypteret dataforbindelse på utrolig kort tid. At der så også sker en logning af målingerne på enheden, der ligeledes kan tilgås og hentes via den opsatte IEC 61850 server, er et stort ekstra plus. Dertil kommer, at systemmet, der er blevet opsat på RPi'en, giver et utal af nye muligheder udover IEC 61850 serveren. Disse muligheder kan - relativt simpelt - tilpasses nye kunders ønsker om funktionalitet. RaspberryPi'en har været et fantastisk udviklerværktøj, og har sparet mange timer fordi den er så let at anvende. Stor tak til folkene bag RaspberryPi.

6.3 Fremtidsarbejde/Videreudvikling

Som det er nævnt flere gange i løbet af rapporten, giver IEC 61850 standarden rig mulighed for at udbygge SGU'en med intelligente funktioner, hvor det vil være åbenlyst at arbejde på, at den skal være den styrende enhed på f.eks en mindre transformerstation. Her vil den kunne sættes til at overvåge alle strømlinjerne og evt. få muligheden for at frakoble disse linjer i tilfælde af fejl/overbelastning/overgang, via kommunikation til nogle styrede relæer. De ting, som er blevet nævnt som afgrænset i prototypen, skal selvfølgelig også implementeres, før enheden kan sættes i produktion, samt en udbygning af de allerede implementerede funktioner. Der skal tages stilling til, om det ønskes at isætte RPi'en i SGU enheden som værende det nye master kontroller board, eller om der skal designes et ny hardwareboard specielt til den, der også er Linux baseret. HTTPS server bør givet funktionaliteten af bruger validering, som en ekstra sikkerheds mulighed. Som systemmet er designet nu, skal indstillinger fortages i en konfigurationsfil, men det kunne overvejs at give enheden en administrations hjemmeside hvor dette vil kunne udføres.

6.4 Udtalelse fra Thiim

[Indsat på næste side]

Herlev, den 1. Juli 2013.

Bachelorprojekt:
Implementering af IEC 61850 i måleudstyr.

Udført af Christian Mühlendorph

Thiim A/S har gennem flere år arbejdet med udviklingen af en enhed til net overvågning af el-distributionen. Enheden kaldet SGU er hardwaremæssigt udviklet til at kunne overvåge op til 6 tre-fasede el-kabler og foretage måling af strømme og spændinger og efterfølgende matematisk beregne aktiv og reaktiv effekt, herunder retningen af effekterne.

En del enheder har med stor succes gennem 2½ år været testet i praksis i el-fordelings-skabe og i transformerstationer. I disse forsøgsudgaver har kommunikationen været baseret på transmission af SMS over mobilnettet, hvilket er en dyr og besværlig løsning, idet der ikke markedsføres generel software, der direkte kan modtage og behandle de indkomne data.

En videreudvikling med kommunikation via TCP/IP og Modbus har stadig ikke medført, at SGU'en kunne markedsføres som "hyldevare".

Et projekt med implementering af IEC 60870-5-104, er til vores store tilfredshed udført af Christian Mühlendorph under hans 20 ugers praktikperiode. Gennemførelsen af projektet har bragt os nærmere generelle anvendelser, og vi samarbejder nu med et stort energifirma om at indføre enhederne i transformerstationer.

For at komme bredt ind på verdensmarkedet har vi indset, at vores udstyr nødvendigvis må understøtte IEC 61850, der i dag er den norm for kommunikation, der vil blive den fremherskende i kommunikationen mellem måleenheder og kontrolcentraler. Normen er omfattende og uoverskuelig, og vi havde svært ved at få overblik over, hvad der ville kræves af hardwareændringer for at kunne foretage den omfattende lagring af data og styring af kommunikationen med omverden.

Christian har gennem sit bachelorprojekt både demonstreret, at vores SGU kan sende data, som krævet i normen, men også klarlagt hvilke hardwareændringer, vi med fordel kan indføre for at opnå den tilsikrede funktion.

Christians projekt har bragt os langt videre i udviklingen af et produkt med meget store eksportmuligheder, og vi er yderst tilfredse med hans indsats.

Med venlig hilsen
Thiim A/S

Søren Thiim

Thiim A/S Transformervej 31, 2730 Herlev - Denmark | CVR / VAT DK34649510 | Tel.: +45 4485 8000 | Fax: +45 4485 8005 | Web/E-shop: www.thiim.com | e-mail: thiim@thiim.com

Appendix A

SGU.scd konfigurations fil

kode/XML/SGU.scd

```
<?xml version="1.0" encoding="utf-8"?>
<SCL xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xmlns="http://www.iec.ch/61850/2003/SCL">
  <Header id="SGU mapping" version="1" revision="1" nameStructure="IEDName">
    <History>
      <Hitem version="1" revision="1" when="01-06-2013 22:08:52" who="cm" what="" why
          ="New SCL" />
    </History>
  </Header>
  <IED name="SGU0001" type="SMART GRID UNIT" manufacturer="Thiim A/S"
      configVersion="1">
    <Services>
      <DynAssociation />
      <GetDirectory />
      <GetDataObjectDefinition />
      <GetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet max="4" maxAttributes="50" modify="true" />
      <ReadWrite />
      <ConfReportControl max="12" />
      <GetCBValues />
      <ConfLogControl max="1" />
      <ReportSettings cbName="Conf" datSet="Conf" rptID="Dyn" optFields="Conf"
          bufTime="Dyn" trgOps="Fix" intgPd="Dyn" />
      <FileHandling />
      <ConfLNs fixPrefix="false" fixLnInst="true" />
    </Services>
    <AccessPoint name="SGU" router="false" clock="false">
      <Server timeout="30">
        <Authentication none="false" password="false" weak="false" strong="false" certificate
            ="true" />
```

```
27        <LDevice inst="CH1">
28          <LN0 lnType="LLN01" lnClass="LLN0" inst="1" />
29          <LN lnType="LPHD1" lnClass="LPHD" inst="1" prefix="" />
30          <LN lnType="MMXU1" lnClass="MMXU" inst="1" prefix="" />
31        </LDevice>
32        <LDevice inst="CH2">
33          <LN0 lnType="LLN01" lnClass="LLN0" inst="1" />
34          <LN lnType="LPHD1" lnClass="LPHD" inst="1" prefix="" />
35          <LN lnType="MMXU1" lnClass="MMXU" inst="1" prefix="" />
36        </LDevice>
37        <LDevice inst="CH3">
38          <LN0 lnType="LLN01" lnClass="LLN0" inst="1" />
39          <LN lnType="LPHD1" lnClass="LPHD" inst="1" prefix="" />
40          <LN lnType="MMXU1" lnClass="MMXU" inst="1" prefix="" />
41        </LDevice>
42        <LDevice inst="CH4">
43          <LN0 lnType="LLN01" lnClass="LLN0" inst="1" />
44          <LN lnType="LPHD1" lnClass="LPHD" inst="1" prefix="" />
45          <LN lnType="MMXU1" lnClass="MMXU" inst="1" prefix="" />
46        </LDevice>
47        <LDevice inst="CH5">
48          <LN0 lnType="LLN01" lnClass="LLN0" inst="1" />
49          <LN lnType="LPHD1" lnClass="LPHD" inst="1" prefix="" />
50          <LN lnType="MMXU1" lnClass="MMXU" inst="1" prefix="" />
51        </LDevice>
52        <LDevice inst="CH6">
53          <LN0 lnType="LLN01" lnClass="LLN0" inst="1" />
54          <LN lnType="LPHD1" lnClass="LPHD" inst="1" prefix="" />
55          <LN lnType="MMXU1" lnClass="MMXU" inst="1" prefix="" />
56        </LDevice>
57        <LDevice inst="BUSBAR">
58          <LN0 lnType="LLN01" lnClass="LLN0" inst="1">
59            <DOI name="Mod">
60              <SDI name="ctlModel">
61                <DAI name="ctlModels" valKind="Spec">
62                  <Val>status_only</Val>
63                </DAI>
64              </SDI>
65              <DAI name="q" valKind="Spec" />
66              <DAI name="stVal" valKind="Spec">
67                <Val>on</Val>
68              </DAI>
69              <DAI name="t" valKind="Spec" />
70            </DOI>
71            <DOI name="Beh">
72              <DAI name="q" valKind="Spec" />
73              <DAI name="stVal" valKind="Spec">
74                <Val>on</Val>
75              </DAI>
76              <DAI name="t" valKind="Spec" />
77            </DOI>
78            <DOI name="Health">
79              <DAI name="q" valKind="Spec" />
80              <DAI name="stVal" valKind="Spec">
81                <Val>on</Val>
82              </DAI>
83              <DAI name="t" valKind="Spec" />
84            </DOI>
85            <DOI name="NamPlt">
86              <DAI name="d" valKind="Spec" />
87              <DAI name="swRev" valKind="Spec" />
88              <DAI name="vendor" valKind="Spec" />
89            </DOI>
90          </LN0>
91          <LN lnType="LPHD1" lnClass="LPHD" inst="1" prefix="">
```

```
92            <DOI name="PhyNam">
93              <DAI name="vendor" valKind="Spec" />
94            </DOI>
95            <DOI name="PhyHealth">
96              <DAI name="q" valKind="Spec" />
97              <DAI name="stVal" valKind="Spec">
98                <Val>on</Val>
99              </DAI>
100             <DAI name="t" valKind="Spec" />
101           </DOI>
102           <DOI name="Proxy">
103             <DAI name="q" valKind="Spec" />
104             <DAI name="stVal" valKind="Spec" />
105             <DAI name="t" valKind="Spec" />
106           </DOI>
107           <DOI name="Mod">
108             <SDI name="ctlModel">
109               <DAI name="ctlModels" valKind="Spec">
110                 <Val>status_only</Val>
111               </DAI>
112             </SDI>
113             <DAI name="q" valKind="Spec" />
114             <DAI name="stVal" valKind="Spec">
115               <Val>on</Val>
116             </DAI>
117             <DAI name="t" valKind="Spec" />
118           </DOI>
119           <DOI name="Beh">
120             <DAI name="q" valKind="Spec" />
121             <DAI name="stVal" valKind="Spec">
122               <Val>on</Val>
123             </DAI>
124             <DAI name="t" valKind="Spec" />
125           </DOI>
126           <DOI name="Health">
127             <DAI name="q" valKind="Spec" />
128             <DAI name="stVal" valKind="Spec">
129               <Val>on</Val>
130             </DAI>
131             <DAI name="t" valKind="Spec" />
132           </DOI>
133           <DOI name="NamPlt">
134             <DAI name="d" valKind="Spec" />
135             <DAI name="swRev" valKind="Spec" />
136             <DAI name="vendor" valKind="Spec" />
137           </DOI>
138         </LN>
139         <LN lnType="MMXU2" lnClass="MMXU" inst="1" prefix="" />
140       </LDevice>
141     </Server>
142   </AccessPoint>
143 </IED>
144 <DataTypeTemplates>
145   <LNodeType id="LLN01" iedType="" lnClass="LLN0">
146     <DO name="Mod" type="Mod" transient="false" />
147     <DO name="Beh" type="Beh" transient="false" />
148     <DO name="Health" type="Health" transient="false" />
149     <DO name="NamPlt" type="NamPlt" transient="false" />
150   </LNodeType>
151   <LNodeType id="LPHD1" iedType="" lnClass="LPHD">
152     <DO name="PhyNam" type="LPHD1PhyNam" transient="false" />
153     <DO name="PhyHealth" type="LPHD1PhyHealth" transient="false" />
154     <DO name="Proxy" type="LPHD1Proxy" transient="false" />
155     <DO name="Mod" type="Mod" transient="false" />
156     <DO name="Beh" type="Beh" transient="false" />
```

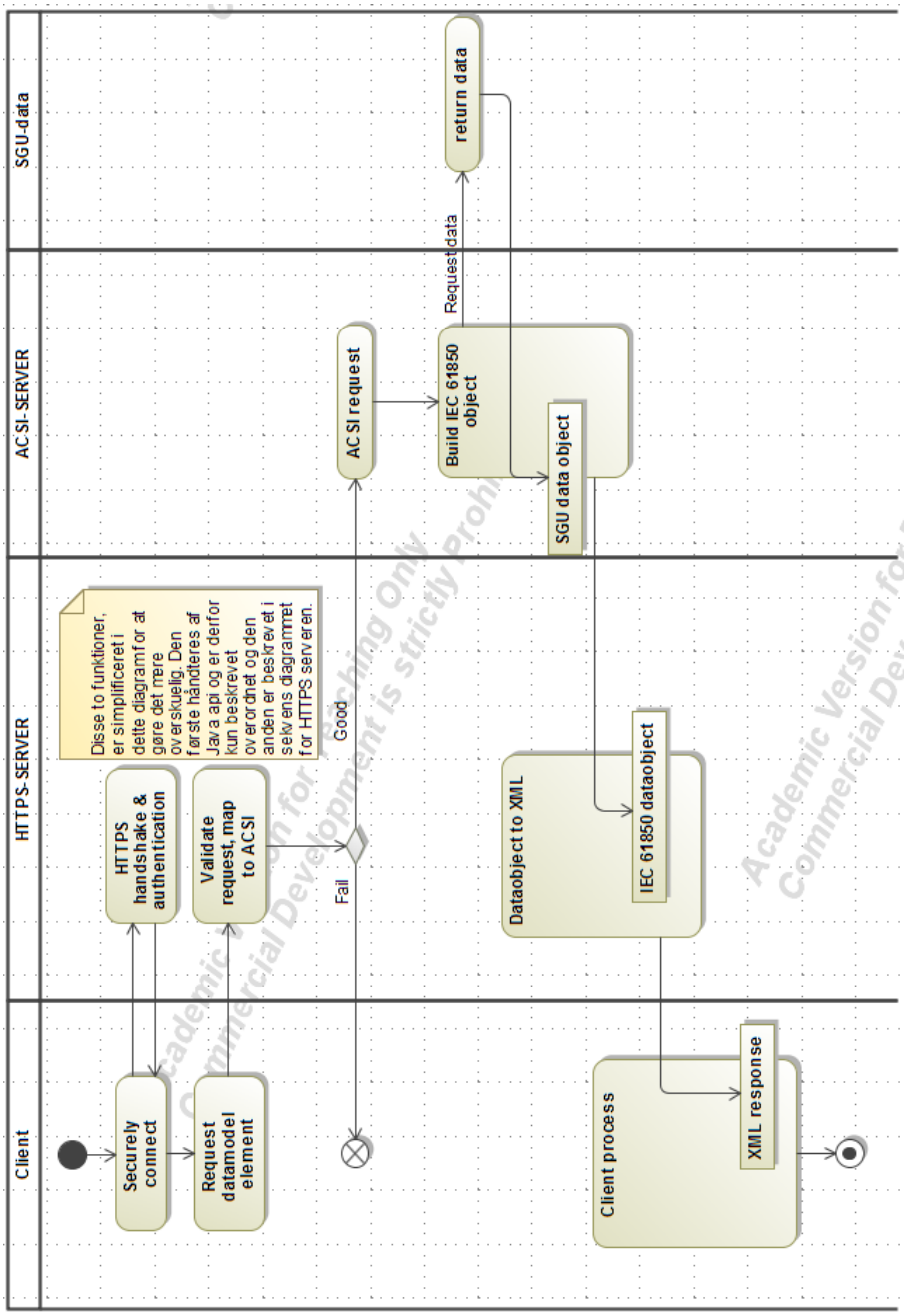
```
157       <DO name="Health" type="Health" transient="false" />
158       <DO name="NamPlt" type="NamPlt" transient="false" />
159     </LNodeType>
160     <LNodeType id="MMXU1" iedType="" lnClass="MMXU">
161       <DO name="A" type="MMXU1A" transient="false" />
162       <DO name="W" type="MMXU1W" transient="false" />
163       <DO name="VAr" type="MMXU1VAr" transient="false" />
164       <DO name="Mod" type="Mod" transient="false" />
165       <DO name="Beh" type="Beh" transient="false" />
166       <DO name="Health" type="Health" transient="false" />
167       <DO name="NamPlt" type="NamPlt" transient="false" />
168     </LNodeType>
169     <LNodeType id="MMXU2" iedType="" lnClass="MMXU">
170       <DO name="Hz" type="MMXU2Hz" transient="false" />
171       <DO name="PhV" type="MMXU2PhV" transient="false" />
172       <DO name="Mod" type="Mod" transient="false" />
173       <DO name="Beh" type="Beh" transient="false" />
174       <DO name="Health" type="Health" transient="false" />
175       <DO name="NamPlt" type="NamPlt" transient="false" />
176     </LNodeType>
177     <DOType id="LPHD1PhyNam" iedType="" cdc="DPL">
178       <DA name="vendor" bType="VisString255" valKind="Spec" count="0" dchg="false"
              qchg="false" dupd="false" fc="DC" />
179     </DOType>
180     <DOType id="LPHD1PhyHealth" iedType="" cdc="INS">
181       <DA name="q" bType="Quality" valKind="Spec" count="0" dchg="false" qchg="false"
              dupd="false" fc="ST" />
182       <DA name="stVal" bType="Enum" valKind="Spec" type=" stValEnum" count="0" dchg
              ="false" qchg="false" dupd="false" fc="ST">
183         <Val>on</Val>
184       </DA>
185       <DA name="t" bType="Timestamp" valKind="Spec" count="0" dchg="false" qchg="
              false" dupd="false" fc="ST" />
186     </DOType>
187     <DOType id="LPHD1Proxy" iedType="" cdc="SPS">
188       <DA name="q" bType="Quality" valKind="Spec" count="0" dchg="false" qchg="false"
              dupd="false" fc="ST" />
189       <DA name="stVal" bType="BOOLEAN" valKind="Spec" count="0" dchg="false" qchg=
              "false" dupd="false" fc="ST" />
190       <DA name="t" bType="Timestamp" valKind="Spec" count="0" dchg="false" qchg="
              false" dupd="false" fc="ST" />
191     </DOType>
192     <DOType id="MMXU1A" iedType="" cdc="MV">
193       <DA name="phsA" bType="Struct" valKind="Spec" type="MMXU1Aphs" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
194       <DA name="phsB" bType="Struct" valKind="Spec" type="MMXU1Aphs" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
195       <DA name="phsC" bType="Struct" valKind="Spec" type="MMXU1Aphs" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
196     </DOType>
197     <DOType id="MMXU1W" iedType="" cdc="WYE">
198       <DA name="phsA" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
199       <DA name="phsB" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
200       <DA name="phsC" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
201     </DOType>
202     <DOType id="MMXU1VAr" iedType="" cdc="WYE">
203       <DA name="phsA" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
204       <DA name="phsB" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
205       <DA name="phsC" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
```

```
206     </DOType>
207     <DOType id="MMXU2Hz" iedType="" cdc="MV">
208       <DA name="all" bType="Struct" valKind="Spec" type="MMXU1obj" count="0" dchg=
              "false" qchg="false" dupd="false" fc="ST" />
209     </DOType>
210     <DOType id="MMXU2PhV" iedType="" cdc="WYE">
211       <DA name="phsA" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
212       <DA name="phsB" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
213       <DA name="phsC" bType="Struct" valKind="Spec" type="MMXU1obj" count="0"
              dchg="false" qchg="false" dupd="false" fc="ST" />
214     </DOType>
215     <DOType id="Mod" iedType="" cdc="INC">
216       <DA name="ctlModel" bType="Struct" valKind="Spec" type="ModctlModel" count="0"
               dchg="false" qchg="false" dupd="false" fc="CF" />
217       <DA name="q" bType="Quality" valKind="Spec" count="0" dchg="false" qchg="false"
              dupd="false" fc="ST" />
218       <DA name="stVal" bType="Enum" valKind="Spec" type=" stValEnum" count="0" dchg
              ="false" qchg="false" dupd="false" fc="ST">
219         <Val>on</Val>
220       </DA>
221       <DA name="t" bType="Timestamp" valKind="Spec" count="0" dchg="false" qchg="
              false" dupd="false" fc="ST" />
222     </DOType>
223     <DOType id="Beh" iedType="" cdc="INS">
224       <DA name="q" bType="Quality" valKind="Spec" count="0" dchg="false" qchg="false"
              dupd="false" fc="ST" />
225       <DA name="stVal" bType="Enum" valKind="Spec" type=" stValEnum" count="0" dchg
              ="false" qchg="false" dupd="false" fc="ST">
226         <Val>on</Val>
227       </DA>
228       <DA name="t" bType="Timestamp" valKind="Spec" count="0" dchg="false" qchg="
              false" dupd="false" fc="ST" />
229     </DOType>
230     <DOType id="Health" iedType="" cdc="INS">
231       <DA name="q" bType="Quality" valKind="Spec" count="0" dchg="false" qchg="false"
              dupd="false" fc="ST" />
232       <DA name="stVal" bType="Enum" valKind="Spec" type=" stValEnum" count="0" dchg
              ="false" qchg="false" dupd="false" fc="ST">
233         <Val>on</Val>
234       </DA>
235       <DA name="t" bType="Timestamp" valKind="Spec" count="0" dchg="false" qchg="
              false" dupd="false" fc="ST" />
236     </DOType>
237     <DOType id="NamPlt" iedType="" cdc="LPL">
238       <DA name="d" bType="VisString255" valKind="Spec" count="0" dchg="false" qchg="
              false" dupd="false" fc="DC" />
239       <DA name="swRev" bType="VisString255" valKind="Spec" count="0" dchg="false"
              qchg="false" dupd="false" fc="DC" />
240       <DA name="vendor" bType="VisString255" valKind="Spec" count="0" dchg="false"
              qchg="false" dupd="false" fc="DC" />
241     </DOType>
242     <DAType id="MMXU1Aphs" iedType="">
243       <BDA name="magavg" bType="Struct" valKind="Spec" type="AnalogueValue" count="
              0" />
244       <BDA name="magpeek" bType="Struct" valKind="Spec" type="AnalogueValue" count=
              "0" />
245       <BDA name="q" bType="Quality" valKind="Spec" count="0" />
246       <BDA name="t" bType="Timestamp" valKind="Spec" count="0" />
247     </DAType>
248     <DAType id="MMXU1obj" iedType="">
249       <BDA name="mag" bType="Struct" valKind="Spec" type="AnalogueValue" count="0"
              />
250       <BDA name="q" bType="Quality" valKind="Spec" count="0" />
```

```
251       <BDA name="t" bType="Timestamp" valKind="Spec" count="0" />
252     </DAType>
253     <DAType id="AnalogueValue" iedType="">
254       <BDA name="f" bType="FLOAT32" valKind="Spec" count="0" />
255     </DAType>
256     <DAType id="ModctlModel" iedType="">
257       <BDA name="ctlModels" bType="Enum" valKind="Spec" type="ctlModelsEnum" count
              ="0">
258         <Val>status_only</Val>
259       </BDA>
260     </DAType>
261     <EnumType id="ctlModelsEnum">
262       <EnumVal ord="0">status_only</EnumVal>
263       <EnumVal ord="1">direct_with_normal_security</EnumVal>
264       <EnumVal ord="2">sbo_with_normal_security</EnumVal>
265       <EnumVal ord="3">sbo_with_enhanced_security</EnumVal>
266       <EnumVal ord="4">direct_with_enhanced_security</EnumVal>
267     </EnumType>
268     <EnumType id=" stValEnum">
269       <EnumVal ord="0">on</EnumVal>
270       <EnumVal ord="1">blocked</EnumVal>
271       <EnumVal ord="2">test</EnumVal>
272       <EnumVal ord="3">test_blocked</EnumVal>
273       <EnumVal ord="4">off</EnumVal>
274     </EnumType>
275   </DataTypeTemplates>
276 </SCL>
```

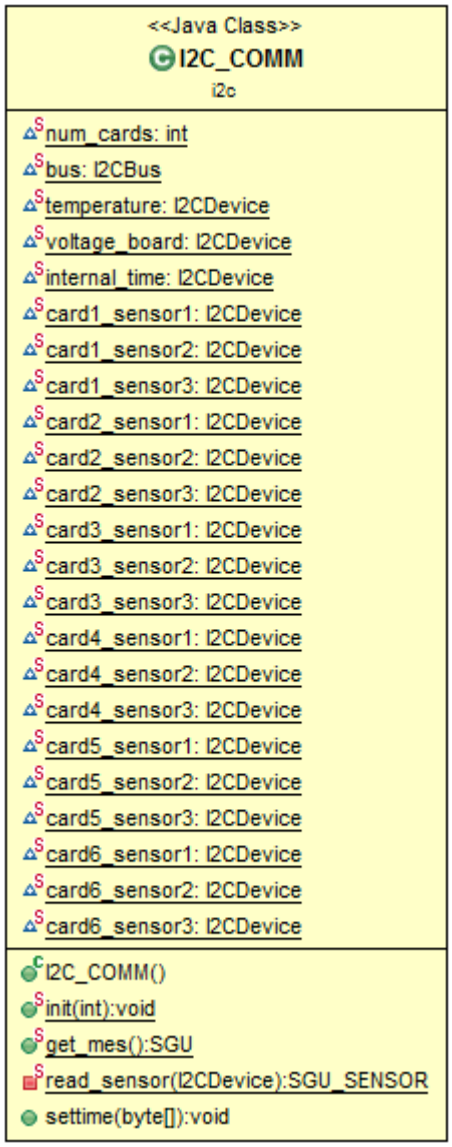
Appendix B

HTTPS kald til REST


Client
HTTPS-SERVER
ACSI-SERVER
SGU-data
Securely connect
HTTPS handshake & authentication
Request datamodel element
Validate request, map to ACSI
Disse to funktioner, er simplificeret i dette diagramfor at gøre det mere overskuelig. Den første håndteres af Java api og er derfor kun beskrevet overordnet og den anden er beskrevet i sekvens diagrammet for HTTPS serveren.
Fail
Good
ACSI request
Build IEC 61850 object
Request data
return data
SGU data object
Dataobject to XML
IEC 61850 dataobject
Client process
XML response

Appendix C

I2C_COMM klasse.


<<Java Class>>
I2C_COMM
i2c
num_cards: int
bus: I2CBus
temperature: I2CDevice
voltage_board: I2CDevice
internal_time: I2CDevice
card1_sensor1: I2CDevice
card1_sensor2: I2CDevice
card1_sensor3: I2CDevice
card2_sensor1: I2CDevice
card2_sensor2: I2CDevice
card2_sensor3: I2CDevice
card3_sensor1: I2CDevice
card3_sensor2: I2CDevice
card3_sensor3: I2CDevice
card4_sensor1: I2CDevice
card4_sensor2: I2CDevice
card4_sensor3: I2CDevice
card5_sensor1: I2CDevice
card5_sensor2: I2CDevice
card5_sensor3: I2CDevice
card6_sensor1: I2CDevice
card6_sensor2: I2CDevice
card6_sensor3: I2CDevice
I2C_COMM()
init(int):void
get_mes():SGU
read_sensor(I2CDevice):SGU_SENSOR
settime(byte[]):void

Appendix D

REST kald 1: /SGU/?GetAllDataValues

kode/XML/SGU.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<server>
    <LD name="CH1">
        <LN name="MMXU1" ref="SGU/CH1/MMXU1" type="MMXU">
            <DO name="A" ref="SGU/CH1/MMXU1/A" type="MV">
                <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/A/phsA" type="
                    struct">
                    <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/A/phsA/magavg"
                        type="struct">
                        <DA name="f" ref="SGU/CH1/MMXU1/A/phsA/magavg/f" type="
                            FLOAT32">23.2</DA>
                    </DA>
                    <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/A/phsA/magpeek"
                        type="struct">
                        <DA name="f" ref="SGU/CH1/MMXU1/A/phsA/magpeek/f" type="
                            FLOAT32">43.3</DA>
                    </DA>
                    <DA name="t" ref="SGU/CH1/MMXU1/A/phsA/magavg/t" type="
                        Timestamp">2013-07-01_13:40:57</DA>
                    <DA name="q" ref="SGU/CH1/MMXU1/A/phsA/magavg/q" type="
                        Quality">Good</DA>
                </DA>
                <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/A/phsB" type="
                    struct">
                    <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/A/phsB/magavg"
                        type="struct">
                        <DA name="f" ref="SGU/CH1/MMXU1/A/phsB/magavg/f" type="
                            FLOAT32">23.2</DA>
```

```
19                  </DA>
20                  <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/A/phsB/magpeek"
                        type="struct">
21                      <DA name="f" ref="SGU/CH1/MMXU1/A/phsB/magpeek/f" type="
                            FLOAT32">43.3</DA>
22                  </DA>
23                  <DA name="t" ref="SGU/CH1/MMXU1/A/phsB/magavg/t" type="
                        Timestamp">2013-07-01_13:40:57</DA>
24                  <DA name="q" ref="SGU/CH1/MMXU1/A/phsB/magavg/q" type="
                        Quality">Good</DA>
25              </DA>
26              <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/A/phsC" type="
                    struct">
27                  <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/A/phsC/magavg"
                        type="struct">
28                      <DA name="f" ref="SGU/CH1/MMXU1/A/phsC/magavg/f" type="
                            FLOAT32">23.2</DA>
29                  </DA>
30                  <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/A/phsC/magpeek"
                        type="struct">
31                      <DA name="f" ref="SGU/CH1/MMXU1/A/phsC/magpeek/f" type="
                            FLOAT32">43.3</DA>
32                  </DA>
33                  <DA name="t" ref="SGU/CH1/MMXU1/A/phsC/magavg/t" type="
                        Timestamp">2013-07-01_13:40:57</DA>
34                  <DA name="q" ref="SGU/CH1/MMXU1/A/phsC/magavg/q" type="
                        Quality">Good</DA>
35              </DA>
36          </DO>
37          <DO name="W" ref="SGU/CH1/MMXU1/W" type="WYE">
38              <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/W/phsA" type="
                    struct">
39                  <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/W/phsA/mag" type=
                        "struct">
40                      <DA name="f" ref="SGU/CH1/MMXU1/W/phsA/mag/f" type="
                            FLOAT32">34.4</DA>
41                  </DA>
42                  <DA name="t" ref="SGU/CH1/MMXU1/W/phsA/mag/t" type="
                        Timestamp">2013-07-01_13:40:57</DA>
43                  <DA name="q" ref="SGU/CH1/MMXU1/W/phsA/mag/q" type="Quality"
                        >Good</DA>
44              </DA>
45              <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/W/phsB" type="
                    struct">
46                  <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/W/phsB/mag" type=
                        "struct">
47                      <DA name="f" ref="SGU/CH1/MMXU1/W/phsB/mag/f" type="
                            FLOAT32">34.4</DA>
48                  </DA>
49                  <DA name="t" ref="SGU/CH1/MMXU1/W/phsB/mag/t" type="
                        Timestamp">2013-07-01_13:40:57</DA>
50                  <DA name="q" ref="SGU/CH1/MMXU1/W/phsB/mag/q" type="Quality"
                        >Good</DA>
51              </DA>
52              <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/W/phsC" type="
                    struct">
53                  <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/W/phsC/mag" type=
                        "struct">
54                      <DA name="f" ref="SGU/CH1/MMXU1/W/phsC/mag/f" type="
                            FLOAT32">34.4</DA>
55                  </DA>
56                  <DA name="t" ref="SGU/CH1/MMXU1/W/phsC/mag/t" type="
                        Timestamp">2013-07-01_13:40:57</DA>
57                  <DA name="q" ref="SGU/CH1/MMXU1/W/phsC/mag/q" type="Quality"
                        >Good</DA>
```

```
58                </DA>
59            </DO>
60            <DO name="VAr" ref="SGU/CH1/MMXU1/VAr" type="WYE">
61                <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsA" type=
                      "struct">
62                    <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsA/mag" type
                          ="struct">
63                        <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsA/mag/f" type="
                              FLOAT32">23.1</DA>
64                    </DA>
65                    <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsA/mag/t" type="
                          Timestamp">2013-07-01_13:40:57</DA>
66                    <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsA/mag/q" type="
                          Quality">Good</DA>
67                </DA>
68                <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsB" type=
                      "struct">
69                    <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsB/mag" type
                          ="struct">
70                        <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsB/mag/f" type="
                              FLOAT32">23.1</DA>
71                    </DA>
72                    <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsB/mag/t" type="
                          Timestamp">2013-07-01_13:40:57</DA>
73                    <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsB/mag/q" type="
                          Quality">Good</DA>
74                </DA>
75                <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsC" type=
                      "struct">
76                    <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsC/mag" type
                          ="struct">
77                        <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsC/mag/f" type="
                              FLOAT32">23.1</DA>
78                    </DA>
79                    <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsC/mag/t" type="
                          Timestamp">2013-07-01_13:40:57</DA>
80                    <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsC/mag/q" type="
                          Quality">Good</DA>
81                </DA>
82            </DO>
83        </LN>
84    </LD>
85    <LD name="BUSBAR">
86        <LN name="MMXU1" ref="SGU/BUSBAR/MMXU1" type="MMXU">
87            <DO name="Hz" ref="SGU/BUSBAR/MMXU1/Hz" type="WYE">
88                <DA fc="SP" name="MMXU1obj" ref="SGU/BUSBAR/MMXU1/Hz/all" type
                      ="struct">
89                    <DA fc="SP" name="mag" ref="SGU/BUSBAR/MMXU1/Hz/all/mag"
                          type="struct">
90                        <DA name="f" ref="SGU/BUSBAR/MMXU1/Hz/all/mag/f" type="
                              FLOAT32">5.0</DA>
91                    </DA>
92                    <DA name="t" ref="SGU/BUSBAR/MMXU1/Hz/all/mag/t" type="
                          Timestamp">2013-07-01_13:40:57</DA>
93                    <DA name="q" ref="SGU/BUSBAR/MMXU1/Hz/all/mag/q" type="
                          Quality">Good</DA>
94                </DA>
95            </DO>
96            <DO name="PhV" ref="SGU/BUSBAR/MMXU1/PhV" type="WYE">
97                <DA fc="SP" name="MMXU1obj" ref="SGU/BUSBAR/MMXU1/PhV/phsA"
                      type="struct">
98                    <DA fc="SP" name="mag" ref="SGU/BUSBAR/MMXU1/PhV/phsA/mag
                          " type="struct">
99                        <DA name="f" ref="SGU/BUSBAR/MMXU1/PhV/phsA/mag/f" type
                              ="FLOAT32">230.0</DA>
```

```
100                     </DA>
101                     <DA name="t" ref="SGU/BUSBAR/MMXU1/PhV/phsA/mag/t" type="
                            Timestamp">2013-07-01_13:40:57</DA>
102                     <DA name="q" ref="SGU/BUSBAR/MMXU1/PhV/phsA/mag/q" type="
                            Quality">Good</DA>
103                 </DA>
104                 <DA fc="SP" name="MMXU1obj" ref="SGU/BUSBAR/MMXU1/PhV/phsB"
                        type="struct">
105                     <DA fc="SP" name="mag" ref="SGU/BUSBAR/MMXU1/PhV/phsB/mag"
                            type="struct">
106                         <DA name="f" ref="SGU/BUSBAR/MMXU1/PhV/phsB/mag/f" type=
                                "FLOAT32">231.0</DA>
107                     </DA>
108                     <DA name="t" ref="SGU/BUSBAR/MMXU1/PhV/phsB/mag/t" type="
                            Timestamp">2013-07-01_13:40:57</DA>
109                     <DA name="q" ref="SGU/BUSBAR/MMXU1/PhV/phsB/mag/q" type="
                            Quality">Good</DA>
110                 </DA>
111                 <DA fc="SP" name="MMXU1obj" ref="SGU/BUSBAR/MMXU1/PhV/phsC"
                        type="struct">
112                     <DA fc="SP" name="mag" ref="SGU/BUSBAR/MMXU1/PhV/phsC/mag"
                            type="struct">
113                         <DA name="f" ref="SGU/BUSBAR/MMXU1/PhV/phsC/mag/f" type
                                ="FLOAT32">232.0</DA>
114                     </DA>
115                     <DA name="t" ref="SGU/BUSBAR/MMXU1/PhV/phsC/mag/t" type="
                            Timestamp">2013-07-01_13:40:57</DA>
116                     <DA name="q" ref="SGU/BUSBAR/MMXU1/PhV/phsC/mag/q" type="
                            Quality">Good</DA>
117                 </DA>
118             </DO>
119         </LN>
120     </LD>
121 </server>
```

Appendix E

REST kald 2: SGU/CH1/MMXU1/W ?GetAllDataValues

kode/XML/w.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DO name="W" ref="SGU/CH1/MMXU1/W" type="WYE">
    <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/W/phsA" type="struct">
        <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/W/phsA/mag" type="struct">
            <DA name="f" ref="SGU/CH1/MMXU1/W/phsA/mag/f" type="FLOAT32">34.4
                </DA>
        </DA>
        <DA name="t" ref="SGU/CH1/MMXU1/W/phsA/mag/t" type="Timestamp">
            2013-07-01_13:40:57</DA>
        <DA name="q" ref="SGU/CH1/MMXU1/W/phsA/mag/q" type="Quality">Good</
            DA>
    </DA>
    <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/W/phsB" type="struct">
        <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/W/phsB/mag" type="struct">
            <DA name="f" ref="SGU/CH1/MMXU1/W/phsB/mag/f" type="FLOAT32">34.4
                </DA>
        </DA>
        <DA name="t" ref="SGU/CH1/MMXU1/W/phsB/mag/t" type="Timestamp">
            2013-07-01_13:40:57</DA>
        <DA name="q" ref="SGU/CH1/MMXU1/W/phsB/mag/q" type="Quality">Good</
            DA>
    </DA>
    <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/W/phsC" type="struct">
        <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/W/phsC/mag" type="struct">
            <DA name="f" ref="SGU/CH1/MMXU1/W/phsC/mag/f" type="FLOAT32">34.4
                </DA>
```

```
20          </DA>
21          <DA name="t" ref="SGU/CH1/MMXU1/W/phsC/mag/t" type="Timestamp">
                2013-07-01_13:40:57</DA>
22          <DA name="q" ref="SGU/CH1/MMXU1/W/phsC/mag/q" type="Quality">Good</
                DA>
23      </DA>
24  </DO>
```

Appendix F

REST kald 3: GetLog

/SGU/CH1/MMXU1/VAr/phsC/mag/f?GetLog&from=2013-07-01T13:49:06,to=2013-07-01T13:49:13

kode/XML/log.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DO>
    <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsC" type="struct">
        <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsC/mag" type="struct">
            <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsC/mag/f" type="FLOAT32">
                23.1</DA>
        </DA>
        <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsC/mag/t" type="Timestamp">
            2013-07-01 13:49:06.0</DA>
        <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsC/mag/q" type="Quality">Good</
            DA>
    </DA>
    <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsC" type="struct">
        <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsC/mag" type="struct">
            <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsC/mag/f" type="FLOAT32">
                23.1</DA>
        </DA>
        <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsC/mag/t" type="Timestamp">
            2013-07-01 13:49:07.0</DA>
        <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsC/mag/q" type="Quality">Good</
            DA>
    </DA>
    <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsC" type="struct">
        <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsC/mag" type="struct">
            <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsC/mag/f" type="FLOAT32">
                23.1</DA>
```

```
20          </DA>
21          <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsC/mag/t" type="Timestamp">
                2013-07-01 13:49:08.0</DA>
22          <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsC/mag/q" type="Quality">Good</
                DA>
23      </DA>
24      <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsC" type="struct">
25          <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsC/mag" type="struct">
26              <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsC/mag/f" type="FLOAT32">
                    23.1</DA>
27          </DA>
28          <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsC/mag/t" type="Timestamp">
                2013-07-01 13:49:09.0</DA>
29          <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsC/mag/q" type="Quality">Good</
                DA>
30      </DA>
31      <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsC" type="struct">
32          <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsC/mag" type="struct">
33              <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsC/mag/f" type="FLOAT32">
                    23.1</DA>
34          </DA>
35          <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsC/mag/t" type="Timestamp">
                2013-07-01 13:49:10.0</DA>
36          <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsC/mag/q" type="Quality">Good</
                DA>
37      </DA>
38      <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsC" type="struct">
39          <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsC/mag" type="struct">
40              <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsC/mag/f" type="FLOAT32">
                    23.1</DA>
41          </DA>
42          <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsC/mag/t" type="Timestamp">
                2013-07-01 13:49:11.0</DA>
43          <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsC/mag/q" type="Quality">Good</
                DA>
44      </DA>
45      <DA fc="SP" name="MMXU1obj" ref="SGU/CH1/MMXU1/VAr/phsC" type="struct">
46          <DA fc="SP" name="mag" ref="SGU/CH1/MMXU1/VAr/phsC/mag" type="struct">
47              <DA name="f" ref="SGU/CH1/MMXU1/VAr/phsC/mag/f" type="FLOAT32">
                    23.1</DA>
48          </DA>
49          <DA name="t" ref="SGU/CH1/MMXU1/VAr/phsC/mag/t" type="Timestamp">
                2013-07-01 13:49:12.0</DA>
50          <DA name="q" ref="SGU/CH1/MMXU1/VAr/phsC/mag/q" type="Quality">Good</
                DA>
51      </DA>
52  </DO>
```

Bibliography

[ARM13] ARM. Arm11 processor family. http://www.arm.com/products/processors/classic/arm11/index.php, 2013.

[Bec13] Ansgar Becker. Heidisql is a lightweight, windows based application for managing mysql and microsoft sql databases. http://www.heidisql.com/, 2013.

[ber13] berryterminal. Berryboot v2.0 - bootloader / universal operating system installer. http://www.berryterminal.com/doku.php/berryboot, 2013.

[Chr13] Finn Kuno Christensen. How to produce graduate theses and reports-guidelines and recommended typesettings. http://www2.imm.dtu.dk/teaching/thesis/, 2013.

[Dha10] Krishnamoorthy Dharmalingam. Sample https server. http://www.javafundu.com/2010/02/sample-https-server.html, 2010.

[Fou13a] The Raspberry Pi Foundation. Raspberrypi - faq. http://www.raspberrypi.org/faqs, 2013.

[Fou13b] The Raspberry Pi Foundation. Raspberrypi - raspbian. http://www.raspberrypi.org/downloads, 2013.

[Goo13] Google. Chrome web browser. http://www.google.com/intl/da/chrome/browser/, 2013.

[IEC13] IEC. Iec 61850 - communication networks and systems in substations. http://webstore.iec.ch/webstore/webstore.nsf/Artnum_PK/33549, 2013.

[Lar01] Craig Larman. *Applying UML and Patterns: An introduction to Obect-Oriented Analysis and Design and the Unifed Process.* Prentice Hall, third edition edition, 2001.

[Log13] LogMeLn. Logmein hamachi for linux (beta) command line version. `https://secure.logmein.com/labs/`, 2013.

[mi13] No magic inc. Magicdraw design and moddelling software. `http://www.nomagic.com/products/magicdraw.html`, 2013.

[mic13] microchip. 2-wire high-accuracy temperature sensor. `http://ww1.microchip.com/downloads/en/DeviceDoc/21909d.pdf`, 2013.

[MS13] Adafruit Mikey Sklar. Raspberry pi – a tour of each plug. `http://www.adafruit.com/blog/2012/06/29/raspberry-pi-a-tour-of-each-plug-piday-raspberrypi-raspberry_pi/`, 2013.

[ORA13] ORACLE. keytool - key and certificate management tool. `http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html`, 2013.

[Ort13] Daniel Espinosa Ortiz. Scl manipulation and configuration tools. `http://sourceforge.net/projects/opensclconfig/?source=navbar`, 2013.

[Pi4] Pi4J. Provide a bridge between the native libraries and java for full access to the raspberry pi, year =.

[Sie13] Siemens. Iec 61850 validator. `http://www.energy.siemens.com/hq/en/energy-topics/standards/IEC61850.htm`, 2013.

[W308] W3. Extensible markup language (xml) 1.0 (fifth edition). `http://www.w3.org/TR/REC-xml/`, 2008.

[Wik13] Wikipedia. Document object model. `http://en.wikipedia.org/wiki/Document_Object_Model`, 2013.