CHALMERS

LoadVisualizer

Övervakning av maskinsystem i webben

Examensarbete inom Högskoleingenjörsprogrammet i datateknik

KRISTOFFER SKJUTAR
VIKTOR SJÖLIND

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2014

LoadVisualizer
Övervakning av maskinsystem i webben

Kristoffer Skjutar, Viktor Sjölind

Institutionen för data- och informationsteknik
Chalmers tekniska högskola
412 96 Göteborg
Tel: 031-772 1000
Fax: 031-772 3663

Institutionen för data- och informationsteknik
Göteborg, 2014

Sammanfattning

Maskinsystem utvecklas ständigt. Det tillkommer nya säkerhetsfunktioner och maskinsystemen får bättre prestanda. Vad gäller service av maskinerna är inte utvecklingen lika stor. Många maskinägare är otillräckligt informerade över hur mycket last olika delar av deras maskiner utsätts för, antingen för att de inte har tillgång till den informationen eller för att informationen är svåråtkomlig, vilket leder till att maskindelar byts ut tidigare än vad de har behov av. Eftersom extra service är dyrt är det ett problem som maskinägare är villiga att investera i lösningar för att lösa. LoadVisualizer är ett projekt som försöker lösa detta problem genom att göra den data som behöver läsas av från maskinsystemen mer tillgänglig. Lösningen sker i tre steg: Extrahera data från maskinsystemen, processa den extraherade datan och presentera den processade datan genom ett användarvänligt gränssnitt. Genom användargränssnittet kan maskinägaren till exempel få information om hur mycket lasttimmar maskinen utfört och tid och datum då nästa service för en specifik maskindel bör ske beräknat utifrån snitt belastningen sedan tidigare service.

Abstract

Machine systems are constantly evolving. Performance is increased and new security features are implemented. However, improvements regarding maintenance is not evolving at the same pace. Machine owners often lack about the load information of the different parts in their machines. Since maintenance is expensive, this is starting to become a problem for machine owners. LoadVisualizer is a project that aims to solve this problem in three steps. The first step is to create a device that collects data from the machines and then transmits the collected data to a server. Secondly the server saves the received data in a database. Finally the information is processed and presented to the user via a web interface.

Förord

Vi skulle vilja tacka KXE Styrteknik för ett bra samarbete under detta projekt, vår handledare Peter Lundin som gett oss vägledning i vårat rapportskrivande och Sakib Sistek som hjälpt oss med allt från teknisk utrustning till att höja vår moral.

Begrepp

JSON (JavaScript Object Notation) är ett textbaserat format som används för att representera data i objektform [13].

REST(Representational State Transfer) är ett designmönster för delning av data mellan datorer som bygger på standardmetoderna i HTTP-protokollet (POST, PUT, GET och DELETE) [AA10].

Black-Box Testing är ett scenario inom testning av mjukvara där man testar en applikations funktionalitet utan att testa dess inre delfunktioner. Det vill säga givet X skall det testade systemet returnera Y [01].

Bootstrap är ett JavaScript och CSS bibliotek för design av webgränsnitt [14c].

Git är ett opensource versionshanteringssystem ursprungligen utvecklat av Linus Torvalds för att hantera källkoden till linuxkärnan [14d].

MVC är ett programmeringsmönster som står för Model View Controller. Programmeringsmönstret går ut på att man delar upp den beräknande delen från gränssnittet och en tredje del som hanterar input från användaren.

Jetty är en webserver som är helt implementerad i Java. Jetty används ofta för testning och maskin till maskin kommunikation via HTTP-protokollet [14e].

Innehåll

Kapitel 1

Inledning

1.1 Bakgrund

Inom industri är det extremt viktigt att dess maskiner är ständigt användningsbara och inte stillastående på grund av oväntade reparationer eller service. Ett sådant scenario kan både försena arbetet samt vara kostsamt för företaget. Samtidigt är det viktigt för företaget att de lätt kan få en översikt över alla deras maskiner och få information om användning och tid till service.

KXE är ett företag som arbetar med industriautomation och vill ta fram en lösning som enkelt kan presentera ovanstående information för lyftdon. Den avsedda information samlas för närvarande in genom att hämta ut logg-filer från PLCer på lyftdon. Logg-filerna behandlas sedan manuellt för att få fram den information man vill ha. KXE vill därför utveckla en lösning där informationen från PLCerna hämtas och behandlas automatiskt. Informationen ska sedan kunna presenteras på ett tillfredsställande sätt tillsammans med information från andra lyftdon.

1.2 Syfte

Projektet skall svara på om det är möjligt att med hjälp av ett nytt informationssystem förenkla insamling och presentation av information om maskinsystem. Informationen som hämtas från informationssystemet skall sedan kunna användas för att planera driftstopp och service av maskinsystemet.

1.3 Mål

Målet är att ta fram ett system som genom direkt anslutning till en maskinutrustning (PLC system) via ett för utrustningen anpassat gränssnitt kan läsa av information i utrustningen. Informationen skall överföras via en uppkoppling mot en tjänst i en webserver med tillhörande databas. Efter lagring i databasen kan informationen presenteras för en användare som är inloggad mot webservern via ett användargränssnitt. För att underlätta underhåll av mjukvaran skall systemet implementeras på ett sådant sätt att endast de direkt berörda delarna i systemet behöver refaktoreras då mjukvaran behöver uppdateras eller ny funktionalitet skall implementeras.

1.4 Avgränsningar

I detta projekt kommer endast kommunikation med en specifik modell av PLC att undersökas. Programmering av PLC kommer att utföras av uppdragsgivaren. För MODBUS / TCP kommunikation samt representation av data i form av grafer i användargränssnittet kommer befintliga kodbibliotek att användas.

Kapitel 2

Metod

Inledningsvis skall det undersökas hur en PLC fungerar och vilka metoder och kommunikationsprotokoll det finns för att kommunicera med PLC system. Utifrån dessa metoder och kommunikationsprotokoll skall de mest lämpliga för detta system väljas. Under utvärderingen av protokollen skall det undersökas vilka möjligheter det finns för att implementera dessa kommunikationsprotokoll med hjälp av programmeringsspråket Java som skall användas för utvecklingen av mjukvaran i den modul som hanterar kommunikationen med PLC, vidare kallad sensor.

Även för användargränssnittet behöver beslut tas angående vilka ramverk som skall användas. Eftersom att användargränssnittet kommer implementeras i form av ett webgränssnitt planeras att använda programmeringsspråket JavaScript.

2.1 Struktur och planering

För att strukturera utvecklingen kommer systemets funktioner delas upp i deluppgifter och bestämmas i vilken ordning de ska genomföras. Varje moment och uppgift ska sedan tidsmässigt uppskattas och ett Gantt-diagram för lättare översikt ska tas fram.

För att hålla arbetet agilt kommer möten hållas i början av varje vecka. På mötena diskuteras vad som behöver göras i kommande veckan. Därefter görs en veckoplanering med tanken att om något viktigt skulle komma upp kan det få prioritet över det andra som är planerat.

2.2 Utveckling av mjukvara

Utvecklingen av mjukvaran kommer att ske iterativt och versionshanteringen kommer stödjas med Git-system för att underlätta det iterativa arbetssättet.

Programutvecklingen kommer först att fokusera på att spara undan data från PLC i databasen hos servern. Detta inkluderar även felhantering hos både sensor och server. Vidare ska funktionen att under körning kunna byta konfigurationer hos sensor implementeras. Under utvecklingen av ovan nämnda funktioner ska även KXE ta fram en kravspecifikation för systemet. Kravspecifikationen ska skriftligt innehålla funktioner, begränsningar och situationer som systemet behöver kunna hantera. Kravspecifikationen ska också innehålla de matematiska funktionerna som skall användas

för beräkna de värden, som skall presenteras för användaren via användargränssnittet, av datan som samlats av de olika systemens sensorer.

2.3 Testning och verifiering

Då systemet har flera enskilda komponenter (sensor, server, användargränssnitt) som kommunicerar med varandra är det svårt att verifiera funktioner som beror på andra komponenter under utvecklingen. Det skulle krävas att motsvarande funktion hos den andra komponenten är färdigställd vilket är svårt att tidsmässigt synkronisera. Därför ligger stort fokus i systemutvecklingen på att enskilt kunna testa varje komponents funktion. Detta ska genomföras genom att andra komponenter simuleras med hjälp av testramverk som sedan genomför test av funktioner, även kallat Black-Box testning [01]. Exempelvis ska sensorer simuleras för att testa serverns förmåga att spara data och hantera uteblivna sändningar.

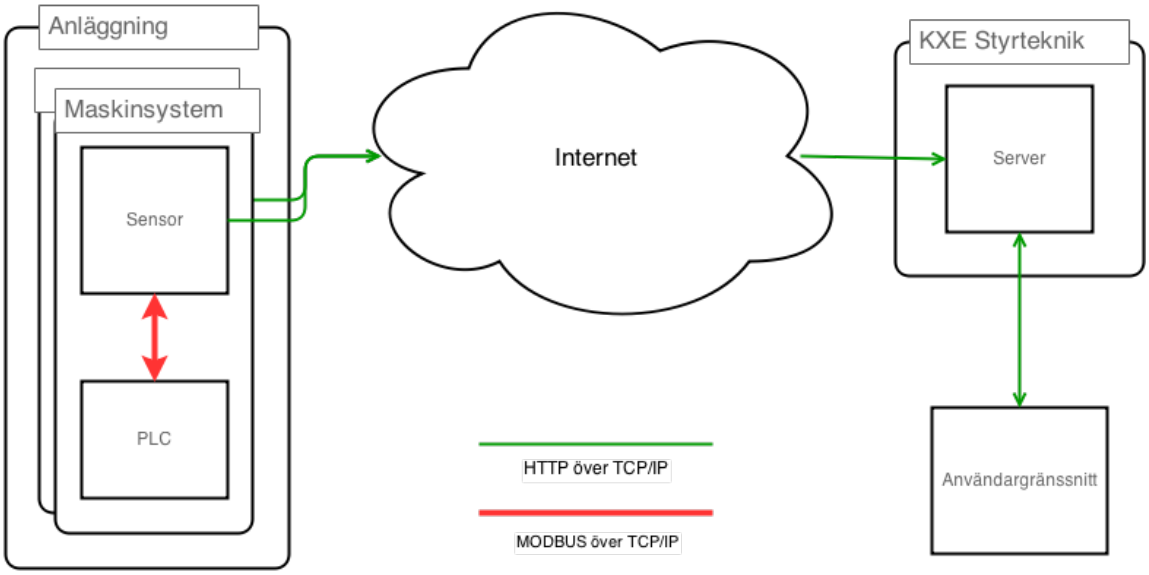
Kapitel 3

Teknisk bakgrund

I detta kapitel presenteras de berörda teknikerna i projektet.

3.1 Vision

En anläggning innehåller ett eller flera maskinsystem. Varje maskinsystem har en tillhörande sensor som läser av maskinsystemet och skickar den samlade datan till en central server med jämna tidsintervall. Servern tar emot datan från sensorerna och sparar den i en databas. Via ett användargränssnitt kan sedan användare logga in och se datan från de maskinsystem de är ansvariga för presenterad i form av grafer och tabeller. Se figur 3.1.



Figur 3.1: Översikt över hela systemet.

3.2 Sensor

Sensorn är den enhet som skall ansvara för att läsa av från PLC system och sända data vidare till server. En sensor kan hantera läsning från flera PLC system.

3.3 Server

Servern refererar till den enhet som tar emot data från sensorer. Servern hanterar databasen i vilken data från sensorer lagrats i. Användargränssnittet hanterar data i databasen via REST-anrop.

3.4 Användargränssnitt

Användargränssnittet är det gränssnitt användare använder för att få data presenterad från servern på ett begripligt sätt.

3.5 Programmable logic controller (PLC)

Programmable logic controller, eller PLC, är en enhet som styr maskinsystem. För att läsa av maskinsystemet tar PLC in information på digitala och / eller analoga portar. Med hjälp av informationen från maskinsystemet beräknar PLC vilka styrsignaler som skall skickas ut på dess digitala och / eller analoga utgående portar. Vilken typ av portar och protokoll en PLC kan hantera varierar mellan olika fabrikat och kan ofta anpassas via extra moduler. I detta projektet används en MODBUS / TCP modul för att få den aktuella PLC:n att kunna kommunicera med sensorn på önskat sätt.

3.6 MODBUS

MODBUS är ett protokoll framtaget för kommunikation med PLC. Protokollet är mycket populärt och det anses oftast vara industristandard för kommunikation med PLC [Buc00]. MODBUS är ursprungligen ett varumärke från Schneider Electronic Group och är implementerat som en simpel version av 'förfrågan och svar' mönstret. Den frågande enheten agerar master i kommunikationen och är den enda av de två kommunicerande som kan skicka förfrågningar. Slaven är den enhet som tar emot förfrågningar och sänder tillbaka svar. En master kan antingen välja att specifikt sända förfrågningar till en slav eller som ett broadcast. Förfrågningar består av adress, funktion, query data och felkontroll där respektive är två bytes. Fältet adress specificerar vilken enhet som skall behandla förfrågan. Fälten funktion och query data representerar vilken slags förfrågan som ska göras och eventuell data som behövs för förfrågan.

3.6.1 Representation av data

MODBUS representerar data genom fyra kategorier; Digital Inputs, Digital Outputs, Input Registers och Holding Registers. De två förstnämnda är representerade digitalt som en bit och skiljer sig åt genom att Digital Input endast kan läsas till skillnad från Digital Output som både kan läsas och skrivas via MODBUS. Samma förhållande gäller mellan Input Registers och Holding Registers som

både är representerade digitalt som 16 bitars och där endast Holding Registers både kan läsas och skrivas. Adresserna för data i MODBUS kan vara strukturerade i det fysiska minnet på olika sätt beroende på tillverkare. För att modulera adresseringen av data har varje kategori av data (Digital Input, Digital Output, Input Registers och Holding Registers) ett visst antal adresser numrerade mellan 1 och ett fixt antal n att använda. I det fysiska minnet kan dessa antingen vara separerade eller överlappande med samma fysiska adresser och då dela data [12].

3.6.2 Kommunikationsprotokoll

Kommunikationen kan ske både via seriell port med protokollet RS-485 eller via Ethernet över TCP eller UDP [Oli11]. Kommunikationen med MODBUS via TCP / IP sker på applikationslagret och kan beskrivas som ett förhållande mellan klient och server där slaven och master agerar klient respektive server i en traditionell kommunikation över HTTP [Buc00].

3.6.3 Jamod

Jamod [Wim10] är ett open-source ramverk till Java för kommunikation över MODBUS protokollet. Jamod är väldokumenterat och har stöd för simulering av PLC system. Ramverket är implementerat i Java 5. Modbus4j [Loh10] är ett alternativt ramverk för kommunikation över MODBUS protokollet som också är open-source men som valts bort, trots att implementationen är gjord i en senare version av Java, på grund av bristfällig dokumentation.

3.7 Spring

Spring är ett utbrett ramverk för utveckling av webapplikationer i Java. Ramverket byggs runt dess lättviktiga kärna och kan lätt utvecklas med flera tillgängliga moduler för Spring. Modulerna kan ge stöd för speciella ramverk, databaser eller strukturer i applikationen. För att importera objekt och moduler använder Spring Dependency Injection som inkluderar dessa medan applikationen kör. Det här underlättar utvecklingen av webapplikationen då mindre tid behövs läggas på att föra samman moduler med varandra [HH12].

3.8 Hibernate

Hibernate är ett ramverk skrivet i Java för att hantera relationsdatabaser. Ramverkets primära uppgift är att hantera mappningen mellan Java-objekt och databasen. Det här gör det möjligt att representera databasen och dess entiteter i form av Java-objekt. Hibernate åstadkommer detta genom att använda annoteringar i Java för att länka referenser mellan objekt som relationer i databasen. Hibernate ger också möjlighet till att byta typ av databas genom att endast byta en modul i webapplikationens konfiguration [Ell05].

3.9 Maven

För att hantera Dependency Injection i Spring används Maven som automatiskt bygger programfilerna med valda moduler [14b]. Maven är även det program som bygger koden och är inställt att alltid köra de automatiserade testerna innan koden byggs. Om något av testerna inte går igenom

kommer inte koden att byggas. På så sätt garanteras att inte otestad kod hamnar i produktionsmiljön.

3.10 AngularJS

AngularJS är ett ramverk skrivet i javascript, framtaget för att ge MVC struktur till webapplikationer. AngularJS ger ett dynamisk användargränssnitt där konceptet 2-way-databinding är applicerat. Detta betyder att modellen och vyn i MVC- strukturen är konstant synkroniserade med varandra. Förändringar i modellen syns direkt i vyn och vice versa. Detta är mycket användbart i applikationer där användare lätt ska kunna manipulera data och se förändringar i vyn. AngularJS har också ett väl utvecklat testramverk. Testramverket gör det möjligt att testa webapplikationen helt avskilt från bakomliggande webserver då anrop för hämtning av t.ex. data kan simuleras i tester.

Kapitel 4

Genomförande

Tre skilda moduler, Sensor, Server och Användargränssnitt är implementerade där vardera har specifika uppgifter i systemet. Nedan följer en mer ingående beskrivning av genomförandet av dessa.

4.1 Sensor

För att läsa av data från PLC implementeras en sensor. Sensorn består av en enkortsdator och mjukvaran är skriven i Java. Anledningen till valet av Java som programmeringsspråk är på grund av goda erfarenheter från tidigare projekt. Internt är mjukvaran uppdelad i tre delar: mottagare, paketerare och sändare, där varje del exekveras som en egen tråd. Då denna enhet kommer sitta i nära kontakt med PLC på lyftdonet kommer miljön för hämtning och sändning av data inte vara optimal. Därför är felhantering vid läsning och vidaresändning av data implementerat. Data hämtas från PLC och skickas till server med ett bestämt tidsintervall. En konfigurationsfil finns lagrad på den aktuella hårdvaran. Denna konfigurationsfil innehåller information om vilka värden som skall hämtas samt vilken PLC och vilken adress på PLCn värden finns att hämtas på. Konfigurationsfilen innehåller även id det identifikationsnummer och lösenord sensorn använder för att sända data till servern.

4.1.1 Val av hårdvara

Mjukvaran som exekveras på sensorn är utvecklad för att vara plattformsoberoende och kräver endast anslutning till PLC och Server via Ethernet samt att hårdvaran kan köra Java 7. Detta ger att det finns många alternativ till val av hårdvara. Då utrymme i produktions miljön kommer vara en bristvara måste en liten dator användas. Den hårdvara som valts för projektet är enkortsdatorn Raspberry Pi modell B. Anledningen till detta val är att Raspberry Pi är en enkel och billigast i sin prestandaklass. Dess lilla formfaktor gör den väl lämpad för labmiljö, dock kan den inte användas i en produktions miljö för att den inte är industriklassad men med tanke på Raspberry Pis låga prestanda kan vi säkerställa att de flesta andra moderna datorer kommer att kunna fungera som ersättare.

4.1.2 Val av kommunikationsprotokoll

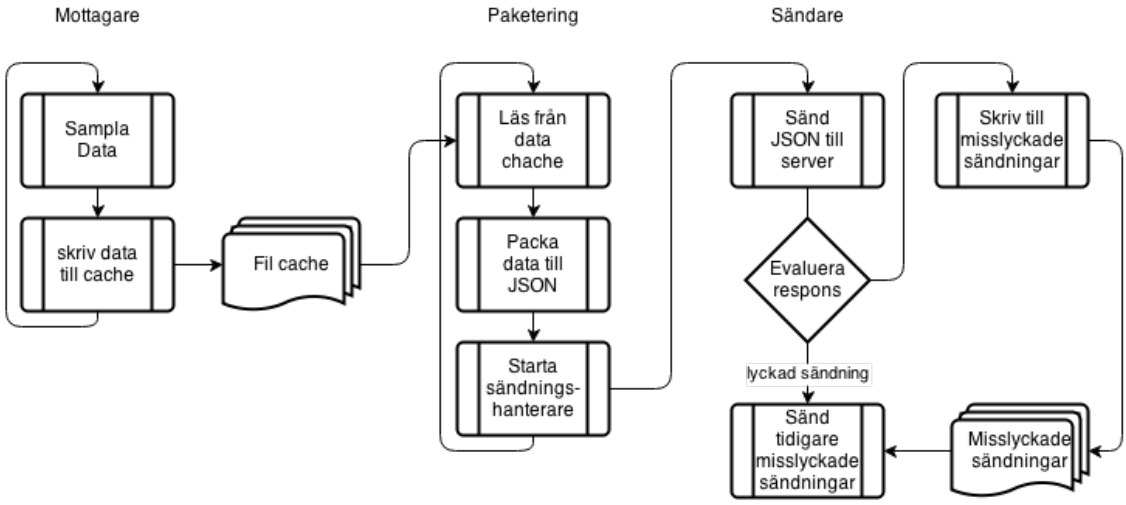
De val som fanns för kommunikation med PLC var protokollet RS-485 via seriell port och TCP / UDP via Ethernet Först och främst valdes seriell kommunikation med RS-485 bort då det är en gammal teknik med begränsad överföringshastighet. En viktig egenskap för sensorn är förmågan att kunna skicka vidare den insamlande datan från PLC över MODBUS till en server över HTTP och Ethernet. Med protokollet RS-485 för PLC skulle sensorn både ha som tekniskt krav att kunna kommunicera över RS-485 via seriell port med PLC och över Ethernet med servern. För att minska de tekniska kraven på kommunikationsmöjligheter för sensorn valdes därför Ethernet för både kommunikation med PLC och server. Eftersom en viktig egenskap för sensorn är att mottagandet av data verifieras så att ingen data går förlorad används TCP som protokoll för både kommunikation med PLC och server. Alternativet till TCP är UDP vilket inte ger verifiering av sändningar, dock är det mycket snabbare men eftersom att systemet inte är tidskritiskt har UDP inga fördelar över TCP.

4.1.3 Cache

Ett gränssnitt kallat Cache är implementerat för att hantera läsning och skrivning till olika cacher. En implementation av gränssnittet är FileCache vilket är en representation av en cache som lagras på disk.

4.1.4 Parallella trådar

Varje del i sensorn exekveras som en parallell tråd, se figur 4.1. Det betyder att den kör helt oberoende av alla andra trådar bortsett från de delar som hanterar överlämning av resurser till andra trådar där trådar kan behöver vänta. Denna fördel som parallella trådar ger är viktigt för att till exempel inte mottagaren skall blockeras för att servern inte svarar då sändaren skickar tidigare samlade sampel samtidigt som mottagaren vill hämta ny data. På samma sätt får inte sändaren bli blockerad av att mottagaren hämtar sampel så att ett fel löses ut på serversidan på grund av att inte sändaren hör av sig tillräckligt ofta. Parallella trådar hjälper även att göra mjukvaran modulär vilket är satt som ett av målen i projektet. För att inte resurserna trådarna använder skall förstöras, genom att flera trådar skriver och läser samtidigt från samma resurs, är ett gränssnitt som använder semaforer implementerat. Dessa typer av gränssnitt kräver mer tid och testning för att implementeras och förlänger på så sätt den totala utvecklingstiden.



Figur 4.1: Schema över processflödet i sensorn. Pilar som kommer in uppifrån markerar start av ny funktion och pilar som kommer in eller går ut från sidan markerar input respektive output från funktionen.

4.1.5 Mottagare

Mottagaren läser sensorns konfigurationsfil och sätter upp kopplingar till alla PLC system som är angivna. Efter att kommunikationerna är upprättade hämtas de specificerade värdena från deras respektive adress och PLC system. För varje värde skapas en plats för lagring av sampel som initialt sätts till null. Null används för att signalera att inget värde har lästs in då 0 är ett godkänt värde. Värdena samplas ett i taget. Varje av ett värde sampling adderas till föregående sampling av samma värde. För att säkra att inte redan insamlade sampel skall gå förlorade, om sändaren skulle bli strömlös eller om mjukvaran skulle krascha, lagras samplingarna i en cache på disk. När X sampel, där x ges av:

$$X = \frac{tidsintervall}{samplingstid}$$

har hämtats divideras varje sampelsamling med X och skrivs sedan till den cache som sändaren läser från. Efter detta nollställs sampelsamlingarna till null och sedan börjar processen om med att samla nya sampel. Ifall att ett PLC system inte skulle svara, till exempel på grund av kabelbrott, skrivs en felkod istället för sampeldata. Skulle endast en specifik adress inte svara från PLC system ersätts endast det värdet av en felkod.

4.1.6 Paketerare

Paketeraren läser sampel och felkoder från en cache som den blivit tilldelad. Den samlade datan packas in i ett gemensamt JSON objekt tillsammans med en tidstämpel och sedan skickas det paketerade JSON objektet vidare till sändaren.

4.1.7 Sändare

Varje gång sändaren skall sända ett JSON objekt läggs även id och lösenord till. Dessa uppgifter använder sändaren för att få access till servern. Anledningen till att inte det läggs till av paketeraren är för att inte gamla uppgifter skall lagras ifall sändningen hamnar i en cache. När en sändningen misslyckas, det vill säga att responskoden från servern inte är 200 eller 201, avlägsnas id och lösenord från JSON objektet som sedan skrivs till en cache som innehåller alla misslyckade sändningar. När en sändning lyckas, det vill säga då responskoden från servern är 200 eller 201, försöker sändaren sända om de misslyckade sändningarna. Skulle de fortfarande inte lyckas skrivs de tillbaka till cachen de lästes från. När responskoden från servern är 201 betyder det att förutom att sändningen lyckats även att det finns en uppdaterad konfigurationsfil för sensorn i responsen från servern.

4.2 Server

För att hantera lagring och mottagning av data från sändaren används en server i form av en webapplikation skriven i Java. Den använder Wildfly, tidigare JBoss som applikationsserver. Servern hanterar utomstående förfrågningar från sensorer och användargränssnittet. Den hanterar och lagrar data mot en databas som kör PostgreSQL.

4.2.1 Val av bibliotek och ramverk

Spring

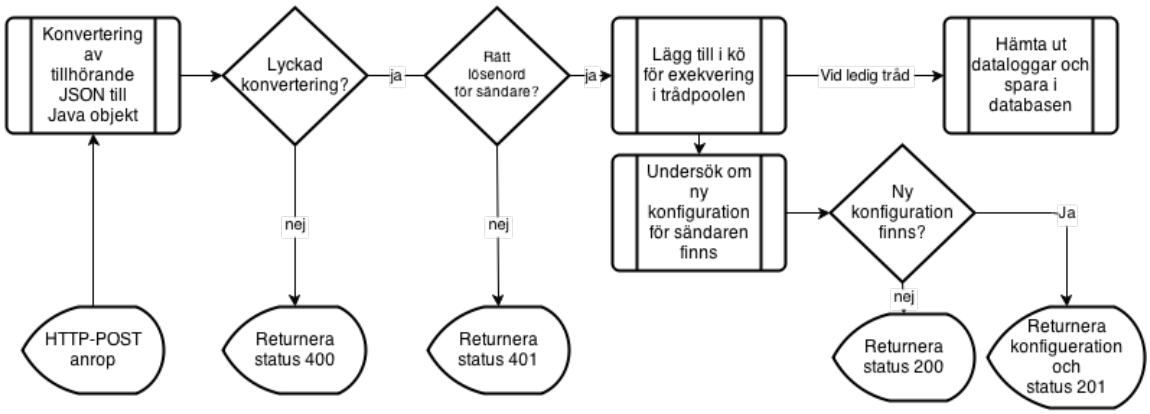
Servern använder ramverket Spring tillsammans med modulen Spring MVC för struktur enligt designmönstret Model, View och Control i webapplikationen. Spring valdes för dels för dess mycket utförliga dokumentation men också på grund av erfarenheter från tidigare projekt.

4.2.2 Hantering av data från sensor

En viktig detalj i kravspecifikationen är att det skall råda strikt envägskommunikation från sensor till server. Därför måste all interaktion från server till sensor ske i samband med att ett anrop från sensor kommer till server. I figur 4.2 kan ett flödesschema för hur hanteringen av data från sensor är implementerad. När ett inkommande anrop anländer till den förutbestämda adressen på servern konverterar det tillhörande JSON-objektet enligt en förutbestämd formatering till Java-objekt. Formateringen är specificerad hos både sensor och server för att kunna sända och läsa av data hos båda parterna. Efter konverteringen till Java-objekt görs en kontroll av det medskickade lösenordet. Om antingen konvertering eller kontrollen av lösenordet misslyckas returneras status 400.

För att minska exekveringstiden av ett anrop från sensor överlåts exekvering av registrering i databasen till parallella trådar från en trådpool. Detta är viktigt för sensorer skall kunna sända över stora mängder data utan att behöva vänta att servern processerar den mottagna datan.

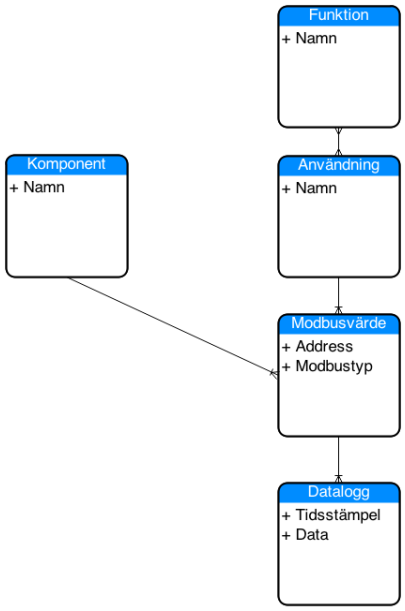
När ett anrop har överlåtit exekveringen görs en kontroll om ny konfigurationsfil finns att hämta (se 4.3.5). Om så är fallet genereras en ny konfigurationsfil och returneras tillsammans med status 201, annars returneras status 200. För att öka säkerheten för sändningar från sensor till server genereras ett nytt lösenord för sensor vid varje ny generering av konfigurationsfil.



Figur 4.2: Flödesschema för hantering av anrop från sensorer.

4.2.3 Databas

Databasen är uppbyggd av relationer liknande ett träd av anläggning, system, PLC, komponenter och värden, se figur 4.3. Varje system tillhör en anläggning och har en sensor som skickar data tillhörande det systemet. För varje system finns ett antal PLC som sensorn kommunicerar med för att hämta värden. Värdena tillhör komponenter där komponent betyder den enhet som flera värden tillsammans beskriver, till exempel en motor. För att sensorn inte skall behöva ha vetskap om strukturen på komponenter har värden även en relation direkt till PLC. Komponenter används för att sortera data när den presenteras i användargränssnittet. Varje PLC, sensor och värde har också möjligheten att kunna användas för att spara felanmälan som uppstår under körtid. Varje värde för en PLC innehar en typ och en enhet för att i användargränssnittet kunna avgöra hur datan ska presenteras.



Figur 4.3: Förenklat ER-diagram av databasen som visar hur funktioner (se tabell *Funktion* i diagrammet) är representerade.

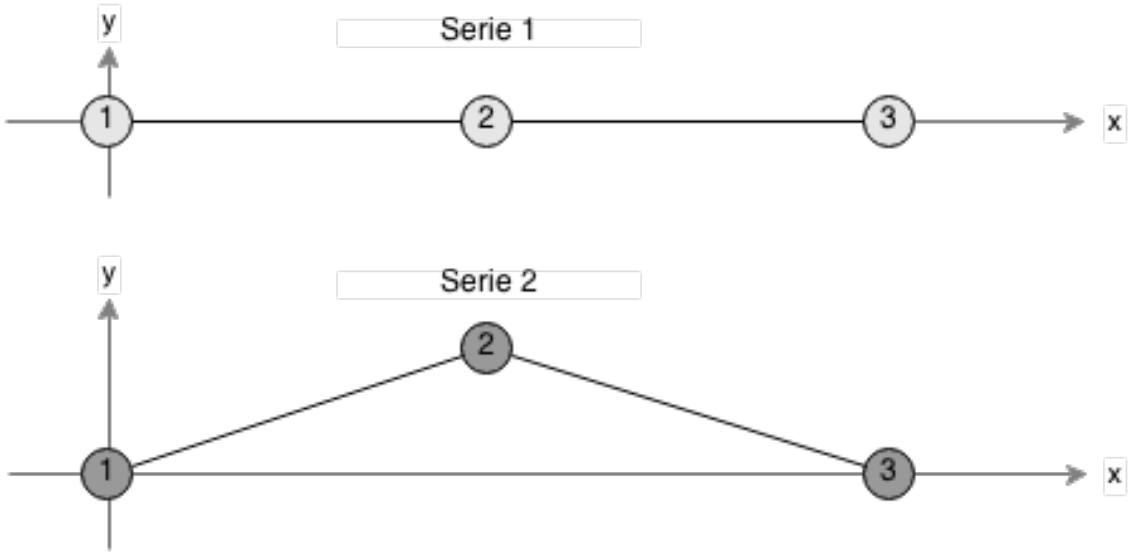
För att underlätta belastningen av databasen togs beslutet att låta klienten (den enhet som exekverar användargränssnittet) beräkna den funktion som ska presenteras i graf eller tabell från specificerade dataserier för den aktuella funktionen. Det här betyder att servern och databasen kan hållas generell då servern och databasen endast ansvarar för att gruppera ihop vilka dataserier som behövs för vilka funktioner. Ett förenklat ER-diagram för representationen av funktioner kan ses i figur 4.3. *Modbusvärde* i en *komponent* har först och främst ett specifikt användningsområde kallad *användning*, till exempel belastning eller aktivitet. *Funktion* är den gruppering av *användning* som utgör en funktion. Användargränssnittet kan därför utifrån en *komponent* lista vilka tillgängliga funktioner det finns och ge användaren möjlighet till att välja vilken *funktion* som ska visualiseras i en graf eller tabell. När användaren valt en *funktion* hämtar servern de berörda dataserierna av dataloggar för funktionen i komponenten som sedan användargränssnittet utför beräkningar på.

4.2.4 Komprimering av mottagen data

För att inte onödig data skall sparas i databasen komprimeras den data som servern tar emot från sensorer. Med tanke på att flera sensorer kan skicka data till en server med korta tidsintervall så blir databasens storlek över tid mycket stor. Därför har följande metod för att spara utrymme i databasen används.

Varje gång en datapunkt skall läggas till undersöks de två närmsta datapunkterna i samma dataserie som den nya datapunkten skall läggas till i. De tre valda datapunkterna (den nya datapunkten och två andra) jämförs, har de alla samma värde tas den datapunkt som kronologiskt sett är i mitten bort. Denna algoritm körs tre gånger. Först körs algoritmen med den nya datapunkten tillsammans med de två kronologiskt nästkommande datapunkterna (om de existerar), sedan används den nya

datapunkten tillsammans med datapunkten kronologiskt föregående och nästkommande och till sist används den nya datapunkten tillsammans med de två kronologiskt föregående datapunkterna. På detta sätt kommer endast de datapunkter som utgör nollpunkter för derivatan av dataserien att sparas i databasen. Se figur 4.4. Denna metod för komprimering implementerades då den är mycket effektiv vid statisk data, exempelvis i en produktionsmiljö där lyftdon inte används på kvällar och nätter. I sådana scenarion sparas för körtid endast två värden för att representera att kranen inte har använts.



Figur 4.4: I dataserie 1 krävs endast *punkt 1* och *punkt 3* för att utgöra grafen, *punkt 2* tillför alltså ingen data och behöver inte sparas i databasen. I dataserie 2 krävs alla tre punkterna för att utgöra grafen.

4.2.5 Autentisering

För att endast autentiserade användare skall komma åt olika system och anläggningar upprättats ett autentiseringssystem där användare via användargränssnittet kan logga in. För autentiseringen används Springs Security [14g] modul som appliceras som ett filter innan anrop når den avsedda controllern. Innan varje anrop gör filtret en kontroll om det aktuella anropet har en Cookie som är giltig och tidigare utfärdad av servern. Securitymodulen gör det också möjligt att genom hela servern hämta information om vilken användare det aktuella anropet kommer ifrån och därför förse den med anpassad information. Informationen anpassas efter vilket system och anläggning användaren tillhör. Det finns även olika nivåer av rättighet för varje tillhörighet: läsa, skriva, ta bort och superadministratör. De tre förstnämnda bestämmer om användaren får läsa av information, lägga till och / eller ta bort servicebestämmelser eller konfigurationer. Den sistnämnda ger fullständiga rättigheter och kan ta bort och lägga till hela anläggningar.

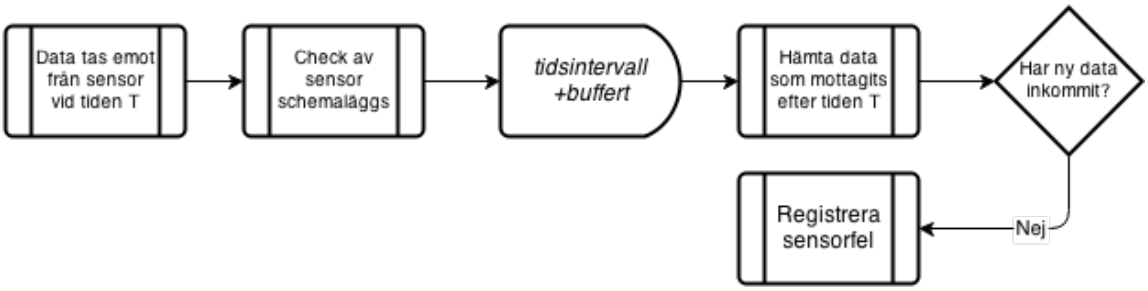
4.2.6 Felhantering

Server hanterar följande fel:

- Sändning av data från sensorer upphör.
- Specifika värden från PLC kunde inte hämtas.
- Kommunikation mellan sensorer och PLC tappas.

När servern tar emot data från sensorer undersöks varje enskilt värde om den innehåller någon felnotering. Ifall ett värde har en felanmälan betyder det att sensorn inte får något svar när den frågar adressen där värdet skall hämtas. När en sådan felanmälan upptäcks lokaliseras det aktuella registret i databasen och en felanmälan registreras med tidpunkten för felet. Ett specialfall av felanmälan för register uppstår när alla register i en PLC är drabbade. Vid ett sådant fall registreras istället en felanmälan på den aktuella PLC i databasen.

En annan typ av felhantering är implementerad för att upptäcka om sensorer inte sänder med det förinställda intervallet, något som kan uppstå vid nätverksfel mellan sensor och server eller att sensorn har kraschat. Då kommunikationen mellan dessa två är en strikt envägskommunikation finns inga möjligheter att aktivt undersöka från servern om det finns ett sensorer svarar. För att lösa detta utnyttjas att sensorer skall sända data med jämna intervall. När en sensor hör av sig schemaläggs en kontroll i servern vid ett halvt tidsintervall efter att sensorn borde sända data nästa gång. När kontrollen exekveras undersöks ifall det inkommit ny data från sensorn. Om inte ny data har mottagits av servern registreras det i databasen som sensorfel.



Figur 4.5: Flödesschema för upptäckande av till exempel nätverksfel mellan sensorer och server.

När servern tar emot ny data enligt 4.2.2 från en sensor schemaläggs den felupptäckande uppgiften enligt följande formel:

$$t_s = t_a + t_{int} + t_{buffert}$$

där t_a är aktuell tid, t_{int} är sensorns intervall för sändning av data till servern, $t_{buffert}$ är en angiven buffert för hur känslig servern ska vara på avvikande sändningsintervaller och t_s är den tid då den schemalagda uppgiften ska exekveras. Tiden då schemaläggningen lades (och när senaste datan togs emot) t_a används i den exekverande processen för att undersöka från och med vilket tillfälle det ska ha inkommit ny data. Det här betyder att en felanmälan registreras för sensorn om den inte hunnit sända ny data inom dess tidsintervall och angiven buffert. Ett flödesschema för denna process kan ses i figur 4.5.

Felanmälan kan sedan ses i användargränssnittet och kan användas för att lokalisera eventuella fel i sensorns konfiguration och andra inställningar. Tack vare att felanmälan registreras på den berörda komponenten i systemet kan man som användare lätt avgöra vart felet i systemet finns och därför lättare kunna åtgärda felet.

4.3 Användargränssnitt

Användargränssnittet avser den del av webapplikationen som visas på användarens skärm. Det vill säga den delen som laddar data från servern och presenterar det för användaren.

4.3.1 Val av bibliotek och ramverk

Javascript

AngularJS är det ramverk som valts för implementation av användargränssnitt till webapplikationen. Anledningen är främst att det följer designmöstret MVC som ger en strikt struktur på källkoden. Databindning i AngularJS gör det möjligt att låta användare få en vy där innehåll laddas dynamisk och reagerar på användares val direkt och därför ge en användarupplevlse som påminner om applikationer för desktop. För detta projekt är en sådan funktion önskvärd för att kunna representera data från PLC på olika sätt, till exempel i form av grafer och tabeller. Ett alternativ som övervägdes är ramverket jQuery [14f] som används av majoriteten av websidor med javascript. Då detta ramverk är mer spritt än AngularJS finns fler exempel på implementationer att ta inspiration ifrån. Däremot använder inte jQuery sig av en MVC modell vilket är huvudanledningen att jQuery inte används.

CSS

För formatering och visuell design används bootstrap, vilket är ett CSS bibliotek för webapplikationer. Detta valdes för att det fungerar bra tillsammans med AngularJS och har ett enkelt och stilrent utseende vilket passar vår produkt.

Visualisering av data

För valet av kodbibliotek för visualisering valdes inledningsvis Angular-charts [Kul13], en avskalad version av D3.js [Bos13] speciellt framtagen för att följa kodstandarder i AngularJS. Angular-charts gör det möjligt att visualisera data i form av exempelvis linje- och stapeldiagram. Dock uppstod det problem när den tidsbaserade datan i databasen försökte visualiseras med hjälp av Angular-charts då stödet för att ha en tidsbaserad x-axel inte fanns. Vid större mängder data som visualiserades var x-axeln oläsbar samt icke skalenlig om datan ej var regelbunden tidsmässigt. Då utveckling av visualiseringsbibliotek inte tillhörde inriktningen på detta projekt gjordes försök att hitta en alternativ implementering av D3.js för AngularJS. Valet föll då på n3-charts [Fra14], ett bibliotek som är mycket likt Angular-charts men som har stöd för tidsbaserad x-axel.

Andra alternativ till d3.js är bland annat Google Charts [14a] som är ett mer utbyggd och omfattande kodbibliotek för visualisering av data. Google Chart är mer fokuserat på diagram än D3.js och i den synpunkten mer lämpat för detta system. Dock kräver Google Charts ständig uppkoppling mot Google via Internet. För att undvika ett beroende av en webtjänst från tredjepart valdes Google Charts därför bort.

4.3.2 Beräkning av funktionsvärden för visualisering

För att funktioner, till exempel total antal fullasttimmar, som bygger på data från flera värden skall kunna presenteras måste funktionsvärdet beräknas. Denna beräkning sker på klientsidan i webapplikationen (användargränssnittet) vid det tillfälle användaren begär att få funktionen visualiserad. Anledningen till att beräkningen sker på klientsidan är att inte servern skall ansträngas så att andra användare får vänta längre när de i sin tur använder applikationen. Ett alternativ är att spara de beräknade värdena i databasen så att de inte behöver beräknas igen. Det betyder dock att vid varje tillfälle en ny datapunkt inte placeras först kronologiskt i en dataserie eller funktioner uppdateras måste alla sparade beräknade värden räknas om. En sådan omberäkningsprocess skulle antingen exekveras omedelbart och göra servern näst intill obrukbar under belastningen, eller så skulle beräkningarna spridas ut över tid vilket skulle resultera i att de funktioner som väntar på att omberäknas visar fel data eller inte kan visas för än de omberäknats. Eftersom ingen av dessa två alternativ är acceptabla är inte alternativet att spara funktionsvärdena i databasen acceptabelt.

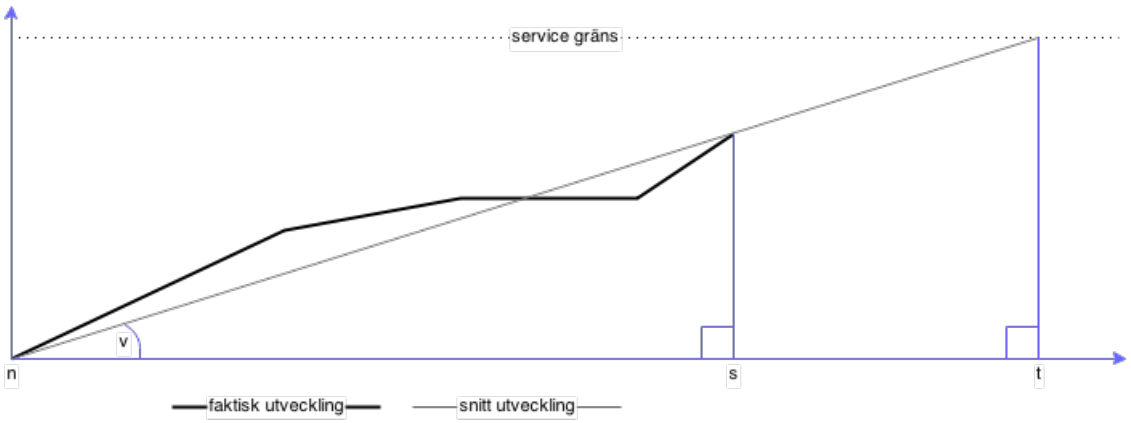
För att visualiseringen i grafen ska vara tydlig kan användaren välja vilken tidsupplösning grafen ska använda. Om användaren exempelvis väljer upplösningen ”per minut” grupperas datan från servern i minuter enligt dess tidsstämpel. Efter grupperingen beräknas det totala värdet för varje gruppering som sedan visualiseras i grafen. Genom att användaren kan välja hur datan ska grupperas kan man på ett enkelt sätt se hur mycket det aktuella värdet har ökat per minut, timme, dag och så vidare.

4.3.3 Beräkning av prognos för service

När beräkningen av funktionsvärdena är genomförda enligt 4.3.2 ovan kan även beräkning av prognos för servicetillfälle genomföras. Först avgörs det om en eller flera dataserier som använts i beräkning av funktioner även har tillhörande serviceobjekt. I sådana fall görs en prognos för när gränsen för service (sparas i ett serviceobjektet när det lagras i databasen) är nådd med nuvarande utveckling enligt funktionsvärdena. Beräkningen använder ekvationen:

$$t = \frac{L(s_x - n_x)}{s_y - n_y} + n_x$$

där t är den tidpunkt som söks, n är punkten då senaste service eller nollställning utförts, s är den senaste punkten i dataserien och L är servicegränsen. Ekvationen använder sig av trigonometri för att beräkna tidsskillnaden mellan t och n. Eftersom de två trianglarna som bildas, se figur 4.6, är likformiga kan tangens av vinkeln beräknas genom $\frac{s_y - n_y}{s_x - n_x}$ och sedan användas för att för att få ut tidsskillnaden mellan t och n_x. När sedan n_x adderas till differensen av t och n_x sedan vilken ger tidsstämpeln för t.



Figur 4.6: Uträkning av den tidpunkt då ett värde beräknas nå en *service gräns*: n är punkten för senaste service, s är senaste datapunkten, t är tidpunkten då grafen beräknas nå *service gränsen* och v är vinkeln mellan x-axeln och *snitt-utvecklingen*.

4.3.4 Notifikation av systemfel

Som nämnts i 4.2.6 sparas fel från sensorer i databasen. För att snabbt informera användare om felen krävs det att de användargränssnittet håller sig uppdaterat om eventuella fel har uppstått. Detta är implementerat genom att ett skript exekveras med jämna mellanrum i användargränssnittet . När skriptet exekveras utfärdas en förfrågan till servern om nya fel har inkommit till den berörda anläggningen användaren har tillgång till. Om ett nytt fel har rapporterats visas felet tillsammans med den berörda komponenten och tid upp för användaren.

4.3.5 Generering av konfiguration

För att skapa anläggningar i databasen har ett grafiskt verktyg tagits fram där användare kan konstruera anläggningar. Anläggningar är representerade i en grafisk trädstruktur där användaren kan lägga till och ta bort komponenter ur strukturen. Användaren kan också bestämma de olika värdena varje komponent behöver. När vald struktur sedan genereras skickas trädstrukturen till servern som registrerar strukturen i databasen. Vid val av generering av konfigurationsfil genererar servern en konfigurationsfil av aktuell struktur och returnerar den till användargränssnittet. Användargränssnittet skapar sedan en tillfällig nedladdningsbar fil som användaren kan ladda ner och direkt använda i en sensor. Verktyget stödjer också möjligheten till att uppdatera befintliga strukturer. Vid ett sådant tillfälle kan användaren välja att låta berörda sensorer automatiskt hämta nya konfigurationsfiler. En sådan hämtning genomförs nästa tillfälle sensorn kontaktar servern för att skicka data.

Kapitel 5

Testning och verifiering

Testning och verifiering är två viktiga moment då de säkerställer de funktioner som är implementerade och därför också de krav som systemet har.

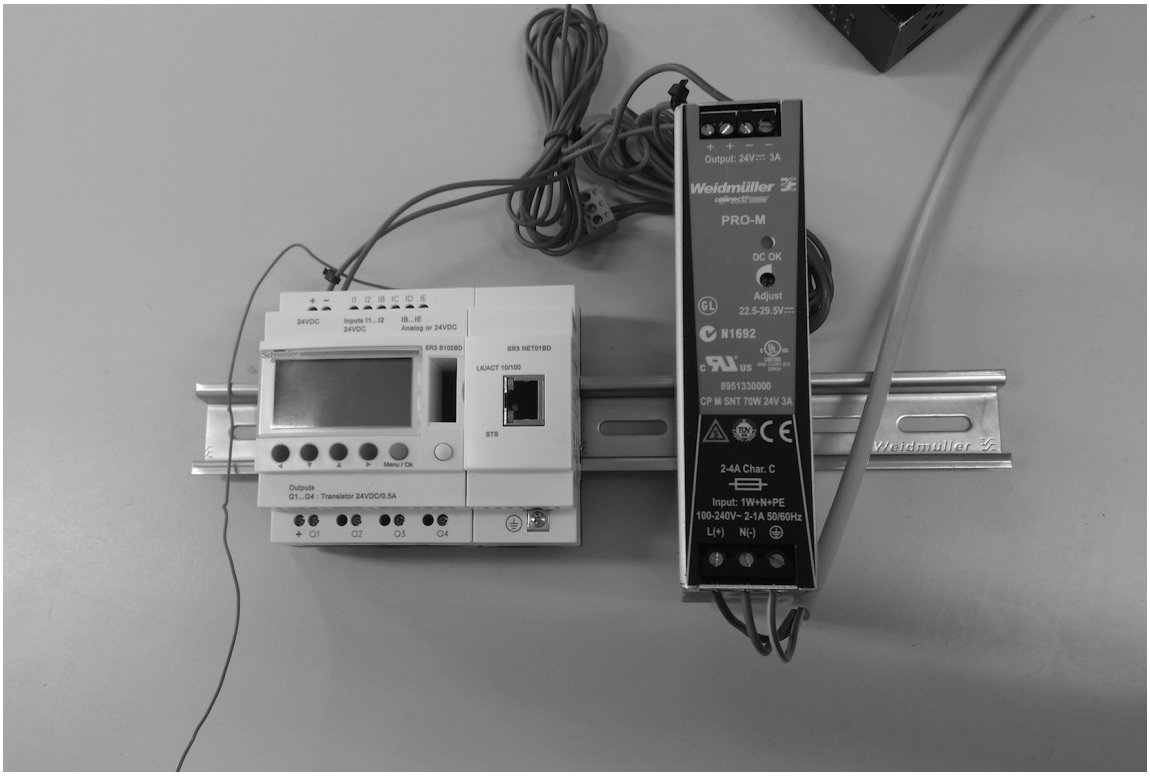
5.1 Kravspecifikation

Uppdragsgivaren tillhandahöll en kravspecifikation som har använts för att implementera tester för systemet. Den kravspecifikation som uppdragsgivaren bestämt för systemet lyder sammanfattningsvis enligt följande:

- *Beräkningar, inställningar och konfiguration ska ske i en del av systemet som kan hantera flera lyftdon.*
- *Databasen ska kunna hantera 1-n lyftdon, 1-n grupperingar av komponenter i lyftdon och 1-n användare.*
- *Systemet ska kunna läsa av en eller flera typer av värden, exempelvis drifttider, lastceller, indikeringar och larm.*
- *Kommunikationen mellan sensor och server ska vara UDP eller TCP. Protokollet för kommunikation måste kunna hantera någon form av kryptering, exempelvis SSL.*
- *All kommunikation mellan sensor och server ska instansieras av sensor. Detta för att enklare kunna sälja systemet då sensorn med största sannolikhet kommer installeras bakom en brandvägg.*
- *Sensor måste kunna hantera bortfall av kommunikation mot server genom buffring av värden.*
- *Sensor ska hantera kommunikationsfel i kommunikationen till PLC. Vid längre avbrott ska larm skickas till server.*
- *Kommandon till sensor bör kunna hanteras, lämpligen läggs de på kö och förs över när sensor kontaktar server nästa tillfälle. Kommandon kan till exempel vara: skicka självdiagnostik, starta om, ändra konfigurering. Ändra konfigureringen är ett krav.*

5.2 Användning av Schneider SR3-B102BD

För att säkerställa att systemet kan hantera en riktig PLC används en PLC från Schneider med betäckningen SR3-B102BD tillsammans med en MODBUS / TCP modul för kommunikation över Ethernet, se figur 5.1. Den tillhandahölls av uppdragsgivaren och innehåller förprogrammerade register med både statiska och varierande värden. Genom att använda en riktig PLC kan Jamod, kodbiblioteket för hantering av MODBUS, funktionstestas samt större simuleringar med PLC, sensor, server och användargränssnitt genomföras.



Figur 5.1: Schneider SR3-B102BD sammankopplad med MODBUS / TCP modul för kommunikation över Ethernet. Till höger om PLC i figuren syns även nätaggregatet.

5.3 JUnit

För att testa funktionaliteten i systemet är alla delar testade via JUnit-tester. Att köra dessa tester är ett krav för att kunna bygga applikationerna för både sensorn och servern, vilket säkerställer att koden alltid är testad innan den kan exekveras. För att se till att en bugg förblir löst implementeras alltid ett test som replikerar buggen innan buggen blir löst i mjukvaran. Detta ger att testerna kommer att fallera då buggen kommer tillbaka och kan på så sätt omedelbart upptäckt.

5.4 Jetty

För att kunna skriva JUnit tester för sändningen från sensor till server används Jetty som tillfällig server när testmiljön exekveras. Jetty konfigureringen är minimal och svarar med responskod 200 på alla anrop. Att sätta upp Jetty på detta sätt gör att testerna kan köras lokalt på samma maskin utan att behöva sätta upp hela servermiljön.

5.5 MockMVC

För att testa servern enligt konceptet Black-Box Testning används MockMVC som är inkluderat i Spring. MockMVC gör det möjligt att för test, bygga upp hela servern med en tillfällig databas för att testa anrop mot applikationen. Detta gör det möjligt att testa anrop via REST och att simulera sensorer i interaktion med servern i riktiga scenarion.

5.6 PLCMock

För att testerna skall kunna köras utan kommunikation med en riktig PLC har en mjukvaruimplementation av en PLC implementeras kallad PLCMock. PLCMock läser av konfigurationsfilen i sensorn och agerar PLC med given adress och svarar med slumpmässiga värden på de värde adresser specificerade för PLC systemet i konfigurationsfilen.

Kapitel 6

Resultat

Samtliga utav de tre modulerna: sensor, server och användargränssnitt följer de krav som ställdes i kravspecifikationen. Mjukvaran för sensor och server är färdig men användargränssnittet är en prototyp. Användargränssnittet kan demonstrera alla funktioner i systemet men gränssnittet har ingen övergripande helhet, vilket gör att användare inte intuitivt kan använda gränssnittet utan behöver leta efter funktioner.

6.1 Sensor

Sensorn klarar utan problem de krav som är definierade för den. Följande funktioner är fullt implementerade:

- Hårdvaran som sensorn använder har inga problem att exekvera mjukvaran.
- Sensorn klarar av att hämta data från flera PLC i samma nät.
- Schemaläggningen av de olika processerna i sensorn är robust och håller tidsintervallen. Den största differensen som uppmätts under belastningstester är 5 millisekunder.
- När en uppdaterad konfigurationsfil skickas från servern uppdateras inställningarna i sensorn utan att programmet startas om.

6.2 Server

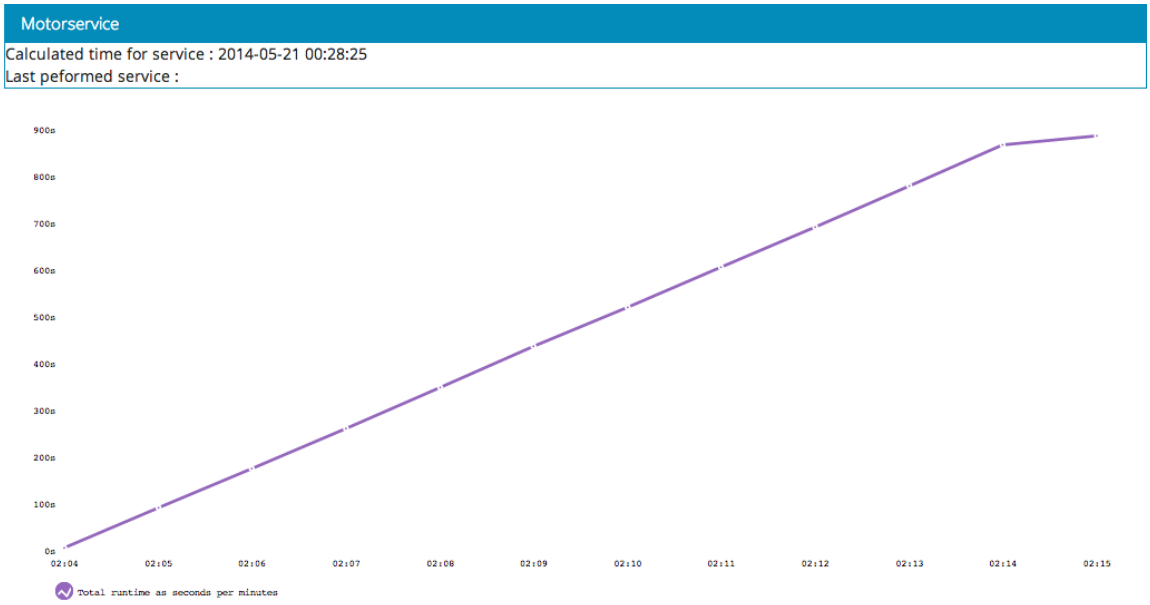
Servern kan hantera alla sensor-relaterade funktionerna enligt specifikationen. Det behövs dock mer testning av uppdatering av konfigurationer för att fullt säkerställa dess funktionalitet gällande utökning och borttagning av komponenter samt felhantering vid dåligt formaterad data.

Följande funktioner är fullt implementerade:

- Strikt envägskommunikation från sensor till server.
- Identifiera inkommande data från sensor och spara den i databasen.
- Distribuera inkommande data till flera trådar för att snabba upp exekveringen.

- Schemalägga processer för att upptäcka fel i sändning från sensor.
- Autentisering av användare.
- Generera konfigurationsfil för sensorer samt dynamiskt skicka med dessa vid anrop.

6.3 Användargränssnitt

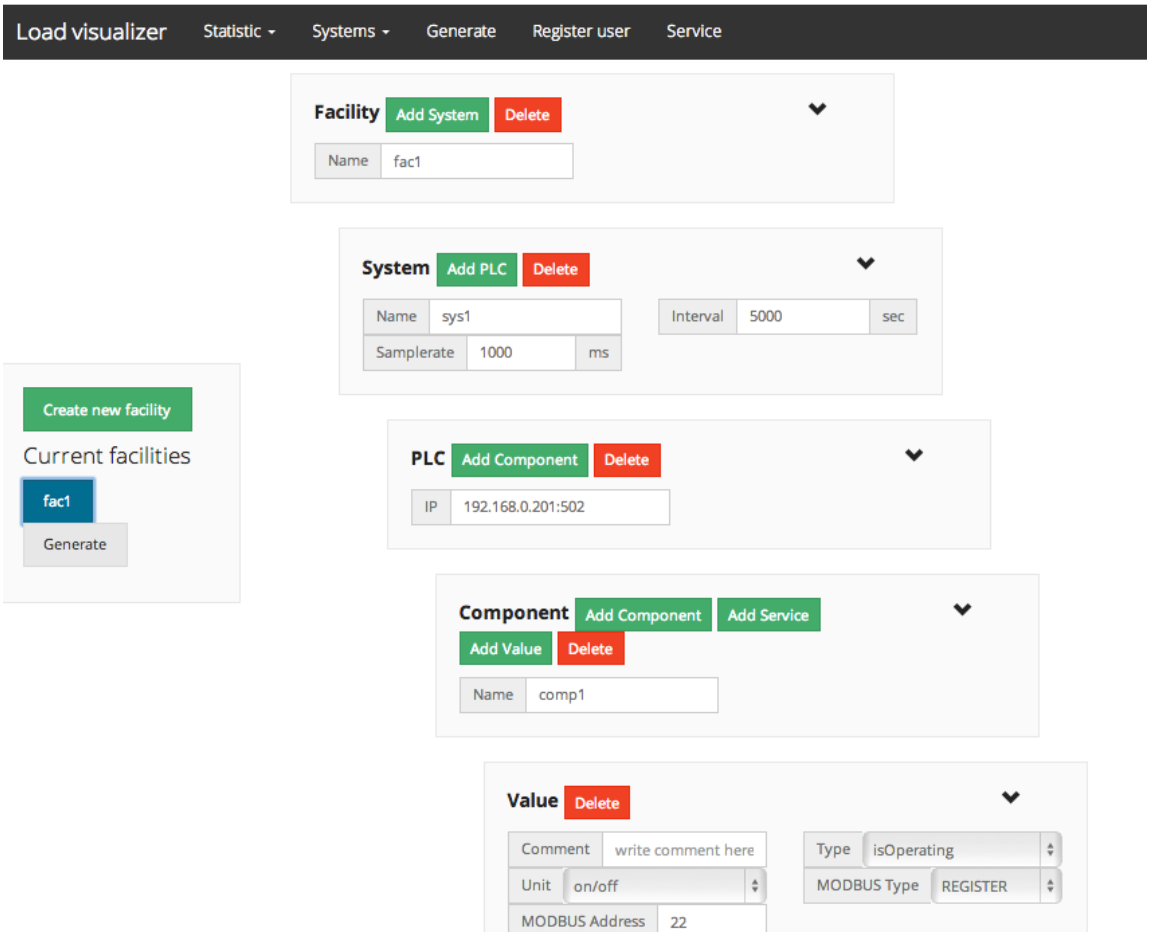


Figur 6.1: Användargränssnittet visandes utvecklingen av antalet körtimmar på en komponent. Prognosen för när komponenten behöver genomföra en service kan ses ovanför grafen i figuren.

Följande funktioner är fullt implementerade:

- Rapportera om genomförd service samt ge prognos för när nästa service bör ske, se figur 6.1.
- Informera användaren om fel vid sensorers läsning av PLC eller sändning till server uppstår, se figur 6.2.
- Beräkna specificerade funktionsvärden som beror på en eller flera dataserier från servern.
- Visuellt visa upp funktioner i form av tabeller och grafer, se figur 6.1.
- På ett överskådligt sätt skapa konfigurationer, se figur 6.3.

Figur 6.2: Användargränssnittet visandes en notifikation om fel vid sändning från sensor till server.



Figur 6.3: Användargränssnittet vid genereringen av anläggningar och konfigurationer. Visualiseringen av strukturen kan ses i trädstrukturen till höger i bild.

6.4 Modularitet

6.4.1 Sensor

Sensorn är implementerad i ett eget projekt med de olika delkomponenterna implementerade som olika processer. Sensorn tar ingen hänsyn till vad det är för data som den läser och formaterar den inte heller. Det enda sensorn gör med datan är att paketera den och skicka vidare till servern. Detta gör sensorn väldigt generell och kan hantera alla typer av data lästa från olika PLC system. Den enda restriktionen som finns är att den endast kommunicerar via MODBUS protokollet över Ethernet / TCP med PLC systemen. Skulle ett annat kommunikations protokoll vara önskvärt är det enkelt att byta då det endast är inbyggt i en mottagar del.

6.4.2 Server

Efter som att systemet inte är byggt för någon specifik typ av data är det enkelt att applicera systemet på alla typer av lyftdon från lyftkranar till hissar. Genom att användare i användargränssnittet kan generera hela system kan man på ett enkelt sätt anpassa systemet till specifika anläggningar. Servern har också stöd för att dynamiskt lägga till nya grupper av dataserier för en funktion, exempelvis funktionen för fullasttimmar där en dataserie som beskriver om systemet har varit aktivt över tid behövs. Detta är möjligt genom att användargränssnittet är den del som sammanställer dataserierna och beräknar det beräknande värdet. På grund av detta har servern bevarats väldigt generell då den endast tillhandahåller användargränssnittet med rådata som tidigare inkommit från sensorer.

6.4.3 Plattformsoberoende

Både sensor och server är utvecklade plattformsoberoende. För att köra någon av dem behöver endast Java Runtime Environment 7 (JRE 7) vara installerad på värdmaskinen vilket finns tillgängligt för samtliga av de mest populära operativsystemen. Skulle behovet av att flytta systemet till ny hårdvara uppstå kommer alltså operativsystem, med största sannolikhet, inte att vara ett problem. Det krav som finns på hårdvaran är tillgång till Ethernet anslutning, denna kan emuleras i mjukvara ifall att en fysisk Ethernet anslutning inte är möjlig.

Kapitel 7

Slutsats och diskussion

7.1 Återkoppling till kravspecifikationen

Som nämnt i avsnitt 6 så är många funktionaliteter implementerade. Vid en återkoppling till den ursprungliga kravspecifikationen (se 5.1) kan man se att i princip alla av de funktioner och krav som fanns på systemet har implementerats. Detta har möjliggjorts dels genom de metoderna som nämnts i kapitel 5 men också genom att simuleringar av hela kedjan med sensor, server och användargränssnitt gjorts och säkerställt dess kompatibilitet med varandra.

7.2 Val av kodbibliotek och ramverk

Utvecklingen har utan större motgångar fortlöpt under hela projektet. Ramverk och kodbibliotek har fungerat och gjort det som förväntats. Men då ramverket Jamod är implementerat i Java 5 och resterande implementationer i projektet är implementerat i Java 7 har tid lagts på att gå igenom och uppdatera kodstandarden i ramverket. Dessutom har JSONSimple och Jamod dessvärre inte mött förväntningarna vid mer avancerad användning. Detta beror dels på att de har en approach som löser deras mål på simpla sätt. Detta gör dem enklare att använda men sänker även deras möjlighet att lösa mer avancerade mål. Ett liknande scenario skedde även för valet av kodbibliotek för visualisering där angular-charts hade bristfälligt stöd för att visa grafer med tidsbaserad upplösning. En lärdom av detta är således att lägga mer tid på val och framförallt testning av ramverk och kodbibliotek så att liknande situationer inte uppstår igen.

7.3 Etiska aspekter

Att stora lyftdon får service i god tid innan de går sönder är viktigt både för ägaren av donet men även för de som arbetar på arbetsplatsen där donet verkar då det uppstår en säkerhetsrisk för verksamma kring lyftdonet om en service blir försenad eller uteblir. Därför är det viktigt att systemet skapat under detta projekt (LoadVisualizer) är säkert. Mycket fokus har legat på testning för att se till att information inte går förlorad på grund av till exempel kabelbrott. Förlorad data skulle exempelvis kunna ge felaktig information om när en service bör utföras. Användare av lyftdonet kan genom detta projekt få information i realtid om hur mycket återstående användning det finns innan

service och när lyftdonet beräknas nå det tillfället. På detta sätt gör LoadVisualizer att lyftdonet blir servat och arbetsplatsen blir säkrare.

7.4 Miljömässiga aspekter

Systemet ger ägaren möjlighet att övervaka hur mycket last som lyftdonet faktiskt utsetts för. Om lyftdonet inte används fullt ut kan donet köras längre innan det behöver service vilket ger färre utryckningar för servicebolagen. Det gör även att man inte byter ut delar som fortfarande har kapacitet och minskar på så sätt användningen av naturresurser.

7.5 Generalitet och användningsområden

De maskiner som projektet var avsätt för att hantera var främst lyftdon. Men för den faktiska hämtningen av information från lyftdon ansluter sig systemets sensorer mot lyftdonens PLC. Då en PLC inte är något specifikt för ett lyftdon utan används även i andra maskiner, till exempel personhissar, kan systemet även användas i för andra maskiner. Det här gör systemet generellt och ger det ett brett användningsområde. De krav som finns för att kunna använda systemet är följande:

- Möjlighet till att fästa en eller flera sensorer i nära kontakt till maskinens PLC.
- Att sensorerna genom anslutning till PLC kan hämta data som berör maskinen.
- Att den miljö och plats som sensorerna placeras i tillåter och har möjlighet till att skicka informationen över Ethernet till en server.

Om en maskin uppfyller ovanstående kraven kan systemet installeras och därigenom ge driftansvariga eller användare information om belastning, körtid och tid till service.

7.6 Projektplanering

Som Gantt-diagrammet för projektet visar (se bilaga A) så var det inledande målet att få sensorn att kommunicera med PLC. Då implementationen med Jamod gick snabbare och lättare än förutspått så kunde andra funktioner för sensorn implementeras snabbare. Diagrammet visar också att servern var beräknad till cirka fyra veckor, något som inte kunde hållas på grund av de problem som uppstod med att modellera databasen i Java, se 7.7.2. Förutom problem med databasen lades mycket tid på att få ner svarstiderna för anrop från sensorer. Även felhantering för både sensor och server var moment som ständigt utvecklades genom projektet, vilket ledde till att utvecklingen av användargränssnitt, autentisering och auktorisering senarelades.

7.7 Förbättringar till framtida projekt

7.7.1 Jamod

Inledningsvis var det enkelt att använda Jamod. Sensorn nådde snabbt det stadium att den kunde kommunicera med PLC system. När det sedan gällde att göra mer avancerade operationer var inte kodbiblioteket lika enkelt. Anledningen var att kodbiblioteket var implementerat i en gammal version av Java. Detta åtgärdades tillsammans med andra finjusteringar vilket löste de problem som hade uppstått runt användningen av kodbiblioteket. Priset för detta var ett par dagar som gick förlorade och frågan är då ifall projektet hade vunnit mer på att välja ett annat kodbibliotek för MODBUS kommunikation än Jamod istället för att åtgärda problemen i Jamod.

7.7.2 Val av plattform för servern

Utvecklingen av servern i projektet har varit ansträngd. Framförallt hantering av databasen. Det ramverk vi arbetat mot JPA (Java Persistence API) lägger mycket vikt vid att modularisera databaslänkningen så att själva databashanteraren, i detta fall PostgreSQL, skall kunna bytas ut utan att koden för att hantera databasen behöver skrivas om. Denna satsning gör dessvärre JPA väldigt tungarbetat och som konsekvens ha många kryptiska fel har uppstått under projektet. Skulle projekt göras om bör en annan plattform som är enklare att arbeta med väljas. Det är viktigt att enkelt kunna underhålla mjukvaran i en produkt och byta ut de delar som inte längre fungerar, men när priset för uppnå ett såpass modulärt system gör själva utvecklingen av produkten tung är det inte rätt väg att gå.

7.7.3 Testning av användargränssnittet

För att underlätta belastningen av servern beslutades att beräkningar av den insamlade datan i servern skulle ske i användargränssnittet när användaren begär att få se informationen. Det här gjorde att komplexiteten i användargränssnittet växte oväntat, vilket var anledningen till att denna del av systemet inte hade ett testramverk bestämt från början av utvecklingen, precis som i sensorn och servern. Därför fick mycket resurser läggas på att lösa buggar och problem, något som hade minimerats vid användning av testramverk.

7.8 Framtida utvecklingsplaner

7.8.1 Användargränssnitt

Ett testramverk behövs implementeras för att bestämma funktionalitet av till exempel beräkningar av grafer samt för att möjliggöra vidare utveckling av komplexa funktioner. För att användargränssnittet ska bli mer överskådligt och lättanvänt behövs också en tydligare struktur för navigering i gränssnittet. Tänkbara alternativ till webbgränssnittet är att utveckla mobilapplikationer för Android och iOS. Systemet är tänkt att användas från mobila enheter men efter som att ett webbgränssnitt fungerar på de flesta plattformar har det stannat vid det under detta projektet. Den bästa upplevelsen får användare oftast genom applikationer som körs direkt på enheten varför Android och iOS fortfarande är intressanta för framtida projekt.

7.8.2 Flera kommunikations protokoll

Den enda delen av systemet som inte är modulär är kommunikations länken. Anledningen till detta är att det PLC system som projektet haft tillgång till har kommunicerat via MODBUS / TCP då det är det som är angivet i kravspecifikationen. I mjukvaran finns stöd för att andra kommunikationsprotokoll kan användas men detta har prioriterats bort då det inte är viktigt för uppdragsgivaren. För att öka målgruppen för försäljning av systemet är detta intressant för framtiden.

7.8.3 Felrapportering via email

I nuvarande implementation rapporteras fel endast som notifikationer till användare som är inloggade i användargränssnittet. Det här gör att användare inte kan få information om fel har inträffat om de inte aktivt använder gränssnittet. En möjlig lösning till detta problem är att servern sänder iväg email till användare av det berörda systemet. På så sätt kan användare snabbare få information vid fel och därför kunna minimera skadan och / eller driftstoppet.

7.8.4 Larmhantering

Tanken med larmhanteringen var att användaren skulle kunna skapa larm som larmade när ett valt värde har passerat en given gräns. Olika larmnivåer för larm skulle ge olika betydelser så att de kan filtreras och de viktigaste larmen kan skickas, via email eller sms, till driftpersonalen för det berörda systemet. Larmhantering var planerat som en extra funktion att implementera ifall tiden tillät vilket den inte har gjort och är därför en av de funktioner vi har planerat att implementera i framtiden.

7.8.5 Stöd för språkval

Flerspråksstöd har inte prioriterats i projektet då det inte är viktigt för att göra prototyp på systemet. När systemet sedan skall säljas som en produkt är det viktigt att användaren skall kunna välja språk i användargränssnittet då det bidrar till att öka marknaden.

Referenser

[01] *Standard for Software Component Testing*. Version 3.4. BCS SIGIST, 2001. URL: http://www.testingstandards.co.uk/Component%20Testing.pdf.

[12] *MODBUS APPLICATION PROTOCOL SPECIFICATION*. Version 1.1b3. The Modbus Organization, 2012. URL: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

[13] *Introducing JSON*. 2013. URL: http://json.org/ (hämtad 2014-04-10).

[14a] *angular directives for commonly used d3 charts*. 2014. URL: https://developers.google.com/chart/ (hämtad 2014-04-10).

[14b] *Apache Maven Project*. 2014. URL: http://maven.apache.org/ (hämtad 2014-04-04).

[14c] *Designed for everyone, everywhere*. 2014. URL: http://getbootstrap.com/ (hämtad 2014-04-15).

[14d] *Git(software)*. 2014. URL: http://en.wikipedia.org/wiki/Git_(software) (hämtad 2014-05-02).

[14e] *Jetty - Servlet Engine and Http Server*. 2014. URL: http://www.eclipse.org/jetty/ (hämtad 2014-05-20).

[14f] *jQuery*. 2014. URL: http://jquery.com/ (hämtad 2014-04-03).

[14g] *Spring Security*. 2014. URL: http://projects.spring.io/spring-security/ (hämtad 2014-05-27).

[AA10] S. Allamaraju och M. Amundsen. *RESTful web services cookbook*. O'Reilly, 2010. ISBN: 9780596801687.

[Bos13] M. Bostock. *Data-Driven Documents*. 2013. URL: http://d3js.org/ (hämtad 2014-04-10).

[Buc00] W. Buchanan. *Computer Busses*. Butterworth-Heinemann, 2000. ISBN: 9780340740767.

[Ell05] J. Elliott. *Hibernate: A Developer's Notebook*. O'Reilly, 2005. ISBN: 9780596006969.

[Fra14] S. Fragnaud. *Versatile charts for AngularJS*. 2014. URL: http://n3-charts.github.io/line-chart/ (hämtad 2014-05-19).

[HH12] C. Ho och R. Harrop. *Pro Spring 3*. Apress, 2012. ISBN: 9781430241072.

[Joh+13] R. Johnson m. fl. *Spring Framework Reference Documentation*. 2013. URL: http://docs.spring.io/spring-framework/docs/4.0.0.RELEASE/spring-framework-reference/html/ (hämtad 2014-03-27).

[Kul13] C. Kulkarni. *angular directives for commonly used d3 charts*. 2013. URL: http://chinmaymk.github.io/angular-charts/ (hämtad 2014-04-10).

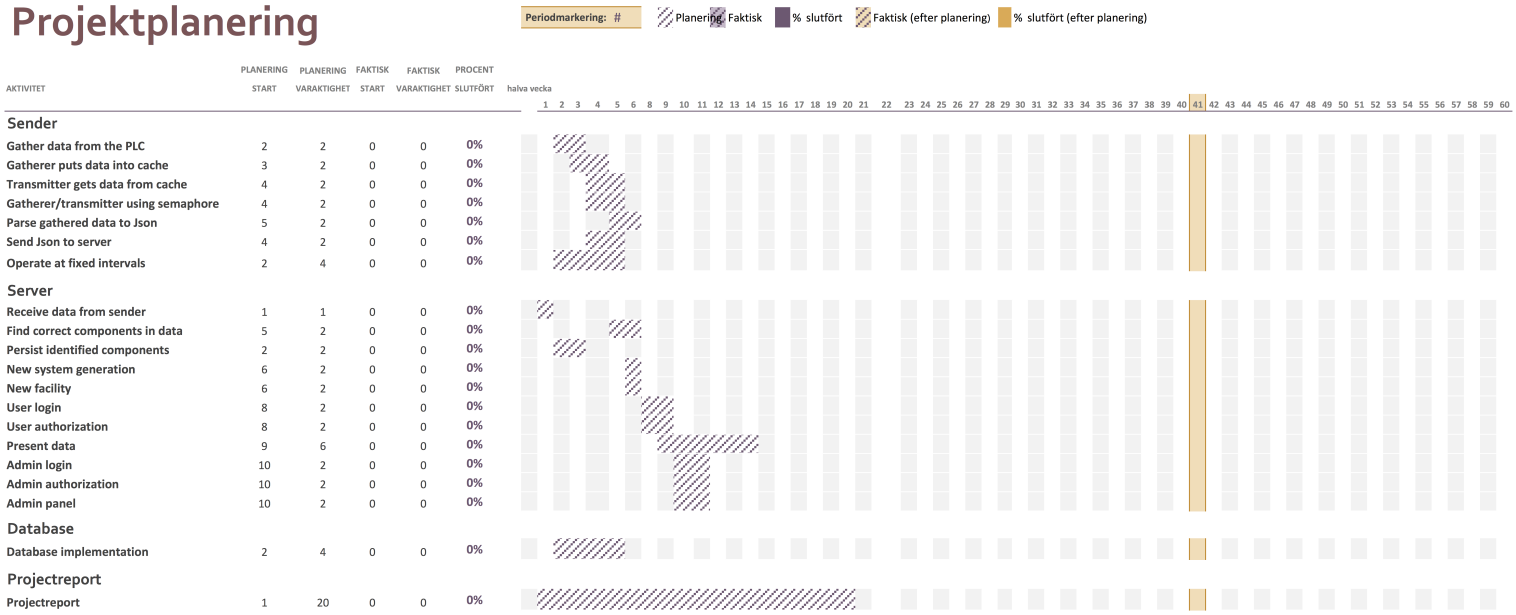
[Loh10] M. Lohbihler. *Modbus for Java*. 2010. URL: http://modbus4j.sourceforge.net/ (hämtad 2014-04-03).

[Oli11] O. E. Olivier Hersent David Boswarthick. *Internet of Things : Key Applications and Protocols (2nd Edition)*. Wiley, 2011. ISBN: 9781119994350.

[Wim10] D. Wimberger. *Java Modbus Library*. 2010. URL: http://jamod.sourceforge.net/ (hämtad 2014-03-26).

Bilaga A

Projektplannering



Figur A.1: Ursprungligt Gantt-diagram för projektet. Notera att skalan för tidsaxeln är i halva veckor.