ARKITEKTURBESKRIVNING

Ansvarig: Emanuel Bergsten

Version 1.0

Status

Granskad		
Godkänd		

PROJEKTIDENTITET

Övervakningsfunktion för en mätplattform för mätning i bil

TDDD77/TDDD76 grupp 7, 2014/VT

Linköpings tekniska högskola

Namn	**Ansvar**	**E-post**
Tony Fredriksson	Teamledare (TL)	tonfr314@student.liu.se
Max Halldén	Dokumentansvarig (DOK)	maxha651@student.liu.se
Viktor Andersson	Analysansvarig (AA)	vikan930@student.liu.se
Emil Berg	Testledare (TST)	emibe709@student.liu.se
Johan Classon	Kvalitetssamordnare (KS)	johcl379@student.liu.se
Emanuel Bergsten	Arkitekt (ARK)	emabe390@student.liu.se
Niklas Ljungberg	Utvecklingsledare (UTV)	niklj028@student.liu.se
Johan Sjöberg	Specialist (SPC)	johsj101@student.liu.se

E-postlista för hela gruppen: tddd77grupp7@googlegroups.com

Kund: Fordonssystem, ISY, Linköpings Universitet,
Kontaktperson hos kund: Per Öberg, per.oberg@liu.se

Kursansvarig: Kristian Sandahl, kristian.sandahl@liu.se
Handledare: Eric Elfving, eric.elfving@liu.se

Innehåll

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
1.0	2014-03-10	Första versionen	Alla	Johan C

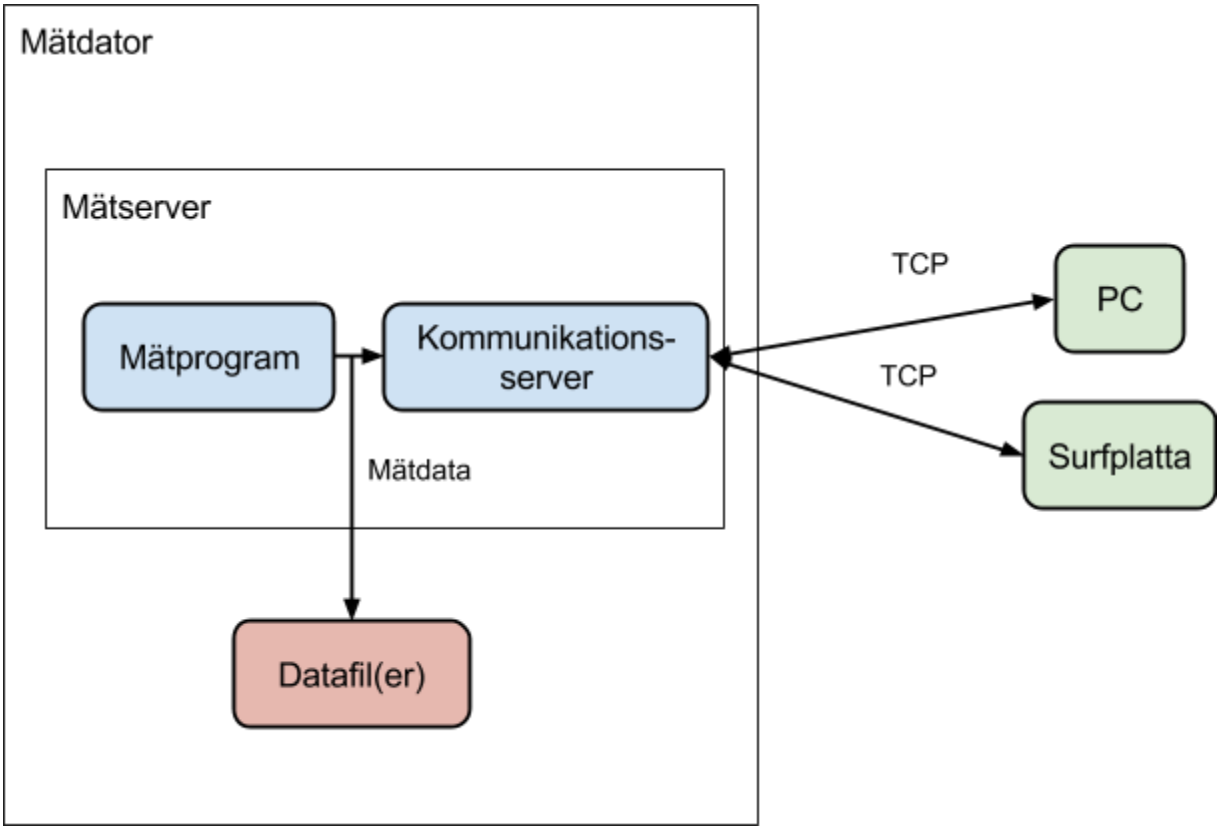
1. Syfte

Syftet med det här dokumentet är att samla uppdaterad information om hur mjukvaran i projektet är uppbyggd ur ett arkitekturellt perspektiv. Den beskriver vilka val som gjorts i projektet och varför de gjorts.

Dokumentet skrivs enligt mallen architectural notebook [1].

2. Arkitektoniska mål och designfilosofi

Systemet är ämnat att förbättra det gamla systemet genom att bidra med välutvecklade gränssnitt och lättare åtkomst till mätdata. Från det nya gränssnittet ska systemet kunna styras så att användaren kan starta och stoppa mätningar och välja hur datan från mätningarna ska sparas. Då mätningarna ofta pågår under relativt lång tid ska systemet klara att köras under långa tider utan fel.
Figur 1 nedan visar en översiktlig beskrivning av systemet.



Figur 1: Översikt över systemet.

Mjukvaran som utvecklas ska anpassas till den befintliga mjukvaran. Detta innebär både att utöka det befintliga programmet och att inte sänka prestandan så att problem uppstår. Om inte kund explicit säger något annat så ska ingen funktionalitet från det gamla systemet tas bort eller förändras. Systemet kommer med största sannolikhet att utvecklas vidare i framtiden och detta ska hållas i åtanke under utvecklingen för att systemet lätt ska kunna byggas ut.

Mjukvaran ska vara flexibel, så att det är lätt att utöka programmet med nya sensorer med tillhörande data.

Gamla mätfiler ska kunna konverteras för att användas i det nya systemet.

2.1 Kommunikationsserver

Kommunikationsservern ska vara ett tillägg till den befintliga mätprogramvaran och dess uppgift kommer vara att agera länk mellan mätprogrammet och klienterna. Kopplingen mellan server och klienterna ska abstraheras så att förändringar och utökning av mätprogramvaran och sensorer varken kräver förändringar av protokoll eller stora förändringar av mjukvaran i sig. Mjukvaran på servern ska vara plattformsoberoende i så stor mån det är möjligt.

Mjukvaran ska agera i mjuk realtid vilket medför att hastigheten för hanteringen av data är viktig.

2.2 Klient

Detta avsnitt redogör för de olika typerna av klienter som kommer användas i projektet.

Klienten består av två delar; back-end och front-end. Back-end ska i stor utsträckning vara plattformsoberoende, genom att skrivas i Java 6. Front-end kan inte lika lätt göras plattformsoberoende. Därför skapas en klient med egen front-end till de viktigaste plattformarna.

2.2.1 Applikation till surfplatta

Den största delen av programmet rör en applikation för Android. Koden ska följa nya standarder i största möjliga utsträckning för att ha lång livslängd på systemet.

2.2.2 PC-mjukvara

Koden för PC-klienten ska följa standarder för grafiska gränssnitt i Java i största möjliga utsträckning.

3. Antaganden och beroenden

Tillgång till den hårdvara och mjukvara som används samt beskrivning av sagd mjukvara förväntas. Exempeldata samt full tillgång till all data som samlas vid fälttester förväntas. Projektgruppen förväntas också ha tillgång till hårdvara med liknande, eller bättre, prestanda för exekvering av mjukvara. Projektgruppen förväntas även ha tillgång till mätbilen och dess tillhörande sensorer i rimlig mån under projektets gång.

Projektgruppen ska ha tillgång till en stationär dator med linuxinstallation på och denna dator ska vara så identisk som möjligt till den mätdatorn som finns i bilen. Denna ska kunna användas för att testa programmet som utvecklas och för att simulera sändning av riktig mätdata.

4. Signifikanta krav ur arkitektursynpunkt

Se kravspecifikationen för systemet.

5. Val, begränsningar och motiveringar

Nedan följer beskrivningar av de begränsningar som ställts på programmet, de val som gjorts i arkitekturen samt motiveringar för dessa.

5.1 Kommunikationsserver

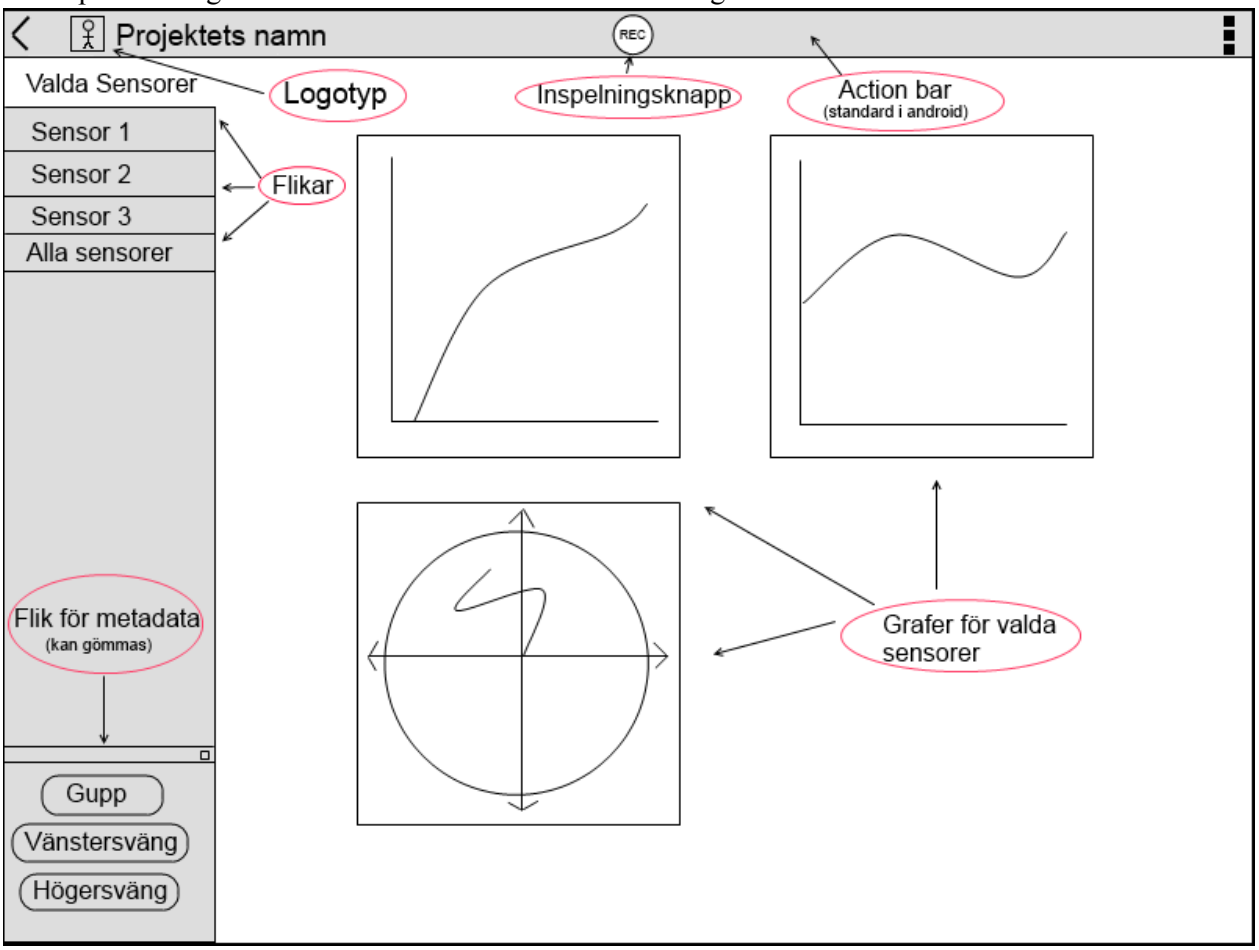
Koden för kommunikationsservern skrivs i C++ för att underlätta kommunikation med det befintliga mätprogrammet. Samma Protobuf-protokoll som används för kommunikation (se 5.3) används också för att lagra mätdata på fil. Anledningen till att samma protokoll används är för att det blir lätt att läsa något

från fil för att sedan direkt skicka iväg det.

Eftersom biblioteket Boost redan används i det befintliga systemet ska det användas så länge det underlättar utvecklingen av kommunikationsservern. Boost ska användas istället för plattformsberoende bibliotek i så stor utsträckning som möjligt.

5.2 Klient

Java används för att skriva större delar av de olika klienterna. Surfplattans front-end måste skrivas för Android (eftersom den ska köras på en Androidsurfplatta och utvecklas i enlighet med Googles riktlinjer) medan back-enden skrivs i Java. Android-delen av programmet kommer sedan behöva skrivas om i Java under utvecklandet av applikationen till PC, så att hela PC-applikationen körs i Java. Figuren nedan visar en idé på hur det grafiska interfacet kan se ut vid en mätning



Figur 2: Gränssnittsprototyp

Appen består av två delar: Front-end och back-end. Front-end hanterar det grafiska, allt som användaren kan se på skärmen och styr de valen som användaren kan göra. Back-end tar emot olika kommandon från front-enden för att ta fram den information användaren vill se. Ett exempel på ett kommando är att plocka fram en viss sensors data med avsikt att representera datan som en graf på skärmen.

Att dela upp en app på detta sätt följs i många mjukvaruprojekt och kallas för two-tier architecture. Det är bra för att skapa en abstraktion mellan användare och det faktiska programmet.

5.3 Kommunikationsprotokoll

Det protokoll som används för kommunikationen mellan server och klienter är definierat med *Google Protobuf*[1]. Protobuf kan användas för att enkelt och plattformsoberoende kommunicera mellan olika språk, och eftersom applikationen till surfplattan är skriven i Java/android och servern är skriven i C++ ses det som ett utmärkt sätt att minimera kommunikationsproblemen mellan server och klienter.

När data skickas från servern med Protobuf måste först meddelandets längd i bytes skickas som ett variabelt heltal, kallat Varint. Sedan skickas själva meddelandet. Denna funktionalitet finns inbyggd i Protobuf för Java (se writeDelimitedTo[2]) men inte för C++ vilket gör att denna funktionalitet måste implementeras i servern.

5.4 Allmänna anvisningar

Nedan följer listor över vad vi ska fokusera på i projektet samt vad vi inte ska fokusera på.

5.4.1 Gör

- Abstrahera så mycket som möjligt.
- Systemet ska fungera även om delar/sensorer förändras.
- Systemet ska fungera via kommunikationsvägar som har lägre överföringshastighet.
- All mjukvarutveckling ska ske med åtanke på framtida utvecklingar. All mjukvara ska vara väldokumenterad av samma skäl.
- Externa bibliotek ska bestå av öppen källkod i så stor utsträckning som möjligt.
- Systemet ska vara plattformsoberoende i så stor utsträckning som möjligt.
- Kommunikationsservern ska i möjligaste mån använda sig av samma bibliotek som den befintliga mjukvaran, som t.ex. C++ standardbibliotek och Boost.

5.4.2 Gör ej

- Systemet ska inte automatiskt kunna detektera nya sensortyper som inte fanns vid utvecklandet av programmet.

6. Arkitektoniska Mekanismer

Under denna rubrik beskrivs syftet, attribut och funktionaliteten hos olika delar av arkitekturen.

6.1 Sensorkonfiguration

Mätservern ska automatiskt kontrollera att experimentuppställningen motsvarar en fördefinierad konfiguration, samt att sensorerna ger rimliga värden. Dessa värden kan ställas in av användaren för att anpassa det aktuella testet.

6.2 Dataöverföring

Den insamlade datan ska skickas till klienter serialiserat genom Protobuf-protokollet. Protobuf används för att på ett effektivt och plattformsoberoende sätt serialisera och deseriealisera data.

6.3 Dataöversättning

Data från tidigare mätsessioner, även sådana som skett innan implementationen av det utvecklade systemet, ska kunna läsas in och översättas till en form som programmet kan läsa.

[1] https://code.google.com/p/protobuf/
[2] Länk till writeDelimitedTo

6.4 GUI

En användare ska enkelt kunna starta en mätning och välja vilken information som ska visas.

6.5 Datavisualisering

Den mottagna datan ska kunna visas på klienter. I de senare versionerna av klienterna kommer data att presenteras i grafer.

7. Nyckelabstraktioner

Här listas kritiska koncept och abstraktioner för systemet.

7.1 Server

Serverpaketet är en utökning av mätprogrammet som tillåter sändning av data till flertalet back-ends.

7.2 Klient

En klient är antingen en surfplata eller PC som kommunicerar med servern för att ge användare möjlighet att använda systemet.

7.2.1 Back-end

Den del av klienterna som kommunicerar direkt med servern och hanterar all data som mottagits under körning.

7.2.2 Surfplatteklient

Surfplatteklienten körs på en surfplatta och är tänkt att sitta i bilen för att ge föraren möjlighet att använda systemet. Kommunicerar med servern via back-end.

7.2.3 PC-Klient

En klient som körs på PC och används av användaren för att använda systemet. Kommunicerar med servern via back-end. PC-klienten ska använda samma back-end som surfplatteklienten och bete sig på liknande sätt som surfplatteklienten.

8. Lager eller ramverk för arkitektur

Arkitekturen på surfplattan följer “front-end” och “back-end”-struktur. Denna innebär att front-end hanterar användarens input och den grafiska representationen för applikationen. Back-end hanterar all kommunikation med utomstående enheter, såsom server, helt bortabstraherat från front-end. Back-end tar även hand om all sparning av data som ska ske.

Servern anpassas så långt det är möjligt till den befintliga arkitekturen.

9. Arkitektonisk vy

Nedan följer olika arkitektoniska högnivå-vyer över systemet.

9.1 Logisk vy

Mätservern består av det gamla programmet (mätprogrammet) som integrerats med den nya delen av programmet (kommunikationsservern). Kommunikationsservern består av en serverdel som hanterar uppkopplingar, samt en klientdel, en för varje uppkopplad klient, som serverdelen håller reda på. Mätprogrammets arkitektur har förändrats för att underlätta påbyggnad. Nya komponenter läggs nu till som antingen *controllers*, vilket är en typ av tjänster som kör parallellt med programmet och t.ex. ser till

att skriva meddelanden till fil eller att synkronisera klockan. Det finns också *sensors*, där en sensormodul ska skapas för varje ny sensor. Mätprogrammet hanterar all exekvering av controllers och sensors, så de startar upp i rätt ordning, och endast exekverar när de ska.

Back-end-delen av systemet är uppbyggd så att de större delarna är egna objekt och med klasser som identifierar sig med interface, exempelvis paketen Message och Sensor som beskrivs nedan.

Front-end har en lista av fragment som används för att grafiskt visualisera mätdata samt hantering av detta.

9.1.1 Paketstruktur

Paketstrukturen består av ett mätserverpaket, ett back-endpaket, ett front-endpaket för surfplatta och ett front-endpaket för PC. Dessa är sedan uppdelade i mindre subpaket.

9.1.2 Kritiska gränssnitt

Kommunikationsprotokollet som används mellan kommunikationsservern och klienten är kritiskt.

9.1.3 Viktiga klasser och subsystem

Mätserverns viktigaste klass är *CommunicationServer*, som sköter alla TCP-anslutningar mellan servern och klienter. Den representerar serverdelen av kommunikationsservern. Klassen *ProjectHandler* tillhandahåller ett gränssnitt för att hantera uppdelningen av mätdata i projekt och experiment. Den har funktionalitet för att skriva och läsa till projekt/experiment. Klassen *ClientCallbackImpl* hanterar händelserna vid kommunikation mellan servern och klienten och är gränssnittet mot klienten. *SensorMinMax* heter klassen som läser in begränsningar på sensorers värden, dvs vilka värden som är för höga eller för låga på sensordata för att räknas som giltiga. *CombinedDataCollector* är klassen där logiken för att samordna alla subsystemen finns.

För back-enden gäller att varje sensorobjekt innehåller en tidsstämpel för att kunna sortera när mätdatan kom in, och ett namn för att lätt kunna identifiera vilken sensor som används. Vilken sorts data som finns i de olika sensorobjekten styrs av protokollet. Detta innebär att om någon sensor ska läggas till i systemet eller om någon information ska representeras på ett annat sätt så måste detta ändras i protokollet samt i de olika objekten.
Alla sensorer innehåller data som läggs in i arrayer med listor. Dessa kan sedan hämtas ut av front-end delen av applikationen och användas för att presentera mätdata.

En sensor är uppbyggd av generic message-klassen, som innehåller funktioner för att hämta ut sensorns olika attribut (exempelvis namn, status, mm) såsom de kommer från servern. Klassen innehåller ett egenskrivet read/write-lås då vissa resurser annars kan uppdateras samtidigt som läsningar sker.

En av de stora classerna i back-enden är ListenerHandler som hanterar alla lyssnare. Dessa lyssnare används för att notifiera front-end om att data har ändrats eller om klienten har fått ett svar från servern, vilket exempelvis kan ske om ett namn är upptaget vid skapande av projekt eller experiment.

Lyssnare anropas bland annat i klassen MessageBuilder som tar in det som kommer från servern och bygger upp ett meddelande som kan användas i back-end eller front-end beroende på syfte. Meddelandet är standardiserat efter kommunikationsprotokollet, mer om detta nedan, och består bland annat av en Enum sub_type som säger vilken typ av meddelande det är som ska hanteras. Beroende på detta skickas meddelandet till olika handlers som hanterar meddelandet på lämpligt sätt. Alla meddelanden som skickas

mellan server och klient är s.k. “General Message” vilket är ett generellt meddelande. Det finns alltså inte egentliga skräddarsydda meddelanden, utan de bara hanteras annorlunda beroende på innehåll.

En av back-endens viktigaste funktioner är att kunna ta mot information från servern. Detta sker genom ett gemensamt protokoll. Via protokollet skickas olika sorters information såsom felmeddelanden, generella meddelanden, serverns status och sensorinformation (se figur 3 och 4).
Front-end måste alltid kunna nå back-end, i nuläget tänkt via en statisk klass som kallas *frontendConnection.Backend*. Front-end får informationen från back-end som en lista av sensorer, som i sig innehåller listor av information. Om en sensors status ändras notifieras lyssnare.
Användaren ska kunna välja vilka sensorer som anses vara viktiga. Varje vald sensor kopplas till ett fragment som används för att visualisera dess information. Dessa fragment samt ett par andra såsom översikt och inställningar sparas och visas som en lista.

9.2 Operativ vy

Nedan följer de fysiska noder, processer och trådar som används av programmet.

9.2.1 Fysiska noder

Fysiskt består systemet av sensorer, en mätdator, en surfplatta och eventuellt en persondator.

9.2.2 Processer, trådar, komponenter

Mätservern består av en process; mätprogrammet och kommunikationsservern kör i samma process. Mätprogrammet har en tråd för varje sensor som sensordata ska inhämtas från samt en tråd för varje aktiverad tjänst. Varje sensortråd skickar mätdatan till två eller tre tjänster (beroende på vilka tjänster som aktiverats) som har till uppgift att både spara sensordatan till fil och skicka iväg samma data till kommunikationsservern. Kommunikationsservern använder sig av flera trådar i en trådpool enligt designmönstret *Proactor-pattern*, och sköter kommunikation med klienterna med de trådar som tilldelats.

Applikationen på surfplattan består av back-end och front-end-processerna, vilka är uppdelade i flera trådar. Back-end processen skapar en ny tråd för varje dataström från respektive sensor; detta för att man i körning ska kunna stänga av flödet från vissa sensorer utan att påverka de andra sensorerna.

I front-end kör gränssnittet på en tråd. Uppdatering av sensorers data görs i separata trådar, medan uppdatering av den visuella representationen sker i en tråd. Uppkopplingen till mätservern i back-end hanteras i en tråd.

10. Referenser

Nedan följer de referenser som använts i dokumentet, rubricerade efter typ.

10.1 Elektroniska källor

[1] OpenUp Architectural notebook (Eclipse public license)
http://epf.eclipse.org/wikis/openup/practice.tech.evolutionary_arch.base/guidances/templates/architecture_notebook_BCD3507B.html (Hämtad 2014-03-10, 13:30)