CHALMERS

Säker uppkoppling mot radiostyrda bilar

Att uppnå ett distribuerat system av javabaserade applikationer med hjälp av MQTT-protokollet.

Examensarbete inom Dataingenjörsprogrammet

BEHROZ KEYVANNIA
SOROUSH D.NEJAD

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

EXAMENSARBETE

Säker uppkoppling mot radiostyrda bilar

Behroz Keyvannia
Soroush D.Nejad

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2015

Säker uppkoppling mot radiostyrda bilar

Behroz Keyvannia
Soroush D.Nejad

Examinator: Lars Svensson

Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon: 031-772 1000

Institutionen för data- och informationsteknik
Göteborg 2015

Säker uppkoppling mot radiostyrda bilar

Behroz Keyvannia
Soroush D.Nejad
Institutionen för data- och informationsteknik, Chalmers Tekniska Högskola

Examensarbete

Sammanfattning

Idag ökar inte bara applikationsutveckling enbart för smarttelefoner, tablets och smart-TV utan även inom fordonsindustrin. Applikationsutvecklingen bidrar till interaktion mellan människa och fordon för att få bättre överblick över fordonet och omgivningen. Målet i detta projekt är att öka säkerheten för integrationen mellan utomstående applikationer med fordonets infotainment-system. Projektet avgränsar sig dock till att sända och mottaga signaler från en radiostyrd bil. Detta inkluderar en implementation av en mäklartjänst på radiostyrda bilen tillsammans från kommunikationen från utomstående applikationer. Ett distribuerat system erhålles och kommunikationen mellan alla parter skapades framgångsrikt. Alla noders kommunikation av det distribuerade systemet är testat och verifierat och är representativt exempel för säker uppkoppling inom fordonsindustrin.

Nyckelord: MQTT, Secure Gateway, Android, integration, Mosquitto, MOPED, Eclipse-Paho.

Abstract

The application development is a fast growing trend in the past few years. This, however, is not only associated with smartphones, tablets and smart-TV:s but with the automotive industry as well. The development of applications enhances human-vehicle interaction and provides a comprehensive overview of the vehicle and surroundings for the driver. The goal of this project is to increase the safety and security of the integration between external applications with the vehicle infotainment-system. The project scope is limited to send and receive signals from a radio-controlled car. We have successfully implemented a broker service on a test model radio-controlled car. This distributed system makes a secure communication from/to external applications possible. We have tested and verified connection and communication of the nodes. The result of this project can be used to build a secure connection for the automotive industry.

Keywords: MQTT, Secure Gateway, Android, integration, Mosquitto, MOPED, Eclipse-Paho.

Förord

Vi vill ge ett speciellt tack till vår handlare **Håkan Burden** för möjligheten att göra ett spännande examensarbete hos Viktoriainstitutet samt vår andre handledare **Sakib Sistek** för hans handledning och vägledning under projektets gång samt **Sebasthian Karlsson**, som tidigare jobbat med MOPEDerna, för hans hjälp och rådgivning.

Vi vill även tacka **Jakob Axelsson** och **Arndt Jonasson** på SICS Swedish ICT för deras arbete gentemot MOPEDerna samt för deras hjälp och vägledning vid påträffade svårigheter.

Tills sist vill vi visa våran tacksamhet för **Jonas Berg** på Semcon och **Niklas Mellegård** på Viktoriainstitutet för deras introduktion och ökad förståelse till ämnet.

Ordlista

AGA - Förkortning för "Automotive Grade Android". En plattform som möjliggör integration av en Android-applikation med signaler från ett fordon. Med AGA-plattformen kan man exempelvis avläsa signaler från fordonet.

API - Förkortning för "Application Program Interface". En regeluppsättning som beskriver hur en programvara samverkar med en annan programvara.

AUTOSAR - Förkortning för "AUTomotive Open System ARchitecture". En programvara riktat mot fordon som bjuder på styrenheter etc.

CAN - Förkortning till "Control Area Network". Detta är ett nätverk av kommunikation som används i riktiga fordon. CAN består av ett flertal noder som kommunicerar med varandra via en CAN-buss.

CarDriver - En klass i mjukvaran på MOPEDen. Sköter bland annat inkommande signaler från WirelessIno.

Eclipse-Paho - Ett open-source MQTT-klient-bibliotek med utvecklarverktyg, utvecklat av Eclipse. Biblioteket stödjer ett flertal programmeringsspråk.

IoT - Förkortning för "Internet of Things". Ett begrepp som syftar till fysiska maskinella komponenter som kopplar upp sig mot internet för att uppnå exempelvis en uppdatering.

Klienter - Klienter i MQTT-sammanhang är de som antingen tar rollen som prenumerant eller utgivare.

MOPED - Förkortning för "Mobile Open Platform for Experimental design of Cyper-physical Systems". En radiostyrd bil för utbildningssyfte bland annat studenter, hobbyister och doktorander.

Mosquitto - En open-source MQTT mäklartjänst.

MQTT - Förkortning för "Messaging Queue Telemetry Transport" som är ett kommunikationsprotokoll över TCP/IP. I protokollet agerar klienter som antingen prenumerant eller utgivare där kommunikation sker via en mäklartjänst.

Mäklare - En tjänst i MQTT-sammanhang som fungerar likt en server. Mäklaren registrerar intresseområden hos prenumeranter och skickar vidare utgivarens meddelanden till rätt prenumerant.

Open-source - Svenskans "Öppen Källkod", programkod som är öppen för allmänheten att läsa, använda, modifiera och vidaredistribuera.

Product backlog - Detta är ett Scrum-begrepp och syftar till en anslagstavla där man skriver ner sina arbetsuppgifter och prioriterar i ordning, för att utvecklare ska kunna ta en uppgift som finns på anslagstavlan. Målet är att alla uppgifter blir gjorda.

PWM-signal - En pulsbreddssignal. Signalen är en typ av signal där spänningen slås på och av snabbare än vad en maskinella kompontenter som exempelvis en motor kan urskilja.

Rasbian Wheezy - Ett operativsystem med en kärna av Debian[18], utvecklat specifikt för Raspberry Pi. Wheezy är versionsnamnet.

SCU - Förkortning för "Sensor Control Unit". En av noderna i det distribuerade datorsystemet på MOPEDen. Noden är en Raspberry Pi som kör AUTOSAR och har bland annat ansvar att samla in data via inkopplande sensorer.

SDK- förkortning för "Software Development Kit". En uppsättning utvecklingsverktyg för att förenkla programutveckling.

Secure Gateway - Detta är ett begrepp som syftar till en säker anslutning till infotainment-system i ett fordon. Secure Gateway är ett koncept för säker kommunikation mellan noder i bilens egna IP-nätverk.

Socket - En anslutning, metaforiskt en tunnel som en programvara skapar till en annan programvara för att skicka och ta emot data.

TCU - Förkortning för "Telematics Control Unit". En av noderna i det distribuerade datorsystemet på MOPEDen. Noden är en Raspberry Pi som kör Rasbian Wheezy och har bland annat ansvar över att sköta den externa kommunikationen.

VCU - Förkortning för "Vehicle Control Unit". En av noderna i det distribuerade datorsystemet på MOPEDen. Noden är en Raspberry Pi som kör AUTOSAR och ansvarar bland annat för att emot börvärden från TCU och skicka PWM-signaler till motor och styrdon.

WirelessIno - En styrapplikation utvecklad av SICS för MOPEDen. Möjligheten att välja hastighet och styrning för att styra MOPEDen.

Innehållsförteckning

1

Introduktion

Detta kapitel täcker introduktionen av examensarbetet. Kapitlet innehåller en inledning som nämner huvudsyftet i detta projekt samt använda tekniker. Kapitlets resterande delkapitel innehåller problembeskrivning, syftet och målet med projektet. Kapitlet avlutas med vad projektet avgränsar sig till.

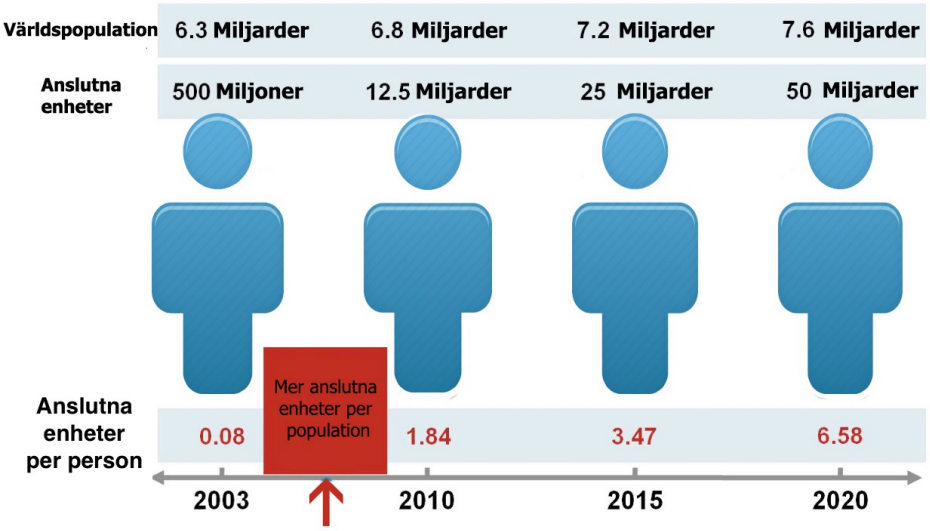
1.1 Inledning

Tekniken spelar stor roll i Sverige. Fordonsindustrin har under många år haft en god utvecklingskurva, både i och utanför Sverige. Utvecklingen av fordonsindustrin beror väldigt mycket på utvecklingen av mjukvara och hårdvara. Vi ser positiva utvecklingskurvor konstant gällande programmeringsspråk, som Java exempelvis, i form av effektivisering av prestanda och kvalité. Men även inbyggda system, distribuerade system, plattformar (som exempelvis Android) och integration mellan olika tekniker som gör att man kan få fördelarna från många olika tekniker.

I det dagliga hör man även om nya begrepp som tar fart, som exempelvis IoT, Internet of Things, som är ett exempel på distribuerade system. På Ciscos räknare av uppkopplade enheter [1] kan man se antalet uppkopplade enheter i dagsläget, för att få en känsla om hur populärt det börjar bli med flera enheter som håller en konstant kommunikation. Figur 1.1 visar antalet enheter per person som har ökat och fortsätter att öka. Kommunikation mellan en användares alla enheter ger ökat värde för användaren, speciellt när enheterna är synkroniserade med varandra.

I detta examensarbetet har vi fokuserat på implementation av en säker metod för uppkoppling mot fordon. Problemen som vi söker svar på är huruvida det är möjligt att koppla upp sig mot ett fordon genom ett specifikt valt protokoll med en handhållen Android-enhet, på ett säkert sätt. Detta examensarbete som framgår, innefattar en vidareutveckling av ett tidigare projekt.

I vår utveckling kommer vi att använda ett protokoll som heter MQTT, Message Queuing Telemetry Transport. För testning av implementationer kommer vi använda oss av en radiostyrd bil som förkortas



Figur 1.1 - En uppskattning av antal enheter per person [2]

MOPED (Mobile Open Platform for Experimental Design of Cyber-Physical Systems).
Mjukvaran som används på MOPEDen är skriven i Java och på de handhållna enheterna är applikationerna utvecklade i Android.

Detta examensarbete utförs i samarbete med Viktoriainstitutet och är ett delprojekt av framtida projekt som utförs på MOPEDerna, för att visa upp visionen som Viktoriainstitutet har för den framtida fordonsindustrin. Vilken vision Viktoriainstitutet har för MOPEDerna framgår i kapitel **1.3 Syfte**.

Det är två projektgrupper som gör examensarbete parallellt på MOPEDerna. Tanken är att dessa två examensarbeten ska slås ihop för att uppnå ett större resultat. Det större resultatet är att kunna utveckla applikationer med AGA-plattformen med en säker uppkoppling med hjälp av Secure Gateway. Rapporten för andra examensarbetet heter ”Applikationsutveckling för radiostyrda bilar” [3].

1.2 Problembeskrivning

I och med den snabba utvecklingen av fordonsindustrin reduceras kvalitén ifall det inte finns säkerhet implementerat. I dagsläget finns det ett sätt att ansluta en Android-applikation till MOPEDen. Anslutningen sker i ett lokalt nätverk och signalen går direkt till MOPEDen. Det problematiska med detta är att i anslutningen finns varken struktur eller säkerhet implementerat. Detta betyder att flera klienter kan ansluta och styra MOPEDen, så länge klienten har ip-adress och portnummer till MOPEDen. Ett behov av ett strukturerat och säkert sätt för att koppla upp sig till MOPEDen finns.

1.3 Syfte

En vision med MOPED är att den ska vara ett bra testverktyg för att kunna testa fordonsanpassade applikationer. MOPED som ska efterlikna mjukvaran på en riktigt fordon fast på ett kostnadseffektivt sätt, kan ge möjligheten att upptäcka felaktigt beteende eller kunskapsläckor när man vill integrera fordonsanpassade applikationer med ett fordons infotainment-system.

Syftet med detta projekt är att vid utveckling av nya applikationer för fordon säkerställa att ingen läckage av känslig information för obehöriga klienter sker. Detta är huvudsakliga konceptet med Secure Gateway som erbjuder en lösning för säkerheten mellan noder i ett fordons IP-nätverk.

I Secure Gateway är det infotainment-systemet i ett fordon som är i fokus. Konceptet är att göra en gateway för applikationer som integreras med ett fordons infotainment-system och uppgiften för denna gateway är att hindra applikationer att nå ut till känslig data i ett fordon.

I ett långsiktigt perspektiv medför detta projekt ett signifikant värde till visionen som erhålls av Viktoriainstitutet och organisationer som står bakom skapandet och underhållning av MOPEDerna.

1.4 Mål

Vid ett lyckat arbete, uppnås målet att öppna dörrarna för en säker uppkoppling och integration mellan applikation och MOPED. Detta kommer i sin tur att ge möjlighet att installera plattformar som AGA, för att kunna utveckla fordonsanpassade applikationer för att integrera dessa på ett säkert och strukturerat sätt med MOPEDen.

Målet med detta projekt är att implementera ett säkert sätt för att koppla upp sig mot MOPEDen. Implementationer ska göras med MQTT-protokollet vilket innefattar en klient-mäklartjänst modell. Mäklaren kommer att vara installerad på själva MOPEDen vilket gör att all kommunikation till MOPEDen kommer ske via denna mäklartjänst.

Målet för projektet är att kunna skicka signaler från styrapplikationen WirelessIno för att kunna köra MOPEDen enligt önskemål samt att kunna få själva MOPEDen att skicka ut signaler om information som exempelvis hastighet, för att intresserade klienter ska kunna utnyttja denna informationen för att åstadkomma något. För att fånga upp signalen är tanken att det ska finnas en annan Android-applikation, som visualiserar hastigheten på skärmen.

Efter att implementationer av MQTT-protokollet är genomfört leder detta till att börja sätta säkerhet på strukturen. Med att sätta säkerhet menas exempelvis att implementera användarnamn och lösenord vid uppkoppling till MOPEDen så att bara en klient ska kunna styra MOPEDen åt gången. En annan säkerhet är även att sätta spärr för obehöriga klienter att få ut känslig information från MOPEDen.

1.5 Avgränsningar

För detta projekt har vi tagit emot en MOPED från tidigare projektgrupp samt en befintlig Android-applikation som SICS [4] har utvecklat. Arbetsuppgiften i detta projekt är som tidigare nämnts att implementera Secure Gateway för MOPEDen och dess Android-applikationen som den ska integreras med. Därav kommer vi enbart påverka anslutningen mellan MOPEDen, styrapplikationen och applikationen för hastighetsvisning. Våra ändringar kommer innebära att implementera en mäklare och att skapa och anpassa klienter som ska kunna kommunicera via mäklaren. Projektet avgränsar sig således till att implementera detta, som också kallas för Secure Gateway.

2

Teknisk och teoretisk bakgrund

I följande kapitel ges bakgrunden till tekniker som projektet bygger på. Här nämns vad MQTT är, vilket hela projektet bygger på samt vilka andra tekniker som använts för att implementera Secure Gateway. Kapitlet avslutas med bakgrunden till MOPEDens uppbyggnad samt vad WirelessIno är för något.

2.1 MQTT

MQTT är ett lättviktigt kommunikationsprotokoll som vanligen används för machine-2-machine (M2M) och Internet-of-Things[5] (IoT) sammanhang. Kommunikationen i MQTT-protokollet sker som klienter, uppdelade som prenumeranter och utgivare via en server som i MQTT-sammanhang kallas för Mäklare (engelskans ”broker”).

MQTT skapades för system som ska vara uppkopplade som bland annat ett distribuerat system. På grund av det är MQTT lättviktigt och följer en arkitektur som möjliggör minimal bandbredd vilket passar bra för maskinella komponenter.

MQTT används idag av bland annat sjukvården och övervakning över rörledningar i energifabriker men även Facebook-Messanger[6]. Tack vare snabbheten och anpassningen i MQTT, har den en bred användning i många områden.

MQTT har bra stöd och utrustning för diverse användning. Några nämnvärda är Quality of Service (QoS), återanslutning till mäklaren (**2.1.2 Mäklare**) samt krypterad anslutning via SSL/TLS[7].

Enligt en jämförelse mellan HTTP och MQTT har man kommit fram till att MQTT har 96 gånger mer mobil genomströmning, 11 gånger mindre batteriförbrukning för att skicka data, 170 gånger mindre batteriförbrukning för att ta emot data och 8 gånger mindre trådlöst bandbredd[8]. HTTP är mest känd för datakommunikation på webben och det mest förekommande protokollet för en klient-server modell. Att jämföra MQTT med HTTP ger en bild av vilka fördelar MQTT har gentemot HTTP. Jämförelsen har skett med en HTC Desire med en Android version på 2.2.2 och med ett Li-Ion batteri 1400 mAh och 3.7 volt.

På senare tid har MQTT blivit en Oasis standard för strukturerad information [9].

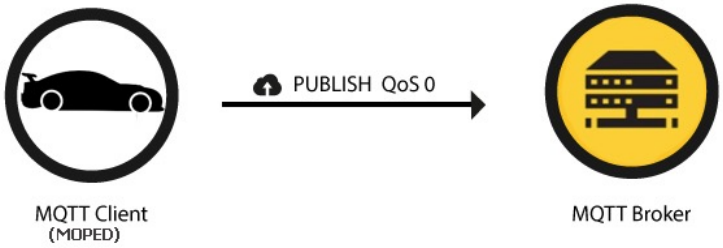
2.1.1 Quality of Service, QoS

QoS är en grad av överenskommelse mellan sändare och mottagare gällande garantier för leverans av meddelanden. MQTT har tre QoS-grader som är benämnda 0, 1 och 2 där de i ordning betyder ”Som mest en gång”, ”Som minst en gång” och ”exakt en gång”. Denna grad kan markeras på meddelandet som skickas mellan klienter och mäklare för att det gör kommunikationen mer pålitlig och ger en chans för omsändning vid paketförluster.

Dessa tre olika grader av QoS beskrivs ytligt nedan. För mer information om QoS, se bifogad referens. [10]

QoS 0 - ”Som mest en gång”

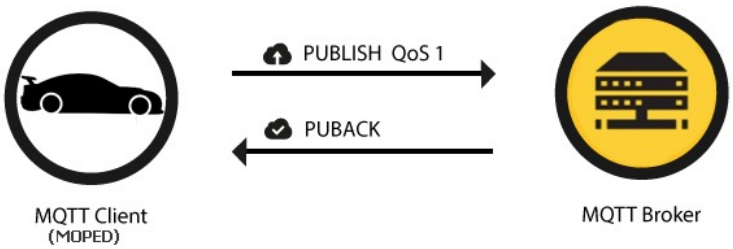
Denna grad brukar kallas för ”fire and forget” eftersom graden inte säkrar någon garanti för leverans av meddelanden, alltså fås ingen bekräftelse från mottagaren (Se figur 2.1). Den kan dock garantera den snabbaste och enklaste sändningsprocessen jämfört med de två graderna av QoS.



Figur 2.1 - QoS-grad 0

QoS 1 - ”Som minst en gång”

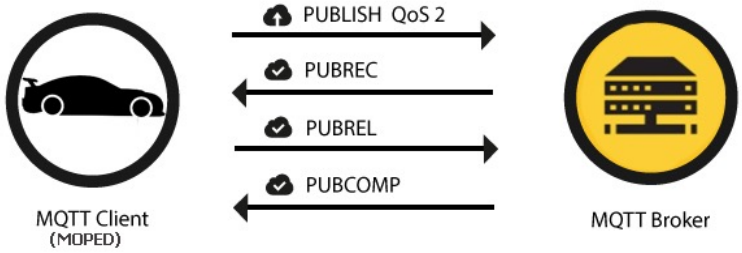
Denna grad av QoS kan garantera att meddelanden tas emot från sändaren minst en gång. Det kan dock vara mer än en gång också. Sändaren sparar meddelandet tills den får bekräftelse från mottagaren. Det kan hända att bekräftelsen aldrig kommer fram till sändaren och då agerar sändaren i QoS 1 att meddelandet skickas om på nytt efter ett visst tidsintervall. Se figur 2.2.



Figur 2.2 - QoS-grad 1

QoS 2 - Exakt en gång

Denna grad kan garantera att meddelanden tas emot från sändaren exakt en gång. Sändaren skickar meddelandet, som mottagaren bekräftar. Men sändaren bekräftar även bekräftelsen och sedan släpper ansvaret. I slutlig fas avslutar mottagaren med en lyckad sändning, se figur 2.3. Det är den mest pålitliga men dock den mest tidskrävande sändningsprocessen jämfört med de två andra QoS-graderna.



Figur 2.3 - QoS-grad 2

2.1.2 Klienter

Klienter i MQTT-strukturen är de som kan agera som prenumerant och utgivare, likt en prenumeration på en tidning. Istället för tidningar i detta sammanhang gäller prenumerationen på olika typer av ”ämnen” (engelskans ”topics”). Klienter med intresse inom alla publikationer från MOPEDen, kan prenumerera på ämnet ”MOPED”. De klienter som enbart är intresserade av MOPEDens hastighet kan prenumerera på ”MOPED/speed” (svenskans ”hastighet”) vilket hänvisar till MOPEDens nuvarande hastighet. Med dessa prenumerationer på olika ämnen kan man sedan sätta i större sammanhang och få en kommunikation.

De klienter som vill agera som utgivare behöver välja ett visst ämne för att publicera meddelandet på. Ett exempel är en Android-applikation som ska fungera som en styrapplikation för MOPEDen. Applikationen uppdaterar MOPEDens vetskap om vilken riktning den ska köra genom att publicera ett meddelande på ämnet ”MOPED/steer” (svenskans ”riktning”) med innehållet ”20 grader till höger” vilket får MOPEDen att svänga enligt meddelandet.

Det kan finnas flera klienter i en applikation och hur många klienter som finns beror på behovet av information och hur man väljer att designa en applikation. MQTT-biblioteket för klienter som vi har använt i detta projekt är Eclipse-Paho, vilket är en open-source variant[11]. Eclipse-Paho är ett bibliotek för klientsidan och har utbud för bland annat programmeringsspråk som C/C++, Java, JavaScript och Python.

2.1.3 Mäklare

En mäklare i MQTT-sammanhang kan liknas en tjänst på en server. Ansvaret som mäklaren har är att registrera klienter samt deras intresseområden i form av ämnen. När nyheter publiceras gällande ett visst ämne, exempelvis ny hastighet, så skickas meddelandet till alla klienter som är intresserade av ämnet. Som tidigare nämnts kan ett ämne vara ”MOPED/speed”. Om en klient har en prenumeration på ämnet ”MOPED/speed” kommer denna klient att uppdateras, via mäklaren, varje gång nyheter gällande ämnet uppstår. Mäklarens uppgift är med andra ord att leverera rätt meddelande till rätt klienter, att länka samman prenumeranter och utgivare.

MQTT har ett flertal anpassade mäklartjänster. Bland annat finns: Mosca (node.js, Javascript), Emqttd (Erlang) och Mosquitto (Java). I detta projekt används Mosquitto vilket är en open-source variant[12].

Med Mosquitto följer vissa säkerhetsfunktioner. Efter installationen av Mosquitto kan man konfigurera inställningarna enligt önskemål. Mosquitto använder sig av sina standardvärden ifall den inte hittar en konfigurationsfil angivet. Kort beskrivet är inställningarna för standardvärden att klienter kan ansluta sig till mäklaren och prenumerera på vilket ämne som helst. Det finns alltså ingen säkerhet av något slag, bara ett öppet fält.

I Mosquittos konfigurationsfil ”mosquitto.conf”[13] (Linux-miljö) finns ett par rader gällande inställningar. Där finns bland annat ”allow-anonymous”, som lägger spärr för anonyma klienter ifall inställningen är på ”false”. För att nämna ett par andra inställningar så finns bland annat: ”username”, ”password”, ”port”, ”max_connections”, ”logging” (för felsökning) och ”use_identity_as_username”. Ett exempel på konfigurationsfilen ”mosquitto.conf” kan hittas i följande referens[14].

2.2 Observer-designmönster

Observer är ett designmönster som används för att ha ett strukturerat sätt att iaktta förändringar i olika objekttillstånd. Själva designmönstret består av ett flertal olika ”karaktärer” som intar roller när man implementerar mönstret.

Dessa karaktärer är observatör (engelskans ”observers”) och observerbar (engelskans ”observable”). Mer förenklat är det ett objekt *X* som blir iakttaget för förändring och då kallas *X* för observerbar medan de som är intresserade av förändring av *X* kallas för observatör.

Designmönstret används oftast för grafiska komponenter och då brukar själva designmönstret ingå i termen ”Model-View-Controller, MVC”.[15]

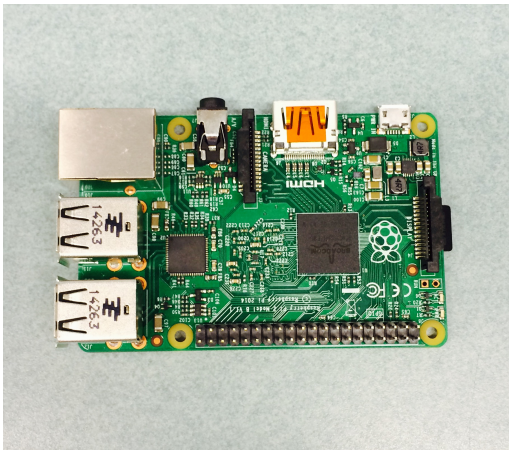
Observer-designmönstret är användbart när man har ett visst objekt *X* och är intresserad av förändringar i objektets tillstånd. När objektet *X* får någon förändring, uppdaterar den alla lyssnare (observers) om sitt nya tillstånd.

2.3 Raspberry Pi

Raspberry Pi (se figur 2.4) är en enkortsdator som är utvecklat av Raspberry Pi Foundation[16]. Själva enkortsdatorn har nästintill allt en vanlig PC har dock i mindre format och i storleken av ett kreditkort. Den finns i ett par olika modeller men den modell som används i detta projekt är B+.

Raspberry Pi använder sig av microSD-kort istället för en hårddisk. MicroSD-kortet använder man för att installera ett operativsystem på. Operativsystemet som används i detta projekt är Linux-versionen Rasbian Wheezy[18]. Raspberry Pi använder sig av en Broadcom-processor med ARM-arkitektur vilket är samma processor som många smarttelefoner använder sig av[19].

Gällande I/O har Raspberry Pi B+ fyra st USB-portar, en Ethernet-port, HDMI-uttag, AUX-port samt micro-USB för strömförsörjning.



Figur 2.4 Raspberry Pi [17]

Raspberry Pi är tillverkad för utbildningssyfte och har blivit väldigt populär bland studenter och hobbyister. Liksom andra Linuxsystem kan den användas för exempelvis skicka e-post, surfa på nätet eller fungera som en webbserver.

2.4 MOPED

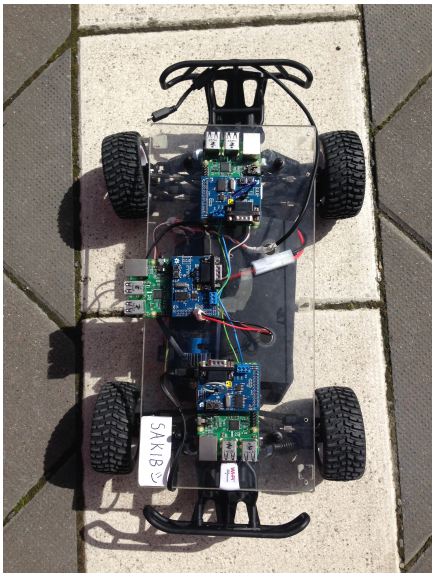
MOPEDen är utvecklad av SICS Swedish ICT. MOPEDen är tillverkad för utbildningssyfte och har som mål att efterlikna industrinära mjukvara och hårdvara. Med hjälp av MOPEDen kan studenter och forskare testa sina system eller mjukvara utan att lägga en stor summa pengar på utrustning samt minimera resursanvändningen.

Eftersom MOPEDens distribuerade datorsystem består av Raspberry Pi enheter kan MOPEDen ha trådat nätverk genom Ethernet-porten. Dock är Raspberry Pi även öppen för trådlöst nätverk genom en WiPi-dongel (figur 2.5), som är kompatibel med Raspberry Pi. WiPi använder den senaste trådlösa 802.11n-tekniken och kan stödja datahastigheter på upp till 150 Mbps. Fördelen med att använda WiPi-dongel för detta projekt är att WiPi-dongel har inbyggt stöd för Rasbian Wheezy operativsystemet vilket gör att man får funktionalitet redan vid insättning av dongeln. Frekvensområdet för WiPi-dongel är 2.4 till 2.48 GHz och har en uteffekt på max 20 dBm [20].

Figur 2.5 - WiPi-dongel

2.4.1 Uppbyggnad

MOPED är en förkortning på Mobile Open Platform for Experimental Design of Cyber-Physical Systems.
MOPEDens distribuerade datorsystem består av tre Raspberry Pi enheter som formar ett distribuerat system. Dessa noder har likartade namn: TCU (Telematics Control Unit), VCU (Vehicle Control Unit), SCU (Sensor Control Unit). Alla noder är kopplade till varandra genom ett CAN-nätverk likt strukturen på ett fordon. I figur 2.6 visas en bild på MOPEDen.



Figur 2.6 - Uppbyggnad av MOPED

TCU

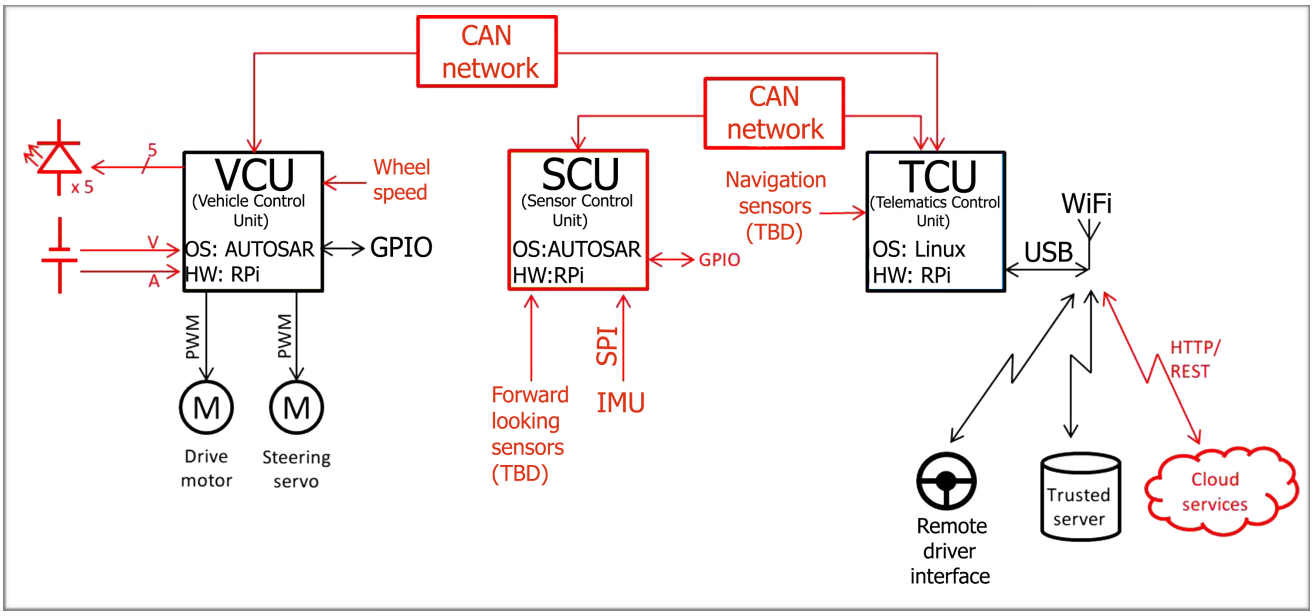
Noden som kallas TCU är den enhet som vi kommer påverka under detta projekt. TCU-enheten har ansvar för att sköta den externa kommunikationen med protokoll som HTTP, REST och MQTT, för att ta emot styrsignaler från Android-applikationen WirelessIno (se **2.5 WirelessIno**) och koppla upp sig mot SICS server för uppdateringar för mjukvara. TCU har kommunikation med VCU-enheten och SCU-

enheten dock har VCU-enheten och SCU-enheten ingen kommunikation med varandra, se figur 2.7 - Uppbyggnad av MOPED. Vid mottagande av styr- och hastighetssignaler, har TCU som ansvar att skicka vidare detta till VCU-enheten.

Operativsystemet hos TCU-enheten är Rasbian Wheezy (Linux). TCU är noden som har Mosquitto-mäklaren installerad. Det är alltså IP-adressen och port 1883 till TCU som öppnar kommunikation med mäklaren.

VCU/SCU

Eftersom implementationer från detta projekt inte påverkar VCU-enheten och SCU-enheten beskrivs dessa noder väldigt kort. Den intresserade läsaren hänvisas till vetenskapliga artiklar kring MOPEDerna[21]. Som tidigare nämnts har inte VCU-enheten och SCU-enheten någon kommunikation med varandra. VCU-enheten sköter kommunikationen med motorn och styrservon medan SCU-enheten är till för att koppla in sensorer och läsa av olika former av mätvärden. Se figur 2.7 för en överblick över MOPEDens struktur. Båda noderna har AUTOSAR [22] som operativsystem och Raspberry Pi som hårdvara.



Figur 2.7 - Uppbyggnad av MOPEDen [23]

2.4.2 Mjukvara

Mjukvaran i MOPEDen är utvecklad av SICS och själva MOPEDen tillsammans med sin styrapplikation (WirelessIno) fanns befintlig innan projektets början. Eftersom programvaran till MOPEDen är stor och mestadels utanför projektets ramar, fokuseras bakgrunden på den delen av programvaran som påverkats under projektets implementationer. Den intresserade läsaren hänvisas till SICS github repository[24] för att få en bättre överblick av hela programvaran.

I TCU-enhetens mjukvara, som heter Ecm-Linux, finns det en Java-klass som tar emot signaler från styrapplikationen WirelessIno. Denna Java-klass heter CarDriver.java och kan hittas i SICS github repository i [24].

Klassens viktigaste funktion är att invänta externa signaler från WirelessIno genom ett lokalt nätverk för att sedan skicka vidare den inkommande informationen. Informationen skickas till VCU-enheten för att få önskad effekt på MOPEDen gällande hastighet och styrning. CarDriver.java skapar en bakgrundstråd för att

parallellt med huvudtråden kunna invänta signaler på hastigheten och styrning för att senare vidarebefordra informationen för att få en önskad finkänslighet under körning.

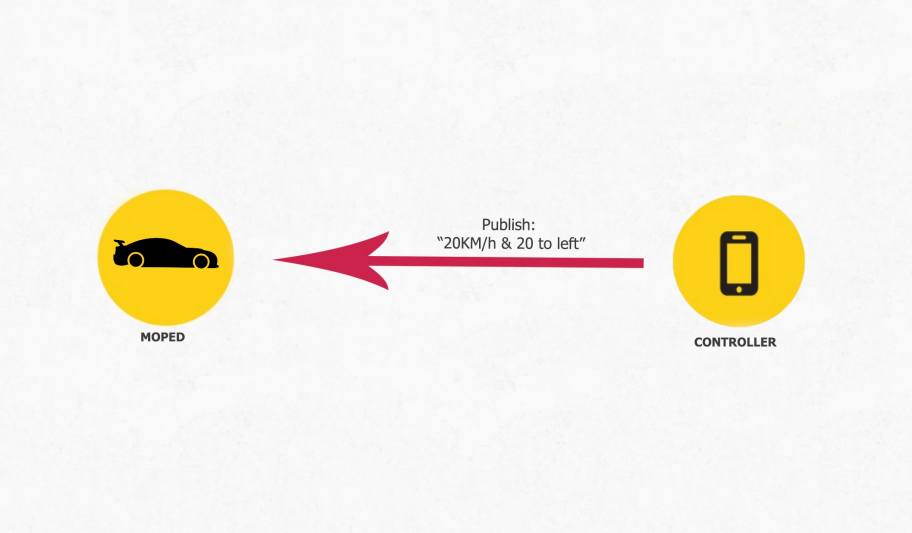
Den inkommande signalen översätts till PWM-signal, vilket en typ av signal där spänning slås på och av snabbare än vad den motordrivna komponenten kan urskilja.

2.5 WirelessIno

WirelessIno fungerar som en styrenhet till MOPEDen där man kan justera hastighet och riktning på MOPEDen enligt önskemål.

Användargränssnittet tillhandahåller en höger- och vänsterspalt där man kan föra en markör från 0 till 100 och 0 till -100 beroende på om man vill köra MOPEDen framåt eller bakåt. Den vänstra spalten, vilket ligger i en vågrätt positionering, tillåter riktning till höger med värden från 0 till 100 och vänster för 0 till -100. För att se hur användargränssnittet på WirelessIno ser ut, se figur 5.3.

Med WirelessIno anslutet till MOPEDens IP-adress, kan WirelessIno ansluta sig till MOPEDens IP-adress och port 9000. Vid lyckad anslutning får man en blå punkt på gränssnittet där det står ”Connected”. Då kan WirelessIno direkt skicka styr- och hastighetsignaler till TCU, se figur 2.8.



Figur 2.8 - Tidigare kopplingen mellan WirelessIno och MOPED

Ansvariga klasser för ovannämnda processen är PadView.java och Main.java. PadView.java är klassen som ritar upp gränssnittet och iakttar börvärden för hastighet och riktning medan Main.java skickar iväg informationen via nätverket, via en socket. En socket kan liknas en tunnel mellan WirelessIno och MOPEDen, där det skickas styrsignaler från WirelessIno till MOPEDen.

3

Metod

Det här är ett projekt inom en vetenskaplig metod som heter Design Science Research, DSR, [25] där vi har använt oss av en Scrum [26] inspirerad arbetsmetodik. I DSR är det centrala en design även nämnd artefakt. I designen följer ett par implementationer där genomförandet beskrivs i nästkommande kapitel samt vad resultatet av designen blev på kapitel 5.

3.1 Arbetssätt

Arbetssättet i projektet har varit en blandning av olika arbetssätt men Scrum-inspirerat. Vi har använt oss av en product backlog där vi har skrivit ner arbetsuppgifter och organiserat dem enligt prioriteringsordning. Webb applikationen som vi har använt oss av heter Trello[27].

Under projektets gång har det varit mycket parprogrammering med stegvis testning efter varje implementation. Vi började med att göra en testapplikation med MQTT för att bli mer insatta i hur det fungerar och sedan, med kunskap, skriva själva anropen som görs i MOPEDen, WirelessIno och Dashboard. Syftet med testapplikationen var även att veta att designen fungerar. På så sätt sparade vi tid på felsökningen och det gav bättre överblick av vart vi bör söka efter fel.

Parallellt med utvecklingen har vi fört en loggbok där vi dokumenterat utveckling på löpande band. I loggboken har nyttiga kommandon skrivits ner och gett en överblick för handledare om hur utvecklingen går. Verktyg för loggboken har varit Google docs.

3.2 Förutsedda uppgifter

I första stadiet för projektet planerade vi att ägna tid åt att bekanta oss med MOPEDens mjukvara och hur det är att köra den. Genom att bekanta oss med MOPEDens mjukvara, var det är lämpligt att rita upp ett

UML-diagram eftersom att detta kan hjälpa oss och nästkommande projektgrupp som vill arbeta med MOPEDerna för bekantskap med MOPEDerna.

Innan själva utvecklingen skapades en design enligt DSR. I designen bestämde vi oss för att ha vilka förutsättningar och tekniker som krävs för att få vår idé att fungera. Enligt våra bestämmelser så krävdes en Mosquitto-mäklartjänst och tre klientsidor för MOPEDen, WirelessIno- och Dashboard-applikationen.

Utöver dessa tekniker krävdes vissa förutsättningar. Till vår användning krävdes en mobil-hotspot, en WiPi-dongel och två Android-enheter för att skapa det distribuerade systemet enligt vår design.

4

Genomförande

I följande kapitel beskrivs genomförandet på MOPEDen, WirelessIno- och Dashboard-applikationen. Kapitlet börjar med implementationer på MOPED med hänvisningar till kodexempel i Appendix . Kapitlet fortsätter med klientsidan och avslutas med nätverksuppsättningen.

4.1 Implementationer på MOPED

MOPEDen bär på den mesta av implementationer som vi har gjort i detta projekt. Detta delkapitel är uppdelat i installation, konfiguration och testning av Mosquitto-mäklaren. Detta följs av MOPEDens klientsida gentemot mäklaren.

Efter alla våra implementationer har vi säkerhetskopierat operativsystemet på MOPEDen i form av en image fil. Denna image är tagen från SICS image[28] och kombinerats med våra implementationer. Där image filen kan hämtas från SICS hemsida, följer även instruktioner för installation av image. Vår egna image fil kan hittas i följande referens. [29]

4.1.1 Mosquitto

Installation

Vi började med att installera Mosquitto-mäklaren på TCU-enheten. Installationen är lik andra installationer på Linux-miljöer. Det praktiska genomförandet av installation av Mosquitto på TCU-enheten kan hittas i **Appendix A.4**.

Konfiguration

Efter installationen bör man konfigurera mäklaren utifrån önskemål. Konfigurationen är nödvändig för att ha exempelvis användarnamn, lösenord och övriga säkerhetsverktyg som Mosquitto har stöd för. Våra konfigurationer för detta projekt kan hittas i **Appendix A.5.**

Testning

För att testa Mosquitto behöver man inte skapa ett Java-program. Med Mosquitto följer ett terminalprogram som heter ”Mosquitto Client Service” där man kan sätta sig in i MQTT-protokollet genom att att vara prenumerant eller utgivare med enkla kommandon. Se vidare i **Appendix A.6**.

4.1.2 Klientsida

Implementationer för klientsidan på MOPEDens mjukvara görs genom användning av Eclipse-Paho-biblioteket (Se 2.1.1 Klienter). Anrop från Java beskrivs i **Appendix A.7**. Varje klient behöver först identifiera sig genom klient-ID och inloggningsuppgifter ifall detta är konfigurerat på mäklaren. Ifall identifieringen går bra, kan klienten hålla en konstant anslutning (engelskans ”connection”) för att kunna skicka och ta emot meddelanden från mäklaren. Hur anslutningen ser ut är upp till hur man vill att klienten ska agera. Med detta menas att man kan välja ifall man vill koppla ner efter att meddelandet är sänt eller mottaget eller fortsätta hålla anslutningen till mäklaren. Möjligheten finns även att skapa en återanslutnings mekanism som ansvarar över att återansluta ifall klienten skulle tappa anslutningen till mäklaren.

Eclipse-Paho stödjer flera andra programmeringsspråk som vi inte har undersökt närmare.

Klientsidan med MQTT-implementationen är generell för klienter vilket betyder att MOPEDen, WirelessIno- och Dashboard-applikationen har en klientsida som är snarlika med varandra.

Observer implementation

På klientsidan på MOPEDen har vi implementerat Observer-mönstret för effektiv uppdatering av variablers nya tillstånd. När WirelessIno publicerar ny data gällande exempelvis hastighet och riktning, kan olika instanser i MOPEDen få notis om detta.

För objekt som vill anmäla sitt intresse för ett objekt finns en metod som heter ”addObserver()”. Själva objektet *X* upprätthåller en lista på vilka som är intresserade av förändringar av tillstånd och löper igenom listan och informerar om förändringen genom att använda en metod som heter ”notify()” (svenskans ”meddela”). Med metoden ”notify()” skickas parametrar som innehåller förändringen i objektet *X* och mottagande intresserade objekt uppdaterar sin information angående objektet *X*.

Ett annat sätt är att det intresserade objektet anropar en annan metod som heter ”read()” (svenskans ”avläs”) i vilket det intresserade objektet själv söker efter förändring hos objektet *X* [15].

Samma implementationer har gjorts på Dashboard-applikationen.

4.2 Implementationer på WirelessIno

Som nämnts i **kapitel 2.4 - MOPED** så är applikationen tidigare utvecklad av SICS och fanns befintlig vid projektets början. Klientsidan som beskrivs i **4.1.2 Klientsida**, gäller för WirelessIno också. WirelessIno är en utgivare som skickar meddelanden om hastighet och riktning vilket MOPEDen är en prenumerant på därmed tar till sig informationen.

Ändringen som gjorts i anropet är att vi har utgivare klass som heter MqttPublisher.java. WirelessIno publicerar sina meddelanden på ämnet ”MOPED/speed” och ”MOPED/steer”. Detta har vi gjort genom att göra anrop enligt **Appendix A.7**.

Med MQTT-strukturen i WirelessIno, har vi döpt om applikationen till MqttWirelessIno, för att minimera risk för förvirring mellan versionerna.

4.3 Dashboard-applikationen

Dashboard-applikationen är en Android-applikation som visualiserar MOPEDens publikationer om hastighet. Dashboard-applikationen är prenumerant på ämnet ”MOPED/speed” vilket betyder att mäklaren kommer skicka ut nyheter gällande hastigheten hos MOPEDen, mot rätt användarnamn och lösenord.

Dashboard-applikationen har samma struktur som klientsidan hos WirelessIno och MOPEDen. Med detta menas MQTT som klientsida och Observer-mönstret för uppdateringar av variabler.

4.4 Testning av systemet

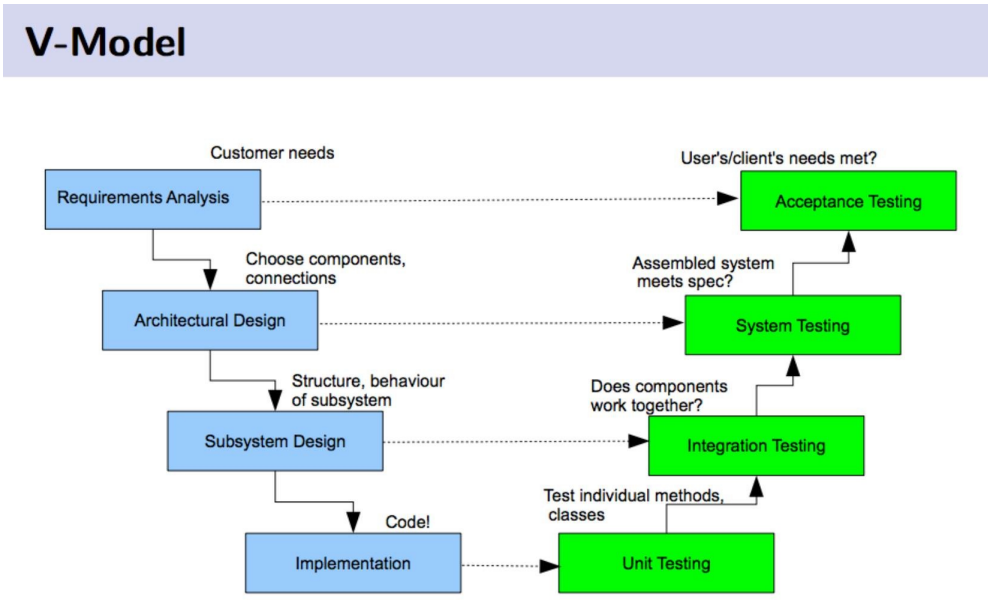
Testning av systemet har skett stegvis efter varje ändring. Testningen är likt V-modellen i testningssammanhang, se figur 4.1. Vi har skrivit unit-testning för olika instanser av objekt för att få önskade värden på olika variabler.

En annan testning av systemet är integrationstestning, som finns med på figur 4.1. Där har vi testat varje applikation enskilt. Applikationerna som har blivit testade för sig självt är MqttWirelessIno, Dashboard och MOPEDens klientsida. Sedan har vi även testat mäklaren enskilt. Detta är gjort i syfte att kunna göra ett systemtest.

Med integrationstestning öppnas dörrarna för systemtestning, enligt figur 4.1. Där testas hur alla applikationer och komponenter fungerar tillsammans och ifall man inte får önskad effekt, vet man vart man ska felsöka.

I slutfasen av testningen är det acceptanstestning. Med detta menas att de förväntningar beställaren har haft på systemet sedan början och som stämmer överens med examensarbetets beskrivning och planeringsrapport, är resultatet av testningen.

Sammanfattningsvis har testerna skett via V-modellen, enligt figuren. Med detta menas att vi skrev ner behovet för att sedan övergå till att göra en förstudie av befintlig programkod för att sedan göra en design och implementera denna. I ett senare skede är man över på den gröna delen av figur 4.1 vilket är enbart testning av klasser som sedan sätts i större sammanhang som applikation, integration mellan applikationer, systemet totalt för att sedan jämföra med det nedskrivna behovet.



Figur 4.1 - V-Modellen [30]

4.5 Nätverk

Tillgång till ett flertal routrar och hotspots har funnits under projektets gång. Det har varit en Netgear router (tråd och trådlöst), Chalmers trådlösa nätverk, två hotspots på två olika Android-enheter samt en tablet-enhet. Eftersom vi har valt att kommunikationen mellan enheterna ska vara krypterat över TCP/IP-protokollet, har vi valt att använda ett lokalt nätverk genom en hotspot. Det har varit fördelaktigt att ha tråd och trådlöst nätverk, då vissa enheter enbart har stöd för trådade gränssnitt.

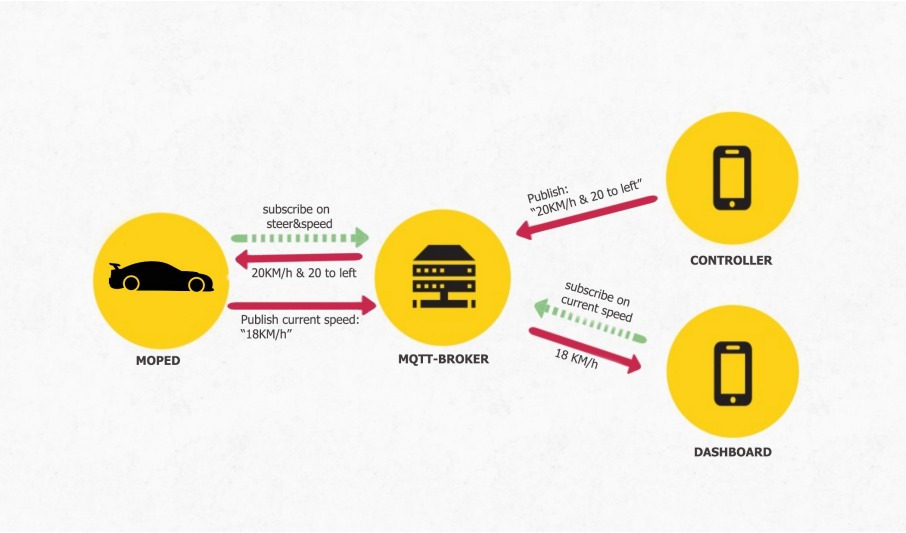
För MOPEDens TCU-enhet har vi haft tillgång till en WiPi-dongel som ger trådlöst nätverk till Raspberry Pi. Dock har detta varit problematisk eftersom den förlorar anslutningen konstant, oberoende av nätverk.

Problemet löstes temporärt genom att använda ett trådat nätverk via en router men vi hittade en lösning under sista veckan av projektet. Problemet samt lösningen finns på **Appendix A.3**.

5

Resultat

Detta kapitel beskriver vilket resultat vi har åstadkommit med hjälp av våra praktiska genomförande i förgående kapitel. Vår design [25] består av en styrapplikation (MqttWirelessIno), en hastighetsvisare (Dashboard), en mäklartjänst (Mosquitto) och en MOPED, som tillsammans utgör ett distribuerat system med säker och strukturerad anslutning till varandra. Se figur 5.1



Figur 5.1 - Koppling mellan MqttWirelessIno, MOPED och Dashboard

Kapitlet är indelat i en allmän resultatbeskrivning samt mer ingående resultat på MOPEDen, WirelessIno och Dashboard. Eftersom mäklaren sitter på MOPEDen, täcks resultatet av mäklaren på **4.2 Funktionalitet på MOPED**.

5.1 Helhetsresultat

Helhetsresultatet som vi har åstadkommit vilket inkluderar tre klienter och en mäklare, fungerar ihop enligt våra mål för projektet. Dessa tre klienter är som tidigare nämnts MOPEDen, MqttWirelessIno och Dashboard-applikationen.

På vår slutgiltiga test, för att se hur alla delar fungerar ihop med varandra, har vi startat MOPEDen och därefter har vi anslutit alla enheter genom IP-adress och portnummer till MOPEDen. Efter lyckad anslutning har vi kunnat köra MOPEDen med önskad hastighet och riktning. Med Dashboard-applikationen kan vi nu ansluta oss som klient, med samma anslutningsmetod som MqttWirelessIno och få uppdateringar från MOPEDen om vilken nuvarande hastighet den håller.

För en mer detaljerad beskrivning av funktionalitet, se **4.2 Funktionalitet på MOPED** samt **4.3 Funktionalitet på WirelessIno/Dashboard**.

5.2 Funktionalitet på MOPED

I det praktiska genomförandet gjorde vi förändringar i klasser som sköter den externa kommunikationen med andra klienter. Där har vi nu skapat två nya klasser som heter MqttSub.java och MqttPub.java, som står för engelskans ”Subscriber” och ”Publisher” vilket är översättningen till svenskans prenumerant och utgivare. Med hjälp av detta kan man skicka och ta emot styr- och hastighetssignaler samt allmänna informationer.

På MOPEDen, efter alla implementationer beskrivet i kapitel 4, har vi nu en fungerande mäklartjänst som startas samtidigt som TCU-enheten startar upp. På grund av detta behöver man inte ansluta till en skärm för att se vad som pågår på TCU-enheten.

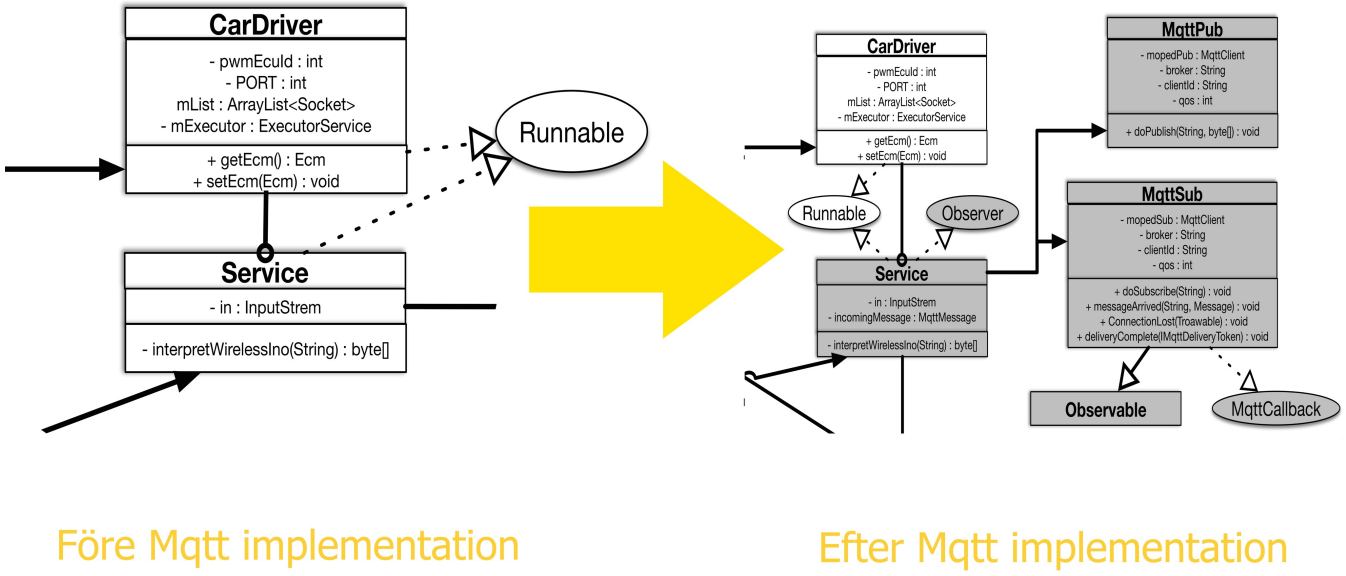
Mäklaren är nu konfigurerad med ett användarnamn och lösenord vilket är gemensamt för alla klienter som vill antingen agera som prenumerant på ett ämne eller publicera med ett givet ämne. TCU-enheten är registrerad som klient till mäklaren så att MOPEDen har både en MQTT-klient och en mäklare.

Med vår implementation kan det nu finnas enbart en styrenhet och en hastighetsvisare (Dashboard-applikationen) till MOPEDen genom att begränsa antalet klienter som kan ansluta till mäklaren till två. Dock kan inte två klienter med samma klient-ID ansluta till mäklaren eftersom det finns ett krav hos mäklaren att klient-ID ska vara unikt. Om en annan klient försöker ansluta till MOPEDen samtidigt som det redan finns en styrenhet trots samma användarnamn och lösenord, fås ett meddelande om misslyckad anslutning eftersom två klienter inte kan ha samma klient-ID, vilket betyder att platsen är fylld. Om man ser det ur en annan synvinkel, sätter mäklaren stopp för enheter som vill publicera meddelanden på ämnet ”MOPED/speed” och ”MOPED/steer” ifall det redan finns en enhet som publicerar meddelanden på dessa ämnen.

Resultatet för klientsidan på MOPEDen är att klienten är prenumerant på ämnet ”MOPED/speed” och ”MOPED/steer” för att få hastighet och styrsignaler från MqttWirelessIno. Klienten på MOPEDen är även utgivare på ämnet ”MOPED/speed” för att sprida sin hastighetsinformation för klienter som är intresserade av detta, i vårt fall Dashboard-applikationen. På figur 5.2 syns ett UML-diagram som visualiserar en före- och efterbild på våra implementationer. På den vänstra delen av diagrammet syns ett klassdiagram utan MQTT implementationer och där en Cardriver.java klass tillsammans med sin innerklass Service.java sköter den externa kommunikationen. På högra sidan av digrammet finns ett klassdiagram på våra MQTT implementationer. Där har ansvaret delats upp i mindre delar där MqttPub.java och MqttSub.java sköter kommunikationen i form av prenumerant och utgivare. För ett komplett UML-diagram samt vad pilarna betyder hänvisas läsaren till **Appendix A.2.**

Med nätverkslösningen kan TCU-enheten koppla upp sig mot angivet nätverk automatiskt vid uppstart av TCU-enheten. Detta resultat förenklar start av MOPEDen utan att behöva koppla enheter som skärm, tangentbord och mus för att göra MOPEDen redo för körläge.

Det totala resultatet för MOPEDen, med ovannämnda resultat, har blivit sparat som en image-fil. I image-filen ingår själva operativsystemet Rasbian Wheezy, Ecm-Linux (MOPED-programmet), Mosquitto-mäklaren och anpassade nätverkskonfigurationer. Med image-filen kan man nu installera allt på en gång. Detta kan vara fördelaktigt för vidaredistribution för andra projektgrupper exempelvis. Image-filen kan installeras som det hänvisas på SICS hemsida [28].

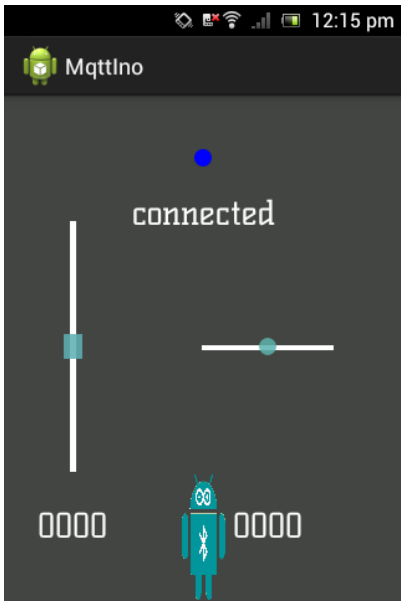


Figur 5.2 - Före och efter implementationer

5.3 Funktionalitet på MqttWirelessIno och Dashboard

5.3.1 MqttWirelessIno

Funktionaliteten på MqttWirelessIno är precis som den befintliga Android-applikationen (WirelessIno) som vi tog emot. Skillnaden ligger i klientsidan, som nu kommunicerar via mäklaren. Nu sker all kommunikation via mäklaren och ifall mäklaren tillåter behörigheten för klienter, skickas detta vidare till TCU-enheten och sedan till VCU-enheten och vidare till motor och styrservon. På figur 5.3 syns MqttWirelessIno-applikationen.



Figur 5.3 MqttWirelessIno

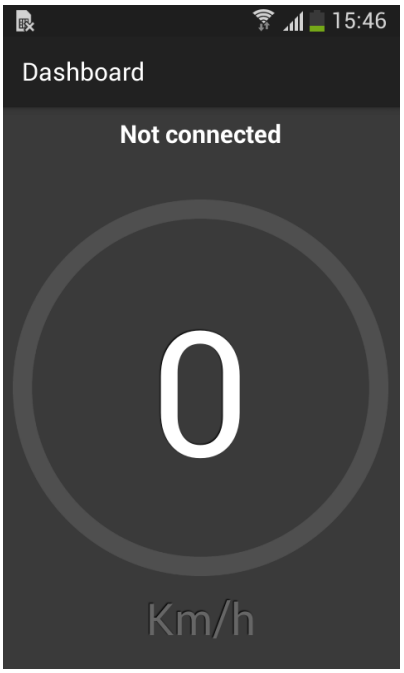
Klientsidan på MqttWirelessIno publicerar meddelanden på ämnet ”MOPED/speed” samt ”MOPED/steer”. Tidigare version av MqttWirelessIno, alltså WirelessIno, fungerar inte längre. När vi kollar på öppna portar på TCU-enheten ser vi att port 9000 inte längre är öppen.

5.3.2 Dashboard

Dashboard har samma funktionalitet som MqttWirelessIno gällande uppkoppling till TCU, via mäklaren. Den kan skapa en socket och agera som prenumerant på ämnet ”MOPED/getSpeed” och få uppdateringar från MOPEDen via mäklaren. På figur 5.4 syns Dashboard-applikationen.

Dashboard-applikationen funktionalitet är inte avancerad. När den tar emot hastigheten från MOPEDen, visualiserar den hastigheten på skärmen konstant, vid varje publikation av hastighet.

Strukturen i Dashboard-applikationen är precis som klientsidorna på MqttWirelessIno och MOPEDen. Där finns nu MQTT och Observer-mönstret implementerat.



Figur 5.4 - Dashboard

5.4 Säkerhet

Säkerheten vid anslutning är till följd av hotspoten på en Android-enhet som har stöd för krypterat nätverk. När enheterna i det distribuerade systemet ansluter sig mot den mobila hotspoten är anslutningen i nätverket krypterat med lösenord.

Bortsett från säkerheten kring krypterat nätverk, finns det användarnamn och lösenord lagrat, i klartext på både mäklaren och klienter som ansluter sig till mäklaren. Detta kan vara sårbarheten i implementationen. Dock är det lagrade lösenordet på mäklaren krypterat, då Mosquitto har stöd för detta. Sårbarheten ligger alltså på klientsidan.

Vi valde att inte gå vidare med säkerheten. Enligt vår resonemang så kan man göra intrång på en krypterad anslutning (SSL/TLSS) som MQTT har stöd för, ifall man lyckas komma förbi Androids egna krypterade nätverk via hotspoten.

6

Diskussion

Detta kapitel täcker den kritiska diskussionen gentemot vårt egna arbete. Här tas upp varför vi valde tekniker som vi gjorde och hur de skiljer sig från varandra. Kapitlet fortsätter med kritik mot klientsidan och hur säker anslutningen till MOPEDen faktiskt är.

6.1 Varför MQTT?

Anledningen till att vi använder MQTT i detta projekt är att huvudsakligen för att MQTT används idag inom fordonsindustrin för den här sortens uppgifter. Vid ett senare skede märkte vi att det är ett bättre alternativ i jämförelse med andra protokoll men även mer förekommande för distribuerade system och maskin till maskin (engelskans"machine-to-machine, M2M") användning. Med detta valet var vi alltså i fas med dagens forskning i valet av protokoll.

	MQTT	**HTTP**
Design orientation	Data-orienterad	Dokument-orienterad
Mönster	Prenumerant/utgivare	Förfrågan/respons
Komplexitet	Enkel	Mer komplex
Storlek av meddelanden	Liten	Större
Service grader	Tre QoS-grader	Alla meddelanden har samma grad
Extra bibliotek	För C (30 KB), för Java (100 KB)	Applikationsberoende
Data distribution	1 till 0, 1 till 1, 1 till N	1 till 1

Figur 6.1 - Jämförelse mellan MQTT och HTTP [32]

Att det är enkelt att förstå konceptet av MQTT var mycket fördelaktigt i detta projekt. MQTT är konstruerat att använda låg bandbredd, vara tidseffektivt och fungera bra för opålitliga nätverk. En annan stor fördel är att på grund av den lätta minnesanvändning av MQTT, sparar den på batteri för mobila enheter jämfört med protokoll som HTTP [8]. Gällande strömförsörjning vinner MQTT-protokollet gentemot andra protokoll, för att nämna några: REST[33], STOMP[34], AMQP[35].

För att nämna några nackdelar med ovannämnda protokoll så använder HTTP mer bandbredd vilket ökar strömförbrukningen och den har inte stöd för Quality of Service.

STOMP, som är likt HTTP, har en för enkel uppbyggnad för att verka effektivt i vår implementation. STOMP passar effektivt för simpla kommunikationer som exempelvis för att skapa websocket [36].

6.2 Olika varianter av mäklare

Det finns ett flertal olika mäklartjänster som fungerar ihop med MQTT. Ett antal olika av dessa har nämnts i **2.1.3 Mäklare.**

Anledningen till vårt val av Mosquitto är att den är lämplig för maskin-till-maskin användning. På grund av att Mosquitto är skriven i programmeringsspråket C [37], passar mäklaren för inbyggda system som exempelvis inte kan använda sig av JVM (Java Virtual Machine)[38]. Mosquitto är en open-source mäklare som kan användas för testning och att skapa prototyper. En annan fördel är dokumentationen för Mosquitto som är bra formulerade och enkel att sätta sig in i. I själva installationen följer det med testprogram vilket gör det ännu enklare att sätta sig in i protokollet. En lista med jämförelser mellan olika mäklartjänster för MQTT finns i följande referens[39].

6.3 Val av programmeringsspråk

Java är det enda programmeringsspråket som vi har använt oss av i detta projekt. Detta valet gjorde vi på grund av att det befintliga programmet på MOPEDen var skriven i Java. Det är dock inte det mest effektiva för detta projekt, med tanke på reaktionstid som man kräver vid körning av MOPEDen.

Eftersom körning av MOPEDen kräver finkänslighet, likt ett riktigt fordon, krävs en bra reaktionstid. Valet vi har gjort med Java fungerar bra men att sänka reaktionstiden är möjligt, eftersom det finns fler programmeringsspråk som har mindre reaktionstid. Ett programmeringsspråk som är lättviktigt och gjort för kommunikation skulle kunna passa in för detta projekt. Eftersom Eclipse-Paho-biblioteket har stöd för ett flertal programmeringsspråk skulle man kunna välja ut ett programmeringsspråk därifrån som gör uppgiften snabbare än Java, exempelvis C[37].

Vi tycker dock att Java passar in väldigt bra för att skicka och ta emot meddelanden, speciellt de meddelanden som inte kräver exakt reaktionstid. Sådana meddelanden som exempelvis sändning och mottagning av motortemperatur fungerar fint med Javas hastighet och prestanda.

6.4 Kritik till klientsidan

Vi är nöjda med implementationer vi har gjort inom tidsramarna för projektet. Dock finns det mycket mer vi kunde ha gjort om tiden inte var knapp. Nedan följer tre kritiska faktorer samt en diskussion om hur säker våra implementationer är.

6.4.1 Återanslutning

I klientsidorna vi har på våra tre olika enheter (MOPEDen, MqttWirelessIno och Dashboard) hann vi inte implementera ”Återanslutning” (engelskans ”reconnection”) vilket är kopplat till delkapitel 4.1.2. Ytligt

beskrivet så gäller återanslutningen för de MQTT-klienter som mot förmodan skulle tappa kopplingen med mäklaren. Eftersom vi inte alltid har tillgång till skärm och tangentbord så kan vi inte se ifall klienten tappat kopplingen eller inte. Det hade varit en bra idé att implementera en återanslutnings mekanism som försöker ansluta till mäklaren ifall anslutningen bryts. MQTT har bra stöd för detta genom sina ”Callback” metoder[40].

6.4.2 Quality of Service, QoS

Vi har valt QoS 0 som grad av leverans av meddelanden för bland annat följande anledningar:

- Vi har prioriterat hastigheten av sändning av meddelanden före pålitlighet eftersom paketförluster inte ger någon signifikant påverkan. Anledningen till att paketförlust inte ger signifikant påverkan är att MqttWirelessIno skickar ett meddelande varje 10 millisekunder vilket gör att ifall det skulle ske paketförlust så fås uppdateringar av meddelanden många gånger om igen. Detta sker väldigt snabbt och man uppfattar knappt någon förändring under körning av MOPEDen.
- Högre grad av QoS har ”överflöd” i nätverket vilket inte ger önskad prestanda. Med överflöd på nätverket menas att varje meddelande kommer att antingen kräva två eller fyra handskakningar. Detta gör att systemet blir långsammare eftersom varje meddelande måste vänta tills handskakningar för meddelandet före har gjorts klart.
- En annan anledning till varför valet blev QoS av grad 0 är för att vi inte har användning av en kö-struktur till skillnad från QoS av grad 1 och 2 som har en kö-struktur och tar meddelanden i tur och ordning.

Fördelarna lämpar sig dock för hastighets- och riktningssignaler. För publikationer av meddelanden som rör motortemperatur eller bränsleförbrukning, som inte har lika många meddelanden under kort period, blir det fördelaktigt med att ha en högre grad av QoS.

När man ger en önskad hastighet till MOPEDen, fungerar MOPEDen på så sätt att den ökar hastigheten tills den når önskat hastighet en gång och sedan, på grund av friktion, saktar den ner och stannar så småningom. Om MOPEDen kör utanför sin räckvidd, så styrapplikationen WirelessIno inte kan nå den längre, fortsätter den inte att köra framåt. Detta beteendet är något som funnits på MOPEDen och som vi inte har påverkat. Med QoS 0 har man alltså större chans att leverera meddelanden med hög hastighet till MOPEDen vid ett scenario där MOPEDen är påväg ut ifrån räckvidden, vilket är fördelaktigt.

Att man väljer en struktur som QoS 0 har, alltså att snabbheten är prioriterat, är något som följs ute i industrin och riktiga fordon. På riktiga fordon gör man alltså inga ”handskakningar” utan bara skickar ut meddelanden med hopp om att man får önskad effekt. Istället finns en mekanism som kallas för Monitor-Actuator[41] som går på en lägre frekvens och håller koll på ifall meddelanden levereras eller inte. Levereras inte meddelanden, ges ett alarm och åtgärder tas av Monitor-Actuator.

6.4.3 Krypterad anslutning via SSL/TLS

MQTT har stöd för krypterad anslutning via nätverk med hjälp av SSL/TLS[42]. Men detta är inget som vi använde oss av i projektet. Främsta anledningen till detta var att vi använde oss av en hotspot på en Android-enhet som skapar ett nätverk och detta nätverk var krypterat. Dock ser vi stor potential med att implementera en krypterad anslutning eftersom detta inte kräver mycket ansträngning för att få gjort. Det skulle vara passande ifall vi inte använde en mobil hotspot utan ett nätverk utan kryptering. Bra dokumentation och exempel på SSL/TLS i följande referens[7].

6.4.4 Hur säker är anslutningen till MOPEDen?

Bedömningen av säkerheten är svår att göra. Jämför man med den tidigare versionen så har säkerheten ökat väsentligt. Tidigare kunde man, genom IP-adress och portnummer, ansluta sig till MOPEDen utan vidare ifrågasättning av vem det var som anslöt sig.

Med våra implementationer blir det svårare att passera till MOPEDens känsliga delar. Nu behöver man alltså IP-adress, portnummer (för mäklaren), rätt användarnamn och rätt lösenord. Det svåra med bedömningen är huruvida det är möjligt att få tillgång användarnamnet och lösenordet. Det är enklare att få tillgång till IP-adress och portnummer som obehörig till skillnad från användarnamn och lösenord, som kräver lite mer arbete. Lyckas man komma förbi det hindret, behöver man även rätt ämne för att publicera meddelanden på eller rätt ämne för att få information ifrån.

Vi ser stor potential i strukturen vi har implementerat och ser möjlighet till vidareutveckling genom att man kan konfigurera mäklaren och lägga fler hinder för att öka säkerheten.

Som nämnts i kapitel **4.4 Säkerhet** finns det en viss sårbarhet i våra implementationer eftersom lösenordet på klientsidan står i klartext. Om man varken har ett krypterat nätverk via en Android hotspot eller krypterat anslutning med SSL/TLS, så kan man även använda en krypteringsmekanism som krypterar lösenord hos klientsidan och samma krypteringsmekanism som dekrypterar lösenordet hos mäklaren. Detta kan lämnas som en vidareutveckling av anslutningen gentemot mäklaren.

6.5 Inträffade svårigheter

Vi har inte haft svårigheter som satt spärr för oss att driva vidare projektet. Dock har vi haft svårigheter som fått oss att ägna mer tid åt problem som har varit utanför projektets ramar. Ett exempel på detta är svårigheter med WiPi-dongeln för Raspberry Pi. Anledningen till problemen vi har stött på med WiPi-dongeln är okända för oss trots felsökning. Samma inställningar och konfigurationer har fungerat på andra Linuxenheter (Ubuntu). Lösningen som presenterades i **Appendix A.3** löser problemet för anslutning till enbart ett nätverk. Dock har lösningen sparat oss tid och energi samt öppnat upp möjligheten för körning av MOPEDen genom trådlöst nätverk.

En annan nämnvärd faktor som sänkte farten på utvecklingen var brist på dokumentation både MOPEDens uppbyggnad och användarmanual men även för mjukvaran. Flera veckors arbete gick till att bekanta sig med MOPEDens mjukvara, enbart för att orientera sig fram till vart våra implementationer bör ske någonstans. Vi tog hjälp av medlemmar från tidigare projektgrupp som har gjort ett projekt på MOPEDerna. Vi är dock tacksamma över den dokumentation som finns och har som mål att öka förståelsen för MOPEDen och dess mjukvara genom denna rapport.

7

Slutsats

Detta avslutande kapitel knyter samman hela examensarbetet. Kapitlet börjar med en resumé av rapporten samt fortsätter med en generalisering av resultatet. Kapitlet avslutas med idéer kring vidareutveckling av MOPEDerna.

7.1 Resumé

I denna rapport för examensarbete på kursen LMTX38 har vi tagit upp ett projekt för Viktoriainstitutet som innefattar att implementera ett säkert och strukturerat sätt för uppkoppling mot radiostyrda bilar. Projektet har till stor del att göra med MQTT eftersom själva strukturen för uppkoppling är gjord enligt MQTT-protokoll.

Implementationen har medfört en mäklartjänst som heter Mosquitto samt klientsida i MOPEDen, WirelessIno- och Dashboard-applikationen. Med prenumeranter och utgivare på rätt plats sker en konstant kommunikation mellan dessa tre enheter i det distribuerade systemet där önskat meddelande gällande uppdateringar på olika ämnen skickas och tas emot. Dessa ämnen kan vara exempelvis riktning, nuvarande hastighet eller motortemperatur.

Med detta gjort öppnas dörrarna för att bland annat implementera AGA-plattformen på MOPEDerna eftersom man nu kan integrera applikationer med säkerhet. Detta betyder att studenter, hobbyister eller doktorander kan bygga applikationer riktat mot fordon och testa dessa mer kostnadseffektivt jämfört med ett riktigt fordon. Det blir även enklare att visa och demonstrera AGA-applikationer på en MOPED.

Detta är ett av många delprojekt som görs på MOPEDen, för att inom en snar framtid kunna förverkliga visionen man har för fordonsindustrin genom MOPEDerna.

7.2 Generalisering

Hur fungerar det distribuerade systemet i bredare villkor? Eftersom vi använder oss av en MOPED med struktur och uppbyggnad som ska efterlikna ett riktigt fordon, har vi under projektet siktat mot att göra ett verklighetsscenario av detta. Med detta menar vi att vi har använt verktyg och protokoll som lika gärna kan sitta på en lastbil eller en personbil. Systemet skulle lika gärna kunna vara placerat i ett riktigt fordon. Detta är också en stor anledning till att vi använder tekniker som MQTT, eftersom detta används i industrin idag.

Ett exempel på ett projekt på Secure Gateway som pågår i industrin idag är ”Plinta” projektet som är en plattform för säker integration av användarfunktioner i fordon, utvecklat av Semcon [43]. Plinta använder Secure Gateway vilket betyder att MQTT används i Plinta projektet likt vår projekt. Plinta används dock inte för att styra ett fordon likt vår projekt utan syftet är att man som förare ska minska interaktionen med applikationer i fordonet under färd för att öka säkerheten. Vi ser våra resultat som en prototyp av Secure Gateway som är open-source och kan implementeras på exempelvis personbilar, lastbilar, båtar och flygplan för uppvisning.

Vi ser vårt arbete som ett representativt resultat för fordonsindustrin eftersom vi har ett koncept av Secure Gateway som fungerar och går att demonstrera.

7.3 Vidareutveckling

Det finns en del förbättringar man kan göra på MOPEDen samt dess applikationer. Förutom punkter som tas upp nedan finns exempelvis byte av programmeringsspråk (Se **5.3 Val av programmeringsspråk**) eller översätta signaler till AGA-signaler.

7.3.1 Utökning av signaler

Det har varit brist på signaler på MOPEDerna under projektets gång. Eftersom vi inte har påverkat CAN-nätverket och inte kunnat skicka upp en signal för att uppdatera klienter om mätvärden på MOPEDen, utnyttjade vi istället hastigheten som skickas in till MOPEDen för önskad effekt, för att sedan få ut samma signal till Dashboard-applikationen.

Lösningen till fler signaler hos MOPEDen ligger i att koppla in mer hårdvara i form av sensorer, temperaturmätare etc för att använda dessa mätvärden för att kunna utnyttja MOPEDens signaler i ett större sammanhang. Att ha Dashboard-applikation med exempelvis temperaturmätare vid sidan om hastigheten liknar mer ett riktig instrumentpanel i ett fordon.

Med fler signaler att prenumerera på hos MOPEDen, kan man även skapa bättre AGA-applikationer eftersom man har mer signaler att utnyttja. I AGA-plattform finns allt från hjulhastighet, växel, bensin- eller strömförbrukning, antal kilometer och mer som kan läggas till MOPEDen för att vara så nära ett riktigt fordon som möjligt.

7.3.2 Justera hastighetsläsningen

Under testningsperioden hade vi mycket tid för att bekanta oss med MOPEDens styrning och körning. Vi tycker att känsligheten på MOPEDen kan justeras för att förenkla körningen genom att göra en bättre hastighetsreglering. Med hastighetsreglering menas också en bättre hastighetsavläsning.

Styrningen av MOPEDen lever upp till våra förväntningar. Där kan vi vrida och vända med fin styrning över MOPEDen. Samma känslighet gällande hastighet hade förbättrat körupplevelsen hos MOPEDen.

7.3.3 Inställningar via applikation

I vanliga fall när man vill ändra inställningar hos MOPEDen gällande dess egna mjukvara, är det enda sättet för ny mjukvara och inställningar att träda i kraft att man överför det via USB-minne, det vill säga att uppdatera hela MOPEDens mjukvara.

En vidareutveckling av detta projekt skulle kunna vara att man kan påverka inställningarna hos MOPEDen via en Android-applikation med administratörsbehörighet. Här skulle man kunna ange användarnamn och lösenord i själva Android-applikationen, för att logga in som administratör. Sedan, ifall man är behörig, skulle man kunna ändra inställningarna hos MOPEDen via Android-applikationen istället för att ha inställningarna hårdkodade i mjukvaran och föra över den till MOPEDen via ett USB-minne.

Referenser

[1] Tillman, K. (2013) How Many Internet Connections are in the World? Right. Now. Cisco blog, http://blogs.cisco.com/news/cisco-connections-counter. (2015-05-25)

[2] Evans, D. (2011). Internet of Things:How the Next Evolution of the Internet Is Changing Everything. Amsterdam: Cisco Internet Business Solutions Group, IBSG. ss 2-3.

[3] Khatab A, Zolic D. (2015) Applikationsutveckling för radiostyrda bilar. Göteborg: Chalmers tekniska högskola. (Examensarbete inom Institution för data- och informationsteknik.)

[4] SICS Swedish ICT. (2015) About SICS Swedish ICT: The Swedish institute of computer science https://www.sics.se/about-sics. (2015-05-25)

[5] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle.(2014) From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. [Electronic] Kinlington, Oxford: Academic Press is an imprint of Elsevier. ss 3-18

[6] Zhang, L. (2011) Building Facebook messenger. Facebook Engineering, http://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920. (2015-05-25)

[7] Light, R. Mosquitto-tls: Configure SSL/TLS support for Mosquitto. Mosquitto official website, http://mosquitto.org/man/mosquitto-tls-7.html. (2015-05-25)

[8] Auguste, E. (2012) Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android. IBM blog, https://www.ibm.com/developerworks/community/blogs/messaging/entry/power_profiling_https_long_polling_vs_mqtt_with_ssl_on_android31?lang=en. (2015-05-26)

[9] Banks, A., Gupta, R. (2014) OASIS MQTT version 3.1.1: OASIS Standard.[Electronic] OASIS Open, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. (2015-05-26)

[10] Light, R. MQTT: MQ Telemetry Transport. Mosquitto official website, http://mosquitto.org/man/mqtt-7.html. (2015-05-26)

[11] Eclipse Paho. Eclipse Foundation, Inc, https://eclipse.org/paho/ .(2015-05-26)

[12] Mosquitto: An Open Source MQTT v3.1/v3.1.1 Broker. (2015) Mosquitto official website, http://mosquitto.org. (2015-05-26)

[13] Light, R. Mosquitto.conf: the configuration file for mosquitto. Mosquitto official website, http://mosquitto.org/man/mosquitto-conf-5.html. (2015-05-26)

[14] Miyamoto, M. (2014) Config file for mosquitto. MQTT and more, http://mm011106.github.io/reference/mosquitto_conf.html. (2015-05-27)

[15] Skansholm, J. (2014) Java Direkt med Swing. Sverige: Studentlitteratur, åttonde upplagan, ss469-470.

[16] Raspberry Pi Foundation website. Raspberry Pi: Teach, Learn and make with Raspberry Pi, https://www.raspberrypi.org. (2015-05-28)

[17] Raspberry Pi Foundation website. What is a Raspberry Pi, https://www.raspberrypi.org/help/what-is-a-raspberry-pi/. (2015-05-28)

[18] Rasbian website. (2012) Welcome to Rasbian, https://www.raspbian.org.(2015-05-28)

[19] Wikipedia, Raspberry Pi, http://sv.wikipedia.org/wiki/Raspberry_Pi. (2015-06-03)

[20] Webbshopen för Raspberry Pi. (2015) WIFI USB DONGEL - FÖR RASPBERRY PI - WIPI, http://www.r-pi.se/wifi-usb-dongel.html. (2015-06-26)

[21] Axelsson, J., Kobetski, A., Ni, Z., Zhang, S., Johansson, E. (2014) MOPED A Mobile Open Platform for Experimental Design of Cyber-Physical Systems. [Electronic]Kista: Swedish Institute of Computer Science (SICS)

[22] Autosar. (2014) AUTOSAR basics, http://www.autosar.org/about/basics/. (2015-06-01)

[23] Swedish Institute of Computer Science (SICS). Hardware configuration, https://moped.sics.se/?page_id=93.(2015-06-01)

[24] Swedish Institute of Computer Science (SICS). (2015) Mobile Open Platform for Experimental Development https://github.com/sics-sse/moped.(2015-05-25)

[25] Hevner, A., March, S., Park, J., Ram, S. (2004) Design science in information systems research. *MIS Quarterly*, Vol. 28, No. 1, pp. 75-115.

[26] Henrysson, D. (2010). Scrum: en fallstudie från lokala företags perspektiv. Blekinge: Blekinge Tekniska Högskola (Kandidatarbete i Datavetenskap)

[27] Trello official Website. (2015) About Trello, https://trello.com/about. (2015-06-02)

[28] Swedish Institute of Computer Science (SICS). Setting up Linux on TCU, https://moped.sics.se/?page_id=328.(2015-05-29)

[29] (2015) Vår image fil "MOPED2015": linux-image-mqtt.gz. http://tinyurl.com/moped2015. (2015-06-03)

[30] Johansson, M. (2014) Testing, Debugging, and Verification: Testing1.pdf. TDA567/DIT082, LP2, HT2014, http://www.cse.chalmers.se/edu/year/2014/course/TDA567_Testing_debugging_and_verification/Testing1.html. (2015-05-29)

[31] Malinen, J. (2013) Linux WPA/WPA2/IEEE 802.1X Supplicant. hostapd and wpa_supplicant, http://w1.fi/wpa_supplicant/. (2015-06-02)

[32] Lampkin, V. Leong, W.T., Olivera, L. Rawat, S. Subrahmanyam, N. Xiang, R. (2012). Building a smarter planet solutions with MQTT and IBM WebSphere MQ Telemetry. [Electronic] NewYork: .IBM Corporation, International Technical Support, ss 7.

[33] Why rest is good, but MQTT is better (2014-04-01) http://iottech.club/three-iot-communication-protocols-comparison-mqtt-vs-coap-vs-lm2m-vs-xmpp-vs-rest-api/ (2015-06-01)

[34] Elsner, J., Meisen, P., Thelen, S., Schillberg, D., Jeschke, S. (2013) EMuRgency - A basic concept for an AI Driven Volunteer Notification System for Integrating Laypersoners into Emegency Medical Services. International Journal on Advances in Life Sciences, Vol. 5 nr 3&4, ss 223-235.

[35] Cohn, R. (2012-02) A comparison of AMQP and MQTT,. OASIS Advancing open standards for the information society,. https://lists.oasis-open.org/archives/amqp/201202/msg00086/StormMQ_WhitePaper_-_A_Comparison_of_AMQP_and_MQTT.pdf. (2015-06-01)

[36] Mesnil, J. (2012) STOMP Over WebSocket, http://jmesnil.net/stomp-websocket/doc/. (2015-05-29)

[37] Axelsson,E., Claessen, K., Devai, G., Horvath, Z., Keijzer, K., Lyckegard, B., Persson, A., Sheeran, M., Svenningsson, J., Vajda, A. (2015) Feldspar: A Domain Specific Language for Digital Signal Processing

algorithms. [Electronic] Sweden: Ericsson Software Research, http://www.cse.chalmers.se/~ms/MemoCode.pdf. (2015-05-26)

[38] Lindholm T, Yellin F, Bracha G, Buckley A, (2015) The Java Virtual Machine Specification. [Electronic] Redwood City: Oracle America, Inc, 8th Edition, https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf. ss 2. (2015-05-27)

[39] LLampkin,V. Leong, W.T.,Olivera,L. Rawat,S. Subrahmanyam,N. Xiang, R. (2012). Building a smarter planet solutions with MQTT and IBM WebSphere MQ Telemetry. [Electronic]NewYork: IBM Corporation,International Technical Support, ss 10

[40] D. Mitchell, J. (1996) Java Tip 10: Implement callback routines in Java: Using interfaces to implement the equivalent of callback functions in Java. Java World, http://www.javaworld.com/article/2077462/learn-java/java-tip-10--implement-callback-routines-in-java.html. (2015-06-03)

[41] Konrad, S., H.C Cheng, B. (2002) Requirements Patterns for Embedded Systems. Proceedings of the IEEE Joint International Conference on Requirements Engineering, September, 2002, Essen.

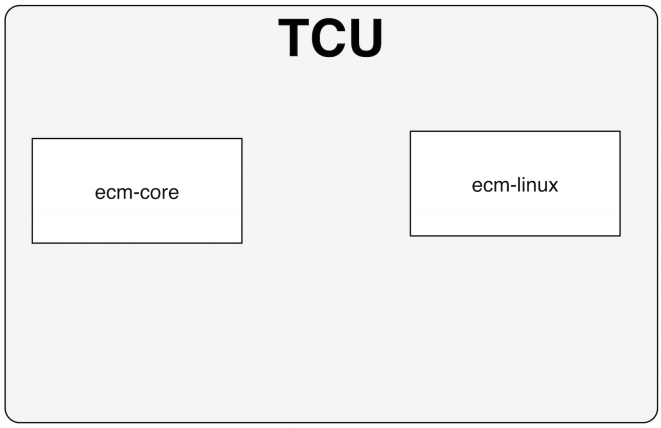
[42] Ristić, I. (2014) SSL/TLS Deployment Best Practices. [Electronic] Redwood City: Qualys SSL Labs

[43] Juul, J. (2012) PLINTA: Plattform för säker integration av användarfunktioner i bil. VINNOVA, http://www.vinnova.se/sv/Resultat/Projekt/Effekta/PLINTA---Plattform-for-saker-integration-av-anvandarfunktioner-i-bil/. (2015-05-30)

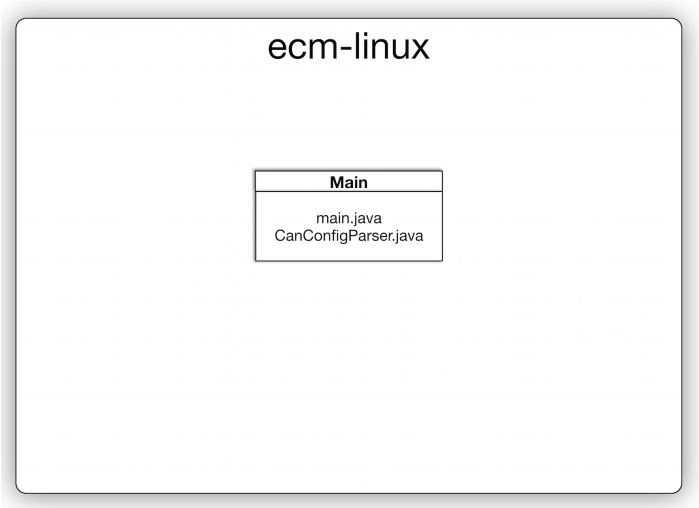
A. Appendix

A.1 Klass- och paketstruktur

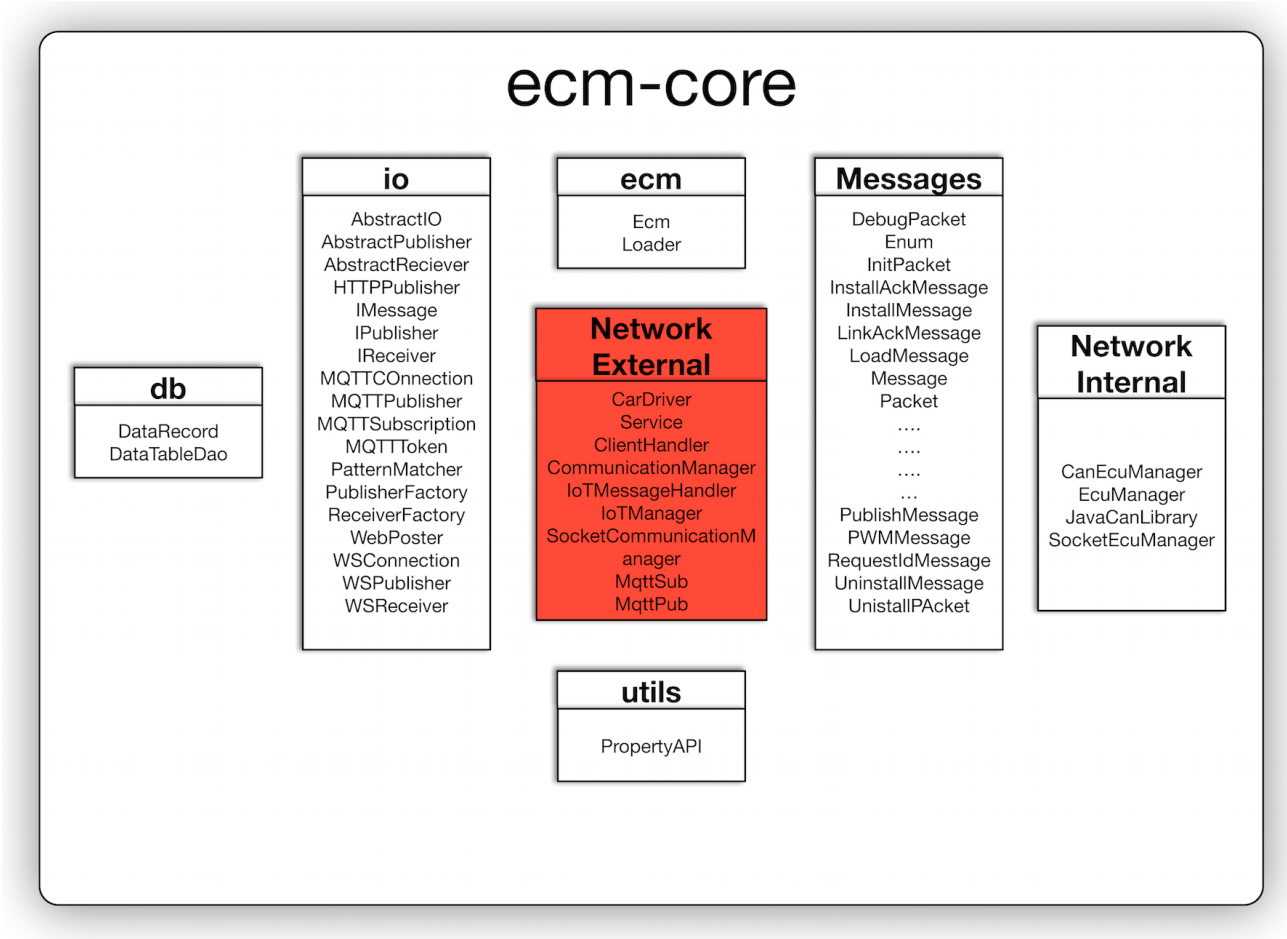
Paketstruktur av TCU.



Paketstruktur av ecm-linux.

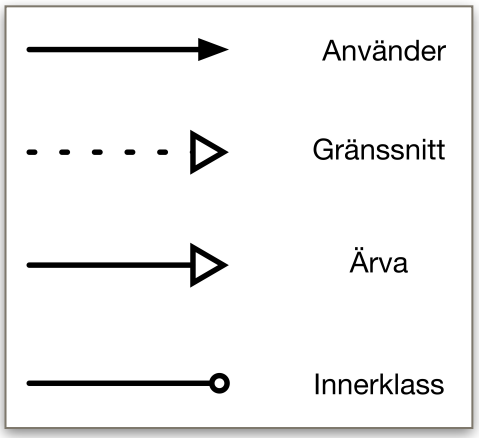


Paketstruktur av ecm-core.

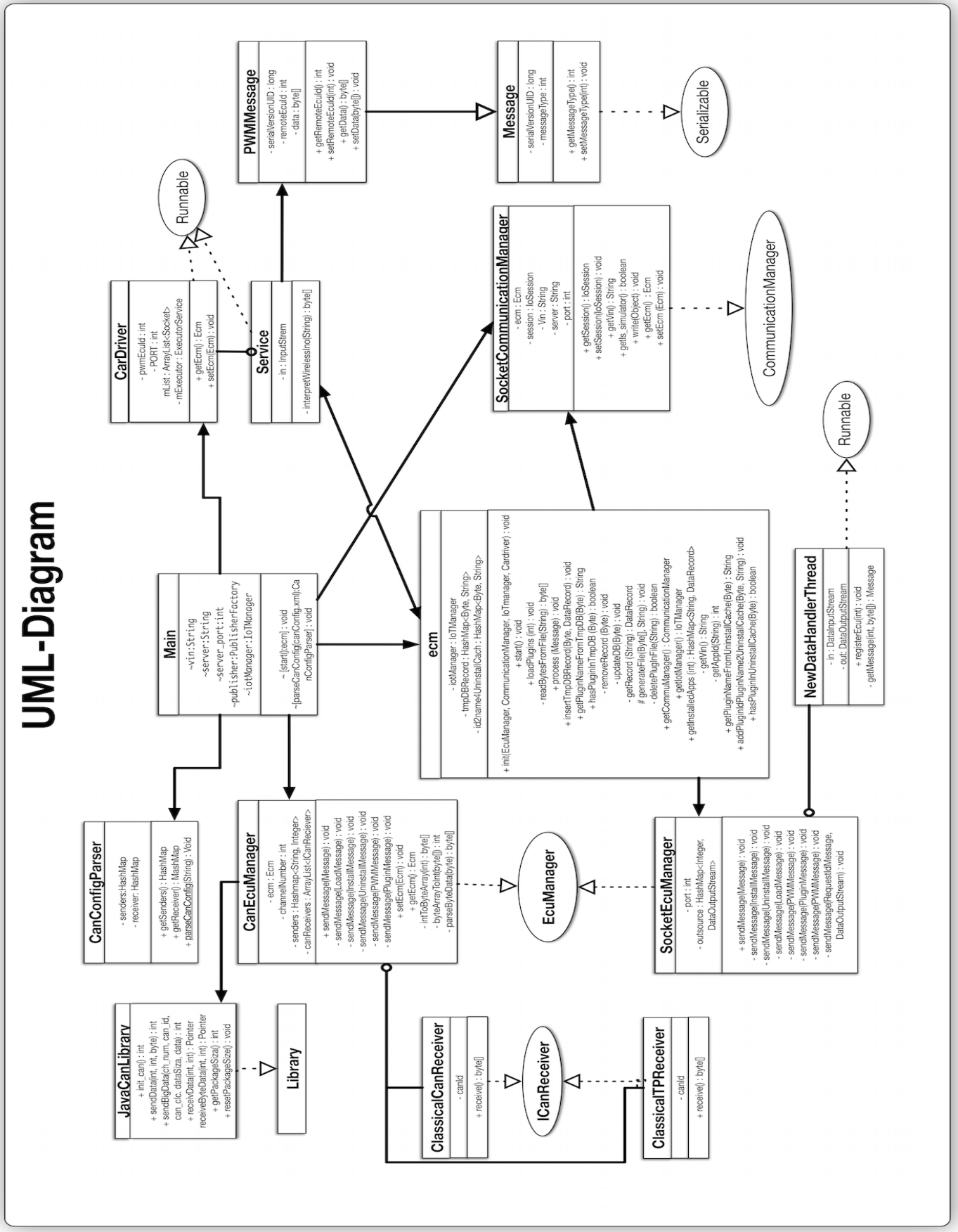


A.2 UML

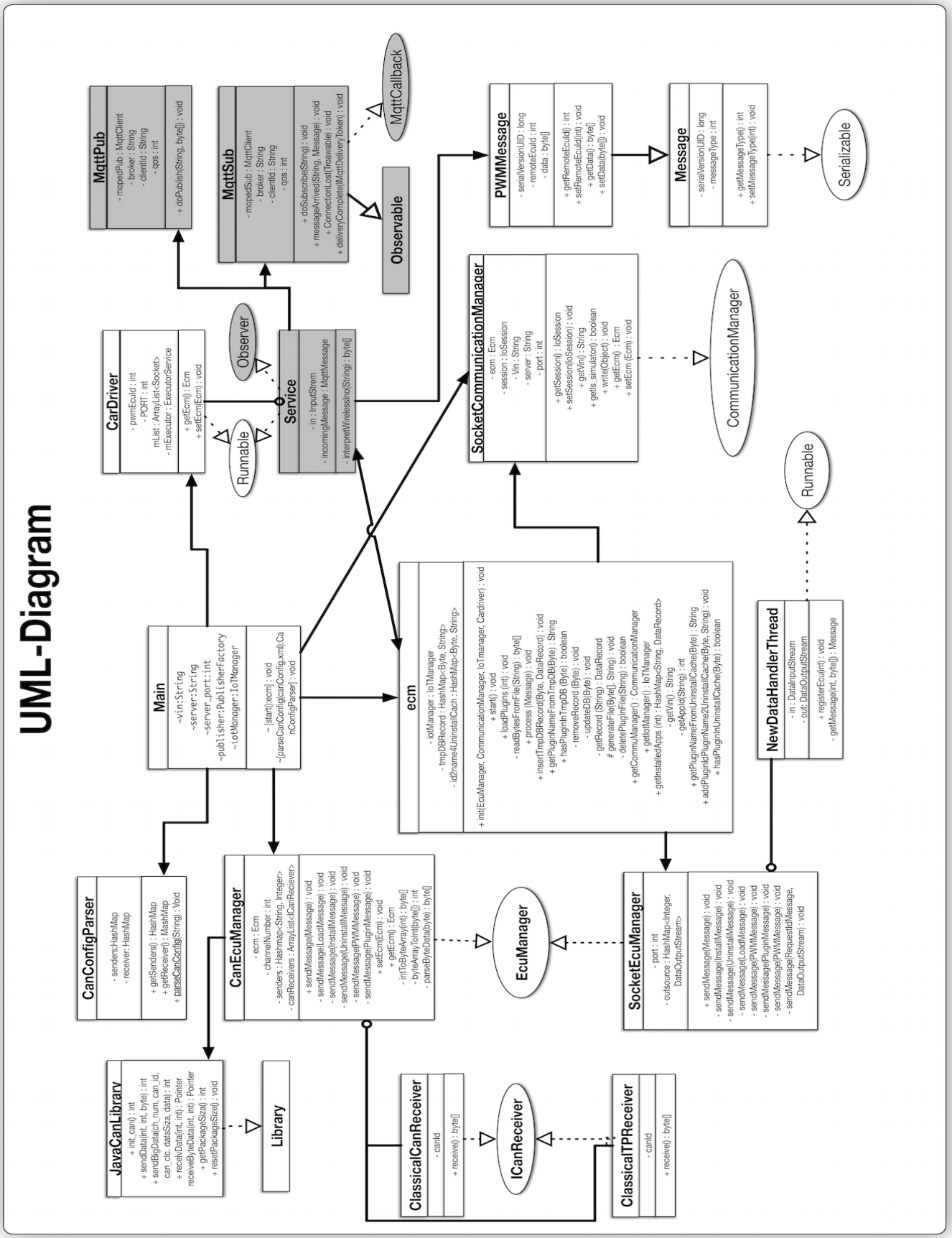
Beskrivning av UML pilar



UML diagram före MQTT implementationer.



UML diagram efter MQTT implementationer.



A.3 Nätverkskonfiguration

Felsökningen av trådlöst nätverk gjorde vi genom att säkerställa att rätt inställningar fanns med på konfigurationsfilen, som ger inställningar för olika nätverk. Detta gjordes genom att orientera sig fram till ”wpa_supplicant.conf” filen[31] och generella gränssnittsinställningarna för nätverk genom följande två kommandon:

```
cd nano /etc/wpa_supplicant/wpa_supplicant.conf
cd nano /etc/network/interfaces
```

Trots rätt inställningar, som testats på andra Linuxenheter, kunde inte WiPi-dongel koppla upp sig automatiskt mot ett nätverk vid uppstart av TCU-enheten.

Vi gav oss in på felsökning gällande automatisk uppkoppling på nätet och under sista veckan av implementationen av projektet fann vi en lösning där man istället för att hänvisa till ”wpa_supplicant.conf”. för vilket nätverk som ska användas, skrivs detta direkt nätverks gränssnittet i ”/etc/network/interfaces”. Varför detta fungerar och inte det andra är oklart och ligger utanför projektets ramar.

För att få fungerande för tråd- och trådlöst nätverk har vi följt följande rader, som finns i ”interfaces"-filen.

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp
allow-hotplug wlan0
auto wlan0
iface wlan0 inet dhcp
wpa-ssid "nätverk-ssid"
wpa-psk "nätverk lösenord"
```

A.4 Installation av Mosquitto

```
wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
sudo wget http://repo.mosquitto.org/debian/mosquitto-wheezy.list
sudo apt-get update
sudo apt-get install mosquitto
```

De första två raderna är till för att hämta nyckel för Mosquittos repository samt att lägga till den.
Den tredje raden hämtar rätt version vilket i vårt fall är för Rasbian Wheezy. Den fjärde raden uppdaterar versionen man nyligen hämtat på rad 3 och efter uppdateringen, installerar man Mosquitto med alla rätta bitar på plats.

A.5 Konfiguration av Mosquitto

Med Mosquitto installerat är det hög tid att konfigurera mäklaren. Efter installationen erhåller mäklaren standardvärden vilket bland annat tillåter anslutning för anonyma klienter.

Mäklaren konfigureras genom att man orientera sig fram till konfigurationsfilen enligt följande kommando:

```
cd /etc/mosquitto/mosquitto.conf
```

Efter att man kommit till rätt mapp, kan man öppna mosquitto.conf med hjälp av följande kommando, ifall man väljer att öppna filen med textredigerare ”nano”:

```
sudo nano mosquitto.conf
```

Efter det steget får man upp konfigurationsfilen och inställningarna som man vill ändra på enligt önskemål. I vårt fall har vi ändrat ”allow-anonymous” (Se 2.1.2 Mäklare) till false vilket nu innebär att anonyma klienter får ett meddelande av mäklaren som säger ”Connection not authorized”.
Med anonyma klienter borta, bör man registrera ett användarnamn följt av ett lösenord för att ge behörighet till klienter för anslutning till mäklaren. Detta gör man genom att orientera sig fram till ”username” och ”password” och deklarera ett användarnamn och lösenord för behöriga klienter.

Det finns ett kommando som följer med installationen av Mosquitto som heter ”mosquitto_passwd”. Denna tjänst hjälper en att sätta användarnamn och lösenord för klienter som vill ansluta sig till mäklaren. Med följande kommando kan man ange användarnamn och lösenord:

```
sudo mosquitto_passwd -c /etc/mosquitto/password_file.txt username
```

Kommandot anropar tjänsten mosquitto_passwd där ”-c” står för skapa (engelskans ”create”) och ”/etc/mosquitto/password_file.txt” står för filen man vill skapa.

Härefter kommer systemet att be om ett lösenord samt bekräftandet av lösenordet. I ett senare skede har man sitt användarnamn samt ett krypterat lösenord i ”password_file.txt” filen, som sedan Mosquitto kan använda sig av.

A.6 Testning av Mosquitto

I MQTT-sammanhang kan man agera som prenumerant på ett ämne. Detta gör man genom att i terminalen knappa in följande kommando (förutsatt att mäklaren tillåter behörighet till alla klienter och att den är igång):

```
mosquitto_sub -t ”MOPED”
```

I kommandot står ”sub” för engelskans ”subscriber” som betyder prenumerant. Prefixet ”-t” står för ämnet (engelskans ”topic”). Med detta kommandot intryckt så kommer terminalrutan att invänta nya publikationer på ämnet ”MOPED” och visa upp meddelanden. För att avsluta programmet kan man trycka Ctrl+C.

Kommandot för utgivare är snarlikt kommandot för prenumerant.

```
mosquitto_pub -t ”MOPED” -m ”Speed: 30”
```

I detta kommandot står ”pub” för utgivare (engelskans ”publisher”) och tillägget jämfört med prenumerant kommandot är i prefixet ”-m” som står för meddelande (engelskans ”message”) följt av själva meddelandet ”Hastighet 30” på ämnet ”MOPED”.

Ifall vi har lösenord och lösenord konfigurerat på mäklaren, behöver vi ha med detta när vi prenumererar eller publicerar genom att ha prefixet ”-u” för användarnamn (engelskans ”username”) och ”-P” för lösenord (engelskans ”password”). Observera att det är versalt ”P” som gäller för lösenord, eftersom det kan förvirras med portprefixet, som har ”-p” i gemener.

Exempel som prenumerant med användarnamn och lösenord:

```
mosquitto_sub -u ”Anders” -P ”hemligt” -t ”MOPED”
```

A.7 Klientsida

```
MqttClient client = new MqttClient(”Adress_till_mäklare:1883”, ”Klient_Namn”);
ConnectOptions connection = new ConnectionOptions();
client.connect(connection);
```

På första raden deklareras klienten med adressen till mäklaren, vilket i vårt fall är adressen till mäklaren som är installerad på MOPEDen, följt av port 1883 som är standard för MQTT. För att kunna initieras behövs även ett klientnamn som klienten kommer introducera sig med. Anslutningen ”connection” är uppsättning av inställningar som man vill ansluta med som innehåller speciella egenskaper för anslutningen. Det som krävs för anslutningen är att koppla samman klienten och anslutningen genom anrop på tredje raden.

När anslutningen är gjord kan man prenumerera och publicera genom följande anrop:

```
client.subscribe(”MOPED”);
client.publish(”MOPED”, ”Speed: 30”);
```

På första raden prenumererar man på ämnet ”MOPED” och får publikationer på det ämnet. På andra raden publicerar man ett meddelande på ämnet ”MOPED” och med meddelandet ”Speed: 30”, precis som exemplet i **4.1.1 Mosquitto**.