PTT-Operatør

Casper Skipper Olsen

Kongens Lyngby 2012
IMM-B.Eng-2011-89

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-B.Eng-2011-89

Resumé

Med dette produkt bliver brugerne i stand til at styre en PABX. Applikationen indgår som en del af et større satellitbasseret PTT kommunikationssystem. Thrane & Thrane har udviklet en mobil VoIP PTT-enhed som kan anvendes i de egne af verden hvor det ikke er muligt at benytte andre former for radio kontakt, som f.eks. en alm. mobil telefon. PTT-enheden benytter i stedet en satellit-antenne til at komme på internettet. Den PABX-server som anvendes i det konkrete system er implementeret ved brugen at Asterisk.

Der kan indhente oplysningerne omkring de tilsluttede enheder i netværket og præsentere dem på operatørens skærmen. Operatøren kan nu vælge at kontakte en bestemt enhed direkte.

Applikationen er udviklet til at blive afvikles på en, Thrane & Thrane fremstillet, Linux-baseret tablet-pc, med navnet "Message Terminal". Selve applikationen er skrevet i programmeringssproget C/C++ samt brugen af Open Source framework'et Qt.

I forbindelse med udviklingen er der implementeret nogle komponenter som på forskellige måder kan bruges til at interagere med en Asterisk-server.
Brugergrænsefladen til applikationen er udviklet specielt ved at anvende Qt Quick. Det grafiske interface er udviklet således at det udnytter de faciliteter som Message Terminalen tilbyder, f.eks. touch-skærm.

Der er udviklet en funktionel applikation der kan indgå i et kommunikationssystem baseret på Thrane & Thrane's PTT-enheder.

Forord

Denne rapport beskriver det afsluttende Diplom-IT eksamensprojekt udført af Casper Skipper Olsen (S081155). Projektet er udført i et samarbejde mellem Danmarks Tekniske Universitet (DTU) og virksomheden Thrane & Thrane A/S (T&T), i perioden 29. august 2011 til 23. januar 2012. Projektet har tilknyttet to vejledere: Hans Henrik Løvengreen (DTU) og Morten Jagd Christensen (T&T)

Jeg vil gerne takke Thrane & Thrane for den store opbakning som jeg har fået undervejs i projektet. Her vil jeg specielt takke Frank Rolsted Jensen (T&T) som har været en stor inspirationskilde. Til sidst vil jeg gerne takke Hans Henrik Løvengreen for den konstruktive sparring undervejs.

Lyngby, 23-Januar-2012

Casper Skipper Olsen

Indhold

Figurer

Tabeller

Kapitel 1

Indledning

Selvom man i dag betragter kommunikation som en naturlig og selvfølgelig del af hverdagen, er der stadigvæk områder af verdenen hvor dette ikke et en selvfølge. Men vil f.eks. i disse områder ikke bare kunne tage sin mobiltelefon frem og lige fortage et opkald. Dette er som regel kun en luksus som mennesker i tæt befolkede områder kan tillade sig.

I de egne af verden hvor der er begrænset mobiltelefondækning har man derfor i mange år været nødsaget til at benytte jordbaseret radiokommunikation. Det kunne f.eks. foregå ved at anvende en mobil VHF/UHF radio. Et sådan radio-systen har desværre sine begrænsninger: Dårlig lydkvalitet, ringe dækning, høje service omkostninger.

Som et alternativ til de almindelige VHF/UHF radio har Thrane & Thrane nu udviklet en PPT-enhed som på en nem måde gør det muligt at kommunikere via satellit. Denne enhed betjenes på samme måde som en traditionel radio men løser mange af de problemer som der kunne opstå i det gamle system. Derudover tilbyder den muligheden for at tilføje mange nye features der kendes fra en almindelig mobiltelefon.

Thrane & Thrane er i dag en af verdens førende producenter af udstyr og systemer til global kommunikation, baseret på avanceret satellit- og radioteknologi. Thrane & Thrane udvikler udstyr inden for 4 indsatsområder: Maritime, aeronautisk, land mobile og jordbaseret systemer.

1.1 Baggrund

Virksomheden har netop lanceret et komplet nyt system til det Maritime marked. Systemet, med navnet ”SAILOR System 6000“, skal varetage det obligatoriske GMDSS system til nød- og sikkerhed som skal være om bord på et skib af hvis størrelse. System 6000 består af en række forskellige typer udstyr, som Thrane & Thrane selv har udviklet og produceret. Et af de produkter som er blevet udviklet i forbindelse med det nye System 6000 er en såkaldt ”SAILOR 6006 Message Terminal“. Dette produkt er en lille Linux-baseret computer med indbygget berøringsskærm. Som et resultat af udviklingen af Message Terminalen har virksomheden set muligheden for at kunne udvide produktsortimentet ved at udnytte det potentiale som ligger i Message Terminalen. Det er derfor et ønske for Thrane & Thrane at få udviklet en række forskellige prototyper af software, som kan afvikles på en Message Terminal. Ideen med disse prototyper er umiddelbart at få afprøvet Message Terminalens kapacitet med henblik alternative anvendelses muligheder.

Thrane & Thrane udvikler primært hardware og er i den forstand ikke leverandører af applikationer. Men med Message Terminalen bliver der åbnet mulighed for at kunne tilbyde kunderne at tilkøbe forskellige programmer som er skræddersyet til at fungere sammen med Thrane & Thrane's øvrige produkter. Det vil sige at det bliver muligt at kunne levere totalløsninger inden for områder hvor virksomheden tidligere har været afhængig af 3.-parts software.

1.2 Problemformulering

Dette projekt omhandler udviklingen af en prototype applikation. Applikationen skal kunne kommunikere med en Thrane & Thrane PTT-enhed.

1.2.1 Begrænsninger

Der er for starten af projektet opstillet nogle rammer omkring de forskellige teknologier der skal anvendes ved udviklingen af applikationen.

- Applikation skal kunne afvikles på en Message Terminal (Linux).
- Brugergrænsefladen skal udvikles ved brugen af Qt-framework'et.
- Der skal anvendes en Asterisk-server som telefoncentral.

1.3 Kapiteloversigt

2 Teknologier
Dette kapitel giver først en beskrivelse af selve projektet. Dernæst beskrives de forskellige teknologier der danner rammen for projektet.

3 Krav
I dette kapitel bliver de forskellige krav til applikationen specificeret. Der bliver defineret tre forskellige typer af krav: Funktionelle, Ikke-funktionelle og Implementering. Derudover bliver der i dette kapitel præciseret hvilken specifikke krav der skal implementeres.

4 Design
Først i dette kapitel beskrives de funktioner, som Asterisk tilbyder, der har haft indflydelse på designet af applikationen. Herefter er der en beskrivelse hvordan de forskellige teknologier og Asterisk-funktioner er blevet udnyttet ved selve designet af applikationen. Kapitlet beskriver designet af: Serversiden, Applikationen og Brugergrænsefladen.

5 Implementering
Dette kapitel beskriver selve implementering af projektet. Kapitlet er opdelt i to afsnit. Først et afsnit som beskriver hvorledes Serversiden er implementeret. Den andet afsnit beskriver implementeringen af Applikationen. Dette afsnit inddelt efter de mindre komponenter som applikationen består af.

6 Test
I dette kapitel er beskrevet hvordan projektet er blevet testet. Der er forklaret hvorledes de enkelte tests, der er blevet udført på de forskellige dele, som systemet består, er blevet udført.

7 Konklusion
Her opsummeres projektet og der konkluderes på resultatet. Herudover giver der nogle forslag til udviklingsmuligheder af applikationen samt nogle løsningsforslag til hvorledes disse udvidelser kan blive implementeret.

Kapitel 2

Teknologier

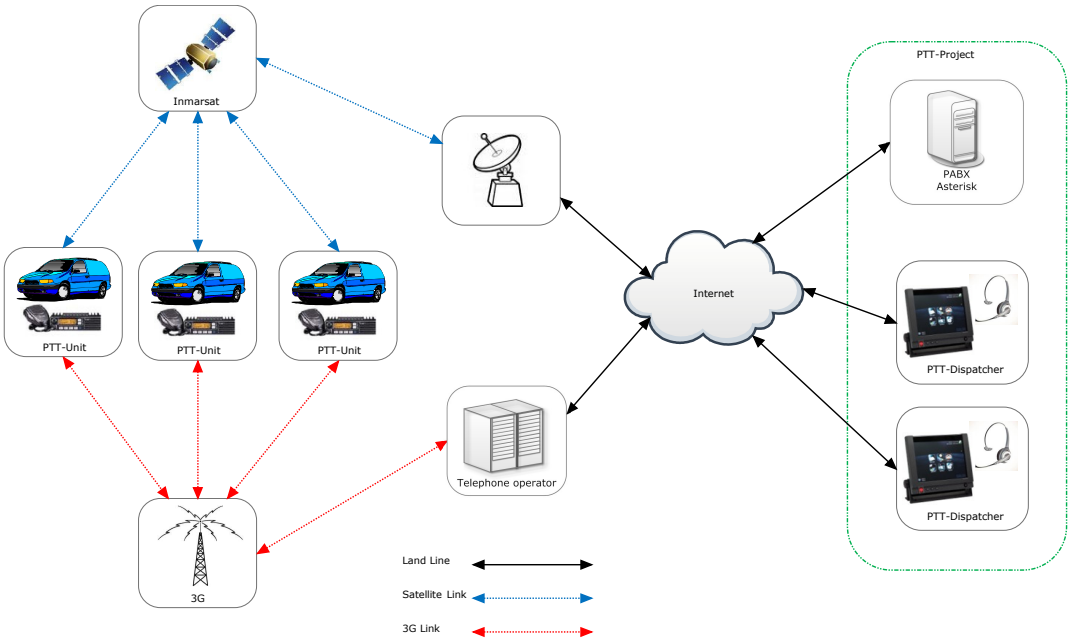
2.1 Produkt Beskrivelse

Prototypen som Thrane & Thrane ønsker at få udviklet har fået arbejdstitlen ”PTT-Dispatcher“ og på dansk ”PTT-Operatør“. Ideen med prototypen er at sætte brugeren i stand til kunne kommunikere med en Thrane & Thrane push-to-talk enhed (PTT-enhed). Push-to-talk er normalt et begreb som anvendes i et ”halv-dupleks“ kommunikationssystem. I sådanne et system kan der foregå kommunikation i begge retninger, men dog kun i den ene retning ad gangen. Grunden til at dette koncept er anvendt ved udviklingen af dette system er at det er meget kostbart at overføre data via satellit. Så for ikke at sende unødige mange data er netop dette koncept implementeret. Man kan også i denne sammenhæng bruge den alternative betegnelse ”Press-to-Transmit“ hvilken antyder at der kun bliver sendt data i det øjeblik der bliver trykket på tasten.

PTT-systemet kunne f.eks. være velegnet i den situation hvor et firma råder over et antal serviceteknikere som skal varetage vedligeholdelse af udstyr, der er placeret på øde steder af verden. Der kunne eksempelvis være tale om elmaster som transporterer strøm over meget lange afstande. PTT-enheden er på den måde beregnet til at fastmontere i en servicevogn eller lignende. PTT-Operatører lokaliseret internt i firmaet, f.eks. på hovedkontoret, kan så anvende applikationen til at kunne kommunikere med de eksternt placerede PTT-enheder.

Applikationen bliver afviklet på Message Terminalen i fuldskærms-mode. Det vil sige at Message Terminalen bliver en slags dedikeret enhed som kun anvendes som PTT-Operatør. Applikationen præsenterer de PTT-enheder som er forbundet til systemet, på en grafisk og overskuelig måde. Message Terminalen betjenes udelukkende via den indbyggede berøringsskærm. Der er altså ikke forbundet hverken mus eller keyboard til terminalen. Brugeren af applikationen kan både modtage og oprette opkald til PTT-enhederne. Message Terminalen

er tilsluttet en ekstern hovedtelefon og mikrofon. Dette gør det muligt at have flere PTT-Operatør applikationer kørende i samme rum samtidige, hvis det er et ønske. Det er også kostbart at sende kommunikation til PTT-enhederne fra PTT-Operatør applikationen. Det er derfor vigtigt at applikationen også implementerer Push-to-talk konceptet. På Figur 2.1 kan ses et eksempel på hvordan et konkret PTT system kan være konfigureret.



Figur 2.1: System Blokdiagram

Både PTT-enhederne og PTT-Operatør applikationen er såkaldte VoIP-enheder (Voice over IP). Det vil sige at de kommunikerer med hinanden via et IP-netværk, som f.eks. internettet. Det resulterer i at alle enhederne i systemet vil fungere på samme måde som man f.eks. kender fra en mobiltelefon. Ønsker to enheder at få forbindelse til hinanden, skal der først fortages et opkald, der så kan besvares, og derved oprette forbindelsen.

På venstre side af internet-skyen er der vist tre PTT-enheder. Disse enheder skal først og fremmest være forbundet til internettet. Det kan lade sig gøre på to måder. Enten via mobiltelefonnettet med det indbyggede 3G-modem, der sidder internt i PTT-enheden. Men er det ikke muligt at få forbindelse til et 3G-netværk kan PTT-enheden komme på internettet via satellit. Dette gøres ved at benytte en ekstern BGAN satellit-antenne udviklet af Thrane & Thrane. På højre side af internet-skyen er afbildet to PTT-Operatør terminaler. Disse er forbundet direkte til internettet via det indbyggede LAN-netkort i Message Terminalen. Hjertet i systemet er en PABX (Private Automatic Branch eXchange) telefoncentral, afbilledet oppe i højre hjørne af figuren. I dette eksempel er der anvendt en softwarebaseret PABX med

navnet Asterisk. PTT-Operatøren og PTT-enhederne skal alle registrere sig, via internettet, hos telefoncentralen som VoIP-enheder. Når man ønsker at komme i kontakt med en anden enhed fortages et opkald til telefoncentralen som så sørger for at skabe forbindelse imellem enhederne.

2.2 Teknologi Baggrund

PTT-Operatør applikationen skal udvikles således at den kan indgå i et system som beskrevet i afsnit Produkt Beskrivelse. Man har hos Thrane & Thrane således allerede fortaget nogle beslutninger om hvilke teknologier som skal anvendes. Nogle af disse teknologier er listet og beskrevet herunder.

2.2.1 Asterisk

Asterisk[6] er en open source software implementering af en PABX. Denne implementering bliver vedligeholdt af firmate "Diginum Inc."[3]. Asterisk kan frit downloades som freeware på deres hjemmeside. Asterisk er Linux-baseret og kan installeres på en standard Linux-server. Asterisk er en telefoncentral hvis primære formål er at kunne rute forskellige telefonlinjer imellem hinanden. Den er både i stand til at håndtere almindelige PSTN (public switched telephone network) telefoner som er tilkoblet et nationalt telefonnetværk samt VoIP-telefoner som er sluttet til server via et IP-netværk. Ønsker man at anvende PSTN telefoner kræver det at den server som afvikler Asterisk har installeret specielt hardware. Serveren som anvendes i dette system har ikke installeret hardware som gør dette muligt.
Asterisk's implementering tilbyder en lang række funktioner som der normalt også er at finde i et mobiltelefonnetværk, såsom: "Voice mail" og "Conference calling". En fordel ved Asterisk, som gør den specielt velegnet til at anvende i dette system er muligheden for at kunne anvende forskellige VoIP-protokoller. Asterisk er et meget fleksibelt system som kan konfigureres på mange forskellige måder, alt efter behov. Asterisk tilbyder f.eks. et "Dialplan" koncept som håndterer alle ind/ud-gående opkald. Denne Dialplan konfigureres med et simpelt script-sprog.

2.2.2 Voice over IP

Voice over IP (VoIP)[9] er et begreb som bruges om overførelse af multimedia informationer via et IP-netværk. VoIP bliver som regel anvendt når der er tale om internet-baseret telefon systemer. VoIP er i sig selv ikke en protokol med derimod en samlet betegnelse for de protokoller som kan bruges til at overføre tale, video elle anden form for multimedia via et IP-netværk. De to VoIP-protokoller som er relevante i forbindelse med dette projekt er IAX og SIP. Disse to er de primære protokoller som anvendes i Asterisk systemer.

VoIP Protokoller

Session Initiation Protocol (SIP)
SIP[10] er en protokol som befinder sig i "program laget" på IP-stakken. SIP kan anvende flere forskellige underliggende transport-protokoller såsom TCP eller UDP. Selve Sip-protokollen er en tekstbaseret protokol på samme måde som f.eks. HTTP. SIP er opdelt i to kanaler. Den ene bruges til kontrol. Denne kanal bliver brugt af klienten til at "forhandle" en forbindelse på plads med serveren. Den anden kanal bruges til at transmittere data (lyd). Denne datakanal benytter Real-time Transport Protocol (RTP). RTP[11] gør det muligt at oprette en real-time datastrøm, end-to-end. RTP er designet specielt til at overfører lyd og video.

Inter-Asterisk eXchange (IAX)
IAX[12] er en protokol udviklet af Asterisk. Den er specielt beregnet til VoIP kommunikation imellem to Asterisk-servere. Til forskeld fra SIP overføres både kontrol og data via samme kanal. IAX supportere "trunking" hvilken vil sige det er muligt at samle (multiplexe) flere igangværende VoIP forbindelser ind på samme kanal. IAX benytter altid UDP som transport-protokol. IAX har et mindre overhead end SIP og så er den NAT transparent.

2.2.3 PTT-enhed

Dette et den enhed som udgør det for selve PTT-enheden. En produktbeskrivelse af PTT-enheden kan ses i Bilag B.1. Den kan forbindes til internette via det indbyggede 3G-modem eller via en ekstern satellit-antenne. Den er meget enkel at betjene. På selve enheden er der kun en knap. Denne knap har flere funktioner. Den bruges både til at oprette et og til at besvare et kald. Enheden er forsynet med en "Fist-mic". Denne fingerer både som højttaler og mikrofon. Derudover er den også forsynet med selve "push-to-talk" tasten. Et billede af PTT-enheden kan ses på Figur 2.2.
Interne i enheden er der implementeret en lille lokal Asterisk-server. Dette betyder at den anvender IAX-protokollen når den registrer sig på selve systemets Asterisk-server.

Figur 2.2: PTT-enhed

2.2.4 Message Terminal

Message Terminalen er en lille PC med indbygget berøringsskærm. En produktbeskrivelse af denne kan ses i Bilag B.2.
Tekniske specifikationer på Message Terminalen:

- **CPU** Intel Atom CPU model Z510 - 1.1 GHz
- **Hukommelse** 1 Gb
- **Harddisk** 2.5” (Standard 2 Gb SSD)
- **Display** 10.4” Touch screen - 800 x 600 pixels - TFT
- **Ethernet** 10/100 Mbit

Et billede af Message Terminalen kan ses på Figur 2.3.

Figur 2.3: SAILOR 6006 Message Terminal

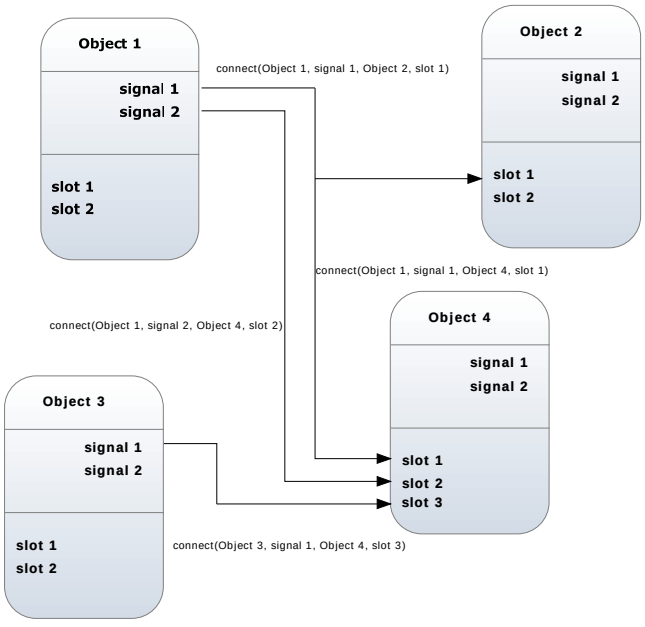
2.2.5 Qt

Man har hos Thrane & Thrane besluttet at anvende Qt-framework’et[1], som i dag ejes og vedligeholdes af Nokia, til de applikationer som skal afvikles på Message Terminalen. Qt er et Open Source framework beregnet til udvikling i C/C++. Dog findes der et utal af software ”binding libraries“ som gør det muligt at anvende Qt i forbindelse med andre programmeringssprog. Thrane & Thrane har først og fremmest besluttet at anvende Qt på grund af den platform som det tilbyder til udvikling af grafiske brugergrænseflader. En anden grund til denne beslutning er at Qt-framework’et tilbyder et udvalg af biblioteker således at det bliver muligt at oversætte (compile) samme kode til flere forskellige platforme. Dette er en fordel hvis man f.eks. på et tidspunkt beslutter at anvende et andet operativsystem i den næste generation af Message Terminalen.

Qt-framework’et tilbyder meget mere end bare grafiske biblioteker. Faktisk kan man betragte Qt som et helt lag oven på C++. Qt-framework’et består af en lang række forskellige

biblioteker. Derudover inkluderer Qt et komplet SDK, med specielle oversættere og udviklingsværktøjer. Herunder er listet nogle af de vigtigste egenskaber inddelt efter bibliotek.

QtCore
Dette er grund-bibliotek i Qt[2]. Det indeholder alle de basale Qt funktionaliteter, såsom specielle Qt-typer og file-IO. I Qt er det muligt at oprette selvstændige program-tråde. Dette gør det muligt at lave flere-trådede programmer. Qt tilbyder en række forskellige synkroniseringskoncepter såsom mutex og semaphore, hvilket gør det simpelt at lave fler-trådede programmer. Qt er et event-baseret system og alle tråde i Qt indeholder deres egen event-kø, som de lytter på. Qt tilbyder en speciel "signal/slot" mekanisme. Dette koncept er beregnet til at kunne kommunikere imellem objekter. Det kan også anvendes til asynkron kommunikation imellem to forskellige tråde. Ideen bag dette koncept er at en klasse kan have implementeret et antal "signaler". Disse "signaler" kan betragtes som events. Signal-events kan så modtages af en eller flere andre klasser. Dette gøres ved at implementere nogle "slots" som kan bruges til at "lytte" på "signaler". Metoden "connect" anvendes til at forbinde et singnal og en slot sammen. Et eksempel kunne være en knap på en GUI. Hver gang brugeren trykker på knappen bliver der sendt et signal. Dette signal kan så fanges af de objekter som lytter på signalet. På den måde slipper man f.eks. for at lave sin egen implementering af et "Observer-pattern". Et eksempel på hvordan signaler og slots kan være forbundet imellem forskellige objekter kan ses på Figur 2.4.



Figur 2.4: Signal/Slot

Til at håndtere alle de muligheder som Qt tilbyder, og som ikke allerede findes indbygget i C++, genererer Qt automatisk nogle såkaldte "Meta-Objekter". Disse objekter indgår som en del af Qt's underliggende infrastruktur. Meta-informationerne angives i selve koden ved hjælp af nogle specielle Qt makroer. Meta-Objekter generes med en medfølgende Qt oversætter med navnet "moc" (Metaobject compiler). Før den egentlige oversættelse af selve Qt programmet, genereres den ekstra infrastruktur-kode med moc-oversætteren, og placeres i nogle moc-filer.

Disse moc-filer indgår herefter, sammen med de øvrige kode-filer, til den endelige oversættelse af programmet.
Der er muligt at oprette properties på en klasse i Qt. Disse properties kan så tilgå direkte eller ved at binde til dem. Properties indgår som en del af Qt meta-objekt systemet. På den måde vil det være muligt runtime at anvende reflection og indhente informationer omkring en enkelte objekter. Properties har også den egenskab at hvis der er oprette en bind til en property vil den som har bindret til den automatisl blive informeret hver gang property'en bliver opdateret.

QtGui
Hvis man ønsker at lave programmer med en grafisk brugergrænseflade skal QtGui biblioteket anvendes. Qt anvender et speciel variation af et klassisk Model-View-Controler pattern til den grafiske del. Qt benytter en mere simpel Model-View arkitektur. Ved denne arkitektur er view- og controler-objektet blevet slået sammen til et view-objekt. Selvom selve controller-objektet er fjernet i denne arkitektur, er den måde som data gemmes på stadig adskilt fra den måde som data præsenteres på. Model-opjektet repræsenterer de data som der arbejdes med i applikationen. View-objektet bestemmer den måde som modellens data skal præsenteres på. Det kunne f.eks. være en liste eller en træstruktur. I stedet for det sædvanlige controller-objekt, som kan ændre på modellens data, har Qt tilføjet et delegate-objekt. Dette delegate-objekt beskriver præcist hvorledes det enkelte data-element skal præsenteres på. Herudover kan de enkelte delegate-objekt også ændre på de data i modellen som de præsentere. Det smarte er at samme data-model-objekt kan anvendes til flere forskellige view's. Qt har som udgangspunkt lavet nogle standard model-, view- og delegate objekter som man kan benytte Det er også muligt at udvikle signe egne objekter som er tilpassede de data man anvender.

QtNetwork
Qt tilbyder en lang række forskellige netværksklasser. Disse klasser arbejder på forskellige niveauer af OSI-stakken. På det højeste niveau findes klasser som QHttp og QFtp. Disse klasser implementerer specifikke netværksprotokoller. Der er også mulighed for at anvende klasser på et lavere niveau. Disse klasser arbejder på socket-niveau. Det er således mulighed for at anvende klasser som QTcpSocket og QUdpSocket.

QtSql
Dette bibliotek giver mulighed for at kunne oprette forbindelse til en database. Qt supporterer en lang række at forskellige databasedrivere f.eks. Oracle og MySql.

Kapitel 3

Krav

Der har fra Thrane & Thrane's side ikke været defineret nogen funktionsmæssige krav til applikationen. Det eneste overordnede krav som der har været til projektet er at applikationen skal kunne kommunikere med en PTT-enhed i et system som beskrevet i kapitel 2. Det har således været en del af projektet at få identificeret og specificeret kravene til applikationen. Der er blevet taget udgangspunkt i Message Terminalen ved definitionen af kravene. Et af formålene med denne prototype-applikation er at få afdækket mulighederne ved Message Terminalen. Så tre overordnede spørgsmål i forbindelse med kravspecifikationen har været:

- Hvilke funktioner skal man som minimum kunne udfører som PTT-operatør?
- Hvilke funktioner kan man forlange Message Terminalen kan håndtere?
- Hvilke forventninger til brugergrænseflade kan man have når en Message Terminalen anvendes?

3.1 Applikationsbeskrivelse

Da systemet kan bestå af mange PTT-enheder er det meningen af de skal kunne opdeles i mindre grupper. Dette skal hjælpe med til at gøre overskueligheden større. Det vil være begrænset hvor mange PTT-enheder en enkelt PTT-Operatør kan have ansvaret for på en gang. De enkelte PTT-Operatører får derfor tildelt en mindre gruppe af PTT-enheder som de skal varetage.

3.2 Aktører

Ser man på systemet som en helhed er der blevet identificeret tre primære aktører. Disse aktører er beskrevet i Tabel 3.1

Aktør	Formål
Worker	En Worker er en person som arbejder ude i marken. Denne person har adgang til en PTT-enhed så han kan komme i kontakt med en PTT-Operatør
Operatør	Operatøren har overblikket over de PTT-enheder som er i den gruppe som han fået tildelt. Operatøren har mulighed for at kontakte en eller flere PTT-enheder i gruppen
Super-user	En Super-user opretter og navngiver nye PTT-enheder i systemet. Derudover har Super-user'en mulighed for at fordele PTT-enhederne ud til de enkelte grupper. Super-user'en er også den person som kan ændre i de settings som er nødvendige for at kunne komme i forbindelse med Asterisk serveren

Tabel 3.1: Aktøre

3.3 Kravspecifikation

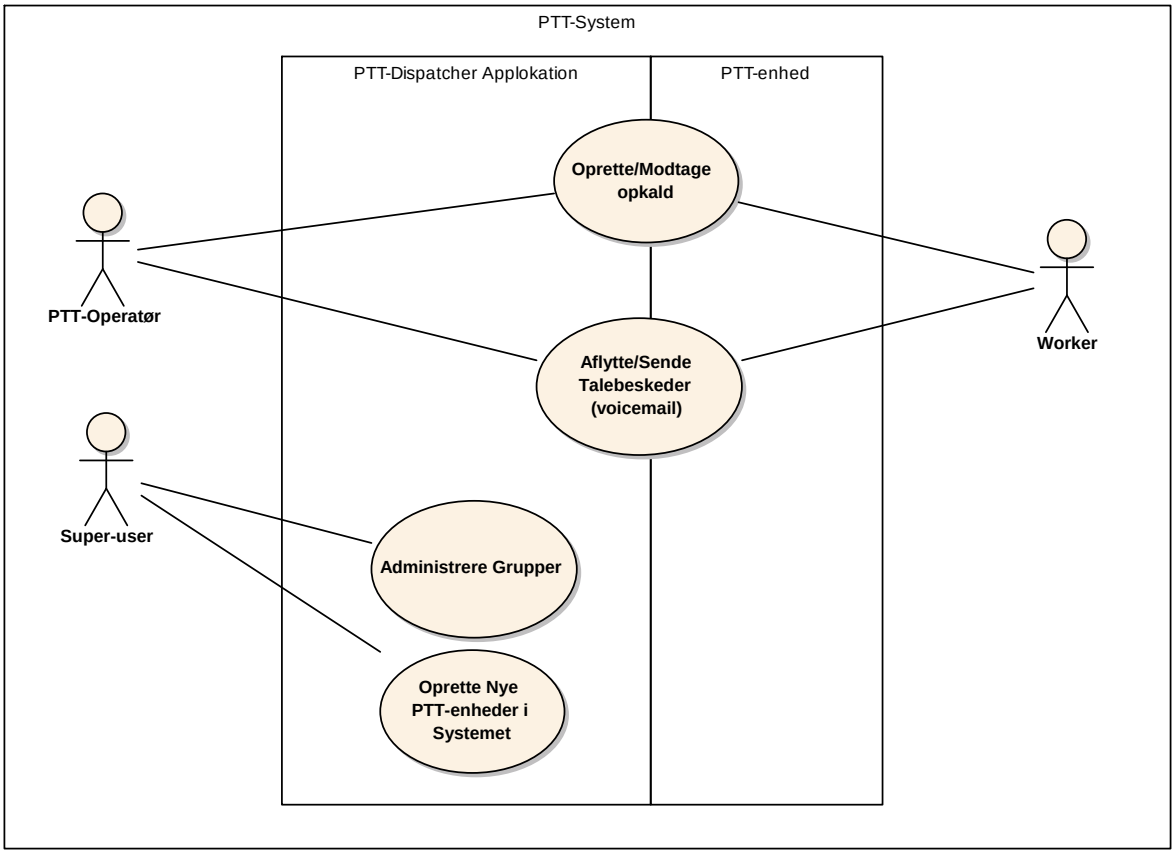
3.3.1 Use-Case model

Til at skabe et overblik over hvilken funktioner som de enkelte aktører skal kunne udfører, med applikationen, er der udarbejdet en Use-Case model. Denne model kan ses på Figur 3.1

For bedre at kunne overskue hvordan systemet fungerer og hvordan de forskellige aktører interagere, er der udarbejdet et eksempel på et use-case-scenarie som beskriver almindelig brug af applikationen. Scenarie: En Operatør vil i kontakt med en Worker. Dette senarie kan ses i Tabel 3.2.

3.3.2 Requirements Model

På baggrund af Use-Case modellen er der udarbejdet en Requirements Model med de specifikke krav til applikationen. Modellen indeholder Funktionelle krav, Ikke funktionelle krav og Implementering.



Figur 3.1: Use-Case model

Funktionelle Krav

De funktionelle krav er inddelt i to kategorier.

- **Basisfunktioner**
  Disse funktioner er et minimum for at applikationen kan fungere i kommunikationssystemet.
- **Udvidelser**
  Disse funktioner er ikke et krav for at applikationen kan anvendes, men derimod en række funktioner som vil kunne forbedre anvendeligheden af applikationen.

Alle de funktionelle krav som man skal kunne udfører med PTT-operatør applikationen er listet i Tabel 3.3. Basisfunktionerne er prioriteret efter vigtighed. Udvidelserne er ikke prioriteret.

Ikke-funktionelle Krav

Alle de ikke-funktionelle krav er listet i Tabel 3.4

Implementering

Tabel 3.5 indeholder forhold som vedrører selve udviklingen af systemet.

Oprette/Modtage opkald
ID: 1
En PTT-Operatør opretter kontakt til en Worker
Primær aktør: Operatør
Sekundær aktør: Worker
Forudsætninger: • PTT-enheden skal have forbindelse til Asterisk serveren. • Operatør-applikationen skal have forbindelse til Asterisk serveren.
Forløb: 1 Operatøren udvælger den PTT-enhed som han ønsker at komme i kontakt med. 2 Applikationen viser detaljer vedrørende den valgte PTT-enhed. 3 Operatøren vælger at oprette forbindelse til den valgte PTT-enhed. 4 Applikationen kontakter Asterisk serveren og beder om at blive forbundet til den pågældende PTT-enhed. 5 Asterisk server giver PTT-enheden besked om at der er 6 PTT-enheden "ringer" 7 Worker'en kan trykker på PTT-enheden og besvare derved opkaldet.
Postkondition: Ingen
Alternativt forløb A: Worker'en er ikke i nærheden af PTT-enheden og kan derfor ikke besvare opkaldet. A7 Asterisk serveren registrerer at opkaldet ikke bliver besvaret og giver Operatøren mulighed for at aflevere en talebesked (voicemail). A8 PTT-enheden indikerer nu at der er en besked. A9 Worker'en kan aflytte beskeden ved at trykke på knappen.

Tabel 3.2: Use-Case: Oprette/Modtage opkald

Både ikke-funktionelle krav og de krav der er specificeret i implementering er fremkommet som et resultat af de begrænsninger og rammer der har været givet på forhånd i projektet.

3.3.3 Brugergrænseflade

Der er ikke nogle krav til brugergrænsefladen. Hverken med hensyn til udseende eller til hvordan de enkelte funktioner skal være tilgængelige for brugerne. Thrane & Thrane har på de applikationer, som allerede bliver afviklet på Message Terminalen i dag, fastlagt præcist hvordan udtrykket skal være. Man da dette projekt handler om at få udviklet en prototype applikation er der ikke nogen grænser for hvordan brugergrænsefladen skal se ud. Der er derimod et ønske om af få afprøvet nogle ny grænser og muligheder. I stedet for nogle klare og præcise krav til brugergrænsefladen er der blevet udarbejdet nogle få visioner om hvordan den skal fremstå over for brugerne. Vision:

- Den skal være overskuelig.
- Den skal være nem at betjene.

ID	KRAV
	Basisfunktioner
FB1	- indhente information omkring hvor mange PTT-enheder der er forbundet
FB2	- indhente status omkring de enkelte PTT-enheder
FB3	- foretage et opkald til en bestemt PPT-enhed
FB4	- modtage indgående opkald fra de enkelte PTT-enheder
	Udvidelser
FU5	- modtage/aflytte voicemail beskeder
FU6	- sende samme voicemail besked til alle PTT-enheder
FU7	- tilføje nye PTT-enheder til systemet
FU8	- administrere de overordnede grupper af PTT-enheder
FU9	- oprette forskellige undergrupper af PTT-enhederne i den pågældende gruppe
FU10	- overskue den aktuelle call-log
FU11	- logge ind med forskellige rettigheder (Operatør/Super User)
FU12	- gemme operatørens bruger-indstillinger
FU13	- foretage et opkald til alle PTT-enhederne i samme undergruppe på en gang
FU14	- oprette forbindelse imellem to PTT-enheder
FU15	- sætte et opkald på standby (parking). Således at et andet opkald kan besvares samtidig.

Tabel 3.3: Funktionelle krav

Der er den overordnede ramme for udviklingen af brugergrænsefladen at det skal gøres ved brugen af Qt-framework'et.

ID	KRAV
IF1	Der skal udvikles en grafisk brugergrænseflade
IF2	Applikationen skal kunne betjenes udelukkende med den indbyggede berørings skærm
IF3	Det skal være mulighed for at kunne tilslutte eksterne hovedtelefoner og mikrofon til audio I/O
IF4	Der skal være en tast som Operatøren kan anvende når der skal sendes tale til PTT-enhederne. Dvs. at applikationen skal implementere phus-to-talk konceptet
IF5	Applikationen skal kunne optræde som en selvstændig VoIP enhed i systemet
IF6	En enkelt PTT-enhed kan kun være tilknyttet en gruppe.
IF7	En PTT-operatøren kan kun varetage en gruppe af enheder ad gangen
IF8	Applikationen skal kunne håndtere grupper på minimum 15 PTT-enheder
IF9	Applikationen skal kunne afvikles på en Message Terminal
IF10	Den PABX som anvendes i systemet skal være en Asterisk-server

Tabel 3.4: Ikke-funktionelle krav

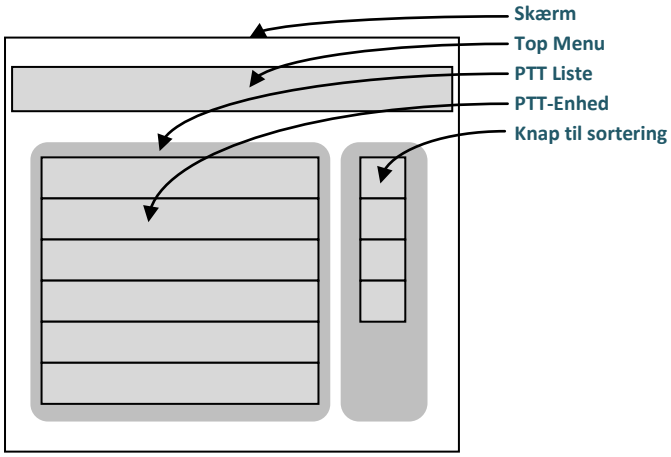
ID	KRAV
I1	Applikationen skal udvikles i C++
I2	Applikationen skal kunne afvikles på et Linux operativsystem
I3	Applikationen skal udvikles med Qt SDK - Qt Creator

Tabel 3.5: Implementering

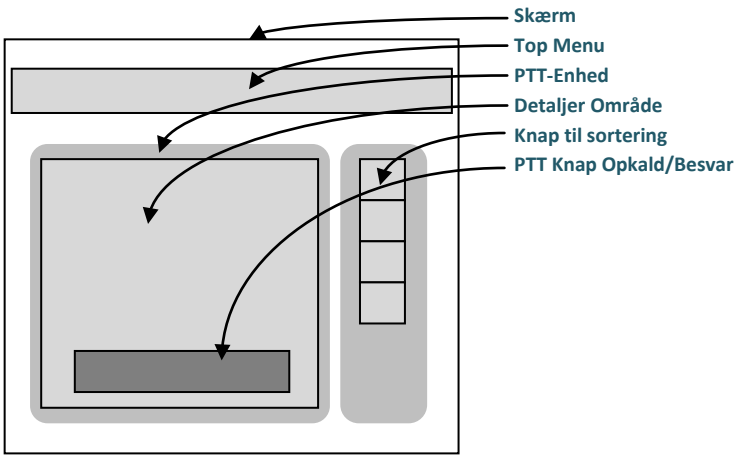
Skitse
Der er udarbejdet to skitser som viser de skærmpræsentationer der skal være i applikationen. Den første præsentation er det primære billede som operatøren skal anvende. Dette billede viser oversigten, i form af en liste, over de PTT-enheder som operatøren varetager. Denne skitse kan ses på Figur 3.2. Den anden præsentation er et billede som bruges til at vise detaljer på en enkelt PTT-enhed. Denne skitse kan ses på Figur 3.3.

3.3.4 Krav/Aktør

Der er udarbejdet en krav-aktør-tabel for at få et bedre overblik over hvorledes de forskellige krav fordeler sig mellem aktørerne. Tabel 3.6 sammenholder de funktionelle krav med aktørerne.



Figur 3.2: Skitse 1. Skærmpræsentation



Figur 3.3: Skitse 2. Skærmpræsentation

3.4 Begrænsninger

Der er ved dette projekt lagt vægt på at udvikle applikationens primære funktion, nemlig at kunne kommunikere med en PTT-enhen. Det er derfor valgt ikke at implementere Super-User-aktørens funktioner. Der antages derfor at de data som applikationen har brug for allerede findes tilgængeligt i systemet. Det kunne f.eks. være tale om selve PTT-enheder eller om administration af de grupper som de enkelte enheder et tilknyttet. Der kunne også være tale om applikations-settings såsom hvilket IP-nummer Asterisk serveren har osv. Endvidere er det besluttet at koncentrere kræfterne i udviklingsforløbet omkring at få udviklet en brugbar applikation med alle basisfunktionerne. Der er derfor også lagt vægt på at få udviklet et design, af applikationen, som nemt vil kunne udvidedes senere, med yderligere funktioner. Med disse begrænsninger kan man ud krav-aktør-tabellen se at de krav der er blevet fokuseret på ved udviklingen af dette projekt, er de fire første FB-krav.

Krav	Operatør	Super-User
FB1	•	
FB2	•	
FB3	•	
FB4	•	
FU5	•	
FU6	•	
FU7		•
FU8		•
FU9	•	
FU10	•	
FU11	•	•
FU12	•	
FU13	•	
FU14	•	

Tabel 3.6: Krav-aktør-tabel

Kapitel 4

Design

Asterisk er en så fleksibel server, at konfigurationen af den er blevet taget med i designovervejelserne ved dette projekt. Dette har været muligt da der ikke er defineret nogen specifikke krav til hvordan Asterisk-server skal konfigureres i systemet.

4.1 Baggrund for design

4.1.1 Grænsefladen til Asterisk

Command Line Interface
Asterisk tilbyder flere forskellige interfaces som kan benyttes til at interagere med serveren. For det første er det muligt at benytte "Asterisk Command Line Interface" (CLI). Dette interface er et ganske almindelige tekst-baseret shell-interface der kan anvendes lokalt på serveren. Med dette CLI er der mulighed for at give serveren en lang række kommandoer. Disse kommandoer er inddelt i nogle forskellige kategorier alt efter hvilken del af serveren de berører. Iblandt de generelle kommandoer kan der f.eks. nævnes: "show dialplan" eller "restart now", som henholdsvis returnerer den aktuelle dialplan eller genstarter serveren. Se en liste med alle CLI-kommandoerne i Bilag A.2

Asterisk Manager Interface
Et andet nyttigt interface som Asterisk tilbyder, er: "Asterisk Manager Interface" (AMI). Med dette interface er det muligt at forbinde til serveren over et TCP/IP netværk og på den måde interagere med serveren. Dette gøres ved at oprette en simpel telnet forbindelse til serveren. Med denne forbindelse oprettet er der mulighed for at sende forskellige kommandoer. Kommandoer i AMI er "pakker" med et antal "key:value" tekst-linjer som definerer

selve kommandoen. Alle linjer skal afsluttes med CR/LF, og en pakke afsluttes med to gange CF/LF. Der findes tre typer kommandoer i AMI: Actions, Response og Events. Klienten sender en action til serveren som så udfører kommandoen og returnerer et response. Et response indeholder typisk kun beskeden "success" eller "failure" alt efter om kommandoen blev udført eller ej. Men i visse tilfælde indeholder response-kommandoen større beskeder.
Den sidste type af kommandoer er event. Events bliver også kun sendt fra serveren til klienten. Som AMI-klient kan man vælge, ved login, at modtage events fra serveren. Events indeholder beskeder om ændringer i den tilstand som serveren har. Det kunne f.eks. være en forbindelse er blevet oprettet imellem to "VoIP-linjer" eller at en "VoIP-klient" har registeret sig hos serveren.

Der er en række forudbestemte kommandoer som kan anvende som actions. Med AMI er det mere begrænset med udvalget af kommandoer end med CLI. Men der findes en speciel action-kommando som hedder `Command`. Med denne specielle kommando er det muligt at sende alle de kommandoer som findes i CLI. En liste med alle AMI-kommandoer kan ses i Bilag A.3

4.1.2 Konfiguration af Asterisk

Den måde som Asterisk serveren rent praktisk konfigureres på er ved at udfylde nogle konfigurationsfiler. Disse filer, som er skrevet i klar tekst, indlæses i det øjeblik serveren startes op. Der findes forskellige områder af konfigurationsfiler. Nogle filer beskriver hvorledes Asterisk skal håndtere de forskellige VoIP- / Telefon-protokoller. Andre beskriver f.eks. hvordan dialplan'en skal fungerer. Nogle filer er bare simple lister. Et eksempel kunne være filen `sip.conf`. Denne fil indeholder en liste med alle de VoIP enheder, som anvender sip-protokollen, der kan registrere sig hos serveren. Et andet eksempel på en, mere avanceret, konfigurationsfil kunne være `extensions.conf`. Dette er den fil som beskriver hele dialplan'en. Denne fil er et script som fortæller serveren hvad der skal ske når en "telefon-klient" ringer et nummer. Man har fra dialplan'en således mulighed for at benytte en række forskellige indbyggede funktioner. Det kunne f.eks. være funktionen "Answer" som besvarer et indkommende kald, eller funktionen "Dial" som sender det indkommende kald videre til en bestemt klient. Hvis en af disse konfigurationsfiler ændres skal de reloades manuelt på serveren.

Asterisk Realtime Architecture
Som et mere fleksibelt alternativ til de tekstbaseret konfigurationsfiler, tilbyder Asterisk en mere avanceret mulighed: "Asterisk Realtime Architecture" (ARA). Denne mulighed tillader at gemme informationerne fra en, elle flere, af konfigurationsfilerne, i en database. Vælges det at anvende ARA kan det gøres på to forskellige måder, enten statisk eller dynamisk. Ved statisk ARA læses konfigurationsinformationerne fra databasen ved opstart. Den eneste forskel er at informationerne bliver læst fra en database i stedet for konfigurationsfilerne. Dvs. at hvis nogle af informationerne ændres skal der her igen reloades. Anvendes dynamisk ARA bliver informationerne læst fra databasen i det øjeblik at de skal bruges. Dette betyder at det er muligt at ændre i informationerne og holde dem ved lige uden at reload eller genstarte serveren.

4.1.3 Asterisk Dialplan

Dialplan'en indeholder en beskrivelse af hvad der skal ske med indkommende og udgående opkald. Asterisk anvender en script-agtig formatering som bruges til at beskrive det forløb man ønsker de forskellige kald skal gennemløbe. Scriptet indeholder også en beskrivelse af hvordan de enkelte forbindelser skal rutes.

Alle de telefoner som kan registrere sig på Asterisk serveren har et unikt id, typisk et nummer. Dette id bruger serveren når der skal oprettes forbindelse imellem to eller flere telefoner. Derudover er enhederne også tilknyttet en bestemt kontekst. Dialplan'en er opdelt i sektioner svarende til disse forskellige kontekster. Når en telefon ringer ind i dialplan'en vil det ske i den kontekst som den hører til. En indgang i dialplan'en svare til der nummer som der bliver ringet fra en telefon. Alle indgange i dialplan'en kan have selvstændige forløb.

Der kan fra dialplan'en kaldes en række forskellige funktioner. Disse funktioner kan f.eks. bruges til at oprette forbindelse imellem to telefoner.

Asterisk Gateway Interface
Hvis man ønsker en mere avanceret dialplan kan man anvende "Asterisk Gateway Interface" (AGI). Med AGI har man mulighed for at udvikle programmer som kan kaldes og afvikles direkte fra dialplan'en. Disse programmer kan være skrevet i en lang række forskellige script udviklingssprog alt efter eget ønske. Det kunne f.eks. være Python, Perl eller Shell-script. I det øjeblik programmet bliver kaldt fra dialplan'en leveres en række forskellige variabler med som argument. Disse variabler kan så tilgås fra programmet. Et eksempel på en variabel kunne være `agi_extension`. Denne variabel indeholder information om det nummer som telefon-klienten har ringet. Alle variablerne kan ses i Bilag A.5. Det er muligt at sende kommandoer til Asterisk serveren fra programmet. Disse kommandoer gives ved at anvende "standard out" kanalen af programmet. Ved at aflæse "standard in" kanalen kan man modtage retur værdierne af kommandoerne. En kommando med navnet `Exec` giver mulighed for at kunne udføre alle de funktioner som der normalt er adgang til fra dialplan'en. En komplet liste med alle action-kommandoerne kan ses i Bilag A.4

4.1.4 PTT-enhed

En egenskab ved PTT-enheden, som har indflydelse på designet af applikationen, er det faktum at alle enhederne ringer samme nummer når de opretter kontakt til serveren. Dvs. at når en PTT-enhed vil i kontakt med en operatør, ringer den op til Asterisk serveren med et forudbestemt nummer. Det er så op til serveren at finde ud af hvilken bestemt operatør som skal modtage opkaldet.

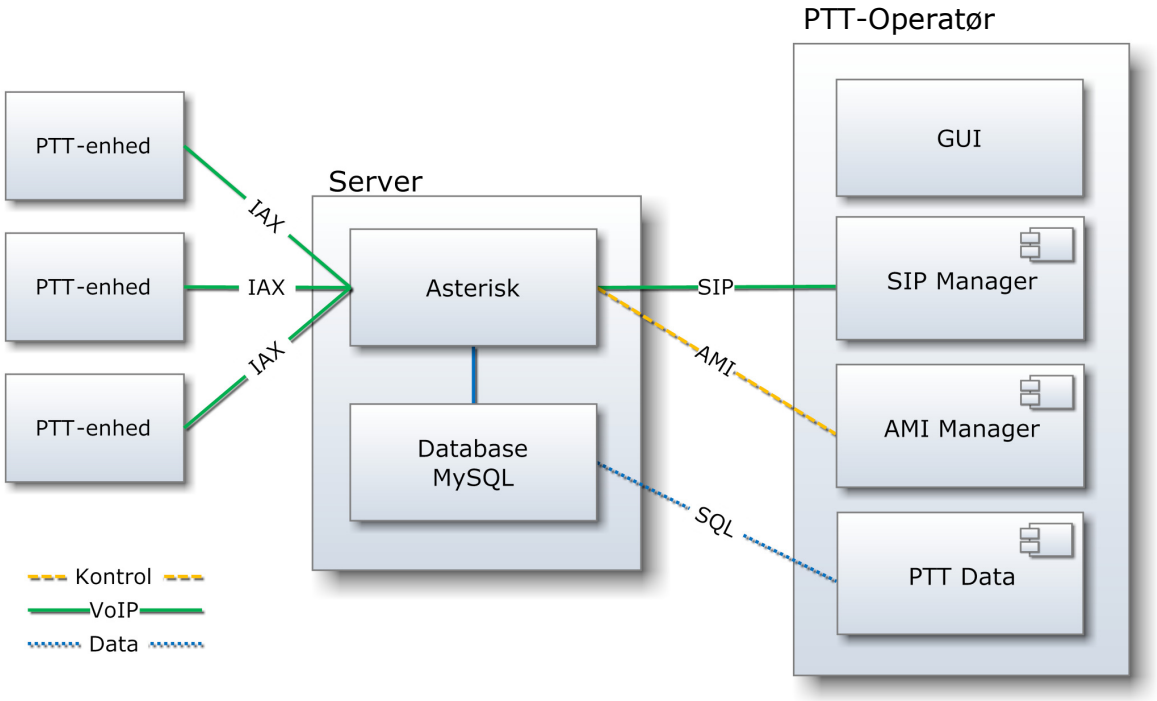
4.1.5 SIP Biblioteker

PjSIP
Dette er en open source implementering af en SIP-stack skrevet i C. Med dette bibliotek er det muligt at optræde som en SIP-telefon overfor Asterisk-serveren. PjSip[8] er godt dokumenteret og med små program eksempler til at illustrere brugen af biblioteket. Der er "liv" i projektet og koden bliver jævnligt vedligeholdt og opdaterer.
PjSip består at en række forskellige biblioteker som tilbyder forskellige funktioner. En af disse funktioner, som er specielt velegnet til udviklingen af netop denne applikation, er muligheden for at kunne interagere direkte med lydkortet i den enhed som afvikler applikationen. Bibliotekets kan også oversættes til forskellige platforme og er derved også specielt velegnet til programmer som benytter Qt-framework'et, da dette også er et multi-platforms-system.

oSIP
Dette er ligeledes et open source projekt som implementerer en SIP-skack. Projekt er dog ikke så aktivt som PjSip. oSIP[7] er skrevet i C. Denne implementering er på et lavere niveau end PjSip. oSIP tilbyder kun selve implementeringen af SIP-stakken. Der er altså ikke mulighed for at interagere med et lyd-kort. Det er desværre ikke så godt dokumenteret.

4.2 Overordnet design

Applikationen er designet på baggrund af de forskellige teknologier. Figur 4.1 viser et diagram over det overordnede design.



Figur 4.1: System Design Diagram

Der er udviklet nogle forskellige komponenter som er beregnet til at interagerer med Asterisk-serveren på forskellige niveauer. En komponent tager sig af selve VoIP kommunikationen: SIP Manager. Et andet komponent tager sig af kontrol af Asterisk-serveren: AMI Manager. Det tredje komponent tager sig af opsætning af Asterisk: PTT Data. Der er tilføjet en database til systemet så det er mulighed for at anvende ARA. Derved bliver det muligt at ændre i konfigurationen af Asterisk realtime.
I de følgende afsnit beskrives først hvorledes de forskellige teknologier anvendes. Herefter bliver designet af serversiden beskrevet og til sidst beskrives designet af applikationen samt de forskellige komponenter som den er opbygget af.

4.2.1 ARA

Informationerne om de PTT-enheder (VoIP-enheder) som kan registrere sig hos Asterisk-serveren ligger i konfigurationsfilerne `sip.conf` og `iax.conf`. Netop informationerne som disse to filer indeholder, er blevet flyttet over i databasen. Dette betyder at når en operatør starter applikationen op skal databasen kontaktes med henblik på at få oprettet en liste af

PTT-enheder. Databasen indeholder data for hver af PTT-enhederne.
Denne information kunne også havde været opnået ved at anvende AMI med det er i dette tilfælde ikke en god løsning. Ulempen ved at anvende AMI til dette er at de kommandoer som skal kan benyttes er CLI kommandoer. Disse kommandoer er beregnet til at returnere informationerne i en kommando-promt. Hvilket betyder at informationerne skal være læsbare for mennesker. Så for at få en applikation til at kunne forstå informationerne skal de først fortolkes. En anden fordel ved at anvende en database, til disse informationer, er i det tilfælde at der skal oprettes nye PTT-enheder. For at kunne gøre dette med AMI vil det kræve meget fortolkning af tekst informationer. Ved databaseløsningen kan der bare blive tilføjet en ny PTT-enhed i tabellen.

4.2.2 AMI

Til at overvåge om de PTT-enheder, som operatøren varetager, er online eller offline, er der her valgt at anvende AMI. En enhed vil gå offline hvis den ikke har adgang til internettet. Hver gang en PTT-enhed skifter status vil der med AMI kunne modtages en event som opfanges af applikationen.

4.2.3 AGI

Når en operatør vil i forbindelse med en PTT-enhed kaldes den metode, på det tilsvarende PTT-objekt, som udfører denne funktion. PTT-objektet ved på forhånd hvilket nummer der skal ringes til for at komme i kontakt med den tilsvarende PTT-enhed. En PTT-enhed ved derimod ikke hvilken bestemt operatør der skal modtage dens opkald. Så når en PTT-ehned vil oprette forbindelse til en operatør ringer den 500 til Asterisk serveren. Denne indgang i dialplan'en kalder et program ved brugen af AGI. Dette program finder ud af hvilken operatør der skal modtage opkaldet. Programmet slår op i databasen for at få returneret oplysninger om hvilken operatør der i øjeblikket varetager den aktuelle PTT-enhed. Når oplysningerne er indhentet sættes PTT-enheden i forbindelse med den givende operatør.

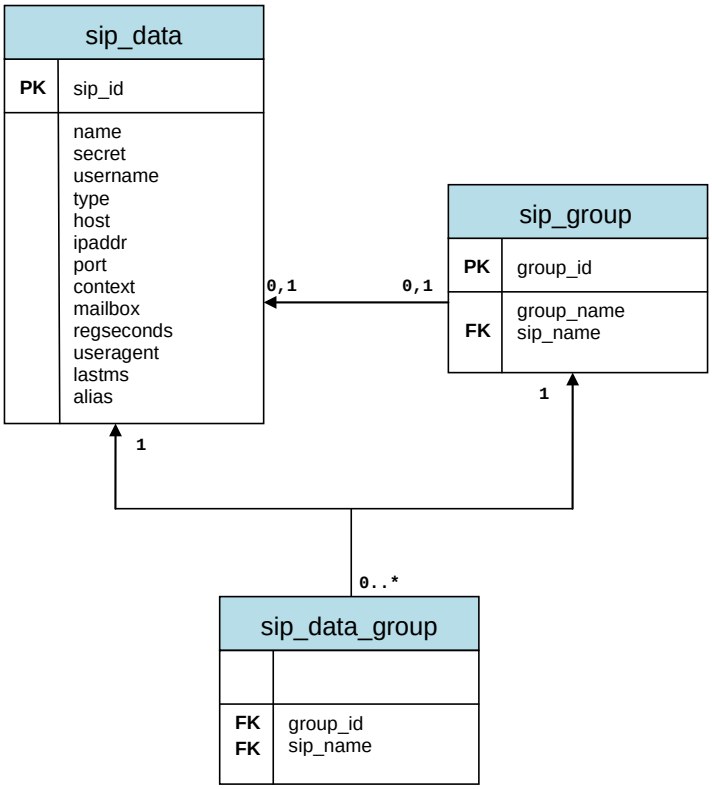
4.2.4 PjSIP

Der er valgt at anvende Pjsip-biblioteket. Selve applikationen kan så registrerer sig hos Asterisk-serveren som en SIP-telefon. Dette gør at applikationen bliver opfattet af Asterisk-serveren som en hver anden VoIP-enhed. Derved kan der drage nytte af de telefon-funkioner som serveren tilbyder på lige vilkår med andre enheder. Dvs. at applikationen f.eks. vil kunne indgå i det voicemail system som Asterisk tilbyder, hvis det ønskes på et senere tidspunkt. Dette gør mulighederne for videreudvikling af applikationen mere fleksibel. Det er muligt, med Pjsip, at få modtaget de forskellige tilstande (SIP-states) som de forskellige PTT-enheder kan være i.

4.3 Serversiden

4.3.1 Database

Databasen indeholder først og fremmest den tabel som Asterisk-serveren bruger. Denne tabel hedder `sip_data` og indeholder alle de data som vedrører de SIP-enheder som kan registrere sig på serveren. Felterne i denne tabel er valgt på baggrund af hvad Asterisk som minimum kræver. Der er ikke blevet oprettet en tilsvarende tabel med alle IAX-enhederne, da der i denne første prototype ikke skal tilsluttes nogle IAX-enheder (de rigtige PTT-enheder bruger IAX-protokollen. Men ved udviklingen af denne applikation er der blevet anvendt en software SIP-enhed, med navnet `X-Lite 4`, som erstatning). Udover den tabel som Asterisk skal bruge er der yderligere to tabeller som gør det muligt at oprette grupper. Den første tabel, `sip_group` indeholder informationerne om de grupper der findes i systemet. Den anden tabel, `sip_data_group` indeholder sammenhængen mellem de enkelte PTT-enheder og den gruppe som de tilhører. En relationelt database model kan ses på Figur 4.2.

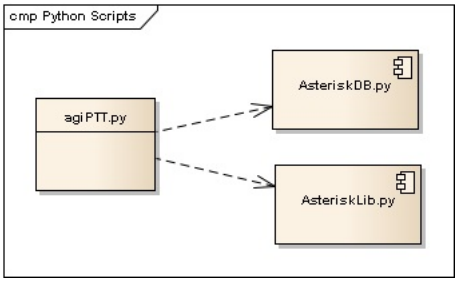


Figur 4.2: Database Diagram

Hver gang en PTT-enhed registrer sig eller afregistrer sig på serveren bliver `sip_data` tabellen opdateret af Asterisk. Dvs. at alle de PTT-enheder som er online har tilknyttet et IP-nummer og et port-nummer i tabellen. Det er kun Asterisk-serveren som ændrer på de data som ligger i `sip_data`.

4.3.2 AGI

Der anvendes AGI i det tilfælde at en PTT-enhed vil i forbindelse med en operatør. Dette AGI-script er skrevet i sproget Python. Det er valgt da der findes mange forskellige tilhørende biblioteker som kan anvendes. Der er udviklet to klasser som er specielt velegnede til denne applikation og som bruges i script'et. Den første klasse med navnet `AsteriskLib` gør det muligt at sende AGI-kommandoer til, og modtage retur beskeder fra, en Asterisk-server. Det er ikke alle AGI-kommandoer som er blevet implementeret, men hele funktionaliteten som gør det muligt er implementeret, så hvis man har behov for det kan der simpelt blive tilføjet nye kommandoer. Den anden klasse, med navnet `AsteriskDB`, gør det muligt at oprette en forbindelse til en MySql server. Denne klasse anvender et Python-bibliotek med navnet `MySQLdb` til at interface med databasen.
Det specielle ved `AsteriskDB`-klassen er at det er muligt at få returneret nummeret til den operatør som PTT-enheden skal i forbindelse med. Dette køres med metoden `findDispatcherFromExt`. Denne metode slår op i databasen og finder frem til den operatør som varetager den PTT-enhed som har ringet til Asterisk serveren.



Figur 4.3: AGI Pakkediagram

4.4 Applikation

Til selve udviklingen af applikationen er der lavet tre komponenter som varetager hvert sin funktion. Fordelen ved at opdele de forskellige funktioner i forskellige komponenter er at man på den måde får lavet en helt klar opdeling af funktionerne i applikationskoden. Koden bliver nemmere at overskue og de enkelte komponenter kan også testes hver for sig. De enkelte komponenter vil også kunne anvendes sammen eller hver for sig ved udvikling af andre programmer. Da der i dette projekt kun fokuseres på at udvikle basiskravene, er der lagt vægt på at komponenterne er designet således at de forholdsvis nemt kan udvides med yderligere funktioner senere.

Komponenterne har nogle fælles overordnede egenskaber. Først og fremmest er de skrevet som Qt-komponenter. Dvs. at de anvender de forskellige komponenter og muligheder som Qt-framework'et tilbyder. Derved kan de kun anvendes til udvikling af Qt-baseret programmer. Det primære interface til komponenterne er således Qt-signaler. Dernæst har alle komponenterne en selvstændig programtråd. Denne egenskab er specielt vigtig ved udvikling af programmer som anvender en grafisk brugerflade. Da komponenterne arbejder med deres egen

tråd vil de ikke kunne ”fryse“ afviklingen af brugergrænseflade-tråden.

4.4.1 AMI Manager

Denne komponent kan anvendes til at oprette en AGI forbindelse til en Asterisk server. Selvom denne komponent, i denne applikation, kun anvendes til at lytte på indkommende ”events“, er al funktionaliteten som gør det muligt at sende ”actions“ til serveren og modtage både ”response“ og ”events“ fra serveren, blevet implementeret. Det er dog kun et begrænset antal af kommandoer som er blevet implementeret, da de ikke skal bruges. Men designet er lavet således at der simpelt kan udvides med yderligere kommandoer al efter behov.

AMI Manager-komponenten arbejder internt med nogle kommando-objekter. Der findes tre typer af kommando-objekter: ”Action“, ”Event“ og ”Response“. Selve komponentet opretter en telnet forbindelse til Asterisk-serveren som både bliver brugt til at sende og modtage kommandoer på. Hver gang AMI Manager'en modtager en event, fyrer den et Qt-signal, svarende til eventen, som så kan opfanges af den kode som benytter komponenten.

For at sende en action til serveren skal der først oprettes et action-objekt. De enkelte action-objekter indeholder alle de informationer som knytter sig til den specifikke action. Der er kun implementeret to forskellige actions som det er muligt at sende til serveren. Den første er: `SIPShowPeersActions`. Denne action kan bruges til at få returneret en liste med alle de SIP-enheder som kan registrerer sig på Asterisk-serveren, samt deres nuværende status. Den anden action hedder: `OriginateAction`. Denne action kan bruges til at bede Asterisk-serveren om af oprette forbindelse imellem to telefoner.
Der er implementeret to forskellige events som AMI Manager'en lytter på. Der første hedder: `PeerEntryEvent`. Dette er den event som bliver returneret, fra Astreisk-serveren som response, på en `SIPShowPeersActions`. Det andet event: `PeerStatusEvent`, er det som bliver brugt i denne applikation. Denne event bliver fyret fra Asterisk-serveren hver gang en enhed registrere eller afregistrere sig på servere.

4.4.2 PTT Data

Denne komponent er et interface til databasen. Der er kun muligt at interagere med en MySql database. Med denne komponent er det muligt at oprette, rette og slette data i tabellen `sip_data`. Dette gøres henholdsvis med funktionerne: `createSipData`, `updateSipData` og `delSipData`. Hvis man ønsker at få data ud omkring en bestemt SIP-enhed skal funktionen `getSipData` anvendes.
Der er ikke mulighed for at ændre på de to andre tabeller som findes i databasen: `sip_group` og `sip_data_group`. Men det vil være forholdsvis enkelt at implementere dette senere da selve funktionaliteten til at interagere med databasen er implementeret. `sip_data` tabellen indeholder informationerne for hvert af de SIP-enheder som kan registrere sig på Asterisk-serveren. Det er muligt at få disse informationer ud, af databasen, repræsenteret i nogle entitets-objekt. Disse objekter hedder `SipData`. Der er implementeret to funktioner som er

specielt beregnet til denne applikation. Den første, `getGroupList`, returnerer en liste med `SipData`-objekter som repræsentere den liste af PTT-enheder som en bestemt operatør varetager. Den anden funktion `getDispatcher` returnerer et `SipData`-objekt som repræsentere informationerne omkring den operatør som varetager en bestemt PTT-enhed.

4.4.3 SIP Manager

Denne komponent udgør det for en SIP-telefon. Kernen i komponenten er Pjsip biblioteket. Man kan betragte SIP Manager som et slags Qt-interface til Pjsip. Med komponenten kan man initialisere de forskellige SIP-egenskaber der ønskes. Herefter er det muligt at registrere sig som en SIP-telefon på en telefonserver. Der er implementeret tre funktioner som kan anvendes til at interagere med telefonserveren på. Den første funktion, `call_make` gør det muligt at oprette et opkald til serveren. Med de to andre funktioner, `call_answer` og `call_hangup`, er det muligt henholdsvis at besvare et indkommende opkald eller afslutte et igangværende opkald. Det er også muligt at fange nogle forskellige SIP-hændelser med denne komponent. De hændelser som der er implementeret er: `incoming_call`, `call_state` og `new_log_message`. `incoming_call`-signalet fyres når der er et indkommende kald. `call_state`-signalet fyres hver gang en af de telefoner, som er registeret på telefonserveren, skifter tilstand (SIP-State). Det sidste signal, `new_log_message`, bliver ikke brugt til SIP-hændelser, men er et signal som kan anvendes til at fange log-beskeder genereret interne i Pjsip biblioteket. Dette signal kan være nyttigt f.eks. i en debug situation. Ikke alle SIP-hændelser er blevet implementeret. Men der er gjort klar i koden således at de nemt kan implimenteres. Alle hændelserne fanges fra Pjsip, men ikke alle fortolkes og sendes videre som et signal.

4.5 Brugergrænseflade

4.5.1 Qt Quick

Ved udviklingen af denne prototype-applikation er det blevet valgt at anvende en nyere version af Qt, end den som Thrane & Thrane normalt anvender. Dette er valgt pga. der i de nyere versioner af Qt er tilføjet en ny måde at udvikle grafiske brugergrænseflader på. Denne nye måde har fået betegnelsen ”Qt Quick“. Qt Quick er blevet udviklet specielt med henblik på at kunne udvikle moderne interfaces som skal benyttes i f.eks. mobile enheder eller andre indlejrede enheder med grafiske brugergrænseflader, såsom en ”set-top box“. Da et af de mere overordnede betingelser med dette projekt er at få afprøvet grænserne for Message Terminalen, er der derfor valgt at udvikle brugergrænsefladen ved brugen af Qt Quick.

4.5.2 QML

Ved udvikling af interfaces med Qt Quick skal selve koden skrives i programmeringssproget "Qt Modeling Language" (QML). QML er et deklarativt sprog. Et deklarativt programmerings paradigme er det diametrale modsatte af et imperativt paradigme. Ved deklarativt sprog skal der beskrives hvad der skal ske, men ikke hvordan det skal ske. C++ og Java er eksempler på imperative programmeringssprog. Disse sprog er generelle og kan anvendes til udvikling af alle typer af computerprogrammer. QML er derimod domæne specifikt og kan kun bruges til at udvikle Qt Quick programmer. QML er script som fortolkes i det øjeblik at de afvikles. Syntaksen som anvendes minder om JacaScript.

QML er beregnet til at beskrive hvordan selve brugergrænsefladen skal se ud. Det er muligt at anvende et antal foruddefinere elementer såsom firkanter og tekster. Disse elementer har tilknyttet nogle "properties", som farve og størrelse, som kan ændres efter behov. Derudover er der mulighed for at beskrive hvordan disse elementer skal opfører sig, f.eks. ændre form eller farve, i forskellige situationer.

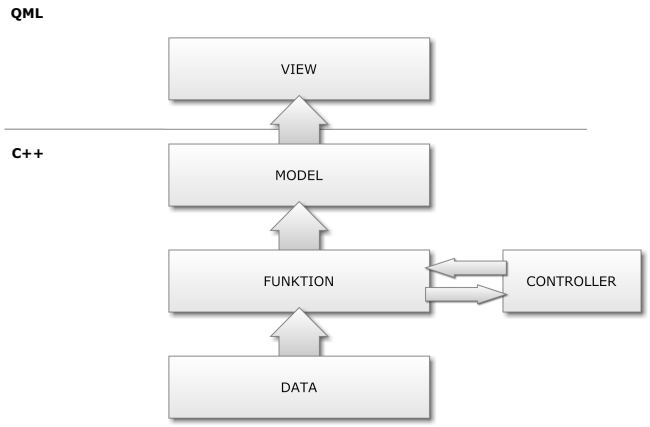
Man kan vælge at udvikle selvstændige Qt Quick programmer udelukkende skrevet i QML. Men hvis man ønsker en mere avanceret logik, i disse programmer, er der muligt at tilføje JavaScript kode. Endeligt er der også mulighed for at interagere med QML elementer fra kode skrevet i C++ igennem Qt framework'et. Qt tilbyder et bibliotek med navnet "Qt Declarative". Med dette bibliotek er der mulighed for at kunne fortolke og vise QML-scripts samt at "binde" data og view sammen.

4.6 PTT Operatør

Selve applikationen er opbygget af forskellige lag. Dette er gjort af flere grunde. Først og fremmest bliver koden lettere at overskue og vedligeholde. Dernæst er det muligt at kunne genbruge de forskellige lag til udvikling af andre programmer. Til sidst er der også mulighed for helt at kunne udskifte et eller flere af lagede. Der kunne f.eks. være tale om at udvikle en anden brugergrænseflade eller måske udskiftning af datalaget. Figur 4.4 viser modellen som den ser ud for denne applikation. Der er blevet taget udgangspunkt i en traditionel 3-lags model ved udviklingen af designet af applikationen.

Det nederste lag i denne model er datalaget. Dette lag svare til de dataobjekter som man får returneret fra databasen. Disse objekter indeholder det data der knytter sig til de enkelte PTT-enheder.

Det næste lag er funktionalitetslaget. Dette lag består af nogle objekter som udnytter Qt-framework'ets muligheder, såsom signaler. Disse objekter bliver oprettet på baggrund af dataobjekterne. Objekterne har tilknyttet et status-felt som fortæller hvilken tilstand de er i. Denne status-information kunne f.eks. afspejle om PTT-enheden er online eller offline. Objekter indeholder al den funktionalitet som knytter sin til de enkelte PTT-enheder. Objekter implementerer således nogle forskellige funktioner som f.eks. skal benyttes til at oprette et



Figur 4.4: Main Model

kald til en PTT-enheden.

Hjertet i modellen er Controller'en. Denne kasse kontrollerer selve applikationens flow. Controller'en interagere fortrinsvis med funktionalitetsobjekterne. Den indeholder et interface til hvert af de tre komponenter: SIP Manager, AMI Manager og PTT Data. Controller'en er den enhed som står for at indhente de forskellige data fra databasen og oprette de forskellige objekter som indeholder disse. Der gælder både objekterne i data- og funktionslaget. Selve main tråden får returneret objekterne fra Controller'en.
Når der sker en hændelse enten fra SIP Manageren eller fra AMI Manageren fortolker Controller'en denne hændelse. Da Controller'en har adgang til objekterne i funktionslaget opdatere den disse objekter direkte ved at ændre deres status. På den måde foregår al interaktion imellem main-tråden og Cntroller'en igennem objekterne i funktionslaget. Hvis der f.eks. kommer et indkommende kald til applikationen, vil liv Controller'en opdatere statussen på det aktuelle objekt. Controller'en selv kommer ikke med nogen form for indikation herom.

De to sidste lag udgør tilsammen det grafiske lag. View består af de QML-scripts der beskriver selve det grafiske udseende af applikationen. Modellaget indeholder de modeller som Qt-framework'et skal bruge til at "binde" mellem QML og C++. Selve modellerne indeholder nogle grafiske dataobjekter som skal præsenteres på skærmen. Denne type af objekter bliver kalde for delegate-objekt i Qt. Delegate-objekter oprettes på baggrund af de objekter som findes i funktionslaget. De grafiske delegate-objekter indeholder den funktionalitet som kræves af Qt-framework'et for at kunne interagere med den via QML.

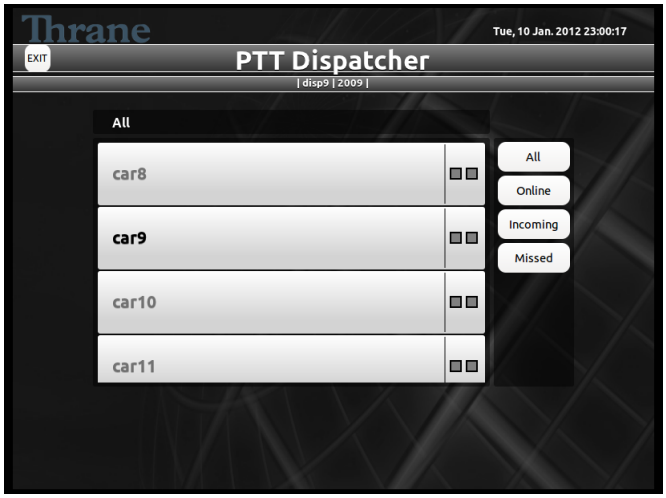
Main-tråden opretter først og fremmest et grafisk vindue som bruges til at afvikle og vise QML-script'et. Derudover sørger den for at få initialiseret og startet Controller'en op. Til sidst sørger main-tråden for at skabe de forskellige sammenhænge mellem modellen og funktionslaget.

4.6.1 Grafiske udseende

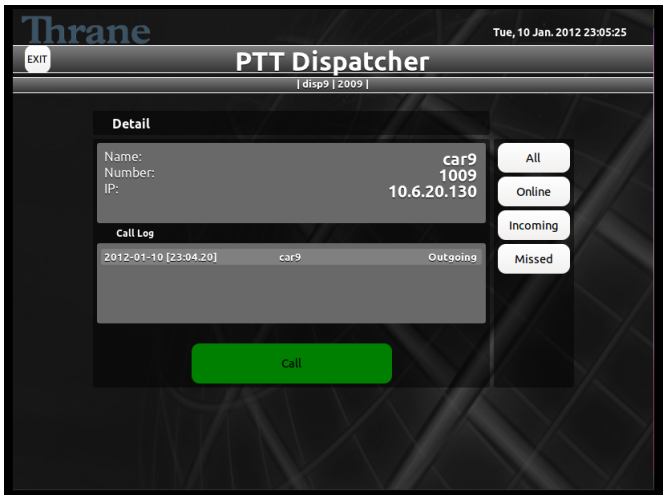
Der er udviklet to forskellige skærmpræsentationer som applikationen kan vise. Disse præsensationer består først af et billede der viser en liste med alle de PTT-enheder som operatøren varetager. Denne kan ses på Figur 4.5. Det er muligt at modificere denne præsentation så listen kun består af nogle forskellige udvalgte PTT-enheder. Det er således mulighed for at vælge imellem at få vist enten:

- Alle enheder.
- Online enheder.
- Enheder som i øjeblikket vil i kontakt med operatøren.
- De enheder som har prøvet at komme i kontakt med operatøren men ikke er blevet besvaret.

Den anden præsentation er et billede som viser de data som knytter sig til en bestemt enhed. Denne kan ses på Figur 4.6. Det er også i denne præsentation hvor det er muligt enten at fortage et opkald eller besvare et opkald fra den aktuelle enhed.



Figur 4.5: 1. Skærmpræsentation



Figur 4.6: 2. Skærmpræsentation

4.7 Design Konklusion

Designet som er udviklet på baggrund af de valgte teknologier og de funktioner som Asterisk tilbyder. Det udarbejdede design imødekommer umiddelbart en række af de definerede krav.

FB1
Der er tilføjet en database til systemet som indeholder informationerne om alle PTT-enhederne (og Operetøere). Derudover er der oprettet nogle tabeller i databasen som kan indeholde informationerne om hvilken grupper, af PTT-enheder, der er i systemet. For at tilgå de informationer som findes i systemet er PTT-Data komponenten udviklet. Med denne komponent kan informationerne omkring de PTT-enheder, som varetages af en bestemt operatør, hentes.

FB2
Til at overvåge den aktuelle status af de enkelte PTT-enheder er der udviklet to biblioteker. Det første hedder AMI Manager. Dette bibliotek indhenter information omkring hvilken PTT-enheder der er online og hvilke der er offline. Det andet bibliotek SIP Manager indhenter information om hvilken SIP-status de enkelte PTT-enheder har.

FB3/FB4
Med SIP Manager'en kan applikationen registrere sig som en SIP-telefon på Asterisk-serveren. Når først applikationen er registeret er det muligt at oprette og modtage opkald.

IF1/IF2
Disse krav bliver imødekommet ved at gøre brug af de funktioner som Qt Quick tilbyder.

IF3
Ved at anvende Pjsip biblioteket bliver det muligt at tilgå de lyd- indgange/udgange som der er tilgængelige på Message Terminalen.

IF4

Det er valgt ikke at implementere en egentlig knap funktion på applikationen. Dette krav bliver imødekommet ved at benytte en ekstern mikrofon som har en knap til at slutte/bryde for lyden. Egenskaberne ved SIP-protokollen bevirker at hvis der ikke komme lyd (tale) fra mikrofonen så bliver der heller ikke transmitteret data.

IF5

Dette er løst ved at anvende Pjsip bibkioteket.

IF8

Da der, ved den grafiske præsentation af PTT-enhederne, er tale om en liste på skærmen er der umiddelbart ikke nogle grænse for hvor mange enheder den enkelte operatør kan varetage.

Kapitel 5

Implementering

5.1 Serversiden

5.1.1 Asterisk Konfiguration

Dette system er implementerer ved brugen af Asterisk version 1.8

Konfigurationsfiler
De filer som bruges til at konfigurere Asterisk-serveren er som standard placeret i mappen "/etc/asterisk" på serveren. En liste over alle konfigurationsfilerne kan ses i Bileg A.1. Listen er inddelt i nogle undergrupper af konfigurationsfiler som har en sammenhæng.

Den fil der bruges til at konfigurere hvordan AMI skal sættes op hedder `manager.conf`. På Figur 5.1 ses et udsnit af `manager.conf`

Filen er inddelt i to sektioner. Først en generel sektion og herefter en sektion som beskriver selve brugeren. I den generelle sektion kan det ses at AMI er aktiveret. Da `bindaddr` er sat til `0.0.0.0` betyder det at alle IP-numre har mulighed for at oprette forbindelse til Asterisk-serveren via AMI. Dette skal gøres på port `5080`. `displayconnects` variablen fortæller om der skal gives besked til konsollen hår en AMI forbindelse oprettes.

Den anden sektion hedder `ami`. Dette er det login som kan anvendes når der oprettes forbindelse. Password'et som denne bruger skal anvende er `astproj`. Variablerne `read` og `write` indeholder en beskrivelse af hvilken rettigheder denne bruger har. I dette tilfælde er alle rettigheder blevet tildelt. Dette er valgt i udviklingsfasen af applikationen, men er nok ikke

```
1  [general]
2  displayconnects = yes
3  enabled = yes
4  port = 5038
5  bindaddr = 0.0.0.0
6
7  [ami]
8  secret = astproj
9  read  = all
10 write = all
11
12 ;system,call,log,verbose,agent,command,dtmf,user,config,reporting,originate
```

Figur 5.1: manager.conf

hensigtsmæssigt i det færdige system. Det er muligt at oprette flere AMI-brugere ved at tilføje den til listen på samme måde som `ami` sektionen.

Der er ændret i to konfigurationsfiler for at aktivere Realtime database adgang. Den første fil hedder `res_config_mysql.conf`. Denne fil konfigurere hvorledes Asterisk-serveren skal forbinde til en MySql database. På Figur 5.2 kan ses et udsnit af filen.

```
1 [general]
2 dbhost = 127.0.0.1
3 dbname = pttproject
4 dbuser = asterisk
5 dbpass = astproj
6 dbport = 3306
7 dbsock = /var/run/mysqld/mysqld.sock
8 dbcharset = latin1
```

Figur 5.2: res_config_mysql.conf

I denne fil er der kun en generel sektion. Her opstilles de paremetre som skal bruges til at forbinde til en bestemt MySql database. Først og fremmest skal IP-nummeret på databasen angives. I dette tilfælde er det `127.0.0.1`, da databasen fysisk er placeret på samme server som Asterisk. Portnummeret som databasen anvender er `3306`. Herudover skal navnet på den specifikke database angives samt brugeroplysningerne på den bruger som skal anvendes. Databasenavnet er `pttproject` og den bruger som anvendes er `asterisk` og password'et er `astproj`. Tilsidst skal angives hvilken driver som der skal bruges samt tegnsættet på databasen.

Den anden fil `extconfig.conf` fortæller Asterisk-serveren hvilken konfigurationsfiler som skal hentes fra databasen, og altså ikke fra den konfigurationsfil som ligger på Asterisk-serveren. Der er muligt at flytte alle konfigurationsfiler over på databasen hvis det ønskes. Men i dette tilfælde er det kun informationerne som filen `sip.conf` indeholder som ligger på databasen. På Figur 5.3 ses er et udsnit fra `extconfig.conf`.

Denne fil indeholder en sektion med navnet settings. I denne sektion er listes alle de filer som

```
1 [settings]
2 sippeers => mysql,general,sip_data
```

Figur 5.3: extconfig.conf

er blevet flytter over på databasen. Denne specifikke fil fortæller Asterisk at listen med de sip-enheder (`sippeers`), der kan registrere sig på serveren, skal hentes i en MySql database og at de brugerinformationer der skal benyttes til at forbinde til databasen ligger i filen `res_config_mysql.conf` under sektion: `general`. Til sidst skal navnet på tabellen angives.

5.1.2 Databasen

Databasen indeholder tre tabeller: `sip_data`, `sip_group` og `sip_group_data`. Den første tabel `sip_data` indeholder alle de data som Asterisk-servern skal bruge angående de forskellige sip-enheder. Denne tabel kan ses på Figur 5.4

```
1  CREATE TABLE pttproject.sip_data(
2    sip_id INT(11) NOT NULL AUTO_INCREMENT,
3    name VARCHAR(80) NOT NULL,
4    secret VARCHAR(80) DEFAULT NULL,
5    username VARCHAR(80) NOT NULL,
6    type ENUM('user', 'peer', 'friend') NOT NULL DEFAULT 'friend',
7    host VARCHAR(31) NOT NULL DEFAULT 'dynamic',
8    ipaddr VARCHAR(15) DEFAULT NULL,
9    port SMALLINT(5) UNSIGNED NOT NULL DEFAULT 0,
10   context VARCHAR(80) DEFAULT NULL,
11   mailbox VARCHAR(50) DEFAULT NULL,
12   regseconds INT(11) NOT NULL DEFAULT 0,
13   useragent VARCHAR(20) DEFAULT NULL,
14   lastms INT(11) NOT NULL DEFAULT 0,
15   alias VARCHAR(80) DEFAULT NULL,
16   PRIMARY KEY (sip_id),
17   UNIQUE INDEX name (name),
18   UNIQUE INDEX sip_id (sip_id)
19 )
```

Figur 5.4: sip_data

Hver enkelt record i denne tabel har et unikt id med navnet `sip_id`. Dette id er primær nøgle i tabellen. Id'et er fortløbende og bliver automatisk tilføjet når en ny record oprettes. `name` værdien bliver brugt til at identificere den enkelte enhed. Dette `name`, er et nummer, og det er det som referere til en bestemt enhed når Asterisk-serveren skal oprette forbindelse imellem to enheder. Variablen `name` er unik, da det ikke skal være muligt at kunne oprette flere enheder med samme nummer. Der er tilføjet en ekstra værdi til tabellen, `alias`. Det er meningen at denne værdi kan indeholde et mere sigende navn på de enkelte enheder, hvilket vil gøre den lettere at overskue og skelne imellem dem. `context` variablen definere den overordnede indgang (sektion) i dialplan'en hvor enheden skal være placeret.

Den næste tabel hedder `sip_group`. Denne tabel indeholder informationerne om de grupper af PTT-enheder som der findes i systemet. Tabellen har som sådan ikke noget med Asterisk at gøre. Tabellen kan ses på Figur 5.5

```
1 CREATE TABLE pttproject.sip_group(
2   group_id INT(11) NOT NULL AUTO_INCREMENT,
3   group_name VARCHAR(255) DEFAULT NULL,
4   sip_name VARCHAR(80) DEFAULT NULL,
5   PRIMARY KEY (group_id),
6   INDEX FK_sip_group_sip_data_name (sip_name)
7 )
```

Figur 5.5: sip_group

`sip_group` tabellen indeholder tre variabler. Først et unikt id som bruges som primær nøgle. Næste variabel, som hedder `group_name`, indeholder navnet på den enkelte gruppe. Til sidst er der en variabel med navnet `sip_name`. Denne variabel fortæller hvilken operatør som har ansvaret for gruppen. Denne variabel har `sip_name` fra tabellen `sip_data` som foreign key. Dette beskytter systemet imod at knytte en operatør, som ikke findes, til en gruppe. Derudover er denne variabel unik hvilket betyder at en operatør kun kan varetage én gruppe ad gangen.

Den sidste tabel med navnet `sip_data_group` indeholder sammenhængen imellem sip-enheder og grupper. Den beskriver således hvilken gruppe de enkelte PTT-enheder tilhører. Tabellen kan ses på Figur 5.6

```
1  CREATE TABLE pttproject.sip_data_group(
2    sip_name VARCHAR(80) NOT NULL,
3    group_id INT(11) NOT NULL DEFAULT 0,
4    INDEX `FK_sip-data-group_sip-data_sip_id` (sip_name),
5    INDEX `FK_sip-data-group_sip-group_group_id` (group_id),
6    UNIQUE INDEX sip_name (sip_name),
7    CONSTRAINT FK_sip_data_group_sip_data_name FOREIGN KEY (sip_name)
8    REFERENCES pttproject.sip_data (name) ON DELETE RESTRICT ON UPDATE RESTRICT,
9    CONSTRAINT FK_sip_data_group_sip_group_group_id FOREIGN KEY (group_id)
10   REFERENCES pttproject.sip_group (group_id) ON DELETE RESTRICT ON UPDATE
         RESTRICT
11 )
```

Figur 5.6: sip_data_group

Tabellen indeholder kun to variabler. `sip_name` og `group_id`. Disse to variabler referer henholdsvis til `sip_name` fra `sip_data`-tabellern og til `group_id` fra `sip_group`-tabellen. Det vil sige at en record indeholder informationen om en PTT-enhen og hvilken gruppe den tilhører. `sip_name`-variablen er unit hvilket vil sige at det kun er muligt for én enhed at være tilknyttet én gruppe ad gangen.

5.1.3 Dialplan

Dialplan'en er beskrevet i konfigurationsfilen, `extensions.conf`. Der er oprette to context'er i dialplan'en. En til PTT-enhederne med navnet `ptt-unti` og en til operatørerne med navnet `ptt-dispatcher`. Et udsnit af dialplan'en kan ses på Figur 5.7

```
1  [ptt-unit]
2  exten => 500,1,AGI(/home/cao/workspace/agiProject/src/agiPTT.py)
3
4  exten => i,1,Playback(vm-invalid)
5   same => n,HangUp()
6
7  exten => t,1,Playback(vm-invalid)
8   same => n,HangUp()
9
10
11 [ptt-dispatch]
12 exten => _#XXXX,1,Dial(SIP/${EXTEN:1}, 10)
13   same => n,HangUp()
14
15 exten => i,1,Playback(vm-invalid)
16  same => n,HangUp()
17
18 exten => t,1,Playback(vm-invalid)
19  same => n,HangUp()
```

Figur 5.7: extensions.conf

Den første context `ptt-unit` beskriver det forløb som en PTT-enhed skal gennemløbe når de vil i forbindelse med en operatør. Som det kan ses er der kun en indgang i denne context. Dette er indgangen `500`. Det vil sige at enheder som er i denne context kun kan ringe 500. Denne indgang kalder et program med navnet `agiPTT.py`. Dette gøres med funktionen `AGI`. Program er i denne udviklingsfase placeret i en udviklingsmappe på serveren med navnet "/home/cao/workspace/agiProject/src/".

Den anden context, `ptt-dispatcher`, har også kun en indgang. Denne indgang beskriver et bestemt mønster som der skal overholdes. Det mønster som skal overholdes for at ringe til en PTT-enhed er: Nummerer skal startes med "#" og nummeret skal bestå af præcis fire cifre. Hvis dette overholdes bliver funktionen `Dial` kaldt. Det nummer som dial-funktionen skal ringe til er den SIP-enhed der har det `name` som svare til de fire cifre efter #-tegnet. (I den endelige version af systemet skal dette ændres til IAX, da de rigtige PTT-enheder anvender denne protokol). Udover det nummer som der skal ringes til leveres der også et argument med til dial-funktionen som fortæller hvor lang tid som Asterisk skal vent på at opkaldet bliver besvaret. Bliver opkaldet ikke besvaret inden for 10 sek. vil der opstå en timeout.

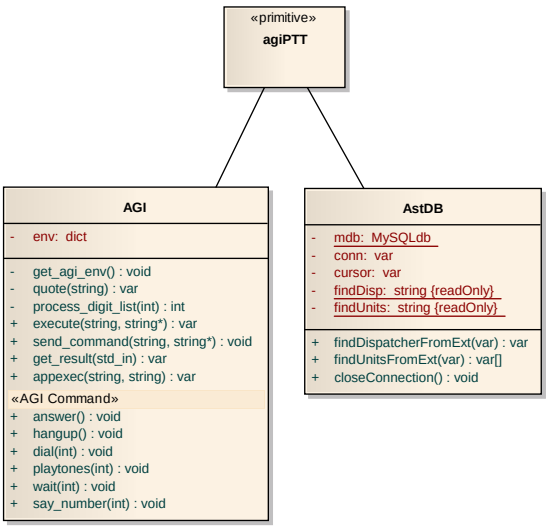
I både `ptt-unit` og `ptt-dispatcher` context'en er der implementeret to standard indgange. Den først hedder `i`. Denne indgang beskriver hvad der skal ske i tilfældet af fejl. Opstår der en fejl vil der blive afspillet en fejl-tone og herefter vil forbindelsen blive afbrudt med funktionen `HangUp`. Den anden standard indgang hedder `t`. Denne bliver kaldt i det til fælde at der opstår en timeout. Der vil også har blive afspillet et fejl-tone og herefter lagt på.

Hvis der bliver forsøgt at oprette forbindelse via en indgang som ikke er beskrevet i dialplan'en vil der blot blive afspillet en fejl-tone og herefter lagt på.

Skal applikationen på et senere tidspunkt udvides med voice mail funktionen, skal der laves om på de to forskellige forløb som der gennemløbes i henholdsvis `ptt-unit` og `ptt-dispatcher` context'en.

5.1.4 AGI

`agiPTT` er navnet på det script som bliver kaldt fra dialplan'en. Dette script er skrevet i programmeringssproget Python. Formålet med dette program er at slå op i databasen og finde ud af hvilken operatør som den PTT-enhed der ringer skal sættes i forbindelse med. Når denne information er indhentet skal programmet så oprette forbindelsen. For at kunne løse denne opgave et der udviklet to forskellige klasser med hver deres funktion. Den første klasse med navnet `AstDB` (`AsteriskDB.py`) er fremstillet med henblik på at kunne forbinde til en MySql database og indhente de informationer som knytter sig til en operatør. Den anden kalsse med navnet `AGI` (`AsteriskLib.py`) kan benyttes til at interagere med en Astarisk-server. Et klassediagram kan ses på Figur 5.8.



Figur 5.8: Klassediagram: AGI Script

Programforløbet i `agiPTT.py` scriptet er ganske simpelt. Først oprettes en instans af henholdsvis `AstDB` og `AGI`. Med `AGI` bliver informationen om hvilken PTT-enhen som der ringer indhentet. Med denne information bliver der slået på i databasen med `AstDB`. Databasen returnerer informationen om hvilken operatør som skal modtage kaldet. Med denne information bliver der nu oprette forbindelse, på samme måde som i dialplan'en, med funktionen `Dial` ved at bruge `AGI`.

`AstDB` klassen benytter et Python bibliotek med navnet `MySQLdb` til at oprette forbindelse med database. Man kan sende to forskellige SQL-statements til databasen med `AstDB`. Det første

beder databasen om at returnere den operatør som varetager en bestemt PTT-enhed. Dette statement kan ses på Figur 5.9 under med navnet "Find Dispatcher". Det andet statement kan bruges til at få returneret en liste med de PTT-enheder som en bestemt operatør varetager. Dette statement kan ses på Figur 5.9 under med navnet "Find Units".

```
1 Find Dispatcher:
2 SELECT * FROM sip_data WHERE name =
3   (SELECT sip_name FROM sip_group WHERE group_id = \
4     (SELECT group_id FROM sip_data_group WHERE sip_name = %s))
5
6 Find Units:
7 SELECT * FROM sip_data WHERE name IN
8   (SELECT sip_name FROM sip_data_group WHERE group_id =
9       (SELECT group_id FROM sip_group WHERE sip_name = %s))
```

Figur 5.9: SQL statements

Funktionerne `findDispatcherFromExt` og `findUnitsFromExt` bruges henholdsvis til at kalde "Find Dispatcher" og "Find Units". Brugerinformationerne som skal bruges til at oprette forbindelse til en database er "hardcoded" ind i `agiPTT`-scriptet. Men det kan nemt ændres da det er et script og derved ikke skal oversættes til en binær fil.

Med `AGI` har man adgang til alle de argumenter som Asterisk sender med når den kalder scriptet. Alle disse argumenter bliver gemt i en dictionary type med navnet `env`. Dette interface er tekst-baseret og kommandoer kan gives til Asterisk ved at skrive på `stdout`. Returværdier modtages ved at aflæse `stdin`.
Al den kode som vedrører `AGI` kan studeres i Bilag D.1.

5.2 Applikation

Applikationen er implementerer ved brugen af Qt version 4.7.4

5.2.1 AMI Manager

Interface
Interface-klassen til denne komponent hedder `AMIManager`. Der er blevet implementeret to signaler samt fire metoder. Al den kode som vedrører AMI Manager kan studeres i Bilag D.2. Herunder er de to signaler listet og beskrevet:

- **peerStatusChanged**
  Dette signal bliver fyret hver gang en telefon-enhed registrerer eller afregistrerer sig på Asterisk-serveren. Med dette signal bliver der leveret fire parametre. Først `peerStatus` denne beskriver den nye status som enden har fået. Denne status kan enten være "Registered" eller "Unregistered". Den næste parameter er navnet på den enhed som der har ændret status. Hvis det er tale om en SIP-enhed kunne denne parameter f.eks. være: "SIP/2005". `address` parameteren indeholder enhedens IP-adresse. Den sidste, `cause`, er en beskrivelse af grunden til ændringen i status.
- **result**
  Dette er et signal som er blevet brugt til debug formål. Det bliver fyret hver gang at en tekst-linje bliver modtaget fra Asterisk-serveren.

En tabel med signalerne kan ses i Tabel 5.1.

Signal
`void peerStatusChanged(QString peerStatus, QString peer, QString address, QString cause)`
`void result(const QString &result)`

Tabel 5.1: AMI Manager: Signal

Herunder er listet de metoder som er tilgængelige:

- **login**
  Denne metode bruges til at oprette forbindelse til Astrisk-serveren. Der skal leveres to argumenter med denne metode. `username` og `secret` disse to skal bruges som bruger-information når der skal logges ind.

- **sendAction**
  Denne metode kan bruges til at sende en action til Asterisk-serveren. Denne metode er overloadet og kan anvendes to to forskellige måder. Den første metode tager en streng som argument. Denne streng skal indeholde selve kommandoen. Den anden metode tager et action-object som argument. Den første metode som tager en streng som argument et kun velegnet til actions som ikke har nogle tilhørende parametre.
- **sendCommand**
  Denne metode skal anvendes hvis man vil sende den specielle action med navnet `command` til Asterisk. Der skal leveres en streng med som argument. Denne streng indeholder en CLI-kommando.

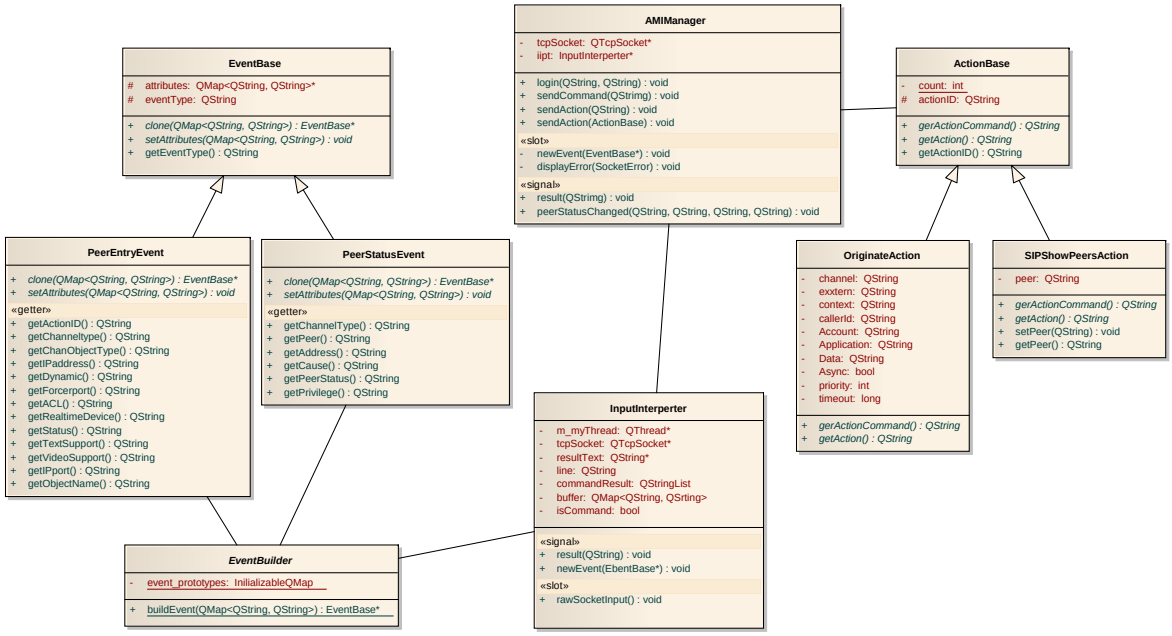
Tabel 5.2 viser en liste med alle metoderne.

Metoder
`void login(const QString &username, const QString &secret)`
`void sendCommand(const QString &command)`
`void sendAction(const QString &action)`
`void sendAction(ActionBase &action)`

Tabel 5.2: AMI Manager: Metoder

Intern opbygning

Figur 5.10 viser et klassediagram over denne komponent.



Figur 5.10: Klassediagram: AMI Manager

Det kan bruges et action-object til at sende en bestemt action. Der er implementeret to forskellige actions som der er muligt at sende. Disse to actions arver fra klassen `ActionBase`. Denne klasse indeholder en abstrakt metode som skal implementeres: `getActionCommand`. Denne metode returnere selve kommando strengen. Det er denne metode der bliver kaldt på det action-object der sendes med som argument til `sendAction` metoden. De objekter som arver fra `ActionBase` indeholder alle de parametre som høre til den specifikke action. Disse parametre sættes så sammen til selve kommando-strengen.

I `AMIManager` klassen bliver der oprette en instans af klassen `QTcpSocket`. Denne klasse bruges til at oprette selve Telnet-forbindelsen til Asterisk-serveren. Med denne forbindelse sendes og modtages kommandoer. Der er implementeret en klasse som bruges til at modtage alt hvad der kommer ind på TCP-socket'en. Denne klasse hedder `InputInterpreter`. Den fortolker beskederne fra Asterisk-serveren og på baggrund af disse opretter den så nogle event-objekter. Disse event-objekter bliver sendt videre til `AMIManager`-klassen som et internt signal. Når `AMIManager`-klassen modtager et event-object bliver det fortolket. På baggrund af dette bliver det tilsvarende signal sendt ud af selve AMI Manageren.

Event-objekterne arver alle fra klassen `EventBase`. Alle event-objekter indeholder et `QMap` med strenge som både key og value. Dette `QMap` bruges til at gemme alle de key/value-par som bliver modtaget fra Asterisk-serveren. Der er implementeret et "prototype pattern" til at oprette de enkelte event-objekter. Til dette formål er der lavet en klasse med navnet `EventBuilder`. Når `InputInterpreter`-klassen modtager en event fra Asterisk gemmes alle eventens tilhørende key/value-par i en `QMap`. Denne `QMap` leveres med som argument til `EventBuilder`'en med henblik på at få returneret det tilsvarende event-object. `EventBuilder`'en læser den key i `QMap`'en som hedder `event`. Denne key indeholder navnet på den event som er blevet modtaget. På baggrund af dette event-navn oprettes det respektive event-objekt som returneres.

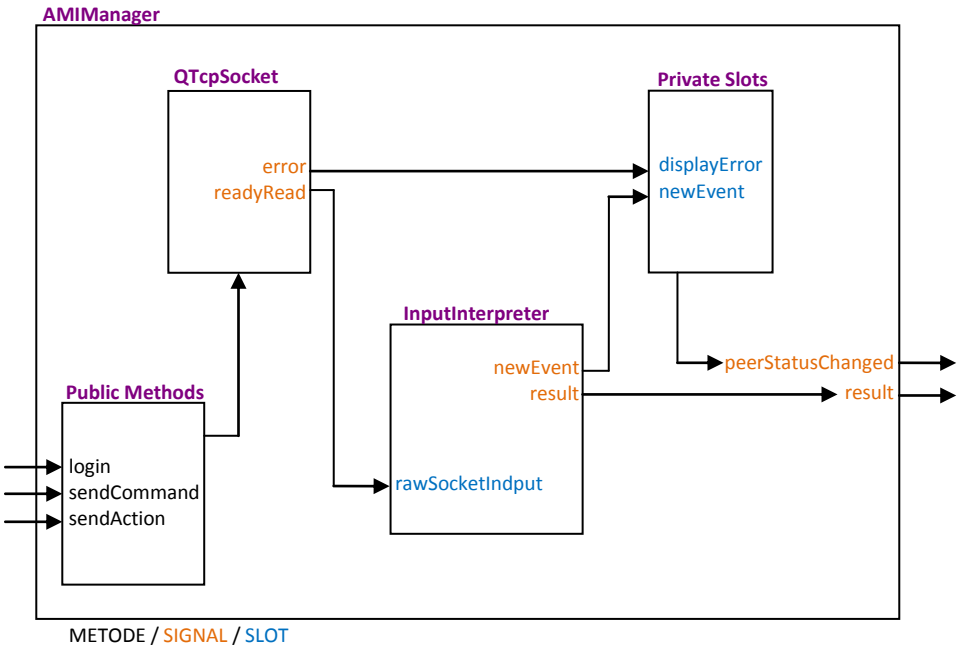
Hvis man ønsker at kunne sende eller modtag henholdsvis actions og events som ikke er implementeret kan disse nemt blive oprette ved at lave nogle nye klasser som arve enten fra `ActionBase` eller `EventBase`.

På Figur 5.11 kan er diagram over hvorledes de forskellige signaler og slots er forbundet internt i AMI Manager komponenten.

5.2.2 PTT Data

Interface
Med dette komponent er det muligt at interagerer med en MySql database. Til at repræsentere de data som er tilgængelige fra databasen er der implementeret nogle "Data Transfer Object" (dto-objekter). Der er kun en tabel i databasen der skal kunne manipuleres med. Denne tabel er `sip_data`-tabellen. Det tilsvarende dao-objekt hedder `SipData`. Et klassedisgram over denne komponent kan studeres på Figur 5.12.

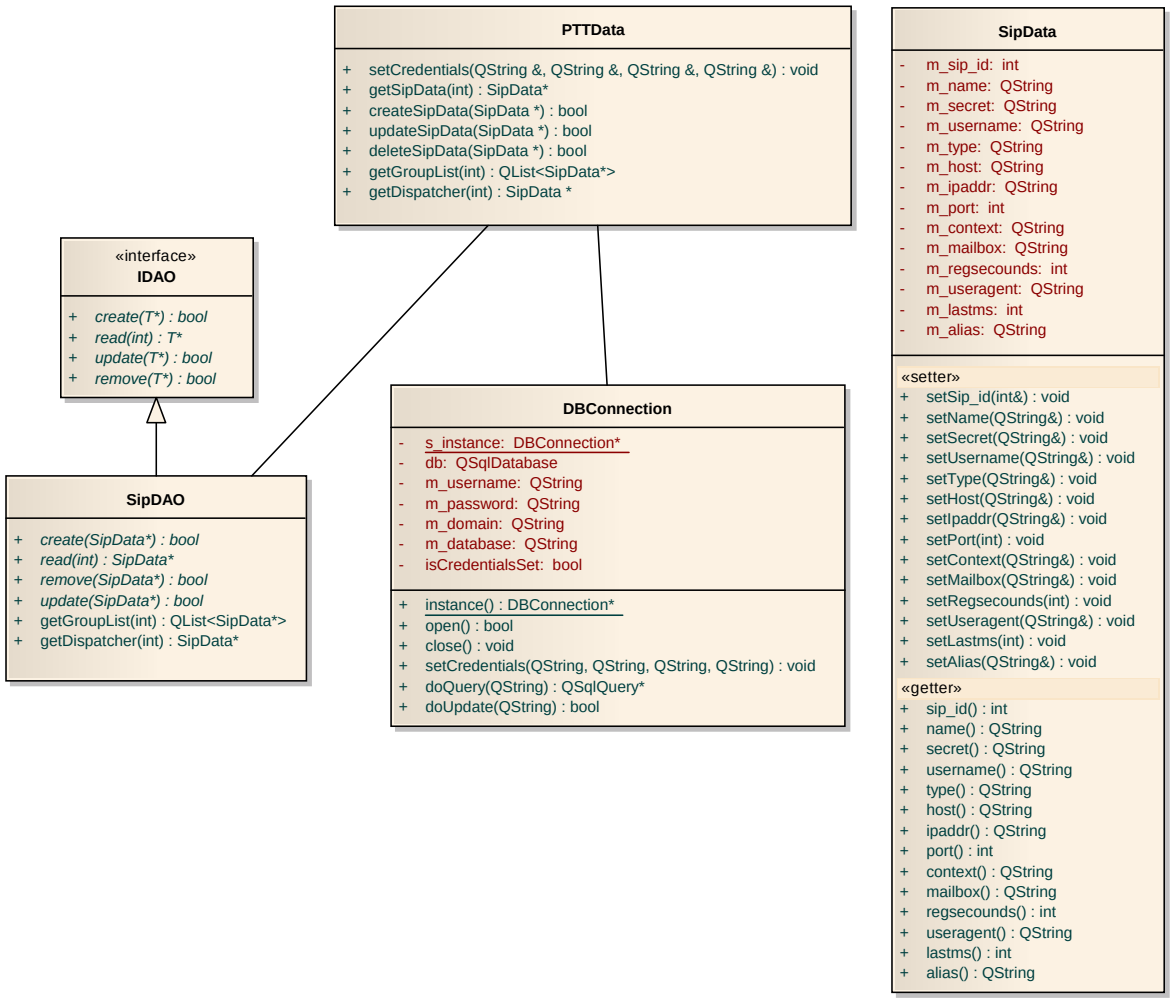
Interface-klassen til dette komponent hedder `PTTData`. Denne klasse implementere nogle metoder som kan anvendes til at interagere med databasen. Al den kode som vedrører PTT Data



Figur 5.11: Signalruting: AMI Manager

kan studeres i Bilag D.4. Herunder er metoderne listet og beskrevet:

- **getSipData**
  Denne metode returnerer et SipData-objekt med de informationer som knytter sig til en enkelt PTT-enhed. Metoden tager et argument, `name`. Dette er navnet på den PTT-enhed som man ønsker at få informationerne returneret på.
- **createSipData**
  Denne metode bruges til at oprette nye PTT-enheder i dadabasen. Denne metode tager et SipData-objekt som argument.
- **delSipData**
  Denne metode bruges til at slette en informationerne omkring en bestemt PTT-enhed fra databasen. Denne metode tager et SipData-objekt som argument.
- **updateSipData**
  Denne metode bruges til at opdatere informationerne omkring en allerede eksisterende PTT-enhed i databasen. Denne metode tager et SipData-objekt som argument. Informationerne som står i dette objekt bliver bruges til at overskrive de informationer som allerede ligger i databasen.
- **getGroupList**
  Denne metode returnerer en liste med SipData-objekter. Denne liste svare til de PTT-enheder som en bestemt operatør varetager. Metoden tager navnet på operatøren som argument.
- **getDispatcher**
  Denne metode returnerer et SipData-objekt. Dette opjekt svare til de informationer som knytter sig til den operatør som varetager en bestemt PTT-enhed. Metoden tager navnet på PTT-enheden som argument.



Figur 5.12: Klassediagram: PTT Data

En tabel med alle metoderne kan ses i Tabel 5.3.

Metoder
`SipData* getSipData(int name)`
`bool createSipData(SipData* sipData)`
`bool delSipData(SipData* sipData)`
`bool updateSipData(SipData* sipData)`
`QList<SipData*>* getGroupList(int dispName)`
`SipData* getDispatcher(int name)`

Tabel 5.3: PTT Data: Metoder

Intern opbygning

Der er implementeret en statisk klasse som varetager selve interaktionen med databasen samt at oprette forbindelse til databasen. Denne klasse hedder `DBConnection`. Før denne klasse kan

bruges er det vigtigt at man har sørget for at den har fået leveret brugeroplysninger omkring databasen og den bruger som man vil benytte. Dette gøres med metoden `setCredentials`. Når dette er på plads er det muligt at oprette en forbindelse til databasen med metoden `open`. Denne forbindelse kan nedlægges igen med metoden `close`. Der kan sende to former for SQL-statements til databasen med `DBConnection`. Det første er statements hvor det ikke bliver returneret noget fra databasen. Det svare til SLQ-statements som ikke er `SELECT`. Disse statements kan blive sendt til databasen med metoden `doUpdate`. Den anden type statements er dem hvor der bliver returneret informationer fra dadabasen. Dette er som regel `SELECT`-statements. Disse kan blive sendt til databasen med metoden `doQuery`.

Til at overføre de informationer, som dto-objekterne indeholder, ind og ud af databasen, er der oprettet et "Data access object" (dao-objekt). Det dao-objekt som anvendes til at overføre `SipData`-objekterne hedder `SipDAO`. Alle dao-klasserne skal implementere interfacet `IDAO`. Dette interface indeholder fire abstrakte metoder som skal gøre det muligt at: Oprette, læse, opdatere og slette informationer i databasen.

På Figur 5.13 er der vist et udsnit at koden fra `PTTData`-klassen. Denne kode viser et eksempel på hvordan `DBConnection`-klassen og `SipDAO`-klassen bruges til at hente informationer ud af databasen. Denne kode viser selve `getSipData`-metoden. Forløbet i metoden er som følger: Først oprettes det forbindelse til databasen med `DBConnection`. Dernæst oprettes der en instans af `SipDAO`-klassen. Denne instans skal bruges til at hente informationerne ud af databasen. Der oprettes en `SipData` reference som peger på det `SipData`-objekt som `SipDAO` returnere. Til sidst lukkes forbindelsen til databasen igen, med `DBConnection` og `SipData` referencen returneres.

```
1  SipData* PTTData::getSipData(int name)
2  {
3      DBConnection::instance()->open();
4      SipDAO sipDAO;
5      SipData* sipData = sipDAO.read(name);
6      DBConnection::instance()->close();
7      return sipData;
8  }
```

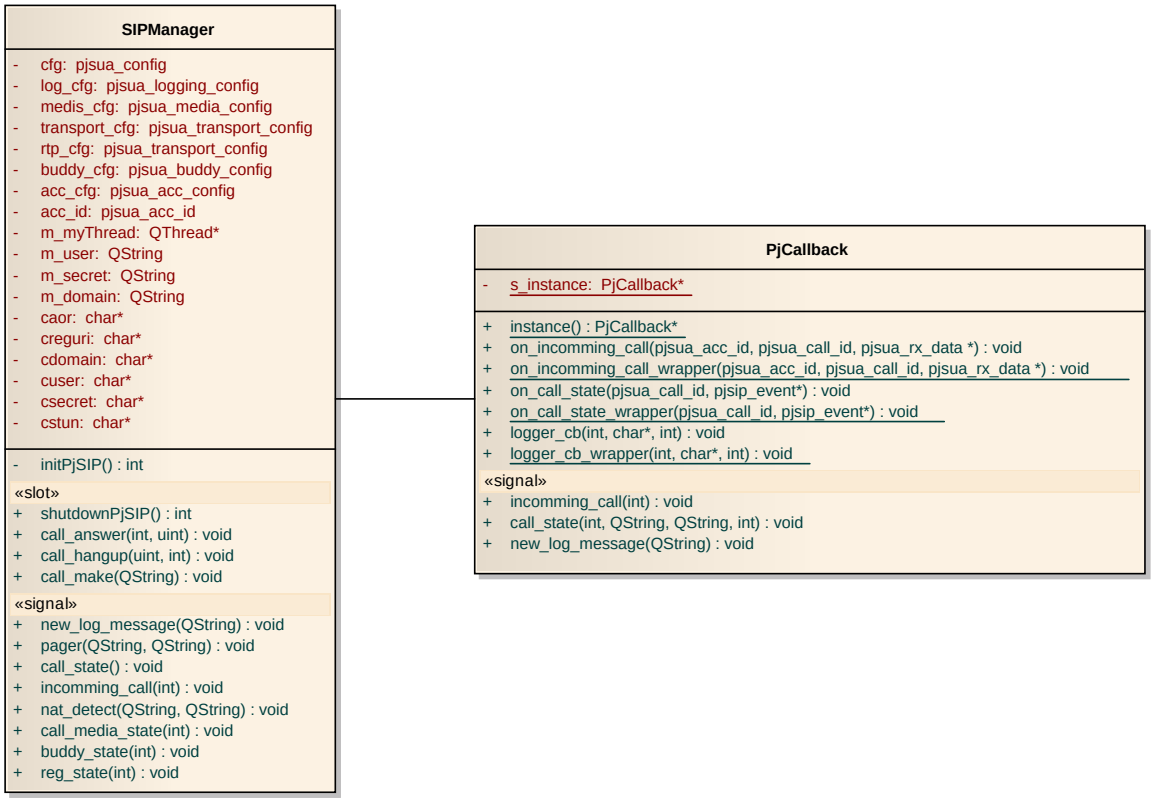
Figur 5.13: PTT Data: getSipData

5.2.3 SIP Manager

Interface

Interface-klassen til denne komponent er klassen `SIPManager`. Der er implementeret en række forskelligt signaler og slots som der kan bruges til at interagere med komponenten. Al den kode som vedrører SIP Manager kan studeres i Bilag D.3. Et klassedisgram over denne komponent kan studeres på Figur 5.14.

Der er tre signaler tilgængelige:



Figur 5.14: Klassediagram: SIP Manager

- **call_state**
  Dette signal bliver fyret hver gang en sip-enhed som er forbundet til Asterisk-serveren skifter sip-status. Der bliver sendt fire parametre med dette signal: Først et id på selve kaldet: `call_id`. Dernæst sendes selve navnet på den enhed, som det drejer sig om, med: `remote_info`. De to sidste parametre er den sip-state som enden har skiftet til. `state_text` er en beskrivelse af staten. `state_id` er det id nummer som sip-stat'en svare til.
- **incoming_call**
  Dette signal bliver fyret hvis en sip-enhed forsøger at lave et opkald. Der sendes to parametre med dette signal. Først et id-nummer: `call_id`. Dette id er et handle til selve opkaldet som er blevet oprette interne i PjSIP biblioteket. Dette handle skal brugse hvis men vil nedlægge forbindelsen igen. Den anden parameter er navnet på den enhed som det drejer sig om.
- **new_log_message**
  Dette er et signal som bliver fyret med interne beskeder fra PjSIP biblioteket. Der sendes en text-besked som indeholder informationen.

En liste med de forskellige signaler kan ses i Tabel 5.4.

Udover de tre signaler er der fire slots tilgængelige. Disse slots kan bruges til at få udført en

Signal
`void call_state(int call_id, QString remote_info, QString state_text, int state_id)`
`void incoming_call(int call_id, QString remote_info)`
`void new_log_message(QString text)`

Tabel 5.4: SIP Manager: Signal

handling:

- **call_make**
  Dette slot kan anvendes hvis man ønsker at oprette et opkald. Der skal angives to argumentet. Først `pjurl`, dette argument er en streng som fortæller hvem man ønsker at ringe til. Hvis man f.eks. vil oprette et kald til en sip-telefon skal denne streng starte med `sip:` efterfulgt at at det nummer man vil ringe. Det andet argument, `call_id` er et handle til opkalds id-nummeret.
- **call_hangup**
  Denne slot skal anvendes hvis man vil nedlægge et igangværende kald. DEr skal veveres to argumenter. Det først, `call_id`, er det handle på den forbindelse som skal nedlægges. Det andet argument, `code`, er en kode som beskriver grunden til at afbryde kaldet. En liste med alle sip response coder kan ses i Bilag A.6.
- **call_answer**
  Ønskes man at besvare et opkald skal det gøres med denne slot. Her skal der leveres to argumenter. Først det handle som peger på det opkald som skal besvares, `call_id`. Det andet argument er den sip response code som man ønsker at bruge. En liste med alle sip response coder kan ses i Bilag A.6.
- **shutdownPjSIP**
  Denne slot bruges til at PjSIP biblioteket ned. Dette vil medfører at man bliver afregistrtet på Asterisk-serveren. Der er ikke mulig at starte PjSIP op igen efter at denne slot er blevet kaldt.

En liste med alle de tilgængelige slots kan ses i Tabel 5.5.

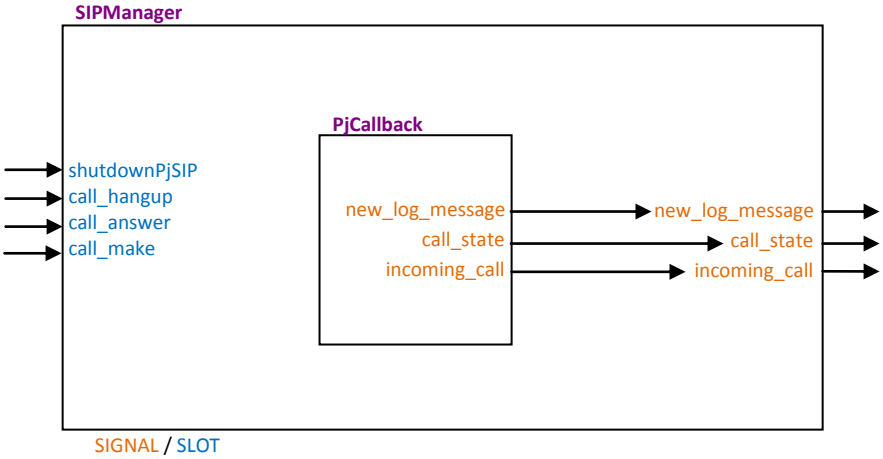
Intern opbygning
`SIPManager`-klassen indeholder et interface til Pjsip-biblioteket. I det øjeblik at klassens konstruktør kaldes bliver Pjsip initialiseret og startet op. Konstruktøren kalder metoden `initPjSIP`. I denne metode foregår al opsætningen af Pjsip. Koden i denne metode tager udgangspunkt i en tutorial der er tilgængelig på Pjsip's hjemmeside: `simple_pjsua.c`[8].

Slot
`void call_make(QString pjurl, int *call_id)`
`void call_hangup(int call_id, uint code)`
`void call_answer(int call_id, uint code)`
`bool shutdownPjSIP()`

Tabel 5.5: SIP Manager: Slot

Denne tutorial løber de trin igennem som skal til for at starte Pjsip op på den korrekte måde. Som argumenter til konstruktøren ska det leveres de brugeroplysninger som man vil anvende når der skal registreres på Asterisk-serveren. Når Pjsip-biblioteket er startet op arbejder det selvstændigt i baggrunden af applikationen. Internt i biblioteket er der defineres nogle callback-funktioner som Pjsip kalder hver gang der opstår en SIP-hændelse. Det kunne f.eks. være et indkommende kald. Disse hændelser resulterer i det tilsvarende signal.

Disse callback-funktioner er implementeret i en klasse for sig med navnet `PjCallback`. Denne klasse er implementeret ved brugen at et "singleton pattern". Da Pjsip biblioteket arbejder i nogle selvstændige program-tråde, som ikke har noget med Qt at gøre, er de callback-funktioner som Pjsip kalder nogle statiske "wrapper"-funktioner som bruges til at kalder de "rigtige" funktioner. Det er nødvendigt at kalde tilbage ind i Qt konteksten igen, med disse wrapper-funktioner, for at have Qt-funktionaliteten tilgængelig. En wrapper-funktion kalder den rigtige funktion med de samme argumenter som den har modtaget. Disse argumenter bliver så fortolket af den rigtige funktion og konverteret til Qt-typer som så bliver sendt med i signalet. På Figur 5.15 kan det ses hvorledes signlerne er forbundet internt i SIP Manager komponenten.



Figur 5.15: Signalruting: SIP Manager

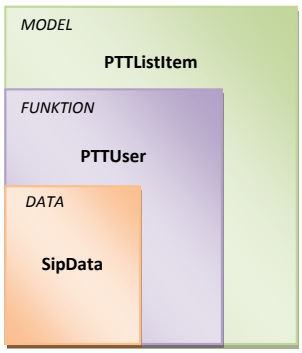
Selve opsætningen af forbindelsen imellem applikationen og Message Terminalens lydkort varetages også af denne komponent. I forbindelse med initialiseringen af Pjsip biblioteket sættes op hvorledes der skal interfaces til lydkortet. Når Pjsip er startet op sørger det selv

for at få rutet lyden fra SIP-forbindelsen til lydkortet og vice versa.

5.2.4 PTT Operatør

Al den kode som vedrører selve applikationen kan studerer nærmere i Bilab D.5.

PTT-abstraktioner
Der er implementeret tre forskellige klasser til at repræsentere en PTT-enhed på forskellige niveauer. Disse klasser tilføjer alle yderligere funktionalitet til den foregående klasse. På Figur 5.16 er denne sammenhæng illustreret. Den inderste klassen `SipData` er en entitets-klasse og indeholder de data som knytter sig til en bestemt PTT-enhed. Disse data kommer fra databasen. Det næste klasse med navnet `PTTData` tilføjer de forskellige funktionaliteter som en PTT-enhed har. Derudover bliver det tilføjet forskellige Qt-framework funktionalitet, såsom signals/slots i denne klasse. Den sidste klasse med navnet `PTTListItem` tilføjer den funktionalitet der gør det muligt at "binde" klassens informationer og funktioner sammen med QML.



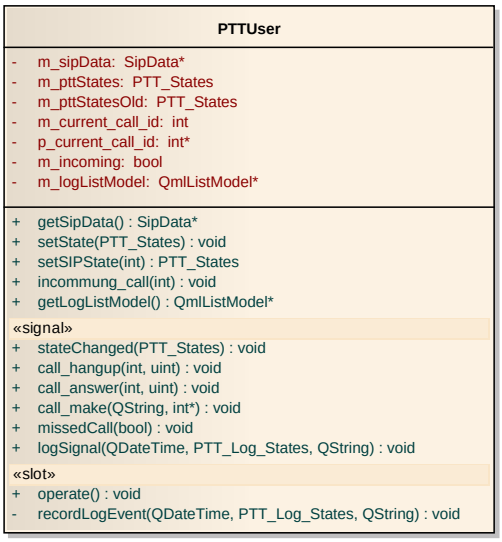
Figur 5.16: PTT Klasse lag

PTTUser
Denne klasse indeholder først og fremmest et `SipData`-objekt. Til at holde styr på hvilken tilstand PTT-enheden er i, indeholder denne klasse en `PTT_States` enum-datatype. Denne type bruges til at definere de forskellige tilstande (state) som en PTT-enheden kan indtage. `PTT_States` kan ses på Figur 5.17. `PTTUser` implementere en "state machine" som bruget til at afgøre hvad der skal ske i forskellige situationer, alt efter hvilken tilstand som PTT-enheden befinder sig i. Et eksempel kunne være den situation at en PTT-enheden vil i forbindelse med operatøren. Her vil `PTTUser`'en skifte tilstand til `RINGING`.
`PTTUser` indeholder en int pointer med navnet `p_current_call_id`. Denne pointer bruges som et handle til det opkald som findes i PjSIP biblioteket. Dette skal bruges hvis det aktuelle kald skal nedlægges. På Figur 5.18 kan `PTTUser`-klassen ses.

Herunder er listet de metoder der et tilgængelige på `PTTUser`:

```
1  typedef enum
2  {
3      UNKNOWN,
4      OFFLINE,
5      ONLINE,
6      INUSE,
7      RINGING,
8      CALLING
9  }PTT_States;
```

Figur 5.17: PTT_States



Figur 5.18: PTTUser Klasse

```
1  typedef enum
2  {
3    STATE_NULL,
4    STATE_CALLING,
5    STATE_INCOMING,
6    STATE_EARLY,
7    STATE_CONNECTING,
8    STATE_CONFIRMED,
9    STATE_DISCONNECTED
10 }SIP_States;
```

Figur 5.19: SIP_States

- **getSipData**
  Returnere `SipData`-klasse.
- **setState**
  Med denne metode kan den tilstand som enheden har ændres. Denne metode tager en `PTT_States` som argument.

- **getState**
  Denne metode returnere en `PTT_States` med den tilstand som PTT-enheden har.
- **setSIPState**
  Denne metode bruges til at fortælle `PTTUser`-objektet at det har skiftet SIP-status. Metode tager en enum som argument. Denne enum skal være at typen `SIP_State`. `SIP_State` indeholder de forskellige SIP-tilstande der kan forekomme. På Figur 5.19 kan `SIP_States` ses. Når denne metode bliver kaldt ændres den tilstand som `PTTUser`-objektet har, på beggrund af SIP-tilstanden. Alle de forskellige datatyper der er blevet defineret er placeret i filen `ptt_datatypes.h`. Denne kan studeres nærmere i Bilag D.5.6.
- **incommung_call**
  Denne metode bruges til at fortælle `PTTUser`-objektet at den tilsvarende PTT-enhed prøver at oprette forbindelse til operatøren. Det er kun hvis `PTTUser`-objektet er i `ONLINE` tilstand at at der bliver reageret på dette.
- **getLogListModel**
  `PTTUser` indeholder en liste med log informationer. Hver gang at `PTTUser`-objektet skifter tilstand logges dette i denne liste. Denne metode returnerer log-listen.

Herunder er de forskellige signaler som er implementeret på `PTTUser` listet:

- **stateChanged**
  Dette signal bliver fyret hver gang at `PTTUser` skifter tilstand.
- **call_hangup / call_answer / call_make**
  Disse tre signaler bruges til at signalere at `PTTUser`-objektet gerne vil afbrude, besvare eller oprette et opkald. Disse signal skal opfanges og sendes videre til "SIP Manageren", som så kan fortage den respektive handling. Hvis operatøren ønsker at oprette et opkald til en PTT-enhed, fyres `call_make`-signalet. Med dette signal sendes to parametre. Først en streng, med navnet `pjurl`, der angiver hvem der skal ringes til. Den anden parameter, `call_id` er et handle til opkaldet. `call_hangup` og `call_answer` leveret begge to parametre. Den første, `call_id` er et handle til opkaldet. Den anden, `code`, er en kode som beskriver grunden til handlingen.
- **missedCall**
  Dette signal fyret hvis et opkald til operatøren fra den specifikke `PTTUser` ikke bliver besvaret og derved timer ud.
- **logSignal**
  Dette signal bliver fyret hver gang der skar en en ændring i PTT-enhedens tilstand. Dette signal indeholder tre informationer. Først et tidsstempel med navnet, `now`, dette er af typen `QDateTime`. Næst en enum type som der fortæller hvilken handling der er tale om. Denne type hedder `PTT_Log_States` og kan ses på Figur 5.20. Den sidste information, `alias`, er en streng som kan bruges til f.eks. at lave en tekst beskrivelse af handlingen.

Herunder er listet de slots som er tilgængelige på `PTTUser`:

- **operate**

```
1  typedef enum
2  {
3      LOG_OFFLINE,
4      LOG_ONLINE,
5      LOG_RINGING,
6      LOG_CALLING,
7      LOG_MISSED
8  }PTT_Log_States;
```

Figur 5.20: PTT_Log_States

Dette er en `public` slot som anvendes til at aktivere `PTTUser`-objektet. Alt efter hvilken tilstand som `PTTUser`-objektet er i vil denne slot have forskellig funktion. Hvis objektet er i `ONLINE`-state vil denne metode bevirke at der bliver fortaget et opkald til PTT-enheden. Men er objektet i `RINGING`-state, hvilken betyder at der er et indkommende kald fra den pågældende PTT-enhed, vil denne metode besvare dette indkommende kald. Er `PTTUser`-objektet i enten `INUSE`- eller `CALLING`-state, som henholdsvis betyder at der er et opkald igang eller at der er ved at blive oprette et opkald til PTT-enheden, vil denne metode bevirke at kaldet vil blive nedlagt.

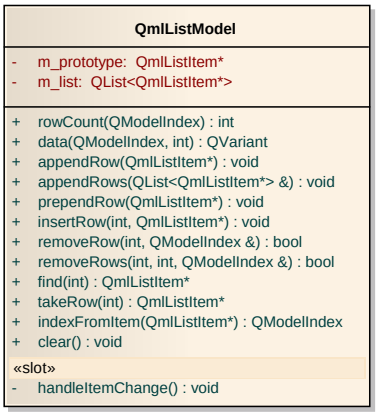
- **recordLogEvent**
  Denne slot bruges til at gribe `logSignal`-signalet. `recordLogEvent` opretter en ny handling i log-listen `m_logListModel`.

QmlListModel

For at kunne binde data imellem C++ og QML koden er der implementeret nogle specifikke klasser. Der er fremstillet en model-klasse, `QmlListModel`. Denne model-klasse udgør det for en liste. `QmlListModel` arver fra `QAbstractListModel` der er standard liste-model klasse fra Qt-framework'et. Sådanne en liste-model beskriver de abstrakte metoder som skal implementeres før at QML kan anvende den. Selve listen i lest-modellen indeholder nogle specielle objekter (ListItem's).
For at gøre det muligt for Qt-framework'et at kunne hente data ud af modellen skal der først og fremmest defineres nogle såkaldte "role's". Disse role's er en liste (QHash) men navnene på de data som skal kunne hente fra de enkelte objekter i listen. Det er disse navne som kan bindes til QML. Når data hentes ud via sådanne en binding bliver metoden `data` kaldt igennem Qt-tramevork'et. Metoden modtager indekset på det element i liste, der skal hentes data fra. Metoden finder så det respektive listitem frem og returnere den ønskede data. Udover `data`-metoden indeholder modellen nogle klasser som gør det muligt at tilføje, indsætte, slette og finde listitem's i listen. `QmlListModel`-klassen kan ses på Figur 5.21.

PTTListItem

Listen i `QmlListModel` består af `QMLListItem`'s. Dette er en abstrakt klasse der skal arves fra for at kunne ligge i listen på en `QmlListModel`. `QMLListItem` beskriver to metoder som der skal implementeres. Disse to metoder er `data` og `roleNames`. Disse to metoder er blevet flyttet ud af `QmlListMelod`. Dette er gjort for at gøre `QmlListModel` til en general list-model som kan anvendes til alle lister af typen `QMLListItem`. Da en `QmlListModel` skal vide hvilken role's der er tilgængelige skal man i konstruktøren til en `QmlListModel` angive den `QMLListItem`-klasse som der skal ligge i listen. Denne `QMLListItem`-klasse bliver så brugt



Figur 5.21: QmlListModel Klasse

som en prototype. `QmlListModel` henter så listen med role's fra `QMLListItem` prototypen med metoden `roleNames`. `PTTListItem` arver fra `QMLListItem` og er beregnet til at kunne ligge i en `QmlListMelod`. På Figur 5.22 ses `QMLListItem` og `PTTListItem`-klasserne.

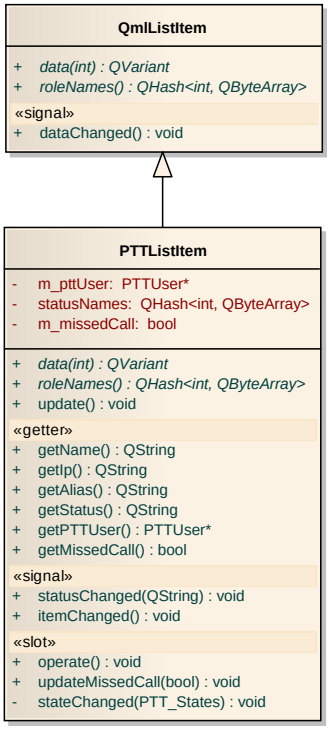
`PTTListItem` er den yderst skal at de forskellige PTT abstraktioner. Klassen er implementeret således at den også kan bruges til at binde til QML uden at ligge i en `QmlListMelod`. Til dette formål er der blevet tilføjet nogle Qt-properties. Disse properties benytter de respektive getter-metoder til at tilgå de forskellige data felter.

Der er implementeret to signaler på `PTTListItem`. Disse et listet herunder:

- **statusChanged**
  Dette signal bruges til at fortælle status-propperty'en at `PTTUser`-objektet har ændret tilstand.
- **itemChanged**
  Dette signal btuges til at fortælle `name`-, `ip`-, `alias`- og `missedCll`-properties'en at de respektive data er blevet opdateret.

Der er implementeret tre slote som er listet herunder:

- **operate**
  Dette er en slot som det er beregnet at til at blive kaldt fra QML-koden. Det kunne f.eks være når en knap på skærmen bliver trykket ned. Denne slot kalder operate metoden på det `PTTUser`-objekt som `PTTListItem` indeholder.
- **updateMissedCall**
  Denne slot fanger `missedCall`-signalet fra `PTTUser`'en. Dette bruges til at indikere om denne enhed har et ubesvaret opkald.
- **stateChanged**
  Denne slot fanger `stateChanged`-signale fra `PTTUser`'en. Dette bruges til at fortælle `status`-property'en at den skal opdateres.



Figur 5.22: QMLListItem og PTTListItem Klasserne

Der er implementeret en anden klasser der også arver fra `QMLListItem` tilsvarende `PTTListItem`. Denne klasse hedder `PTTLogListItem` og bruges som objekt i den liste (`QmlListModel`) som lideholder log-beskederne på `PTTUser`. Denne klasse kan ses på Figur 5.23

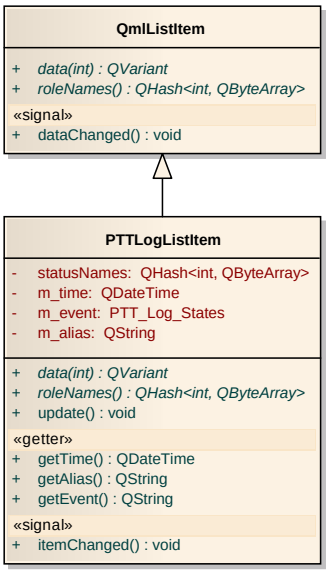
Controller

`PTTController` indeholder en instans at hver at de tre komponenter: `SIPManager`, `AMIManager` og `PTTData`. Figur 5.24 viser klassen.
I konstruktøren skal navnet på den operatør som der skal arbejde med angives. Dette navn bruger `PTTController`'en så til at hente de `SipData`-objekter der er knytte til den gruppe som operatørerne varetager. Dette gøres ved brugen af `PTTData`-komponenten. `SIPManager`'en og `AMIManager`'en startes op med de relevante brugeroplysninger. Disse oplysninger er "hardcodet" ind i koden.

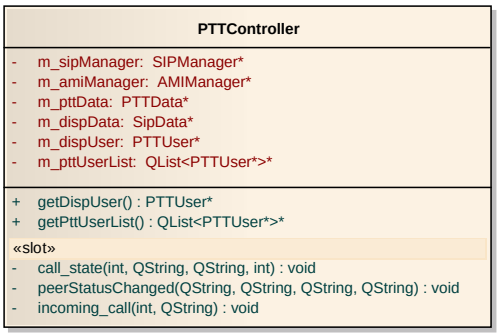
På baggrund af listen med `SipData`-objekter oprettes der en liste med `PTTUser`-objekter. Denne liste er tilgængelig via `getPttUserList`-metoden. Alle `PTTUser`-objekterne forbindes sammen med `SIPManager`'en. Således at signalerne `call_make`, `call_answer` og `call_hangup` fra hver af `PTTUser`'en fanges af `SIPManager`.

Det er implementeret nogle interne slot's til at fange signaler fra `SIPManager`'en og `AMIManager`'en. Disse slot's er beskrevet herunder:

- **incoming_call - call_state**
  Disse to slots bruges til at fange de tilsvarende signaler der kommer fra `SIPManageren`.



Figur 5.23: PTTLogListItem Klassen



Figur 5.24: PTTController Klasse

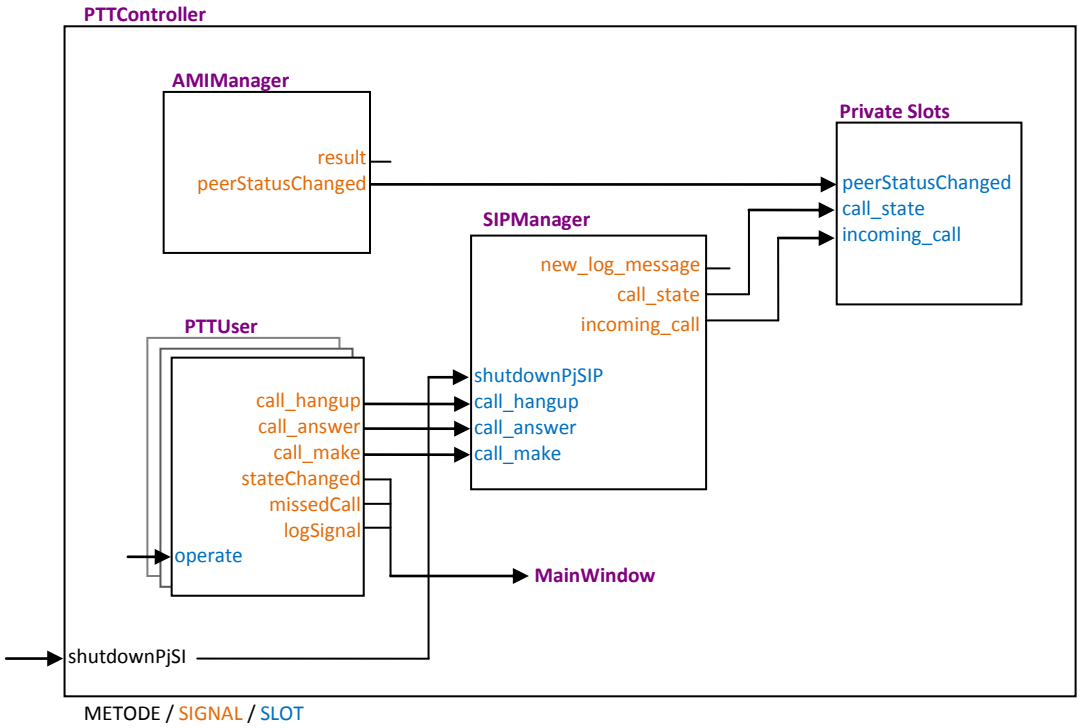
Hvis et af disse signaler fanges gås listen med `PTTUser`-objekteren (`m_pttUserList`) igennem. Dette gøres for at finde den `PTTUser` som svare til den det navn som bliver sente med signalet, `remote_info`. Når `PTTUser`'en er fundet ændres enten dens tilstand eller og så kaldes metoden `incomming_call`.

- **peerStatusChanged**
  Dette slot brures til at fange `peerStatusChanged`-signalet fra `AMIManager`'en. Opfanges dette signal findes den respektive `PTTUser` i listen og dens tilstand ændres enten til `ONLINE` eller `OFFLINE`.

Et diagram over hvorledes de forskellige signaler er forbundet interne i `PTTController` kan ses på Figur 5.25.

MainWindow

Dette er den klassen hvor selve applikationen starter. Det er fra denne klasse de forskellige komponenter bliver startet op. Dette gøres ved at oprette en instans af `PTTController`'en.



Figur 5.25: Signalruting: PTT Controller

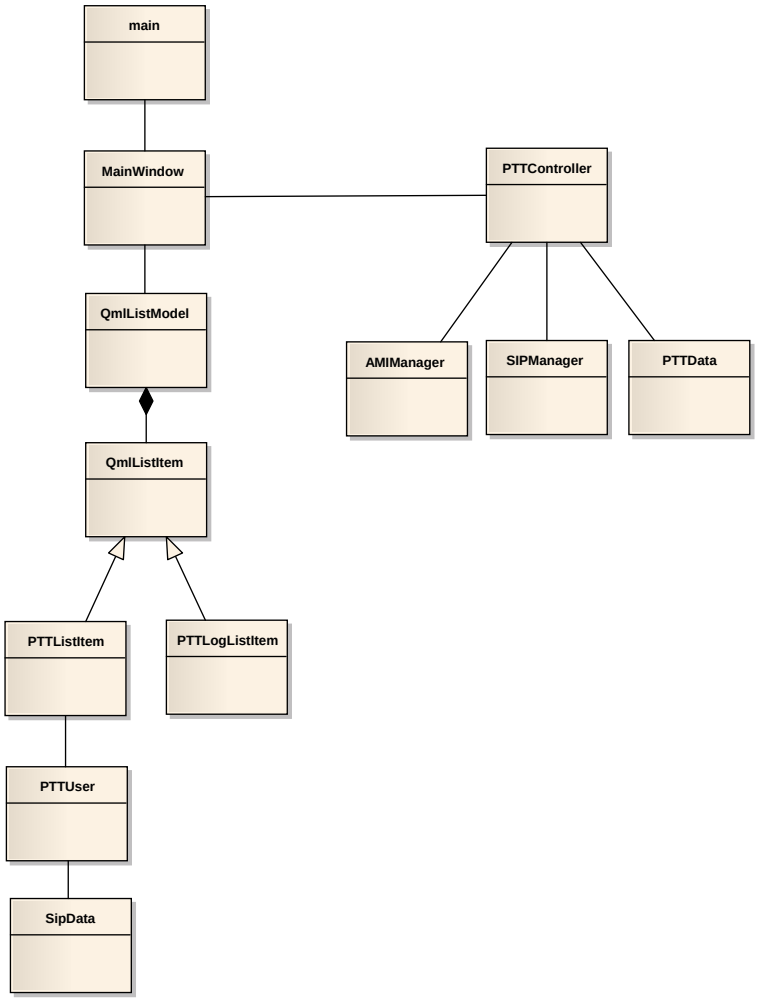
Udover dette bruges denne klasse også til at starte QML-fortolkeren op og oprette de forskellige sammenhænge mellem QML og C++. På Figur 5.26 viser et klassediagram over `MainWindow`.

Vindue

Der oprettes en instans af klassen `QDeclarativeView` der er en del af Qt-framework'et. Dette er det vindue som bruges til at præsentere grafikken. Denne klasse er også den som fortolke QML-scriptet. Når vindiet er blevet oprettet sættes det til at fylde hele skærmen ud. Dette gøres med metoden `SetFullScreen`.
På Figur 5.27 kan ses et udsnit af koden fra `MainWindow`. Dette udsnit viser først hvorledes vinduet, med navnet `ui`, bliver tilføjet de objekter som skal være tilgængelige fra QML. Alle disse objekter får samtidig tilføjet et navn som er det der skal refereres til fra QML. Efter disse objekter er blevet tilføjet, QML-konteksten, bliver selve det QML-scriptet, der skal vises i vinduet, loadet. Dette script hedder `window.qml`.

PTT Lister

Der er oprette en række af forskellige liste der bruges til at holde styr på PTT-enheder. Først og fremmest er der en liste med navnet `m_allPttUsersList`. Denne liste indeholder alle de `PTTUnite` som operatøren skal varetage. Denne liste bliver leveret af `PTTController`'en. Dernæst er der en liste med navnet `m_allQmlItemsList`. Denne liste indeholder også alle PTT-enhederne. Objekterne i denne listen er af typen `PTTListItem`. Disse objekter skal bruges til at komme ind i den liste som præsentere objekterne på skærmen. Den liste hedder `m_qmlGraphicListModel` og er af typen `QmlListModel`.



Figur 5.26: PTT Dispatcher Klassediagram

```
1 this->ui = new QDeclarativeView;
2 this->ui->rootContext()->setContextProperty("mainWindow", this);
3 this->ui->rootContext()->setContextProperty("buttonIndicator",
4                                             &m_buttonIndicator);
5 this->ui->rootContext()->setContextProperty("listModel",
6                                             m_qmlGraphicListModel);
7 this->ui->rootContext()->setContextProperty("detailItem", NULL);
8 this->ui->rootContext()->setContextProperty("dispItem", m_dispItme);
9 this->ui->setSource(QUrl("qrc:qml/window.qml"));
```

Figur 5.27: MainWindow

Det er muligt at vise forskellige sorteringer at PTT-enhederne. F.eks. alle enheder eller kun de enheder der er online. Dette betyder at `m_qmlGraphicListModel`-liste nogle gange kun skal indeholde nogle udvalgte PTT-enhederne. De forskellige udsnit af PTT-enheder bliver vedligeholdt i tre forskellige liste som indeholder `PTTUser`-objekter. Disse lister med navene: `m_onlineUsersList`, `m_incomingCallsList` og `m_missedCallsList` indeholder henholdsvis

alle de online enheder, de enheder som ønsker at komme i forbindelse med operatøren og de enheder som har ringet forgæves. I det øjeblik at operatøren vælger f.eks. at se alle de online enheder. Bliver alle de objekter der ligger i `m_qmlGraphicListModel` fjernet og herefter bliver de objekter som ligget i listen `m_onlineUsersList` flyttet over i `m_qmlGraphicListModel`. Da disse to lister ikke indeholder objekter af samme type bliver de objekter der ligger i `m_onlineUsersList` sammenlignet med de objekter som ligger i `m_allQmlItemsList` og det tilsvarende objekt er det der bliver flyttet over i `m_qmlGraphicListModel`. Grunden til denne konstruktion er at når et objekt i `m_qmlGraphicListModel` tilføjes eller fjernes vil det kunne ses grafisk med en lille animation. Derfor er der kun en liste som bruges til at blive præsenteret på skærmen. Et skema over de forskellige lister og deres type kan ses på Figur 5.6.

Type	Liste

`QList<PTTUser*>`	`m_allPttUsersList`
	`m_onlineUsersList`
	`m_incomingCallsList`
	`m_missedCallsList`

`QList<PTTListItem*>`	`m_allQmlItemsList`
`QmlListModel*` (`PTTListItem*`)	`m_qmlGraphicListModel`

Tabel 5.6: Lister

Metoder
Der er implementeret nogle forskellige metoder som er beregnet til at blive kaldt fra QML-koden. Alle disse metoder fortæller `MainWindow` hvilken liste der skal vises: `showOnline` - `showAll` - `showIncoming` - `showMissed`
Der er implementeret en enum, `ListViewState`, der bruges til at holde styr på hvilken sortering af `m_qmlGraphicListModel` som er den aktuelle.

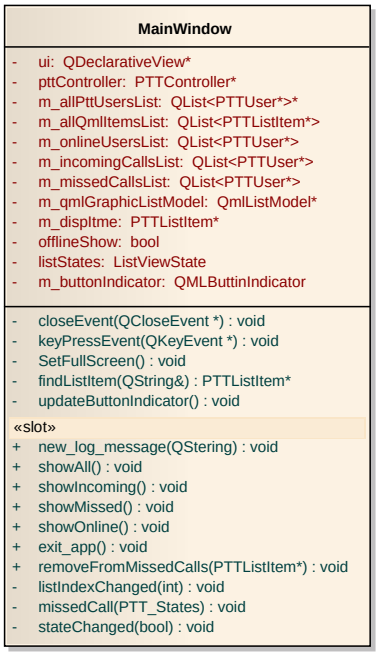
Der er også en metode som kan kaldes fra QML hvis man ønsker at afslutte applikationen: `exit_app`

En detaljeret `MainWindow`-klasse kan ses på Figur 5.28.

Når `PTTController`'en ændre på den tilstand som et `PTTUser`-objekt har, vil dette objekt udsende forskellige signal som bliver fanget af `MainWindow`. Der er implementeret to slote til at fange disse signaler:

- **stateChanged**
  Denne slot bruges til at flytte `PTTUser`-objekterne rundt imellem de forskellige lister alt efter hvilke tilstand de har.
- **missedCall**
  Dette er en slot som fanger den specielle situation at: Et opkald ikke er blevet besvaret. Er dette tilfældet bliver den respektive `PTTUser` flytter over i listen `m_missedCallsList`.

Figur 5.29 illustrerer hvordan de forskellige signaer er forbundet interne i `MainWindow`.



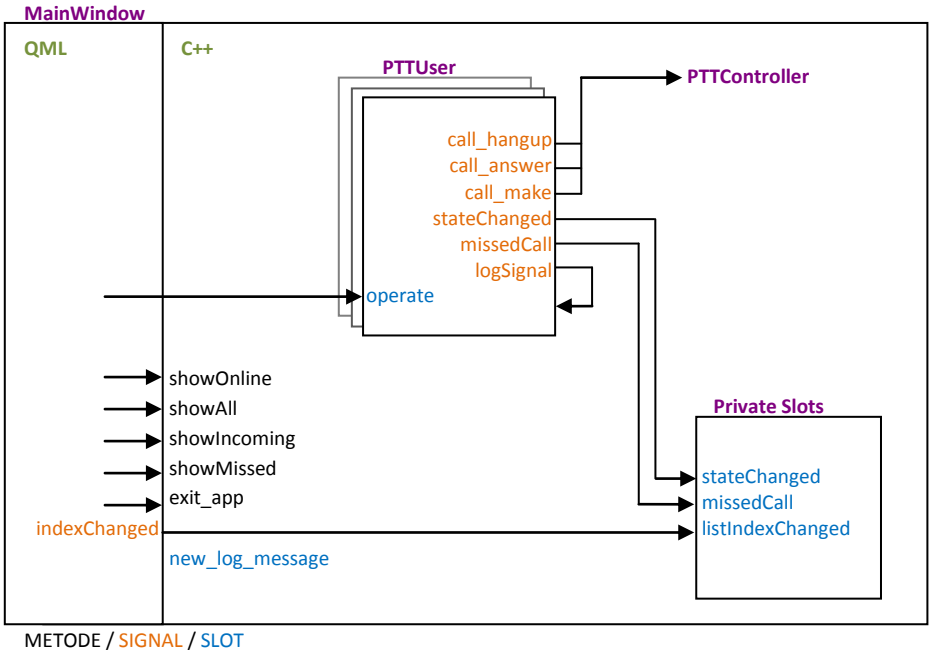
Figur 5.28: MainWindow Klasse

5.2.5 QML Scripts

Hele brugergrænsefladen er implementeret ved brugen af QML. Koden er opdelt i flere forskellige script-filer med hver deres funktion. Alle diees filer kan studeres i Bilag D.6. Der er i princippet kun et stort script som beskriver hele brugergrænsefladen. Hoved script-filen hedder `window.qml`. De andre filer er selvstændige grafikelementer som så er blevet anvendt ved opbygningen af `window`-filen.

Binding
Et eksempel på sammenhængen mellem QML og C++ er den liste som anvendes til at præsentere alle PTT-enhederne på skærmen. I C++ koden er der implementeret en list-model som kan bindes til QML. Denne liste er `m_qmlGraphicListModel`. QML-koden som beskriver hvordan denne liste skal vises på skærmen, er placeret i filen `PTTList.qml`. På Figur 5.30 kan denne kode ses. Listen er laver ved at bruge det QML-element som hedder `ListView`. Dette er et standard QML-element som følger med Qt. Der er en række forskellige properties som bliver brugt til at beskrive hvordan listen skal tage sig ud. F.eks. id som beskriver det navn som skal bruges til at referere til listen i QML. Den property der er interessante i forbindelse med sammenhængen med C++ er `model`. Denne propperty angiver den model som liste skal anvende. `listModel`-modellen referere til det navn som `m_qmlGraphicListModel` fik tildelt da den bliv loadet ind i QML ved opstarten af applikationen i `MainWindow`-klassen.

En anden interessant property på `ListView` er `delegate`. Denne fortæller hvordan de enkelte objekter, der er i modellen, skal tegnes på skærmen. `delegate`-property'en referere i dette tilfælde til en anden QML-scriptfil med navnet `PTTListDelegate.qml`. Denne fil beskriver altså udseendet på det enkelt objekt i listen.



Figur 5.29: Signalruting: MainWindow

Dette element er opbygget at et Qt Item element og har fået navnet `pttItem`. Når der skal tilgår data omkring et objekt, som ligger i modellen, gøres dette via de role's der er blevet difineret i `QmlListModel`-klassen. Dette gøres f.eks med den role som hedder `status`. Denne role indeholder den streng der beskriver tilstanden af PTT-enheden. På Figur 5.31 ses hvorledes denne role bindes til `state`-proprtty'en på `PTTListDelegate`-elementet. Ligeledes kan de forskellige tilstande som elementet kan indtage ses. Hver tilstand beskriver hvilken property på elementet der skal ændres i den givende tilstand. F.eks. skal property'en `onlineOpacity` som hører til `pttItem` sættes til 1 i `online` tilstanden. På den måde ændres udseende at brugergrænsefladen alt efter hvilken tilstand de enkelte PTT-enheder er i.

Detail view

Når en operatør trykker på en PTT-enhed som ligger i listen skal der vises detaljer omkring den enkelte enhed. Til at fange selve trykket er der på `pttItem` placeret et såkaldt `MouseArea`. Dette aera beskriver et område på grafikken der er modtageligt for musse-tryk samt hvas der skal ske når der bliver klikket i dette område. På Figur 5.32 ses det udsnit af `pttItem` som beskriver dette `MouseArea`. Når der klikkes her sker der tre ting: det property der beskriver hvad der skal stå i toppen af skærmen skiftes til `"Detail"`. Dernæst vælges det index som vise detalje-billede på skærmen. Ser man på Figur 5.30, der viser `listView` koden, kan man se at der her er implementeret en funktion med navnet `updateIndex`. Denne funktion bliver brugt til at fyre et signal til `MainWindow`. Dette signal indeholder indekser på det objekt i listen som skal vises. Den sidste ting der skar når der trykkes på `MouseArea`'et er at dette signal bliver fyret.

Den slot i `MainWindow` som medtager dette signal hedder `listIndexChanged`. Når dette signal modtages findes det `PTTListItem` i `m_qmlGraphicListModel` som svare til indekser frem.

```
1     ListView {
2         id: listView
3         objectName: "listView"
4         anchors.fill: parent
5         model: listModel
6         delegate: PTTListDelegate{}
7
8         interactive: pttListBase.interactive
9
10        width: parent.width
11        height: parent.height
12        clip: true
13        smooth: true
14        cacheBuffer: 10
15        currentIndex: -1
16
17        boundsBehavior: Flickable.DragOverBounds
18
19        ScrollBar{
20            flickable: listView
21            vertical: true
22        }
23
24        function updateIndex(indexx) {
25            currentIndex = indexx
26            pttListBase.indexChanged(currentIndex)
27        }
28    }
```

Figur 5.30: ListView

```
1     state: status
2     states:[
3         State {
4             name: "offline"
5             PropertyChanges { target: missedLight; color: "gray" }
6             PropertyChanges { target: incomingLight; color: "gray" }
7             PropertyChanges { target: pttItem; onlineOpacity: offlineOpacity;}
8             PropertyChanges { target: pttItem; incomingActiveOpacity: 0;}
9         },
10        State {
11            name: "online"
12            PropertyChanges { target: pttItem; onlineOpacity: 1;}
13            PropertyChanges { target: pttItem; incomingActiveOpacity: 0;}
14        },
15        State {
16            name: "ringing"
17            PropertyChanges { target: pttItem; onlineOpacity: 1;}
18            PropertyChanges { target: pttItem; incomingActiveOpacity: 1;}
19        }
20    ]
```

Figur 5.31: PTTListDelegate: State

```
1     MouseArea {
2         anchors.fill: parent
3         onClicked:{
4          titleTop.text = "Detail"
5             view.currentIndex = 1
6             listView.updateIndex(index)
7         }
8     }
```

Figur 5.32: PTTListDelegate: MouseArea

Dette `PTTListItem` loades nu ind i QML-koden så de data som dette objekt indeholder kan blive vist i detalje-vinduet. Metoden `listIndexChanged` kan ses på Figur 5.33

```
1  void MainWindow::listIndexChanged(int index)
2  {
3      //qDebug()<<__FUNCTION__;
4      qDebug()<< "index: "<<index;
5      PTTListItem* pttItem = qobject_cast<PTTListItem*>(m_qmlGraphicListModel->
           find(index));
6      this->ui->rootContext()->setContextProperty("detailItem", pttItem);
7      this->ui->rootContext()->setContextProperty("loglistModel", pttItem->
           getPTTUser()->getLogListModel());
8      pttItem->update();
9      removeFromMissedCalls(pttItem);
10 }
```

Figur 5.33: MainWindow: listIndexChanged

5.2.6 Licens

Der er til udviklingen af dette projekt udelukkende blevet brugt open source biblioteker. Alle disse biblioteker er beskyttet af forskellige licenser som muligvis vil have indflydelse på en eventuel udgivelse af applikationen. Men da projekt kun er tiltænkt til udviklingen af en prototype, som ikke skal sælges eller på anden måde bruges uden for virksomheden, er der ingen problemer forbundet ved at benytte biblioteker med forskellige typer af licenser.
De open source projekter som benyttes til udviklingen af applikationen anvender to typer at licenser:

GNU General Public License (GPL)
Alle programmer der benytter kode som er beskyttet at en GPL[4] licens skal selv være open source. Dette gør der umuligt for virksomheder, som gerne vil holde deres source kode hemmelig, at bruge kode med denne licens.

GNU Lesser General Public License (LGPL)
Det er tilladt at anvende kode beskyttet med LGPL[5] licens til udvikling af programmer som ikke er open source. Dette betyder et programmer som benytter LGPL kode ikke automatisk skal overtage denne licens.

Qt, der er open source, et beskyttet med LGPL. Dette betyder at det er lovligt at udgive både åbne og lukkede programmer som anvender Qt. Qt udgives dog også med en kommerciel licens. Hvis denne licens vælges har men også rettighed til at ændre i selve Qt koden, uden at offentliggøre det.
PjSip er underlage en GPL licens. Dette bevirker i princippet at denne applikation skal udgives som open source. Men hvis det kom så langt vil der sikkert også kunne købes et antal kommerciel licenser af PjSip.

Kapitel 6

Test

Alle de tests der er gennemført i forbindelse med udviklingen af denne applikation er udført som "Black-box" tests. Dvs. at de testede enheder er blevet tilført en form for indput. Derefter er resultatet (output) af denne handling så blevet kontrolleret for at se om det levede op til forventningerne.

Først og fremmest er der udført nogle unit-tests for hver af de enkelte dele, som applikationen er opbygget af. Herudover er der der udført en "acceptance" test af det færdige produkt. Dette er gjort for at finde ud om produktet lever op til de opstillede krav.

For at kunne udfører de enkelte teste er der lavet nogle forskellige test-setup. Disse setup beskriver hvad der skal til for at teste en given unit og hvorledes selve testen er blevet udført.

For hver af de tests der er blevet udført er der udfyldt et testskema. Dette skema beskriver selve testen og hvad det forventes, samt selve resultatet af testen. Nederst på skemaet er der et felt som bruges til at konkludere på testen. Skemaet som er blevet brugt kan ses på Figur 6.1. Resultaterne af alle testene kan se i Bilag C.

Overskrift:		
Test nummer:	**Testtype:**	
Metode testet:	**Forventet handling:**	**Faktisk handling:**
Konklusion:		

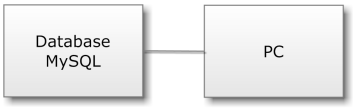
Figur 6.1: Testskema

6.1 Unit

6.1.1 Database

Databasen er for det første testet ved at logge ind på den lokalt. Herefter er den testet af ved at logge ind via et netværk. Databasen lytter på port 3306. Databasen er testet ved at udførere fire forskellige typer af statements på `sip_data`-tabellen. De fire typer at statements der er testet med er: `INSERT`, `SELECT`, `UPDATE` og `DELETE`.

Setup

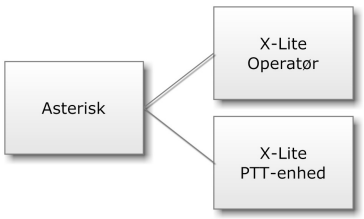


Figur 6.2: Testsetup: Database

6.1.2 Dialplan

For at teste dialplan'en er det nødvendigt at have mindst to SIP-enheder registreret på Asterisk-serveren. Den ene enhed skal udgøre det for en operatør og den anden skal udgøre det for PTT-enhed. Dette kræver at der er defineret mindst en af hver type i den liste af SIP-enheder som Asterisk-serveren bruger. Til denne test defineres dette i konfigurationsfilen `sip.conf`. Der er brugt et PC program men navnet `X-Lite 4`, som SIP-klienter ved denne test.

Setup



Figur 6.3: Testsetup: Dialplan

6.1.3 Python scripts

AsteriskDB.py
Dette script er testet ved at afvikle det lokalt på database-server. Det kræver at Python er

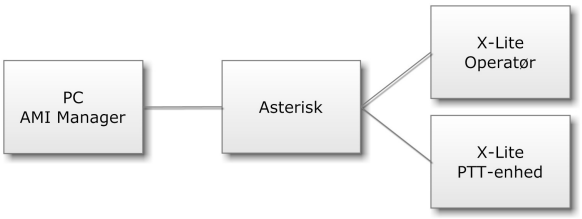
installeret på serveren, samt database-pakken med navnet `MySQLdb`. Dette script indhenter oplysninger fra alle tabellerne i databasen. Dette betyder at det skal være valid data tilgængelig i alle tabeller for at kunne udføre denne test.

AsteriskLib.py
Dette script skal kaldes fra dialplan'en så det er nødvendigt med samme setup som blev brugt ved selve testen af dialplan'en. Dette setup kan ses på Figur 6.3. En PTT-enhed skal foretage et opkal for at dette script bliver aktiveret. De forskellige funktioner som scriptet tilbyder er så testet af en ad gangen. Det er nødvendigt med to SIP-enheder for at være sikker på at scriptet kan ringer videre til en anden SIP-enhed.

6.1.4 AMI Manager

Denne komponent er testet ved at lave et setup som vist på Figur 6.4. Først er det testet om det er muligt at logge ind på Asterisk-serveren vis AMI. Derefter er de forskellige AMI-kommandoer der er implementeret afprøvet. Til sidst er det testet om det er muligt at modtage de forskellige events. For at kunne teste action kommandoerne kræver det at der mindst to SIP-enheder er registeret på Asterisk-serveren, da den ene kommando bruges til af forbinde to SIP-enheder. Før de enkelte kommandoer er afsendt til serveren er de blevet udskrevet for at se om de var korrekt struktureret.

Setup



Figur 6.4: Testsetup: AMI Manager

6.1.5 SIP Manager

Med SIP Manageren skal man først og fremmest kunne registrer sig på Asterisk-serveren som en SIP-enhed. Derudover skal det så være muligt af oprette opkald og modtage opkald. til dette formål der der brugt en PC SIP-klient. Til sidst er det testet om det er muligt at modtage oplysningerne fra Asterisk-serveren vedrørende de SIP-enheder som er registeret.

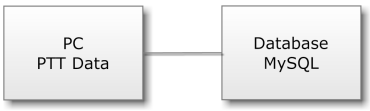
Setup

Figur 6.5: Testsetup: SIP Manager

6.1.6 PTT Data

Denne komponent skal kunne oprette forbindelse til databasen. Derudover er det teste af om det er muligt af læse, oprette, rette og slette informationer i `sip_data`-tabellen med komponenten.

Setup



Figur 6.6: Testsetup: PTT Data

6.1.7 Brugergrænsefladen

Brugergrænsefladens funktioner og udseende er testet ved at afvikler selve QML-scriptet på en PC. De PTT-data der er nødvendige, for at kunne vise noget på skærmen, er oprettet som stubbe. De forskellige hændelser som kan opstå i systemet er testet af ved at genereret dem kunstigt ved at ændre på tilstanden af de forskellige stubbe manuelt.

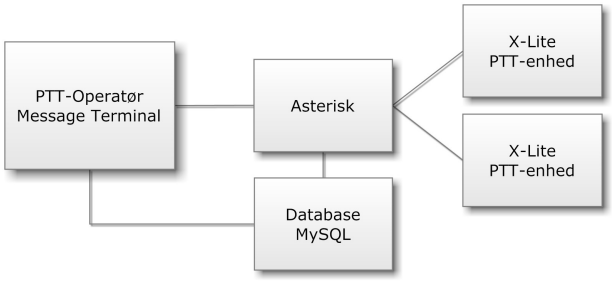
6.2 Acceptance

Der er udført en integration test hvor alle de forskellige dele, som hele projektet pestår af, er sat sammen og teste som et system, på samme tid. Dette er gjort for at se om de enkelte dele kan fungere sammen i selve systemet og for at se om applikationen lever op til kravene.

6.2.1 PTT-Operatør Applikation

Selve applikationen er testet i et system som skitseret på Figur 6.7. Ved denne test er de forskellige krav blevet afprøvet. Da der ikke er muligthed for at tilgå ”lyd-kortet“ i Message Terminalen er applikationen også blevet testet ved at afvikle den på en PC, i et tilsvarende system.

Setup



Figur 6.7: Testsetup: PTT Operstør

6.3 Test Konklusion

De forskellige unit-tests der er blevet udført af de enkelte dele har vist at de fungerer. Ved acceptance testen er der konstateret to forskellige problemer. Der først problem forekommer kun når applikationen afvikles fra en Message Terminal. Der kan forekomme at programmet konflikter med lyd-driveren (Alsa) og derved går med. Dette skylles formentlig af lyd-driveren ikke er sat rigtigt op på Message Terminalen. Det andet problem, der er observeret, er at en PTT-enhed periodisk kan skifte tilstand til `OFFLINE` selvom den stadig er `ONLINE`. Dette hænger måske sammen med den SIP-klient der anvendes (`X-Lite 4`).

Kapitel 7

Konklusion

Der er udviklet en prototype applikation. Denne applikation er i stand til at indgå, som operatør, i et kommunikationssystem som beskrevet i kapitel 2.

Brugeren af applikationen har mulighed for at kommunikere med forskellige PTT-enheder. Disse PTT-enheder er inddelt i grupper som brugerne at applikationen kan få tildelt.
Alle de funktionelle krav der er betegnet som Basisfunktionerne er implementeret i applikationen (Krav: FB1-FB4). Dvs. at der et muligt at oprette og modtage et opkald til og fra en PTT-enhed. Derudover er der muligt at få overblik over de PTT-enheder der er i den gruppe som operatøren har fået tildelt samt at se detaljer omkring hver enkelt PTT-enhed.
Alle de funktioner der knytter sig til Super-user aktøren, som er specificeret i kapitlet 3, er ikke implementeret.
Det er kun et af de funktionelle krav der er beskrevet som Udvidelser som er blevet implementeret. Dette er krav: FU10. Denne funktion gør det muligt for operatøren at se den aktuelle opkaldsoversigt (call-log) fro hver af de PTT-enheder som han varetager.

Et af de væsentlige ikke-funktionelle krav, som er specificeret, omhandler tilslutning af eksterne hovedtelefoner og mikrofon (IF3). Dette krav har det ikke været muligt at imødekomme. Dette bevirker at det ikke er muligt, med den implementeret prototype, hverken at høre hvad PTT-enheden siger eller tale tilbage til PTT-enheden. Dette skyldes at det ikke har været muligt at installere den korrekte lyd-driver på Message Terminalen. De stik som gør det muligt at tilslutte både hovedtelefoner og mikrofon findes på Message Terminalen. Hvis applikationen afvikles på en almindelig PC er dette ikke et problem.

Der er i forbindelse med testen af applikationen konstateret to fejl. Den ene hænger sammen med problemet med lyd-driveren på Message Terminalen. Der kan opstå en konflikt imellem applikationen og driveren som får applikationen til at gå ned. Denne fejl opstår ikke hvis applikationen afvikles fra en PC.

Den anden fejl som er observeret er at en PTT-enhed periodisk kan skifte tilstand til `OFFLINE`, selvom den stadig er `ONLINE`. Dette skyldes muligvis samspillet med den SIP-klient (`X-Lite 4`) der er valgt som erstatning for en PTT-enhed.

7.1 Komponenter

I forbindelse med udviklingen at applikationen er det implementeret en række forskellige komponenter. Alle disse komponenter indgår som en del af systemet. De enkelte komponenter kan bruges uafhængigt af hinanden til udvikling af andre programmer.

AMI Manager
Denne komponent kan anvendes til at oprette en AMI forbindelse til en Asterisk-server. Det er muligt at sende action-kommandoer til serveren. Der er kun implementeret et begrænset antal af de kommandoer der findes i forbindelse med dette interface. Det er også muligt at modtage beskeder (response/events) fra serveren med denne komponent. Ligeledes er det også her kun implementeret et udpluk af er mulige beskedtyper. Designet af denne komponent er udviklet således at, hvis yderligere kommandoer eller beskedtyper ønskes, kan de simpelt implementere.

PTT Data
I forbindelse med udviklingen af applikationen er der blevet tilføjet en database til systemet. Denne database indeholder data omkring de enkelte PTT-enheder samt hvilken gruppe de tilhører.
PTT Data komponenten gør det muligt at tilgå denne database og oprette, rette og slette PTT-enheder. Det er ikke muligt at tilgå de tabeller i databasen der vedrører grupper med denne komponent.

SIP Manager
Denne komponent er et Qt-interface til PjSIP-biblioteket. Med komponenten er det muligt at initialisere PjSIP med de ønskede indstillinger og derefter starte det op. Når PjSIP-biblioteket først er startet op er det muligt at oprette og modtage SIP-kald til og fra en Asterisk-serveren. Det er også muligt at overvåge tilstanden på de SIP-enheder der er registeret på serveren.

Ptyhon Scripts
Der er udviklet to forskellige Python-script som er beregnet til at blive anvendt i forbindelse med AGI. Det første script er et interface til AGI. Med dette interface er det muligt at interagere med Asterisk dialplan'en. Det er muligt at afvikle et udvalg af de forskellige funktioner som er tilgængelige via dialplan'en. Ikke alle funktioner er implementeret, med al den omliggende kode der gør interaktionen med serveren muligt er implementeret. Dette betyder at der ment kan tilføjes yderligere funktioner.
Det andet script gør det muligt at forbinde til databasen. Med dette script er det muligt at hente oplysninger ud omkring sammenhængen mellem PTT-enheder, operatøere og grupper.

7.2 Resultat

Et af de overordnede mål, med at af udviklet denne prototype, var at bruge Message Terminalen på en ny måde og derved få testet grænserne for brugen af denne. Derfor blev valgt at udvikle brugergrænsefladen til applikationen ved brugen af Qt Quick, hvilket ikke her været prøvet før. Den udviklede applikation har bevist at dette sagtens har kunne lade sig gøre og derved åbnet det mulighed for at anvende denne teknologi ved udviklingen af applikationer, til Message Terminalin, i fremtiden.
I forbindelse med det generalle PTT-projektet der foregår hos Thrane & Thrane kan den udviklede applikation fremvises som en slags "proof of concept". Applikationen vil kunne bruges som inspiration hvis det på et senere tidspunk besluttes at udvikle en tilsvarende applikation med henblik salg.

7.3 Udviklingsmuligheder

Det er selvsagt at der er en lang række forskellige funktionaliteter som vil kunne tilføjes applikation for at kunne udvide anvendelses muligheder af den. For det første er i dette projekt beskrevet et udvalg at forskellige udvidelser. Alle disse udvidelser er oplagte og ligger lige for at kunne implementere. Derudover er det også muligt at tilføje funktioner til applikationen som ligger udenfor hvad Asterisk kan tilbyde.
Herunder er beskrevet to muligt udvidelser som vil kunne gøre applikationen mere anvendelig. For det første er der givet et eksempel på hvordan voicemail kunne implementeres. Dette er en af de allerede beskrevet udvidelser. Den anden er et eksempel på hvordan applikationen kunne få tilføjet en helt ny anvendelse.

7.3.1 VoiceMail

Der er flere problemer ved den måde som Asterisk implementere voicemail på, som vanskeliggøre tilføjelsen af denne funktion til applikationen. Den måde som voicemail funktionen normalt implmenteres på er ved at alle PTT-enhederne skal have tilknyttet en unik mailbox. Dette gøres ved at oprette en mailbox til hver PTT-enhed i konfigurationsfilen med navnet `voicemail.conf` på serveren. Navnet (nummeret) på mailbox skal så tilføjes listen af informationer omkring de enkelte PTT-enheder i tabellen på databasen. Denne måde vil give en række problemer. For det første er der et problem i forbindelse med at informere operatøren om at der er tilgængelige voice-beskeder. SIP protokollen tilbyder muligheden for at få en notifikation om tilgængelige voice-beskeder. Men det vil ikke være mulighed for at få at vide hvor de enkelte beskeder kommer fra. Med får bare en information om et der en eller flere beskeder tilgængelige. Det vil bevirke at man ikke har mulighed for at afspille en besked fre en bestemt PTT-enhed. Den eneste mulighed er at afspille beskederne i kronologisk rækkefølge. I og med de enkelte PTT-enheder kan flyttes rundt imellem de forskellige grupper vil dette også kunne skabe problemer. Hvis der er afgivet en voice-besked til en operatør som ikke er blevet aflyttet endnu og PTT-enheden flytter til en anden gruppe. Så vil beskeden måske ikke

være længere være relevant for den operatør som har beskeden. Operatøren har heller ikke mulighed for at besvare beskeden da han ikke længere kan kontakte PTT-enheden.

En anden måde at implementere voicemail-funktionen på kunne være at gøre brug af databasen. Det er nemlig muligt at sætte Asterisk-serveren op således at den gemmer voicemail beskederne på direkte i databasen i stedet for at gøre det lokalt på serveren. Udover selve "lyden" leverer serveren også nogle forskellige informationer omkring den enkelte besked, såsom tidspunke og hvem der har afgivet beskeden og hvem der skal modtage den. Frem for at oprette en mailbox til hver af operatørerne kan man oprette en stor mailbox hvor alle voice-beskeder havner. For at indikere om der er voice-beskeder tilgængelige kan de enkelte `PTTUser` objekter selv kontakte databasen og få oplyst om den tilsvarende PTT-enhed har afgivet en besked. Er det tilfældet vil dette kunne blive indikeret grafisk. På den måde kan man se præcist hvilken PTT-enhed som har afgivet beskeder. `PTTUser`-objektet kan så hente beskeden lokalt ind i applikationen fra databasen. Det er ganske simpelt at afspille "lyd-sekvensen" med Qt-fraevork'et. Idet beskeden er blevet afspillet kan den blive slettet på databasen. I dette system vil beskederne ikke have nogle bestemt modtager. Men til gengæld vil den operatør som varetager PTT-enheden kunne afspille beskeden.

7.3.2 Google Map

Det vil være muligt at få til informationer omkring hvor de enkelte PTT-enheder rent geografisk befinder sig. Den satellit-antenne, som PTT-enheden anvender til at komme på internettet med, har også GPS-informationerne omkring den aktuelle position af PTT-enheden. Der skal først udvikles et interface som gør det muligt for PTT-operatør applikationen få adgang til GPS-informationerne. Med sådanne et interface vil det forholdsvis simpelt kunne præsentere alle PTT-enhederne på et grafisk kort. En oplagt mulighed er at anvende Google Maps. Dette er en web-baseret tjeneste som vil kunne anvendes ved at bruge Qt web browser engine. Qt har lavet en portering af open source projektet WebKit. Dette Qt modul hedder `QtWebKit`. Denne web engine kan afvikle både HTML og JavaScript kode som er hvad der skal til for at kunne anvende Google Maps.

7.4 Opsummering

Der er lykkedes at udvikle en applikation der lever op til, og løser det oprindelige problem som er beskrevet i problemformuleringen. Applikation kan afvikles på en Message Terminal og gør det muligt at optræde som PTT-operatør i et kommunikationsystem bestående af Thrane & Thrane PTT-enheder.

Litteratur

[1] Nokia Corporation. Qt - cross-platform application and ui framework. http://qt.nokia.com/. Besøgt Januar 22, 2012.

[2] Nokia Corporation. Qt developer network. http://developer.qt.nokia.com/. Besøgt Januar 22, 2012.

[3] Diginum Inc. Asterisk. http://www.asterisk.org/. Besøgt Januar 22, 2012.

[4] GNU General Public License. Gpl. http://www.gnu.org/copyleft/gpl.html. Besøgt Januar 22, 2012.

[5] GNU Lesser General Public License. Lgpl. http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html. Besøgt Januar 22, 2012.

[6] Leif Madsen, Jim Van Meggelen, and Russell Bryant. *Asterisk: The Definitive Giude*. O'Reilly Media, Inc, 3 edition, 2011.

[7] osip.org The GNU oSIP library. osip. http://www.gnu.org/software/osip/. Besøgt Januar 22, 2012.

[8] pjsip.org Open Source SIP Stack. Pjsip. http://www.pjsip.org/. Besøgt Januar 22, 2012.

[9] A referance guide to all things VOIP. voip-info. http://www.voip-info.org/. Besøgt Januar 22, 2012.

[10] RFC3261. Session initiation protocol. http://tools.ietf.org/html/rfc3261.

[11] RFC3550. A transport protocol for real-time applications. http://tools.ietf.org/html/rfc3550.

[12] RFC5456. Inter-asterisk exchange version 2. http://www.rfc-editor.org/rfc/rfc5456.txt.

Bilag A

Asterisk

A.1 Asterisk konfigurationsfiler

```
1  Master configuration file:
2  asterisk.conf: Tell Asterisk the directories where everything is, including the
       directory containing all the other configuration files. By default,
      Asterisk looks for the asterisk.conf file in the /etc/asterisk directory,
      but you can supply a command line parameter to use a different asterisk.conf
       file.
3
4  Configuration of Asterisk channels:
5  adtranvofr.conf: Configure voice over frame relay (Adtran style) channels
6  agents.conf: Configure agent channels
7  h323.conf: Configure H323 channels
8  iax.conf: Configure IAX channels
9  mgcp.conf: Configure MGCP channels
10 modem.conf: Configure Modem channels (for ISDN, not for modems!)
11 phone.conf: Configure phone channels (Linux Telephony devices)
12 sip.conf: Configure SIP channels
13 sip_notify.conf: Configure SIP NOTIFY messages
14 skinny.conf: Configure Asterisk Skinny channels (Cisco SCCP)
15 vpb.conf: Configure vpb channels (Voicetronix cards)
16 zapata.conf: Configure Zap channels (Digium cards)
17
18 Configuration of Analog Display Services Interface:
19 adsi.conf
20 asterisk.adsi
21 Asterisk config telcordia-1.adsi
22
23 Configuration of the Dialplan:
24  extconfig.conf: Used by res_data to arrange external configuration (e.g. thru
       ODBC)
25  extensions.ael: The Asterisk Extensions Language
26  extensions.conf: The Dialplan
```

```
 parking.conf: Call Parking configuration. Note: This file has been renamed to
     features.conf as of Asterisk 1.0rc1 (17 july 2004)

Configuration of specific Dialplan Commands:
 alarmreceiver.conf: AlarmReceiver configuration
 chan_dahdi.conf: Asterisk cmd DAHDiLookup configuration
 dundi.conf: DUNDiLookup configuration
 enum.conf: EnumLookup configuration
 festival.conf: Festival configuration
 indications.conf: Playtones tone definitions
 meetme.conf: MeetMe conference configuration
 musiconhold.conf: MusicOnHold configuration
 queues.conf: Queue configuration
 voicemail.conf: VoiceMail configuration

Uncategorized configuration files:
alarmreceiver.conf: Configuration file for the AlarmReceiver application
alsa.conf
Asterisk config cdr_odbc.conf
Asterisk config cdr_pgsql.conf: Configuration of PostgreSQL CDR database for
    billing
cel.conf
cel_pgsql.conf
codecs.conf
dnsmgr.conf: Background DNS update manager (new in Asterisk v1.2)
features.conf: Call Parking and other features
http.conf: Built-in mini HTTP server
logger.conf: Configuration of what to log and where to log it
manager.conf: Configuration of the Asterisk manager API
modules.conf: Configuration of Asterisk module loading
Asterisk config odbc.conf: Configuration of UnixODBC drivers for Asterisk
oss.conf
privacy.conf
res_odbc.conf
rpt.conf
rtp.conf: Configuration of RTP ports for media
say.conf: Internationalised numbers and dates (new in Asterisk v1.4)
users.conf: Generate a "user" (phone, dioalplan, and just about everything)
```

A.2 CLI Commands

```
General commands
 !<command>: Executes a given shell command
 abort halt: Cancel a running halt
 add extension: Add new extension into context
 add ignorepat: Add new ignore pattern
 add indication: Add the given indication to the country
 debug channel: Enable debugging on a channel
 dont include: Remove a specified include from context
 help: Display help list, or specific help on a command
 include context: Include context in other context
 load: Load a dynamic module by name
 logger reload: Reopen log files. Use after rotating the log files.
 no debug channel: Disable debugging on a channel
```

```
14  originate: originate a call.
15  remove extension: Remove a specified extension
16  remove ignorepat: Remove ignore pattern from context
17  remove indication: Remove the given indication from the country
18  save dialplan: Overwrites your current extensions.conf file with an exported
        version based on the current state of the dialplan. A backup copy of your
        old extensions.conf is not saved. The initial values of global variables
        defined in the [globals] category retain their previous initial values; the
         current values of global variables are not written into the new extensions
        .conf. (:exclaim:) Using "save dialplan" will result in losing any comments
         in your current extensions.conf.
19  dialplan save (1.4): BROKEN, doesn't parse correctly. Overwrites your current
        extensions.conf file with an exported version based on the current state of
         the dialplan. A backup copy of your old extensions.conf is not saved. The
        initial values of global variables defined in the [globals] category retain
         their previous initial values; the current values of global variables are
        not written into the new extensions.conf. (:exclaim:) Using "save dialplan"
         will result in losing any comments in your current extensions.conf.
20  set verbose: Set level of verboseness
21  show agents: Show status of agents
22  show applications: Shows registered applications
23  show application: Describe a specific application
24  show channel: Display information on a specific channel
25  show channels: Display information on channels
26  show codecs: Display information on codecs
27  show conferences: Show status of conferences
28  show dialplan: Show dialplan
29  show hints: Show registered hints
30  show image formats: Displays image formats
31  show indications: Show a list of all country/indications
32  show locals: Show status of local channels
33  show manager command: Show manager commands
34  show manager connect: Show connected manager users
35  show parkedcalls: Lists parked calls
36  show queues: Show status of queues, see details here
37  show switches: Show alternative switches
38  show translation: Display translation matrix
39  soft hangup: Request a hangup on a given channel - in Asterisk 1.6.2: "channel
        request hangup <name>"
40  show voicemail users: List defined voicemail boxes
41  show voicemail zones: List zone message formats
42  devstate change: Change state of a custom device (new in Asterisk 1.6.0)
43
44 Server management
45  restart gracefully: Restart Asterisk gracefully, i.e. stop receiving new calls
        and restart at empty call volume
46  restart now: Restart Asterisk immediately
47  restart when convenient: Restart Asterisk at empty call volume
48
49 Note for Asterisk 1.2: Restart now is like a reload, not a real restart it just
       run the reload routines (thus open ports are not closed). Often you don't
      need really need to restart asterisk, instead just need to issue e.g. '
      unload chan_sip.so' and 'load chan_sip.so'.
50
51  reload: Reload configuration
52  stop gracefully: Gracefully shut down Asterisk, i.e. stop receiving new calls
      and shut down at empty call volume
```

```
53  stop now: Shut down Asterisk imediately
54  stop when convenient: Shut down Asterisk at empty call volume
55  dialplan reload: Reload extensions and only extensions (formerly extensions
        reload)
56  unload: Unload a dynamic module by name
57  show modules: List modules and info about them
58  show uptime: Show uptime information
59  show version: Display Asterisk version info
60
61 AGI commands
62  show agi: Show AGI commands or specific help
63  dump agihtml: Dumps a list of agi command in html format
64
65 Database handling commands
66  database del: Removes database key/value
67  database deltree: Removes database keytree/values
68  database get: Gets database value
69  database put: Adds/updates database value
70  database show: Shows database contents
71  database showkey: Shows database contents: An alternative to showing keys by
        family with database show, this command shows all the families with a
        particular key
72
73 IAX Channel commands
74  iax2 debug: Enable IAX debugging
75  iax2 no debug: Disable IAX debugging
76  iax2 set jitter: Sets IAX jitter buffer
77  iax2 show cache: Display IAX cached dialplan
78  iax2 show channels: Show active IAX channels
79  iax2 show netstats: Show network and jitter buffer statistics for active IAX
        calls
80  iax2 show peers: Show defined IAX peers
81  iax2 show registry: Show IAX registration status
82  iax2 show stats: Display IAX statistics
83  iax2 show users: Show defined IAX users
84  iax2 trunk debug: Request IAX trunk debug
85
86  iax debug: Enable IAX debugging
87  iax no debug: Disable IAX debugging
88  iax set jitter: Sets IAX jitter buffer
89  iax show cache: Display IAX cached dialplan
90  iax show channels: Show active IAX channels
91  iax show peers: Show defined IAX peers
92  iax show registry: Show IAX registration status
93  iax show stats: Display IAX statistics
94  iax show users: Show defined IAX users
95  init keys: Initialize RSA key passcodes
96  show keys: Displays RSA key information
97
98
99 H323 channel commands
100  h.323 debug: Enable chan_h323 debug
101  h.323 gk cycle: Manually re-register with the Gatekeper
102  h.323 hangup: Manually try to hang up a call
103  h.323 no debug: Disable chan_h323 debug
104  h.323 no trace: Disable H.323 Stack Tracing
105  h.323 show codecs: Show enabled codecs
```

```
106  h.323 show tokens: Manually try to hang up a call
107  h.323 trace: Enable H.323 Stack Tracing
108
109 SIP channel commands
110 DebuggingEnablesip debug
111 sip set debug on (valid on 1.6.2.7)
112
113 Disable sip no debug
114  sip set debug off (valid on 1.6.2.7)
115  sip reload: Reload sip.conf (added after 0.7.1 on 2004-01-23)
116  sip show channels: Show active SIP channels
117  sip show channel: Show detailed SIP channel info
118  sip show inuse: List all inuse/limit
119  sip show peers: Show defined SIP peers (clients that register to your Asterisk
       server), see details here
120  sip show registry: Show SIP registration status (when Asterisk registers as a
       client to a SIP Proxy)
121  sip show subscriptions: Lists all sip presence (busy lamp indication)
       subscriptions
122  sip show users: Show defined SIP users
123
124 Zap channel commands
125  zap destroy channel: Destroy a channel
126  zap show channels: Show active zapata channels
127  zap show channel: Show information on a channel
128  zap show status: lists all the Zaptel spans. A span will apear here whether or
       not its channels are configured with chan_zap.
129  zap show cadences: Show the configured ring cadences (available e.g with Zap/1
       r2).
130  zap set swgain(<= 1.6): set the (software) gain for a hannel. Temporary
       equivalents of rxgain and txgain in zapata.conf.
131  zap set hwgain(<=1.6): set the hardware gain for channels that support it.
132  zap set dnd(<=1.6) set a channel's do-not-disturb mode on or off.
133
134 The following commands are available if the channel is built with support for
      libpri:
135
136  pri debug span: Enables PRI debugging on a span
137  pri intense debug span: Enables REALLY INTENSE PRI debugging
138  pri no debug span: Disables PRI debugging on a span
139  pri show spans: List spans and their status.
140  pri show span: Information about a span.
141  pri show debug: show where debug is enabled.
142
143 Console channel commands
144  dial : Dials the given extension, if specified, from the console. Can be used
       to initiate a call, or to dial digits during an existing call.
145  answer: Answer a call if one is currently ringing on the console.
146  hangup: Hangup the call if there is currently one on the console.
147
148 Asterisk channel MGCP commands
149  mgcp audit endpoint: Audit specified MGCP endpoint
150  mgcp debug: Enable MGCP debugging
151  mgcp no debug: Disable MGCP debugging
152  mgcp show endpoints: Show defined MGCP endpoints
153
154 skinny channel commands
```

```
155  skinny debug: Enable Skinny debugging
156  skinny no debug: Disable Skinny debugging
157  skinny show lines: Show defined Skinny lines per device
158
159 Asterisk channel CAPI commands
160  capi debug: Enable CAPI debugging
161  capi no debug: Disable CAPI debugging
162  capi info: Show CAPI info
163
164 Sirrix ISDN channel commands
165  srx reload: Reload channel driver configuration; active calls are not being
         terminated!
166  srx show ccmsgs: Disable / enable output of incoming callcontrol messages.
167  srx show chans: Show info about B-Channels
168  srx show globals: Show info about global settings
169  srx show groups: Show info about configured groups
170  srx show layers: Show info about ISDN stack (Layer 1, 2, 3)
171  srx show sxpvts: Show private info about active channels
172  srx show timers: Show info about running timers
```

A.3 AMI Commands

```
 1 Manager Actions
 2 Output from the CLI command show manager commands:
 3 (For Asterisk 1.4 and greater, use manager show commands)
 4
 5  AbsoluteTimeout: Set Absolute Timeout (privilege: call,all)
 6  ChangeMonitor: Change monitoring filename of a channel (privilege: call,all)
 7  Command: Execute Command (privilege: command,all)
 8  Events: Control Event Flow
 9  ExtensionState: Check Extension Status (privilege: call,all)
10  GetVar: Gets a Channel Variable (privilege: call,all)
11  Hangup: Hangup Channel __(privilege: call,all)
12  IAXpeers: List IAX Peers (privilege: system,all)
13  ListCommands: List available manager commands
14  Logoff: Logoff Manager
15  MailboxCount: Check Mailbox Message Count (privilege: call,all)
16  MailboxStatus: Check Mailbox (privilege: call,all)
17  Monitor: Monitor a channel (privilege: call,all)
18  Originate: Originate Call (privilege: call,all) NOTE: starting from 1.6:
        originate,all
19  ParkedCalls: List parked calls
20  Ping: Ping
21  QueueAdd: Queues (privilege: agent,all)
22  QueueRemove: Queues (privilege: agent,all)
23  Queues: Queues
24  QueueStatus: Queue Status
25  Redirect: Redirect (privilege: call,all)
26  SetCDRUserField: Set the CDR UserField (privilege: call,all)
27  SetVar: Set Channel Variable (privilege: call,all)
28  SIPpeers: List SIP Peers (chan_sip2 only. Not available in chan_sip as of
        9/20/2004) (privilege: system,all)
29  Status: Status (privilege: call,all)
30  StopMonitor: Stop monitoring a channel (privilege: call,all)
```

```
31  ZapDialOffhook: Dial over Zap channel while offhook
32  ZapDNDoff: Toggle Zap channel Do Not Disturb status OFF
33  ZapDNDon: Toggle Zap channel Do Not Disturb status ON
34  ZapHangup: Hangup Zap Channel
35  ZapTransfer: Transfer Zap Channel
36  ZapShowChannels: Show Zap Channels
37
38 (New?) in Asterisk 1.2.1 (was "CVS HEAD") (Taken from the output of CLI command
       show manager commands):
39  AgentCallbackLogin: Sets an agent as logged in by callback (Privilege: agent,
       all)
40  AgentLogoff: Sets an agent as no longer logged in (Privilege: agent,all)
41  Agents: Lists agents and their status (Privilege: agent,all)
42  DBGet: Get DB Entry (Privilege: system,all)
43  DBPut: Put DB Entry (Privilege: system,all)
44  QueuePause: Makes a queue member temporarily unavailable (Privilege: agent,all
       )
45  SIPshowPeer: Show SIP peer (text format) (Privilege: system,all)
46
47 New in Asterisk 1.4.0
48  GetConfig: Display a configuration file, used mainly by AJAM/Asterisk-gui. (
       Privilege: config,all)
49  PlayDTMF: Play DTMF signal on a specific channel. (Privilege: call,all)
50  UpdateConfig: Updates a configuration file, used mainly by AJAM/Asterisk-gui.
       (Privilege: config,all)
51
52
53 Available in Asterisk 1.6.0
54  AbsoluteTimeout: Set Absolute Timeout (Priv: system,call,all)
55  AgentLogoff: Sets an agent as no longer logged in (Priv: agent,all)
56  Agents: Lists agents and their status (Priv: agent,all)
57  AGI: Add an AGI command to execute by Async AGI (Priv: call,all)
58  Bridge: Bridge two channels already in the PBX (Priv: call,all)
59  Challenge: Generate Challenge for MD5 Auth (Priv: <none>)
60  ChangeMonitor: Change monitoring filename of a channel (Priv: call,all)
61  Command: Execute Asterisk CLI Command (Priv: command,all)
62  CoreSettings: Show PBX core settings (version etc) (Priv: system,reporting,all
       )
63  CoreShowChannels: List currently active channels (Priv: system,reporting,all)
64  CoreStatus: Show PBX core status variables (Priv: system,reporting,all)
65  CreateConfig: Creates an empty file in the configuration directory (Priv:
       config,all)
66  DAHDIDialOffhook: Dial over DAHDI channel while offhook (Priv: <none>)
67  DAHDIDNDoff: Toggle DAHDI channel Do Not Disturb status OFF (Priv: <none>)
68  DAHDIDNDon: Toggle DAHDI channel Do Not Disturb status ON (Priv: <none>)
69  DAHDIHangup: Hangup DAHDI Channel (Priv: <none>)
70  DAHDIRestart: Fully Restart DAHDI channels (terminates calls) (Priv: <none>)
71  DAHDIShowChannels: Show status dahdi channels (Priv: <none>)
72  DAHDITransfer: Transfer DAHDI Channel (Priv: <none>)
73  DBDel: Delete DB Entry (Priv: system,all)
74  DBDelTree: Delete DB Tree (Priv: system,all)
75  DBGet: Get DB Entry (Priv: system,reporting,all)
76  DBPut: Put DB Entry (Priv: system,all)
77  Events: Control Event Flow (Priv: <none>)
78  ExtensionState: Check Extension Status (Priv: call,reporting,all)
79  GetConfigJSON: Retrieve configuration (JSON format) (Priv: system,config,all)
80  GetConfig: Retrieve configuration (Priv: system,config,all)
```

```
 81  Getvar: Gets a Channel Variable (Priv: call,reporting,all)
 82  Hangup: Hangup Channel (Priv: system,call,all)
 83  IAXnetstats: Show IAX Netstats (Priv: system,reporting,all)
 84  IAXpeerlist: List IAX Peers (Priv: system,reporting,all)
 85  IAXpeers: List IAX Peers (Priv: system,reporting,all)
 86  ListCategories: List categories in configuration file (Priv: config,all)
 87  ListCommands: List available manager commands (Priv: <none>)
 88  Login: Login Manager (Priv: <none>)
 89  Logoff: Logoff Manager (Priv: <none>)
 90  MailboxCount: Check Mailbox Message Count (Priv: call,reporting,all)
 91  MailboxStatus: Check Mailbox (Priv: call,reporting,all)
 92  MeetmeMute: Mute a Meetme user (Priv: call,all)
 93  MeetmeUnmute: Unmute a Meetme user (Priv: call,all)
 94  ModuleCheck: Check if module is loaded (Priv: system,all)
 95  ModuleLoad: Module management (Priv: system,all)
 96  Monitor: Monitor a channel (Priv: call,all)
 97  Originate: Originate Call (Priv: originate,all)
 98  ParkedCalls: List parked calls (Priv: <none>)
 99  Park: Park a channel (Priv: call,all)
100  PauseMonitor: Pause monitoring of a channel (Priv: call,all)
101  Ping: Keepalive command (Priv: <none>)
102  PlayDTMF: Play DTMF signal on a specific channel. (Priv: call,all)
103  QueueAdd: Add interface to queue. (Priv: agent,all)
104  QueueLog: Adds custom entry in queue_log (Priv: agent,all)
105  QueuePause: Makes a queue member temporarily unavailable (Priv: agent,all)
106  QueuePenalty: Set the penalty for a queue member (Priv: agent,all)
107  QueueRemove: Remove interface from queue. (Priv: agent,all)
108  QueueRule: Queue Rules (Priv: <none>)
109  Queues: Queues (Priv: <none>)
110  QueueStatus: Queue Status (Priv: <none>)
111  QueueSummary: Queue Summary (Priv: <none>)
112  Redirect: Redirect (transfer) a call (Priv: call,all)
113  Reload: Send a reload event (Priv: system,config,all)
114  SendText: Send text message to channel (Priv: call,all)
115  Setvar: Set Channel Variable (Priv: call,all)
116  ShowDialPlan: List dialplan (Priv: config,reporting,all)
117  SIPpeers: List SIP peers (text format) (Priv: system,reporting,all)
118  SIPshowpeer: Show SIP peer (text format) (Priv: system,reporting,all)
119  SIPshowregistry: Show SIP registrations (text format) (Priv: system,reporting,
         all)
120  Status: Lists channel status (Priv: system,call,reporting,all)
121  StopMonitor: Stop monitoring a channel (Priv: call,all)
122  UnpauseMonitor: Unpause monitoring of a channel (Priv: call,all)
123  UpdateConfig: Update basic configuration (Priv: config,all)
124  UserEvent: Send an arbitrary event (Priv: user,all)
125  VoicemailUsersList: List All Voicemail User Information (Priv: call,reporting,
         all)
126  WaitEvent: Wait for an event to occur (Priv: <none>)
```

A.4 AGI Commands

```
1 AGI commands
2  answer: Asserts answer
3  asyncagi break: Break Async AGI loop (since Asterisk 1.6)
```

```
4  channel status: Returns status of the connected channel
5  control stream file: Send the given file, allowing playback to be controlled
       by the given digits, if any. (since Asterisk 1.2)
6  database del: Removes database key/value
7  database deltree: Removes database keytree/value
8  database get: Gets database value
9  database put: Adds/updates database value
10 exec: Executes a given Application. (Applications are the functions you use to
       create a dial plan in extensions.conf ).
11 get data: Gets data on a channel
12 get full variable: Gets a channel variable, but understands complex variable
       names and builtin variables. (since Asterisk 1.2)
13 get option: Behaves similar to STREAM FILE but used with a timeout option. (
       since Asterisk 1.2)
14 get variable: Gets a channel variable
15 hangup: Hangup the current channel
16 noop: Does nothing
17 receive char: Receives one character from channels supporting it
18 receive text: Receives text from channels supporting it
19 record file: Records to a given file
20 say alpha: Says a given character string (since Asterisk 1.2)
21 say date: Say a date (since Asterisk 1.2)
22 say datetime: Say a formatted date and time (since Asterisk 1.2)
23 say digits: Says a given digit string
24 say number: Says a given number
25 say phonetic: Say the given character string.
26 say time: Say a time
27 send image: Sends images to channels supporting it
28 send text: Sends text to channels supporting it
29 set autohangup: Autohangup channel in some time
30 set callerid: Sets callerid for the current channel
31 set context: Sets channel context
32 set extension: Changes channel extension
33 set music: Enable/Disable Music on hold generator, example "SET MUSIC ON
       default"
34 set priority: Prioritizes the channel
35 set variable: Sets a channel variable
36 speech activate grammar: Activates a grammar (since Asterisk 1.6)
37 speech create: Creates a speech object (since Asterisk 1.6)
38 speech deactivate grammar: Deactivates a grammar (since Asterisk 1.6)
39 speech destroy: Destroys a speech object (since Asterisk 1.6)
40 speech load grammar: Loads a grammar (since Asterisk 1.6)
41 speech recognize: Recognizes speech (since Asterisk 1.6)
42 speech set: Sets a speech engine setting (since Asterisk 1.6)
43 speech unload grammar: Unloads a grammar (since Asterisk 1.6)
44 stream file: Sends audio file on channel
45 tdd mode: Activates TDD mode on channels supporting it, to enable
       communication with TDDs.
46 verbose: Logs a message to the asterisk verbose log
47 wait for digit: Waits for a digit to be pressed
48
49 Cam Farnells AGI Documentation - November 2002
50 AGI Documentation
51 AGI Documentation (with long descriptions)
52 Use show agi [agi-command] and dump agihtml <filename> at the Asterisk CLI to
       get information on the commands. For debugging purposes you can type agi
       debug to see each command and response as they happen.
```

A.5 AGI Variables

```
agi_request - The filename of your script
agi_channel - The originating channel (your phone)
agi_language - The language code (e.g. "en")
agi_type - The originating channel type (e.g. "SIP" or "ZAP")
agi_uniqueid - A unique ID for the call
agi_version - The version of Asterisk (since Asterisk 1.6)
agi_callerid - The caller ID number (or "unknown")
agi_calleridname - The caller ID name (or "unknown")
agi_callingpres - The presentation for the callerid in a ZAP channel
agi_callingani2 - The number which is defined in ANI2 see Asterisk Detailed
    Variable List (only for PRI Channels)
agi_callington - The type of number used in PRI Channels see Asterisk Detailed
    Variable List
agi_callingtns - An optional 4 digit number (Transit Network Selector) used in
    PRI Channels see Asterisk Detailed Variable List
agi_dnid - The dialed number id (or "unknown")
agi_rdnis - The referring DNIS number (or "unknown")
agi_context - Origin context in extensions.conf
agi_extension - The called number
agi_priority - The priority it was executed as in the dial plan
agi_enhanced - The flag value is 1.0 if started as an EAGI script, 0.0
    otherwise
agi_accountcode - Account code of the origin channel
agi_threadid - Thread ID of the AGI script (since Asterisk 1.6)
```

A.6 SIP response codes

```
1xx—Informational Responses
 100 Trying: extended search being performed may take a significant time so a
     forking proxy must send a 100 Trying response
 180 Ringing
 181 Call Is Being Forwarded
 182 Queued
 183 Session Progress

2xx—Successful Responses
 200 OK
 202 accepted: It Indicates that the request has been understood but actually
     can't be processed
 204 No Notification [RFC5839]

3xx—Redirection Responses
 300 Multiple Choices
 301 Moved Permanently
 302 Moved Temporarily
 305 Use Proxy
 380 Alternative Service

4xx—Client Failure Responses
 400 Bad Request
 401 Unauthorized (Used only by registrars or user agents. Proxies should use
     proxy authorization 407)
```

```
23  402 Payment Required (Reserved for future use)
24  403 Forbidden
25  404 Not Found (User not found)
26  405 Method Not Allowed
27  406 Not Acceptable
28  407 Proxy Authentication Required
29  408 Request Timeout (Couldn't find the user in time)
30  409 Conflict
31  410 Gone (The user existed once, but is not available here any more.)
32  412 Conditional Request Failed
33  413 Request Entity Too Large
34  414 Request-URI Too Long
35  415 Unsupported Media Type
36  416 Unsupported URI Scheme
37  417 Unknown Resource-Priority
38  420 Bad Extension (Bad SIP Protocol Extension used, not understood by the
        server)
39  421 Extension Required
40  422 Session Interval Too Small
41  423 Interval Too Brief
42  424 Bad Location Information
43  428 Use Identity Header
44  429 Provide Referrer Identity
45  433 Anonymity Disallowed
46  436 Bad Identity-Info
47  437 Unsupported Certificate
48  438 Invalid Identity Header
49  479 Regretfully, we were not able to process the URI (479/SL)
50  480 Temporarily Unavailable
51  481 Call/Transaction Does Not Exist
52  482 Loop Detected
53  483 Too Many Hops
54  484 Address Incomplete
55  485 Ambiguous
56  486 Busy Here
57  487 Request Terminated
58  488 Not Acceptable Here
59  489 Bad Event
60  491 Request Pending
61  493 Undecipherable (Could not decrypt S/MIME body part)
62  494 Security Agreement Required
63
64 5xx—Server Failure Responses
65  500 Server Internal Error
66  501 Not Implemented: The SIP request method is not implemented here
67  502 Bad Gateway
68  503 Service Unavailable
69  504 Server Time-out
70  505 Version Not Supported: The server does not support this version of the SIP
        protocol
71  513 Message Too Large
72  580 Precondition Failure
73
74 6xx—Global Failure Responses
75  600 Busy Everywhere
76  603 Decline
77  604 Does Not Exist Anywhere
```

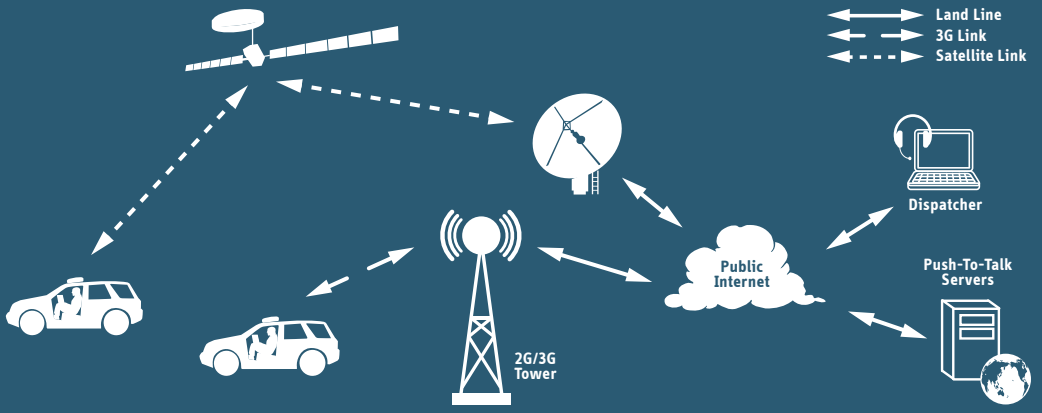
```
78  606 Not Acceptable
```

Bilag B

Product Sheet

B.1 EXPLORER Push-to-talk Product Sheet

EXPLORER PUSH-TO-TALK OVERVIEW



The complete EXPLORER 325 Push-To-Talk solution (3G modem optional)

71-134666-A02.10.11 LMBU

Thrane & Thrane A/S · landmobile@thrane.com · thrane.com

Thrane & Thrane

WHITEPAPER

EXPLORER™

PUSH-TO-TALK VOICE COMMUNICATION

EXPLORER PUSH-TO-TALK

EXPLORER Push-To-Talk (PTT) is a rugged voice dispatch and communication system. It is a cost effective, IP based voice and data communication system designed to replace VHF/UHF based trunk radio systems widely used in the field service, search and rescue, utility, mining and Oil & Gas sectors. The system extends classical Push-To-Talk capabilities to hybrid data networks such as terrestrial 2G/3G/GPRS networks where available supplemented by the Inmarsat BGAN satellite network where no terrestrial network coverage is present. With no user intervention required the system automatically routes voice and data traffic via the least expensive network available.

The EXPLORER PTT solution is the result of 1½ years of field evaluation. The customer, a South American electricity distribution company covering 66.000 miles of distribution lines and a concessional area of 120.000 square kilometers were facing a communication challenge. They were using a traditional VHF trunk radio system provided by an infrastructure of 180 VHF towers with combined satellite and land line backhaul to the dispatch center. The mobile work force consisted of 500+ field engineers maintaining the power grid and serving subscribers on a daily basis. Faced with high infrastructure maintenance costs, poor VHF voice quality and limited coverage the company was looking for an alternative. They were looking for a cost efficient, simple and user friendly communication solution with a PTT/VHF user experience and with improved voice quality, support of data connectivity as well as expanded coverage.

www.thrane.com/explorer
landmobile@thrane.com
+45 39 55 88 00

EXPLORER™

THE PUSH-TO-TALK SOLUTION

The EXPLORER PTT solution solved the majority of the built-in challenges of the previous VHF system.

- The use of BGAN and already existing 2G/3G/GPRS networks extended the coverage area.
- Improved voice quality with IP based digital voice quality compared to the analogue VHF voice quality.
- No expensive infrastructure (no VHF towers, no fixed VSAT) and thereby lower maintenance costs.
- High quality voice but also an on-the-move internet connection.

Simplicity permeates the EXPLORER PTT solution. The look and feel is exactly like the VHF system replaced - only a few buttons (on/off and call) and easy push to transmit communication.

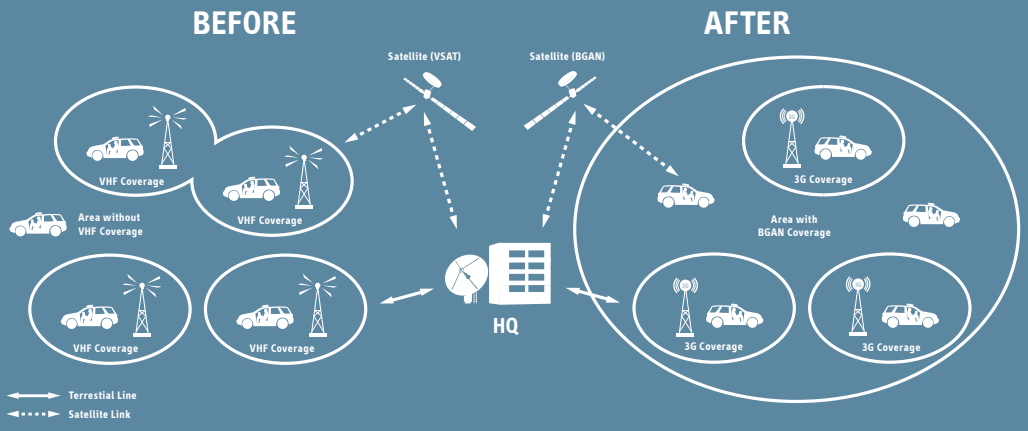


Illustration 1: The communication challenge - before and after

A car installation includes a vehicular EXPLORER BGAN terminal with a roof mounted antenna and a base station with a hand microphone (fist-mike) in the cabin. The all important built-in least cost routing functionality enables automatic switching between the available networks. The system is designed to use existing cellular networks as default. Two independent cellular networks can be supported simultaneously. If one of the cellular networks is congested or unavailable the system will automatically switch to the other 2G/3G/GPRS network. If the vehicle is situated in areas with limited or no cellular coverage the system will switch to the Inmarsat BGAN satellite network.

When the system is connected to the cellular networks the cost of communication is typically based on a flat rate data package. A state of the art voice transfer protocol ensures that the bandwidth usage is minimized to an absolute minimum in order for optimal use on background satellite connections.

In summary EXPLORER Push-To-Talk turns traditional satellite communication, cellular networks and the internet into a closed managed Wide Area Network with beyond line-of-sight communication.

SYSTEM COMPONENTS

- The Mobile Unit
- EXPLORER Push-To-Talk servers
- Dispatch client software

Mobile Unit

The mobile part of the PTT system consists of:

- Push-To-Talk user terminal (PTT box)
- Hand microphone/speaker (fist-mike)
- Vehicular BGAN terminal with a seperate antenna & transceiver
- One or two 2G/3G/GPRS USB modems

The fist-mike is connected to the PTT box and gives the user a "VHF-radio-like" experience. The PTT box has several ways of communicating with the PTT servers, either through the BGAN network or one of the 2G/3G/GPRS modems.

The user interface of the PTT box is extremely simple to use with only a dial-button, a few LEDs and a combined volume control/on-off button. Once the user presses the dial-button a VoIP connection is established between the PTT box and the dispatcher. When the dispatcher answers the call, a simplex communication channel is available. In the same way the dispatcher may call the mobile unit; to answer the call the mobile user simply presses the dial button.

In order to secure communication, the PTT box must always be connected to an EXPLORER BGAN terminal but can also use up to two 2G/3G/GPRS cellular USB modems to ensure better coverage, reachability and cost efficiency. Routing over the multiple network interfaces can be setup for least cost routing. The PTT box can also route data e.g. from a PC connected via a standard Ethernet interface.

A vital part of the system is the BGAN terminal from Thrane & Thrane. All EXPLORER BGAN terminals can be used depending on the environment in which the PTT box is deployed. If the PTT box is mounted in a vehicle the EXPLORER 727 or EXPLORER 325 would be the right choices. For stationary use the PTT box can operate with an EXPLORER 700 (or 300/500) and for maritime use the SAILOR FBB series. The BGAN terminal ensures reliable communication in rural areas where no 2G/3G/GPRS network has been built or if existing 2G/3G/GPRS networks is congested or lost due to natural disaster, extreme weather etc.

EXPLORER Push-To-Talk server infrastructure

The heart of the system is the PTT servers. A server setup initially consists of two standard 19" rack servers that can be placed anywhere connected to the public internet:

1. The PTT server is handling all switching between the mobile units and the dispatchers.
2. The transcoding server handles the voice transcoding and compression

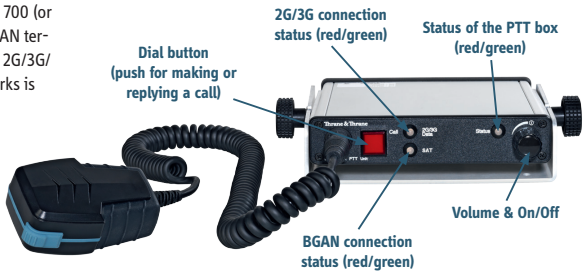
The PTT server can handle any number of users, while the transcoding server can accommodate 20 concurrent calls, more transcoding servers can be added when the need arises. All information is stored in the PTT server including:

- GPS position of each PTT user.
- Statistics on availability of each PTT user on each network.
- Recordings of call logs and voice communication.

Dispatch client software

The Dispatch client software is a combined PABX switchboard and VOIP softphone. The software runs on a standard PC and the dispatcher uses a headset to communicate with the PTT box users in the field. Through the dispatch client software, the dispatcher can

- answer calls from mobile users
- initiate calls to mobile users
- create conference calls between mobile users
- lookup the GPS position of each mobile user on map (google maps or similar).



B.2 SAILOR 6006 Message Terminal Product Sheet

Thrane & Thrane

SAILOR® 6006 MESSAGE TERMINAL

World's first touch-screen GMDSS terminal

The revolutionary SAILOR 6006 Message Terminal is designed to enhance the efficiency of safety communication as the defining component in a SAILOR 6000 GMDSS installation, as Inmarsat mini-C or as a radiotelex terminal. Regardless of its application, it is a game changing product that introduces intuitive touch-screen technology to maritime safety communications for easier, safer and more efficient operation.

The SAILOR 6006 Message Terminal stands out as the world's first touch-screen, Wheelmarked terminal. It enables the safe operation of a vessel's IMO GMDSS safety systems either when connected to the SAILOR 6110 mini-C or when operating in conjunction with the MF/HF installation to provide seamless radiotelex communications.

Features include:

- High contrast touch-screen display with dimming
- Perfect night and day vision
- SD card and USB support
- Integrated GMDSS distress button
- Easy-to-use icon based interface
- ThraneLINK

A number of sub-systems, such as alarm panels can be connected using RJ45 Ethernet cables via the SAILOR 6197 Switch, making installation of the SAILOR 6006 Message Terminal easier and less costly. Additional cost-saving features include a revolutionary approach to cabling with the use of NMEA 2000 style cables and connectors, and the ability to present GMDSS system battery data, therefore removing the need for a battery panel, provided that the SAILOR 6081 Power Supply and Charger is used.

Touch-screen

The SAILOR 6006 Message Terminal features a high resolution 800x600 touch-screen that offers clear benefits to safety and operational efficiency, as it enables the user to quickly and easily carry out GMDSS and communication tasks, therefore leaving more time for other critical jobs on board. Even when using the keyboard and mouse input method, controlling the multimedia style interface, with easy-to-use, intuitive, icon based operation is very straightforward.

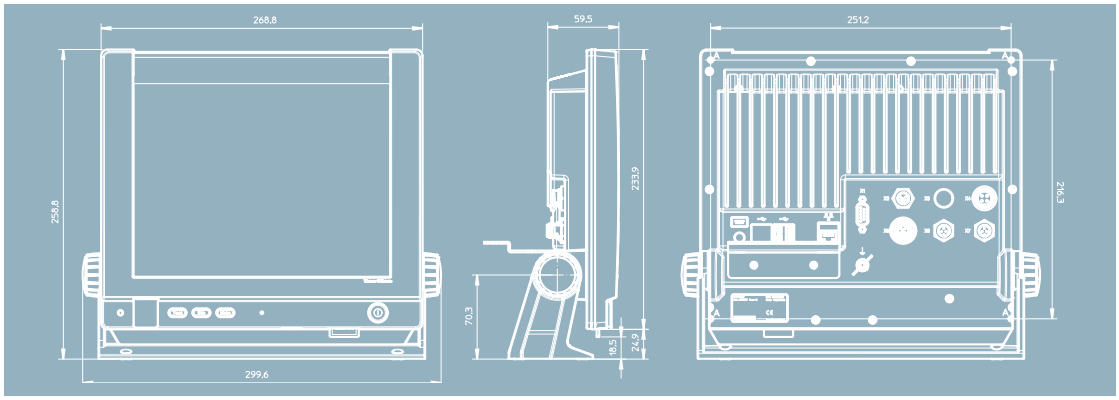
Purpose Built

The SAILOR 6006 Message Terminal is built specifically for use at sea, and with the SAILOR dedication to quality, it is a highly reliable system that is able to withstand the harsh maritime environment.

SAILOR®

SAILOR® 6006 MESSAGE TERMINAL

SAILOR®



TECHNICAL SPECIFICATIONS

CPU	Intel Atom based CPU (1,1 GhZ)
Memory	1 GB
Internal storage	1 GB Flash for OS and application software
Additional storage media	SD card slot (SDHC)
USB Host interface connecter for mass storage devices	

INTERFACES

5 x USB Host interface ports (all up to 480 MB)	
Ethernet (10/100Mbit), RJ45	
RS232, DB9 male	
Isolated CAN-bus interface	
NMEA 0183 compatible talker, reference to chassis (secondary gnd)	
NMEA 0183 compatible listener, isolated, 4800 Baud max.	
NMEA 0183 compatible listener, isolated, 4800/38400 Baud max.	
One digital input pin for simple active/inactive detection	

ENVIRONMENTAL

Meets or exceeds all Inmarsat specifications for the Inmarsat-C Network for SOLAS with distress callfunctions. (CN114 and IEC 945 requirements)	
Meets CE-marking requirements	
IP protection class	IP30 on the rear section, IP33 on the front surface.

AMBIENT TEMPERATURE

-25°C to 55°C operating -40°C to 80°C storage	

POWER

Power 10.8 to 32 V DC, with “remote on/off” and “on/off control output	
Power Consumption	Max. 20 W, typical 12 W

DIMENSIONS

With out bracket	HxWxD: 233.9 x 268.8 x 59.9 mm
	HxWxD: 9.2 x 10.6 x 2.4 inch
With bracket at vertical position	HxWxD: 258.8 x 299.6 x 92.3 mm
	HxWxD: 10.2 x 11.8 x 3.6 inch

WEIGHT

Weight	2.0 KG

Some features and interfaces are not available with standard delivered WinXP OS. These can be enabled and accessed by specific software applications.

ThraneLINK

ThraneLINK is a sophisticated communication protocol that connects the SAILOR products in a network, offering important new opportunities to vessels. It provides facility for remote diagnostics and enables access to all the SAILOR products from a single point for service. This results in optimized maintenance and lower cost of ownership because less time is needed for troubleshooting and service. Installation is made easier as ThraneLINK automatically identifies new products in the system. The uniform protocol is an open standard which provides a future proof solution for all vessels.

Subject to change without further notice.

Thrane & Thrane A/S · maritime@thrane.com · www.thrane.com

71-132181-A1-00 09.10 MBU

Bilag C

Test Resultater

C.1 Unit

C.1.1 Database

Overskrift: `Database`		
Test nummer: `#01`	**Testtype:** `Black-box test`	
Funktionalitet testet:	**Forventet handling:**	**Faktisk handling:**
`INSERT`	Der bliver oprettet en ny PTT-enhed i databasen	Som forventet. `GODKENDT`
`SELECT`	Den givende PTT-enhed bliver returneret	Som forventet. `GODKENDT`
`UPDATE`	Den givende PTT-enhed bliver opdateret	Som forventet. `GODKENDT`
`DELETE`	Den givende PTT-enhed bliver slettet	Som forventet. `GODKENDT`
Konklusion: Alle tests er forløbet som forventet		

C.1.2 Dialplan

Overskrift: `Dialplan`		
Test nummer: `#01`	**Testtype:** `Black-box test`	
Funktionalitet testet:	**Forventet handling:**	**Faktisk handling:**
Indkommende kald i operatør konteksted	Kaldet kommer ind i den rigtige kontekst (Testes ved at afspille en tone bestemt tone)	Som forventet. `GODKENDT`
Indkommende kald i PTT konteksted	Kaldet kommer ind i den rigtige kontekst (Testes ved at afspille en tone bestemt tone)	Som forventet. `GODKENDT`
Timeout af indkommende kald	Afspiller en fejltone efter 10 sek. hvis opkaldet ikke bliver besvaret	Som forventet. `GODKENDT`
Opkald til en enhed som ikke er registeret på serveren	Der vil blive afspillet en fejltone	Som forventet. `GODKENDT`
Konklusion: Alle tests er forløbet som forventet		

C.1.3 Python script

Overskrift: `AsteriskDB.py`		
Test nummer: `#01`	**Testtype:** `Black-box test`	
Metode testet:	**Forventet handling:**	**Faktisk handling:**
`findDispatcherFromExt`	”name“ på den operatør som varetager den givende PTT-enhed bliver returneret	Som forventet. `GODKENDT`
`findUnitsFromEct`	En liste med alle de PTT-enheder som en bestemt operatør varetager bliver returneret	Som forventet. `GODKENDT`
`closeConnection`	Forbindelsen til databasen bliver afbrudt	Som forventet. `GODKENDT`
Konklusion: Alle tests er forløbet som forventet		

Overskrift: AsteriskLib.py		
Test nummer: #02	**Testtype:** Black-box test	
Funktionalitet testet:	**Forventet handling:**	**Faktisk handling:**
answer	Et indkommende kald til Dialplan'en bliver besvaret	Som forventet. GODKENDT
hangup	Et igangværende opkald bliver nedlagt	Som forventet. GODKENDT
dial	Opretter et kald til en bestemt VoIP-enhed	Som forventet. GODKENDT
playtones	Afspiller den givende sekvens at toner (frekvens) ud på linjen	Som forventet. GODKENDT
wait	Pauser afviklingen af Dialplan sekvensen i den givende til	Som forventet. GODKENDT
say_number	Siger det givende nummer ud på linjen (Engelsk)	Som forventet. GODKENDT
Konklusion: Alle tests er forløbet som forventet		

C.1.4 AMI Manager

Overskrift: AMIManager		
Test nummer: #01	**Testtype:** Black-box test	
Metode testet:	**Forventet handling:**	**Faktisk handling:**
login	Der bliver oprettet en AMI forbindelse til den givende Asterisk-server	Som forventet. GODKENDT
sendCommand	Den givende CLI-kommando bliver sendt til Asterisk-serveren	Som forventet. GODKENDT
sendAction	Den givende action-kommando bliver sendt til Asterisk-serveren	Som forventet. GODKENDT
Konklusion: Alle tests er forløbet som forventet		

Overskrift: AMIManager		
Test nummer: #01	**Testtype:** Black-box test	
Signal testet:	**Forventet handling:**	**Faktisk handling:**
peerStatusChanged	Hver gang en SIP-enhed skifter tilstand på Asterisk-serveren bliver dette signal fyret	Som forventet. GODKENDT
result	Hver gang en tekst-linje bliver modtages fra Asterisk-serveren fyres dette signal	Som forventet. GODKENDT
Konklusion: Alle tests er forløbet som forventet		

Overskrift: AMIManager		
Test nummer: #01	**Testtype:** Black-box test	
Kommando testet:	**Forventet handling:**	**Faktisk handling:**
SIP Show Peers	Bliver denne action-kommando sendt til Asterisk-serveren bliver en liste med alle de SIP-enheder, der kan register sig på serveren, returneret	Som forventet. GODKENDT
Originate	Denne action-kommando får Asterisk-serveren til at oprette forbindelse mellem de givende SIP-enheder	Som forventet. GODKENDT
Konklusion: Alle tests er forløbet som forventet		

C.1.5 SIP Manager

Overskrift: SIPManager		
Test nummer: #01	**Testtype:** Black-box test	
Metode testet:	**Forventet handling:**	**Faktisk handling:**
shutdownPjSIP	PjSIP-biblioteket bliver lukket ned	Som forventet. GODKENDT
call_hangup	Det givende kald bliver nedlagt	Som forventet. GODKENDT
call_answer	Et indkommende bliver besvaret	Som forventet. GODKENDT
call_make	Der oprettes et kald til den givende SIP-enhed	Som forventet. GODKENDT
Konklusion: Alle tests er forløbet som forventet		

Overskrift: SIPManager		
Test nummer: #02	**Testtype:** Black-box test	
Signal testet:	**Forventet handling:**	**Faktisk handling:**
new_log_message	Hver gang PjSIP generere en log-meddelelse bliver dette signal fyret med meddelelsen	Som forventet. GODKENDT
call_state	Når en SIP-enhed skifter SIP-tilstand på Asterisk-serven bliver dette signal fyret	Som forventet. GODKENDT
incoming_call	Dette signal bliver fyret når der er et indkommende kald til PjSIP-biblioteket	Som forventet. GODKENDT
Konklusion: Alle tests er forløbet som forventet		

C.1.6 PTT Data

Overskrift: PTTData		
Test nummer: #01	**Testtype:** Black-box test	
Metode testet:	**Forventet handling:**	**Faktisk handling:**
setCredentials	Brugeroplysningerne gives og der kan herefter logges ind	Som forventet. GODKENDT
getSipData	Data returneres på den givende PTT-enhed	Som forventet. GODKENDT
createSipData	Der oprettes en ny PTT-enhed i databasen	Som forventet. GODKENDT
delSipData	Den givende PTT-enhed bliver slettet i databasen	Som forventet. GODKENDT
updateSipData	Data omkring den givende PTT-enhed bliver opdateret i databasen	Som forventet. GODKENDT
getGroupList	Oplysningerne om den givende gruppe bliver returneret	Som forventet. GODKENDT
getDispatcher	Oplysningerne om den operatør som varetager den givende gruppe returneres	Som forventet. GODKENDT
Konklusion: Alle tests er forløbet som forventet		

C.1.7 Brugergrænseflade

Overskrift: `Brugergrænseflade`		
Test nummer: `#05`	**Testtype:** `Black-box test`	
Funktionalitet testet:	**Forventet handling:**	**Faktisk handling:**
Applikationen skal kunne præsentere grupper med mindst 15 PTT-enheder(IF8)	Da det er en liste som anvendes til at vise PTT-enhederne forventes det at der sagtens kan ligge mere end 15 enheder i den	Som forventet: `GODKENDT` (Listen er testet af med 100 enheder)
Præsentation af en bestemt PTT-enheds detaljer	Når der vælges en bestemt PTT-enhed skal denne enheds detaljer blive præsenteret. Grafikken skifte og detalje-præsentationen blive vist	Som forventet. `GODKENDT`
Præsentation af listen med PTT-enheder	Når der vælges ikke længere at få vist detaljer trykkes på der grå detalje-område. Grafikken skal skifte og liste-præsentationen skal blive vist	Som forventet. `GODKENDT`
Sortering af listen der viser PTT-enhederne	Når der trykkes på en af de fire knapper, som vælger et bestemt udsnit af enheder, skal listens indhold opdateres med den ønskede sortering	Som forventet. `GODKENDT`
Indikation af indkommende kald i liste-præsentationen	Den pågældende PTT-enhed får aktiveret en grøn markering. Sorterings-knappen med navnet ”Incoming“ begynder at blinke grøn	Som forventet. `GODKENDT`
Indikation af indkommende kald i liste-præsentationen og med ”Incoming“-sorteringen valgt	Den pågældende PTT-enhed bliver synlig i listen og får aktiveret en grøn markering. Sorterings-knappen med navnet ”Incoming“ begynder at blinke grøn	Som forventet. `GODKENDT`
Indikation af indkommende kald i detalje-præsentationen	Hvis det er den aktuelle PTT-enhed som ringer vil den grønne ”call“-knap begynde at blinke. Sorterings-knappen med navnet ”Incoming“ begynder at blinke grøn. Hvis det ikke er den aktuelle PTT-enhed som der bliver vist detaljer på vil ”call“-knap ikke blinke.	Som forventet. `GODKENDT`

Detalje-præsentation af ”offline“ PTT-enheder	De eneste detaljer som det er muligt at se er ”navn“ og ”alias“. Den grønne ”call“-knap vil være inaktiv og den vil have skiftet udseende til grå og med teksten: ”offline“	Som forventet. `GODKENDT`
Indikation af et ubesvaret opkald i liste-præsentationen	Den pågældende PTT-enhed får aktiveret en gul markering. Sorterings-knappen med navnet ”Missed“ skifter farve til gul.	Som forventet. `GODKENDT`
Indikation af et ubesvaret opkald i liste-præsentationen og med ”Missed“-sorteringen valgt	Den pågældende PTT-enhed bliver synlig i listen og får aktiveret en gul markering. Sorterings-knappen med navnet ”Missed“ skifter farve til gul.	Som forventet. `GODKENDT`
Indikation af et ubesvaret opkald i detalje-præsentationen	Sorterings-knappen med navnet ”Missed“ skifter farve til gul	Som forventet. `GODKENDT`
Visning af tilstands-log for en bestemt PTT-enhed	I detalje-præsentationen skal der være muligt at se en log med de forskellige tilstande som PTT-enheden har været i.	Som forventet. `GODKENDT`
Konklusion: Alle tests er forløbet som forventet		

C.2 Acceptance

C.2.1 PTT-Operstør Applikation

Overskrift: `Acceptance`		
Test nummer: `#01`	**Testtype:** `Black-box test`	
Funktionalitet testet:	**Forventet handling:**	**Faktisk handling:**
Krav FB1: indhente information omkring hvor mange PTT-enheder	Alle PTT-enhederne der tilhører den gruppe som operatøren varetager, kan ses i listen midt på skærmen. (Det er ikke muligt at se det eksakte antal)	Som forventet. `GODKENDT`
Krav FB2: indhente status omkring de enkelte PTT-enheder	Hvis en bestemt PTT-enhed vælges i listen skiftes til detalje-præsentationen og detaljer på den aktuelle PTT-enhed vises	Som forventet. `GODKENDT`
Krav FB3: foretage et opkald til en bestemt PPT-enhed	Grafikken skifter således at det ikke længere er muligt at trykke på andre knapper end ”End“-knappen. Der bliver oprettet et opkald til den aktuelle PTT-enhed	Som forventet. `GODKENDT`
Krav FB4: modtage indgående opkald fra de enkelte PTT-enheder		
Modtager et indkommende opkald i liste-præsentationen	Den aktuelle PTT-enhed i listen indikerer at der er et indkommende kald med en grøn markering. Det er ikke muligt at besvare opkaldet i denne præsentation	Som forventet. `GODKENDT`
Modtager et indkommende opkald i detalje-præsentationen. Med den aktuelle PTT-enhed valgt	Den grønne ”Call“-knap begynder at blinke. Det er muligt at besvare kaldet ved at trykke på knappen	Som forventet. `GODKENDT`
Modtager et indkommende opkald i detalje-præsentationen Med en anden PTT-enhed valgt end den som der ringer	Det er ikke muligt at besvate kalder da det ikke er den aktuelle PTT-enhed der ringer	Som forventet. `GODKENDT`
Konklusion: Der er i løbet at disse teste observeret to problemer: • En PTT-enhed kan periodisk skifte tilstand til `OFFLINE` • Applikationen kan gå ned pga. konflikter med lyd-driveren.		

Bilag D

Programkode

D.1 AGI

D.1.1 agiPTT.py

```
#! /usr/bin/python
'''
Created on Sep 28, 2011

@author: Casper Skipper Olsen
'''

from AsteriskLib import AGI
from AsteriskDB  import AstDB

agi = AGI()
astDB = AstDB('127.0.0.1', 'asterisk', 'astproj', 'pttproject')

callId = agi.env["agi_callerid"]
channel = agi.env["agi_channel"]

try:
    dispId = astDB.findDispatcherFromExt(callId)

    if not dispId == 0:
        dispId = str(dispId)
        agi.answer()
        agi.dial('sip/%s,20,r'%(dispId))
except:
    print "Error"
finally:
    astDB.closeConnection()
```

D.1.2 AsteriskDB.py

```
'''
Created on Oct 4, 2011

@author: Casper Skipper Olsen
'''

import MySQLdb as mdb
from types import NoneType

#SQL Prepared Statements
findDisp = \
'SELECT    * \
    FROM sip_data \
    WHERE name    = \
    (SELECT    sip_name \
    FROM sip_group \
    WHERE group_id = \
        (SELECT    group_id \
        FROM sip_data_group    \
        WHERE sip_name = %s))'

findUnits = \
'SELECT * \
FROM sip_data \
WHERE name IN \
    (SELECT sip_name \
    FROM sip_data_group \
    WHERE group_id = \
    (SELECT group_id \
    FROM sip_group \
    WHERE sip_name = %s))'

class AstDB:
    def __init__(self, server, user, password, database):
        self.conn = 0
        self.cursor = 0

        try:
            self.conn = mdb.connect(server, user, password, database);
            self.cursor = self.conn.cursor(mdb.cursors.DictCursor)
        except mdb.Error, e:
            print "Error %d: %s" %(e.args[0],e.args[1])

    def findDispatcherFromExt(self, unitExt):
  """findDispatcherFromExt(self, unitExt)
  Returnes the Dispatcher
  """
        try:
            self.cursor.execute(findDisp%(unitExt))

            dispExt = 0
            row = self.cursor.fetchone()

            if not type(row) == NoneType:
                dispExt = row["name"]
```

```
56
57              return dispExt
58
59          except mdb.Error, e:
60              print "Error %d: %s" %(e.args[0],e.args[1])
61
62      def findUnitsFromExt(self, dispExt):
63    """findUnitsFromExt(self, dispExt)
64    Returnes a list of PTT-units
65    """
66          try:
67              self.cursor.execute(findUnits%(dispExt))
68
69              unitsExt = []
70              rows = self.cursor.fetchall()
71
72              if not type(rows) == NoneType:
73                  if not len(rows) == 0:
74                      for row in rows:
75                          unitsExt += [row["name"]]
76
77              return unitsExt
78
79          except mdb.Error, e:
80              print "Error %d: %s" %(e.args[0],e.args[1])
81
82      def closeConnection(self):
83    """closeConnection(self)
84    Closes the database connection
85    """
86          if not self.cursor == 0:
87              self.cursor.close()
88          if not self.conn == 0:
89              self.conn.close()
```

D.1.3 AsteriskLib.py

```
1  '''
2  Created on Oct 4, 2011
3
4  @author: Casper Skipper Olsen
5  '''
6
7  import sys
8  import re
9  from types import ListType
10
11 re_code = re.compile(r'(^\d*)\s*(.*)')
12 re_kv = re.compile(r'(?P<key>\w+)=(?P<value>[^\s]+)\s*(?:\((?P<data>.*)\))*')
13
14 """ AGI Varibles
15 agi_request: /home/cao/workspace/FirstTest/src/test1.py
16 agi_channel: SIP/3000-0000000e
17 agi_language: en
18 agi_type: SIP
```

```
agi_uniqueid: 1317744405.15
agi_version: 1.8.7.0-1digium1~natty
agi_callerid: 12345
agi_calleridname: unknown
agi_callingpres: 0
agi_callingani2: 0
agi_callington: 0
agi_callingtns: 0
agi_dnid: unknown
agi_rdnis: unknown
agi_context: mytest
agi_extension: 400
agi_priority: 2
agi_enhanced: 0.0
agi_accountcode:
agi_threadid: 55815024
"""

class AGI:
    def __init__(self):
        sys.stderr.write('AGI Object Created')
        sys.stderr.flush()
        self.env = {}
        self._get_agi_env()

    def _get_agi_env(self):
        while 1:
            line = sys.stdin.readline().strip()

            if line == '':
                break
            key,data = line.split(':')

            if key[:4] <> 'agi_':
                continue

            key = key.strip()
            data = data.strip()

            if key <> '':
                self.env[key] = data

    def execute(self, command, *args):
    """execute(self, command, *args)
    Exectues a AGI function
    """
        try:
            self.send_command(command, *args)
            return self.get_result()
        except:
            raise

    def send_command(self, command, *args):
        command = command.strip()
        command = '%s %s' % (command, ' '.join(map(str,args)))
        command = command.strip()

```

```
76          if command[-1] != '\n':
77              command += '\n'
78          sys.stdout.write(command)
79          sys.stdout.flush()
80
81      def get_result(self, stdin=sys.stdin):
82          code = 0
83          result = {'result':('','')}
84          line = stdin.readline().strip()
85
86          m = re_code.search(line)
87          if m:
88              code, response = m.groups()
89              code = int(code)
90
91          if code == 200:
92              for key,value,data in re_kv.findall(response):
93                  result[key] = (value,data)
94
95                  if data == 'hangup':
96                      raise
97
98                  if key == 'result' and value == '-1':
99                      raise
100             return result
101         else:
102             raise
103
104     def _quote(self, string):
105         return ''.join(['"', str(string), '"'])
106
107     def _process_digit_list(self, digits):
108         if type(digits) == ListType:
109             digits = ''.join(map(str, digits))
110         return self._quote(digits)
111
112     def appexec(self, application, options=''):
113     """appexec(self, application, options='')
114     Executes a Dialplan function
115     """
116         res = self.execute('EXEC', application, self._quote(options))['result'
                ][0]
117         if res == '-2':
118             raise
119         return res
120
121     def answer(self):
122     """answer(self)
123     Answers e call
124     """
125         self.execute('ANSWER')['result'][0]
126
127     def hangup(self, channel=''):
128     """hangup(self, channel='')
129     Ends a call
130     """
131         self.execute('HANGUP', channel)
```

```
132
133     def dial(self, number):
134     """dial(self, number)
135     Dial e given number
136     """
137         self.appexec('dial', number)
138
139     def playtones(self, tone):
140     """playtones(self, tone)
141     Plays a given tone
142     """
143         self.appexec('playtones', tone)
144
145     def wait(self, msec):
146         self.appexec('wait', msec)
147
148     def say_number(self, number, escape_digits=""):
149         """say_number(self, number, escape_digits='')
150         Say a given number
151         """
152         number = self._process_digit_list(number)
153         escape_digits = self._process_digit_list(escape_digits)
154         res = self.execute('SAY NUMBER', number, escape_digits)['result'][0]
155         if res == '0':
156             return ''
157         else:
158             try:
159                 return chr(int(res))
160             except:
161                 raise
```

D.2 AMI Manager

D.2.1 ami_manager.h

```
1  #ifndef AMI_MANAGER_H
2  #define AMI_MANAGER_H
3
4  #include <QTcpSocket>
5  #include "Actions/actionbase.h"
6  #include "inputinterpreter.h"
7
8  class AMIManager : public QObject
9  {
10     Q_OBJECT
11
12 public:
13     AMIManager(QString domain, QObject *parent = 0);
14     ~AMIManager(void);
15
16     void login(const QString &username, const QString &secret);
17     void sendCommand(const QString &command);
18     void sendAction(const QString &action);
```

```
19      void sendAction(ActionBase &action);
20
21  private:
22      QTcpSocket          *tcpSocket;
23      InputInterpreter    *iipt;
24
25
26  private slots:
27      void newEvent(EventBase* ev);
28      void displayError(QAbstractSocket::SocketError socketError);
29
30
31  signals:
32      void result(const QString &result);
33      void peerStatusChanged(QString peerStatus, QString peer, QString address,
            QString cause);
34  };
35
36  #endif //AMI_MANAGER_H
```

D.2.2 ami_manager.cpp

```
1   #include <QtNetwork>
2
3   #include "ami_manager.h"
4   #include "Events.h"
5   //#include "peerentryevent.h"
6   //#include "peerstatusevent.h"
7   AMIManager::AMIManager(QString domain, QObject *parent)
8       :QObject(parent)
9   {
10      tcpSocket = new QTcpSocket(this);
11      connect(tcpSocket, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT
            (displayError(QAbstractSocket::SocketError)));
12
13      iipt = new InputInterpreter(tcpSocket);
14      connect(iipt, SIGNAL(result(const QString &)), this, SIGNAL(result(const
            QString &)), Qt::QueuedConnection);
15      connect(iipt, SIGNAL(newEvent(EventBase*)), this, SLOT(newEvent(EventBase
            *)), Qt::QueuedConnection);
16
17      tcpSocket->connectToHost(QHostAddress(domain),5038);
18  }
19
20
21  AMIManager::~AMIManager(void){}
22
23
24  /*
25  TCP Socket errors slot
26  */
27  void AMIManager::displayError(QAbstractSocket::SocketError socketError)
28  {
29       qDebug()<<__FUNCTION__;
30
```

```
31      switch (socketError) {
32      case QAbstractSocket::RemoteHostClosedError:
33          break;
34      case QAbstractSocket::HostNotFoundError:
35          qDebug()<<tr("The host was not found.");
36          break;
37      case QAbstractSocket::ConnectionRefusedError:
38          qDebug()<<tr("The connection was refused.");
39          break;
40      default:
41          qDebug()<<tr("The following error occurred: %1.")
42              .arg(tcpSocket->errorString());
43      }
44
45  }
46
47  /*
48  AMI Login method
49  */
50  void AMIManager::login(const QString &username, const QString &secret)
51  {
52      //qDebug()<<__FUNCTION__;
53
54      QByteArray *ba = new QByteArray();
55
56      ba->append("Action: Login").append("\n");
57      ba->append("Username: ").append(username).append("\n");
58      ba->append("Secret: ").append(secret).append("\n");
59      ba->append("\n");
60
61      tcpSocket->write(ba->data());
62      delete ba;
63      ba = 0;
64  }
65
66  /*
67  Send CLI Comand method
68  */
69  void AMIManager::sendCommand(const QString &command)
70  {
71      //qDebug()<<__FUNCTION__;
72
73      QByteArray *ba = new QByteArray();
74
75      ba->append("Action: Command").append("\n");
76      ba->append("Command: ").append(command).append("\n");
77      ba->append("\n");
78
79      tcpSocket->write(ba->data());
80      delete ba;
81      ba = 0;
82  }
83
84  /*
85  Send AMI Action
86  */
87  void AMIManager::sendAction(const QString &action)
```

```
88 {
89     //qDebug()<<__FUNCTION__;
90 
91     QByteArray *ba = new QByteArray();
92 
93     ba->append("Action: ").append(action).append("\n");
94     ba->append("\n");
95 
96     tcpSocket->write(ba->data());
97     delete ba;
98     ba = 0;
99 }
100 
101 /*
102 Send AMI Action
103 */
104 void AMIManager::sendAction(ActionBase &action)
105 {
106     //qDebug()<<__FUNCTION__;
107 
108     QByteArray *ba = new QByteArray();
109 
110     ba->append(action.getActionCommand());
111 
112     tcpSocket->write(ba->data());
113     delete ba;
114     ba = 0;
115 }
116 
117 /*
118 AMI Event interperter slot
119 */
120 void AMIManager::newEvent(EventBase* ev)
121 {
122     //qDebug()<<__FUNCTION__<< ev->getEventType();
123 
124     QString type = ev->getEventType();
125 
126     if(type.compare("peerstatus") == 0)
127     {
128         PeerStatusEvent* pev = qobject_cast<PeerStatusEvent*>(ev);
129         emit peerStatusChanged(pev->getPeerStatus(), pev->getPeer(), pev->
                getAddress(), pev->getCause());
130 
131         delete pev;
132         pev = 0;
133     }
134     else if(type.compare("peerentry") == 0)
135     {
136         PeerEntryEvent* pev = qobject_cast<PeerEntryEvent*>(ev);
137         qDebug()<< pev->getIPaddress();
138 
139         delete pev;
140         pev = 0;
141     }
142     else
143     {
```

```
144         qDebug()<< "Unknown Event";
145     }
146 }
```

D.2.3 inputinterpreter.h

```
1  #ifndef INPUTINTERPRETER_H
2  #define INPUTINTERPRETER_H
3
4  #include <QtCore>
5  #include <QObject>
6  #include <QTcpSocket>
7  #include <QMap>
8  #include "Events/eventbase.h"
9
10 class InputInterpreter : public QObject
11 {
12     Q_OBJECT
13
14 public:
15     InputInterpreter(QTcpSocket *socket);
16     ~InputInterpreter();
17
18 signals:
19     void result(const QString &result);
20     void newEvent(EventBase* ev);
21
22 private slots:
23     void rawSocketIndput();
24
25 private:
26     QThread      *m_myThread;
27     QTcpSocket   *tcpSocket;
28     QString      *resultText;
29
30     QString                  line;
31     QStringList              commandResult;
32     QMap<QString, QString>   buffer;
33     bool isCommand;
34 };
35
36 #endif // INPUTINTERPRETER_H
```

D.2.4 inputinterpreter.cpp

```
1  #include "inputinterpreter.h"
2  #include "Events/eventbuilder.h"
3
4  InputInterpreter::InputInterpreter(QTcpSocket *socket)
5      : QObject()
6  {
7      this->tcpSocket = socket;
```

```
8
9       resultText = new QString("");
10      line = QString("");
11      commandResult.clear();
12      buffer.clear();
13      isCommand = false;
14
15      connect(tcpSocket, SIGNAL(readyRead()),
16          this, SLOT(rawSocketIndput()));
17
18      m_myThread = new QThread();
19      m_myThread->start();
20      this->moveToThread(m_myThread);
21
22  }
23
24  InputInterpreter::~InputInterpreter(){}
25
26  /*
27  AMI indput Interpreter slot
28  */
29  void InputInterpreter::rawSocketIndput()
30  {
31      //qDebug()<<__FUNCTION__;
32
33      while(tcpSocket->bytesAvailable() != 0)
34      {
35          //resultText->append(tcpSocket->readLine(256));
36          line = tcpSocket->readLine(256);
37          //qDebug()<<line;
38          emit result(line);
39
40          if(isCommand)
41          {
42              //qDebug()<<"isCommand";
43
44              if((line.startsWith("--END COMMAND--")) || (line.startsWith(" --
                    END COMMAND--")))
45              {
46                  foreach(QString li,commandResult)
47                  {
48                      qDebug()<<li;
49                  }
50                  qDebug()<<"--END COMMAND--";
51                  isCommand = false;
52              }
53              else
54              {
55                  commandResult.append(line);
56              }
57              continue;
58          }
59
60          if (line.startsWith("Response: Follows"))
61          {
62              qDebug()<<"Response: Follows";
63              isCommand = true;
```

```
64              commandResult.clear();
65              continue;
66          }
67
68          if(line.startsWith("Asterisk Call Manager/"))
69          {
70              qDebug()<<"Astrisk Call Manager";
71              continue;
72          }
73
74          //qDebug()<<"Line Size: "<<line.size()<<(line.compare("\r\n")==0);
75          if(line.compare("\r\n")==0)
76          {
77              //qDebug()<<"at the end";
78
79              if(buffer.contains("event"))
80              {
81                  //qDebug()<<QString("event:%1").arg(buffer.value("event"));
82
83                  EventBase* eb = EventBuilder::buildEvent(buffer);
84                  if(eb)
85                  {
86                      emit newEvent(eb);
87                  }
88              }
89              else if (buffer.contains("response"))
90              {
91                  //qDebug()<<QString("response:%1").arg(buffer.value("response
                        "));
92
93                  //QStringList keys = buffer.keys();
94                  //for(int i=0;i<keys.size();i++)
95                  //{
96                  //    qDebug()<<keys[i]+":"+buffer.value(keys[i]);
97                  //}
98              }
99              else
100             {
101                 //qDebug()<<"neither event nor response";
102             }
103             buffer.clear();
104         }
105         else
106         {
107             //qDebug()<<"LINE: "+line;
108             QStringList pair = line.split(": ");
109             if(pair.size() > 1)
110             {
111                 pair[1].replace("\r\n","");
112                 buffer.insert(pair[0].toLower(),QString(pair[1].toLower()));
113             }
114         }
115     }
116 }
```

D.2.5 actionbase.h

```
#ifndef ACTIONBASE_H
#define ACTIONBASE_H

#include <QObject>

class ActionBase : public QObject
{
    Q_OBJECT
    //Q_PROPERTY(QString actionCommand   READ getActionCommand);
private:
    static int count;

protected:
    ActionBase(QObject *parent = 0);
    //virtual ~ActionBase();
    QString actionID;

public:
    virtual QString getActionCommand() = 0;
    virtual QString getAction() = 0;
    QString getActionID(){return actionID;}
};

#endif // ACTIONBASE_H
```

D.2.6 actionbase.cpp

```
#include <QtCore>
#include "actionbase.h"

int ActionBase::count = 0;

ActionBase::ActionBase(QObject *parent)
    : QObject(parent)
{
    int temp = qHash(this) ^ ++count;
    actionID = QString::number(temp);

}
```

D.2.7 sipshowpeerAction.h

```
#ifndef SIPSHOWPEERACTION_H
#define SIPSHOWPEERACTION_H

#include "actionbase.h"

class SIPShowPeerAction : public ActionBase
{
```

```
8       Q_OBJECT
9
10  public:
11      SIPShowPeerAction(QObject *parent = 0);
12      SIPShowPeerAction(QString peer, QObject *parent = 0);
13      ~SIPShowPeerAction();
14
15      virtual QString getActionCommand();
16      virtual QString getAction(){return "SIPShowPeer";}
17
18      void setPeer(const QString &peer);
19
20      QString getPeer(){return peer;}
21
22  private:
23      QString peer;
24
25  };
26
27  #endif // SIPSHOWPEERACTION_H
```

D.2.8 sipshowpeerAction.cpp

```
1   #include "sipshowpeerAction.h"
2
3   SIPShowPeerAction::SIPShowPeerAction(QObject *parent): ActionBase(parent){}
4
5   SIPShowPeerAction::SIPShowPeerAction(QString peer, QObject *parent)
6       : ActionBase(parent), peer(peer)
7   {}
8
9   SIPShowPeerAction::~SIPShowPeerAction(){}
10
11  QString SIPShowPeerAction::getActionCommand()
12  {
13      QString act;
14      act.append("Action: ").append(this->getAction()).append("\n");
15      act.append("ActionID: ").append(this->getActionID()).append("\n");
16      act.append("Peer: ").append(this->peer).append("\n");
17      act.append("\n");
18      return act;
19  }
20
21
22  void SIPShowPeerAction::setPeer(const QString &peer)
23  {
24      this->peer = peer;
25  }
```

D.2.9 originateAction.h

```
#ifndef ORIGINATEACTION_H
#define ORIGINATEACTION_H

#include "actionbase.h"

class OriginateAction : public ActionBase
{
    Q_OBJECT

public:
    OriginateAction(QObject *parent = 0);
    ~OriginateAction();

    virtual QString getActionCommand();
    virtual QString getAction(){return "Originate";}

private:
    QString channel;
    QString exxtern;
    QString context;
    int     priority;
    long    timeout;
    QString callerId;

    QString Account;
    QString Application;
    QString Data;
    bool  Async;

};

#endif // ORIGINATEACTION_H
```

D.2.10 originateAction.cpp

```
#include "originateAction.h"

OriginateAction::OriginateAction(QObject *parent)
    : ActionBase(parent)
{

}

OriginateAction::~OriginateAction()
{

}



QString OriginateAction::getActionCommand()
{
    QString act;
    act.append("Action: ").append(this->getAction()).append("\n");
```

```
20      act.append("ActionID: ").append(this->getActionID()).append("\n");
21      act.append("Channel: ").append("SIP/3000").append("\n");
22      act.append("Context: ").append("mytest").append("\n");
23      act.append("Exten: ").append("400").append("\n");
24      act.append("Priority: ").append("1").append("\n");
25      act.append("Callerid: ").append("12345").append("\n");
26      act.append("Timeout: ").append("30000").append("\n");
27      act.append("\n");
28
29      return act;
30  }
```

D.2.11 eventbase.h

```
1   #ifndef EVENTBASE_H
2   #define EVENTBASE_H
3
4   #include <QObject>
5   #include <QMap>
6
7   class EventBase : public QObject
8   {
9       Q_OBJECT
10
11  public:
12      EventBase(QObject *parent = 0);
13      ~EventBase();
14
15      virtual EventBase* clone(const QMap<QString, QString>  &attributes) = 0;
16      virtual void setAttributes(const QMap<QString, QString>  &attributes) = 0;
17
18      //QMap<QString, QString> getAttributes(){return *attributes;}
19      QString getEventType(){return eventType;}
20
21  protected:
22      QMap<QString, QString>  *attributes;
23      QString                 eventType;
24  };
25
26  #endif // EVENTBASE_H
```

D.2.12 eventbase.cpp

```
1   #include <QMap>
2   #include "eventbase.h"
3
4
5   EventBase::EventBase(QObject *parent)
6       : QObject(parent)
7   {
8       //this->attributes = attributes;
9       //eventType = this->attributes.value("event");
```

```
10 }
11
12 EventBase::~EventBase(){}
```

D.2.13 peerentryevent.h

```
1  #ifndef PEERENTRYEVENT_H
2  #define PEERENTRYEVENT_H
3
4  #include "eventbase.h"
5
6  class PeerEntryEvent : public EventBase
7  {
8      Q_OBJECT
9
10 public:
11     PeerEntryEvent(QObject *parent = 0);
12     PeerEntryEvent(const QMap<QString, QString>  &attributes, QObject *parent
           = 0);
13     ~PeerEntryEvent();
14
15     virtual EventBase* clone(const QMap<QString, QString>  &attributes){return
            new PeerEntryEvent(attributes);}
16     virtual void setAttributes(const QMap<QString, QString>  &attributes);
17
18     QString getActionID(){return attributes->value("actionid");}
19     QString getChanneltype(){return attributes->value("channeltype");}
20     QString getObjectName(){return attributes->value("objectname");}
21     QString getChanObjectType(){return attributes->value("chanObjecttype");}
22     QString getIPaddress(){return attributes->value("ipaddress");}
23     QString getIPport(){return attributes->value("ipport");}
24     QString getDynamic(){return attributes->value("dynamic");}
25     QString getForcerport(){return attributes->value("forcerport");}
26     QString getVideoSupport(){return attributes->value("videosupport");}
27     QString getTextSupport(){return attributes->value("textsupport");}
28     QString getACL(){return attributes->value("acl");}
29     QString getStatus(){return attributes->value("status");}
30     QString getRealtimeDevice(){return attributes->value("realtimedevice");}
31 };
32
33 #endif // PEERENTRYEVENT_H
```

D.2.14 peerentryevent.cpp

```
1  #include "peerentryevent.h"
2
3  PeerEntryEvent::PeerEntryEvent(QObject *parent)
4      : EventBase(parent)
5  {
6
7  }
8
```

```
9  PeerEntryEvent::PeerEntryEvent(const QMap<QString, QString>  &attri, QObject *
       parent)
10     : EventBase(parent)
11 {
12     attributes = new QMap<QString, QString>(attri);
13     eventType = attributes->value("event");
14 }
15
16 PeerEntryEvent::~PeerEntryEvent()
17 {
18
19 }
20
21
22 void PeerEntryEvent::setAttributes(const QMap<QString, QString>  &attri)
23 {
24     attributes = new QMap<QString, QString>(attri);
25     eventType = attributes->value("event");
26 }
```

D.2.15 peerstatusevent.h

```
1  #ifndef PEERSTATUSEVENT_H
2  #define PEERSTATUSEVENT_H
3
4  #include "eventbase.h"
5
6  class PeerStatusEvent : public EventBase
7  {
8      Q_OBJECT
9
10 public:
11     PeerStatusEvent(QObject *parent = 0);
12     PeerStatusEvent(const QMap<QString, QString>  &attributes, QObject *parent
           = 0);
13     ~PeerStatusEvent();
14
15     virtual EventBase* clone(const QMap<QString, QString>  &attributes){return
           new PeerStatusEvent(attributes);}
16     virtual void setAttributes(const QMap<QString, QString>  &attributes);
17
18     QString getPrivilege(){return this->attributes->value("privilege");}
19     QString getChannelType(){return this->attributes->value("channeltype");}
20     QString getPeer(){return this->attributes->value("peer");}
21     QString getPeerStatus(){return this->attributes->value("peerstatus");}
22     QString getAddress(){return this->attributes->value("address");}
23     QString getCause(){return this->attributes->value("cause");}
24 };
25
26 #endif // PEERSTATUSEVENT_H
```

D.2.16 peerstatusevent.cpp

```
#include <QMap>
#include "peerstatusevent.h"

PeerStatusEvent::PeerStatusEvent(QObject *parent)
    : EventBase(parent)
{
}

PeerStatusEvent::PeerStatusEvent(const QMap<QString, QString>  &attri, QObject
    *parent)
    : EventBase(parent)
{
    attributes = new QMap<QString, QString>(attri);
    eventType = attributes->value("event");
}

PeerStatusEvent::~PeerStatusEvent(){}


void PeerStatusEvent::setAttributes(const QMap<QString, QString>  &attri)
{
    attributes = new QMap<QString, QString>(attri);
    eventType = attributes->value("event");
}
```

D.2.17 eventbuilder.h

```
#ifndef EVENTBUILDER_H
#define EVENTBUILDER_H

#include <QtCore>
#include <QMap>
#include "eventbase.h"

template <class Key, class T>
class InitializableQMap : public QMap<Key,T>
{
public:
    inline InitializableQMap<Key,T> &operator<< (const QPair<Key,T> &t)
    {
        insert(t.first,t.second);
        return *this;
    }
};

class EventBuilder
{

public:
    static EventBase* buildEvent(const QMap<QString, QString>  &attributes);

private:
```

```
26      static InitializableQMap<QString, EventBase*> event_prototypes;
27  };
28
29  #endif // EVENTBUILDER_H
```

D.2.18 eventbuilder.cpp

```
1   #include <QMap>
2   #include <QPair>
3   #include "eventbuilder.h"
4   #include "eventbase.h"
5   #include "Events.h"
6
7   EventBase* EventBuilder::buildEvent(const QMap<QString, QString>  &attributes)
8   {
9       QString eventType = attributes.value("event");
10      EventBase* eb = event_prototypes.value(eventType);
11
12      if(eb){
13          //qDebug()<<"ok";
14          eb = eb->clone(attributes);
15          //eb->setAttributes(attributes);
16      }else{
17          //qDebug()<<"null";
18      }
19
20      return eb;
21  }
22
23  /*
24    Event Prototuypes
25  */
26  InitializableQMap<QString, EventBase*> EventBuilder::event_prototypes =
        InitializableQMap<QString, EventBase*>()
27      << QPair<QString, EventBase*>("peerstatus", new PeerStatusEvent())
28      << QPair<QString, EventBase*>("peerentry",  new PeerEntryEvent());
```

D.3 SIP Manager

D.3.1 sip_manager.h

```
#ifndef SIP_MANAGER_H
#define SIP_MANAGER_H

#include <QtCore>
extern "C" {
    #include <pjsua-lib/pjsua.h>
}

class SIPManager : public QObject
{
    Q_OBJECT


public:
    SIPManager(QString user, QString secret, QString domain, QObject *parent =
        0);

public slots:
    bool shutdownPjSIP();
    void call_hangup(int call_id, uint code);
    void call_answer(int call_id, uint code);
    void call_make(QString pjurl, int *call_id);

signals:
    void new_log_message(QString text);
    void pager(QString from, QString text);
    void call_state(int call_id,QString remote_info, QString state_text, int
       state_id);
    void incoming_call(int call_id, QString remote_info);
    void nat_detect(QString text, QString description);
    void call_media_state(int call_id);
    void buddy_state(int buddy_id);
    void reg_state(int acc_id);


private:
    pjsua_config cfg;
    pjsua_logging_config log_cfg;
    pjsua_media_config media_cfg;
    pjsua_transport_config transport_cfg;
    pjsua_transport_config rtp_cfg;
    pjsua_buddy_config buddy_cfg;
    pjsua_acc_config acc_cfg;
    pjsua_acc_id acc_id;

    QThread *m_myThread;

    QString m_user;
    QString m_secret;
    QString m_domain;

    int initPjSIP();
```

```
51      char *caor, *creguri, *cdomain, *cuser, *csecret, *cstun;
52  };
53
54  #endif // SIP_MANAGER_H
```

D.3.2 sip_manager.cpp

```
1   #include "sip_manager.h"
2   #include "PjCallback.h"
3
4   extern "C" {
5       #include <pjsua-lib/pjsua.h>
6   }
7
8
9   SIPManager::SIPManager(QString user, QString secret, QString domain, QObject *
        parent) :
10      QObject(parent),
11      m_user(user),
12      m_secret(secret),
13      m_domain(domain)
14  {
15      PjCallback::instance();
16      if(!initPjSIP())
17          qDebug() << "Pjsua Running";
18
19      connect(PjCallback::instance(), SIGNAL(incoming_call(int, QString)), this,
            SIGNAL(incoming_call(int, QString)));
20      connect(PjCallback::instance(), SIGNAL(call_state(int,QString,QString,int)
            ), this, SIGNAL(call_state(int,QString,QString, int)));
21      connect(PjCallback::instance(), SIGNAL(new_log_message(QString)), this,
            SIGNAL(new_log_message(QString)));
22
23      //m_myThread = new QThread();
24      //m_myThread->start();
25      //this->moveToThread(m_myThread);
26  }
27
28  /*
29  PJSIP initilize method
30  */
31  int SIPManager::initPjSIP()
32  {
33      qDebug() << __FUNCTION__ << QThread::currentThreadId();
34
35      pj_status_t status;
36
37      /* Create pjsua first! */
38      status = pjsua_create();
39      if (status != PJ_SUCCESS) {
40          qDebug() << "Error in pjsua_create()" << status;
41          pjsua_destroy();
42          return -1;
43      }
44
```

```
45      pjsua_config_default(&cfg);
46      /* initialize pjsua callbacks */
47      cfg.cb.on_incoming_call = PjCallback::on_incoming_call_wrapper;
48      cfg.cb.on_call_media_state = PjCallback::on_call_media_state_wrapper;
49      cfg.cb.on_call_state = PjCallback::on_call_state_wrapper;
50      cfg.cb.on_pager = PjCallback::on_pager_wrapper;
51      cfg.cb.on_reg_state = PjCallback::on_reg_state_wrapper;
52      cfg.cb.on_buddy_state = PjCallback::on_buddy_state_wrapper;
53      cfg.cb.on_nat_detect = PjCallback::on_nat_detect_wrapper;
54      cfg.user_agent = pj_str("Thrane PTT Project");
55
56      pjsua_logging_config_default(&log_cfg);
57      log_cfg.msg_logging = true;
58      log_cfg.console_level = 3; // Todo
59      log_cfg.cb = PjCallback::logger_cb_wrapper;
60      log_cfg.decor = log_cfg.decor & ~PJ_LOG_HAS_NEWLINE;
61      pjsua_media_config_default(&media_cfg);
62      media_cfg.no_vad = true;
63      status = pjsua_init(&cfg, &log_cfg, &media_cfg);
64      if (status != PJ_SUCCESS) {
65          qDebug() << "Error in pjsua_init()" << status;
66          pjsua_destroy();
67          return -1;
68      }
69
70      /* Add UDP transport. */
71      pjsua_transport_config_default(&transport_cfg);
72      transport_cfg.port = 5060; // Todo
73      status = pjsua_transport_create(PJSIP_TRANSPORT_UDP, &transport_cfg, NULL)
            ;
74      if (status != PJ_SUCCESS) {
75          qDebug() << "Error creating transport" << status;
76          pjsua_destroy();
77          return -1;
78      }
79
80      pjsua_acc_config_default(&acc_cfg);
81      QString sipAoR;
82      QString regUri;
83      sipAoR = QString("sip:") + m_user + QString("@") + m_domain;
84      regUri = QString("sip:") + m_domain;
85
86      QByteArray temp;
87      temp = sipAoR.toLatin1();     caor = strdup(temp.data());
88      temp = regUri.toLatin1();     creguri = strdup(temp.data());
89      temp = m_domain.toLatin1();   cdomain = strdup(temp.data());
90      temp = m_user.toLatin1();     cuser = strdup(temp.data());
91      temp = m_secret.toLatin1();   csecret = strdup(temp.data());
92
93      acc_cfg.id = pj_str(caor);
94      acc_cfg.reg_uri = pj_str(creguri);
95      acc_cfg.cred_count = 1;
96      acc_cfg.cred_info[0].realm = pj_str("*");
97      acc_cfg.cred_info[0].scheme = pj_str("digest");
98      acc_cfg.cred_info[0].username = pj_str(cuser);
99      acc_cfg.cred_info[0].data_type = PJSIP_CRED_DATA_PLAIN_PASSWD;
100     acc_cfg.cred_info[0].data = pj_str(csecret);
```

```
101
102     status = pjsua_acc_add(&acc_cfg, PJ_TRUE, &acc_id);
103     if (status != PJ_SUCCESS) {
104         qDebug() << "Failure adding the account! Inspect the debug window!" <<
                status;
105         pjsua_destroy();
106         return -1;
107     }
108
109     /* start pjsua */
110     status = pjsua_start();
111     if (status != PJ_SUCCESS) {
112         qDebug() << "Error starting pjsua" << status;
113         pjsua_destroy();
114         return -1;
115     }
116
117     return 0;
118 }
119
120 /*
121 PJSIP Shutdown
122 */
123 bool SIPManager::shutdownPjSIP() {
124     /* shut down */
125     pj_status_t status;
126     bool ret = true;
127     status = pjsua_destroy();
128     if (status != PJ_SUCCESS)
129     {
130         qDebug() << "Error destroying pjsua" << status;
131         ret = false;
132     }
133     return ret;
134 }
135
136 /*
137 Call Answer
138 */
139 void SIPManager::call_answer(int call_id, uint code)
140 {
141     qDebug() << __FUNCTION__;
142     pjsua_call_answer(call_id, code, NULL, NULL);
143 }
144
145 /*
146 Call Hangup
147 */
148 void SIPManager::call_hangup(int call_id, uint code)
149 {
150     qDebug() << __FUNCTION__;
151      pjsua_call_hangup(call_id, code, NULL, NULL);
152 }
153
154 /*
155 Call Create
156 */
```

```
157 void SIPManager::call_make(QString pjurl, int *call_id)
158 {
159     qDebug() << __FUNCTION__<<"PJURJ: "<<pjurl<<"call_id: "<<call_id;
160
161     pjurl.append(QString("@%1").arg(m_domain));
162
163     QByteArray temp;
164     temp = pjurl.toLatin1();
165     char *url;
166     url = strdup(temp.data());
167
168     pj_status_t status;
169     status = pjsua_verify_url(url);
170     if (status != PJ_SUCCESS)
171     {
172         qDebug() << "Invalid URL" << status;
173         return;
174     }
175
176     pj_str_t uri = pj_str(url);
177     status = pjsua_call_make_call(acc_id, &uri, 0, NULL, NULL, call_id);
178     if (status != PJ_SUCCESS){
179         qDebug() << "Error making call" << status;
180         return;
181     }
182 }
```

D.3.3 PjCallback.h

```
1  #ifndef PJCALLBACK_H_
2  #define PJCALLBACK_H_
3
4  #include <QObject>
5
6  extern "C" {
7  #include <pjsua-lib/pjsua.h>
8  }
9
10
11 class PjCallback : public QObject {
12     Q_OBJECT
13
14 public:
15     virtual ~PjCallback();
16     static PjCallback* instance();
17
18     void logger_cb(int level, const char *data, int len);
19     void on_pager(pjsua_call_id call_id, const pj_str_t *from, const pj_str_t
           *to, const pj_str_t *contact, const pj_str_t *mime_type, const
           pj_str_t *text);
20     void on_call_state(pjsua_call_id call_id, pjsip_event *e);
21     void on_incoming_call(pjsua_acc_id acc_id, pjsua_call_id call_id,
           pjsip_rx_data *rdata);
22     void on_nat_detect(const pj_stun_nat_detect_result *res);
23     void on_call_media_state(pjsua_call_id call_id);
```

```
24      void on_buddy_state(pjsua_buddy_id buddy_id);
25      void on_reg_state(pjsua_acc_id acc_id);
26
27      /* callback functions */
28      static void logger_cb_wrapper(int level, const char *data, int len);
29      static void on_pager_wrapper(pjsua_call_id call_id, const pj_str_t *from,
            const pj_str_t *to, const pj_str_t *contact, const pj_str_t *mime_type
            , const pj_str_t *text);
30      static void on_call_state_wrapper(pjsua_call_id call_id, pjsip_event *e);
31      static void on_incoming_call_wrapper(pjsua_acc_id acc_id, pjsua_call_id
            call_id, pjsip_rx_data *rdata);
32      static void on_nat_detect_wrapper(const pj_stun_nat_detect_result *res);
33      static void on_call_media_state_wrapper(pjsua_call_id call_id);
34      static void on_buddy_state_wrapper(pjsua_buddy_id buddy_id);
35      static void on_reg_state_wrapper(pjsua_acc_id acc_id);
36
37  signals:
38      void new_log_message(QString text);
39      void pager(QString from, QString text);
40      void call_state(int call_id,QString remote_info, QString state_text, int
            state_id);
41      void incoming_call(int call_id,QString remote_info);
42      void nat_detect(QString text, QString description);
43      void call_media_state(int call_id);
44      void buddy_state(int buddy_id);
45      void reg_state(int acc_id);
46
47
48  private:
49      PjCallback();
50      static PjCallback *s_instance;
51
52  };
53
54  #endif /*PJCALLBACK_H_*/
```

D.3.4 PjCallback.cpp

```
1   #include <QtCore>
2   #include <QList>
3   #include <QMutex>
4
5   #include "PjCallback.h"
6
7   #define THIS_FILE "PjCallback"
8
9   QList<int> activeCalls;
10
11  PjCallback *PjCallback::s_instance = NULL;
12
13  PjCallback::PjCallback(){}
14  PjCallback::~PjCallback(){}
15
16
17  PjCallback *PjCallback::instance()
```

```
18 {
19     if (!s_instance)
20     {
21         s_instance = new PjCallback();
22     }
23     return s_instance;
24 }
25
26 void PjCallback::logger_cb(int level, const char *data, int len)
27 {
28     //qDebug() << __FUNCTION__;
29
30     PJ_UNUSED_ARG(level);
31     PJ_UNUSED_ARG(len);
32     emit new_log_message(data);
33     //qDebug() << data;
34 }
35
36 void PjCallback::logger_cb_wrapper(int level, const char *data, int len)
37 {
38     /* call the non-static member */
39     PjCallback::instance()->logger_cb(level, data, len);
40 }
41 //------------------------------------------------------------------
42 void PjCallback::on_pager(pjsua_call_id call_id, const pj_str_t *from, const
      pj_str_t *to, const pj_str_t *contact, const pj_str_t *mime_type, const
      pj_str_t *text)
43 {
44     qDebug() << __FUNCTION__;
45 }
46
47 void PjCallback::on_pager_wrapper(pjsua_call_id call_id, const pj_str_t *from,
       const pj_str_t *to, const pj_str_t *contact, const pj_str_t *mime_type,
      const pj_str_t *text)
48 {
49     /* call the non-static member */
50     PjCallback::instance()->on_pager(call_id, from, to, contact, mime_type,
         text);
51 }
52 //------------------------------------------------------------------
53 void PjCallback::on_nat_detect(const pj_stun_nat_detect_result *res)
54 {
55     qDebug() << __FUNCTION__;
56 }
57
58 void PjCallback::on_nat_detect_wrapper(const pj_stun_nat_detect_result *res)
59 {
60     /* call the non-static member */
61     PjCallback::instance()->on_nat_detect(res);
62 }
63 //------------------------------------------------------------------
64 void PjCallback::on_call_state(pjsua_call_id call_id, pjsip_event *e)
65 {
66     qDebug() << __FUNCTION__;
67
68     PJ_UNUSED_ARG(e);
69
```

```
70      pj_status_t status;
71      pjsua_call_info ci;
72      status = pjsua_call_get_info(call_id, &ci);
73      if (status != PJ_SUCCESS) {
74          PJ_LOG(3,(THIS_FILE, "ERROR retrieveing info for Call %d ... ignoring"
                , call_id));
75          return;
76      }
77      PJ_LOG(3,(THIS_FILE, "Call %d state=%.*s", call_id,
78                (int)ci.state_text.slen, ci.state_text.ptr));
79
80      QString state_text = QString::fromAscii(ci.state_text.ptr,(int)ci.
            state_text.slen);
81      QString remote_info = QString::fromAscii(ci.remote_info.ptr,(int)ci.
            remote_info.slen);
82      qDebug() << "----------------------STATE text:"<<state_text;
83
84      emit call_state(call_id,remote_info, state_text, ci.state);
85
86      switch(ci.state) {
87      case PJSIP_INV_STATE_DISCONNECTED:
88
89          break;
90      default:
91          ;
92      }
93  }
94
95  void PjCallback::on_call_state_wrapper(pjsua_call_id call_id, pjsip_event *e)
96  {
97      /* call the non-static member */
98      PjCallback::instance()->on_call_state(call_id, e);
99  }
100 //------------------------------------------------------------------
101 void PjCallback::on_incoming_call(pjsua_acc_id acc_id, pjsua_call_id call_id,
        pjsip_rx_data *rdata)
102 {
103     qDebug() << __FUNCTION__;
104
105     PJ_UNUSED_ARG(acc_id);
106     PJ_UNUSED_ARG(rdata);
107
108     //pjsua_call_hangup(call_id, 486, NULL, NULL);
109     //pjsua_call_answer(call_id, 200, NULL, NULL);
110
111     pj_status_t status;
112     pjsua_call_info ci;
113     status = pjsua_call_get_info(call_id, &ci);
114     if (status != PJ_SUCCESS)
115     {
116         qDebug() << "ERROR retrieveing info for Call %d ... ignoring" <<
                call_id;
117         PJ_LOG(3,(THIS_FILE, "ERROR retrieveing info for Call %d ... ignoring"
                , call_id));
118         return;
119     }
120     PJ_LOG(3,(THIS_FILE, "Call %d state=%.*s", call_id,
```

```
121                 (int)ci.state_text.slen, ci.state_text.ptr));
122     PJ_LOG(3,(THIS_FILE, "----------------------------Incoming call from %.*s
            !!",
123                 (int)ci.remote_info.slen,
124                 ci.remote_info.ptr));
125
126     QString state_text = QString::fromAscii(ci.state_text.ptr,(int)ci.
            state_text.slen);
127     QString remote_info = QString::fromAscii(ci.remote_info.ptr,(int)ci.
            remote_info.slen);
128
129
130     //activeCalls << call_id;
131     emit incoming_call(call_id,remote_info);
132 }
133
134 void PjCallback::on_incoming_call_wrapper(pjsua_acc_id acc_id, pjsua_call_id
        call_id, pjsip_rx_data *rdata)
135 {
136     /* call the non-static member */
137     PjCallback::instance()->on_incoming_call(acc_id, call_id, rdata);
138 }
139 //----------------------------------------------------------------
140 void PjCallback::on_call_media_state(pjsua_call_id call_id)
141 {
142     qDebug() << __FUNCTION__;
143 }
144
145 void PjCallback::on_call_media_state_wrapper(pjsua_call_id call_id)
146 {
147     /* call the non-static member */
148     PjCallback::instance()->on_call_media_state(call_id);
149 }
150 //----------------------------------------------------------------
151 void PjCallback::on_buddy_state(pjsua_buddy_id buddy_id)
152 {
153     qDebug() << __FUNCTION__;
154     //emit buddy_state(buddy_id);
155 }
156
157 void PjCallback::on_buddy_state_wrapper(pjsua_buddy_id buddy_id)
158 {
159     /* call the non-static member */
160     PjCallback::instance()->on_buddy_state(buddy_id);
161 }
162 //----------------------------------------------------------------
163 void PjCallback::on_reg_state(pjsua_acc_id acc_id)
164 {
165     qDebug() << __FUNCTION__;
166     //emit reg_state(acc_id);
167 }
168
169 void PjCallback::on_reg_state_wrapper(pjsua_acc_id acc_id)
170 {
171     /* call the non-static member */
172     PjCallback::instance()->on_reg_state(acc_id);
173 }
```

```
174 //------------------------------------------------------------
```

D.4 PTT Data

D.4.1 ptt_data.h

```
#ifndef PTT_DATA_H
#define PTT_DATA_H

#include <QObject>
#include "dto/sip_data.h"

class PTTData : public QObject
{
    Q_OBJECT
public:
    PTTData(QObject *parent = 0);
    PTTData(const QString &username, const QString &password, const QString &
        domain, const QString &database, QObject *parent = 0);

    void setCredentials(const QString &username, const QString &password,
        const QString &domain, const QString &database);

    SipData*    getSipData(int name);
    bool        createSipData(SipData* sipData);
    bool        delSipData(SipData* sipData);
    bool        updateSipData(SipData* sipData);

    QList<SipData*>* getGroupList(int dispName);
    SipData*         getDispatcher(int name);

};

#endif // PTT_DATA_H
```

D.4.2 ptt_data.cpp

```

#include "ptt_data.h"
#include "db/DBConnection.h"
#include "dao/sip_dao.h"


PTTData::PTTData(QObject *parent) :
    QObject(parent)
{

}

PTTData::PTTData(const QString &username, const QString &password, const
    QString &domain, const QString &database, QObject *parent) :
    QObject(parent)
{
    DBConnection::instance()->setCredentials(username, password, domain,
        database);
```

```
17 }
18
19 void PTTData::setCredentials(const QString &username, const QString &password,
       const QString &domain, const QString &database)
20 {
21     DBConnection::instance()->setCredentials(username, password, domain,
           database);
22 }
23
24 SipData* PTTData::getSipData(int name)
25 {
26     DBConnection::instance()->open();
27     SipDAO sipDAO;
28     SipData* sipData = sipDAO.read(name);
29     DBConnection::instance()->close();
30     return sipData;
31 }
32
33 bool PTTData::createSipData(SipData *sipData)
34 {
35     DBConnection::instance()->open();
36     SipDAO sipDAO;
37     bool res = sipDAO.create(sipData);
38     DBConnection::instance()->close();
39     return res;
40 }
41
42 bool PTTData::delSipData(SipData *sipData)
43 {
44     DBConnection::instance()->open();
45     SipDAO sipDAO;
46     bool res = sipDAO.remove(sipData);
47     DBConnection::instance()->close();
48     return res;
49 }
50
51 bool PTTData::updateSipData(SipData *sipData)
52 {
53     DBConnection::instance()->open();
54     SipDAO sipDAO;
55     bool res = sipDAO.update(sipData);
56     DBConnection::instance()->close();
57     return res;
58 }
59
60 QList<SipData*>* PTTData::getGroupList(int dispName)
61 {
62     DBConnection::instance()->open();
63     SipDAO sipDAO;
64     QList<SipData*>* groupList = sipDAO.getGroupList(dispName);
65     DBConnection::instance()->close();
66     return groupList;
67 }
68
69 SipData* PTTData::getDispatcher(int name)
70 {
71     DBConnection::instance()->open();
```

```
72      SipDAO sipDAO;
73      SipData* sipData = sipDAO.getDispatcher(name);
74      DBConnection::instance()->close();
75      return sipData;
76  }
```

D.4.3 IDAO.h

```
1  #ifndef IDAO_H
2  #define IDAO_H
3
4  template<class T>
5  class IDAO
6  {
7  public:
8      virtual bool create(T *to) = 0;
9      virtual T*   read(int id) = 0;
10     virtual bool update(T *to) = 0;
11     virtual bool remove(T *to) = 0;
12 };
13
14
15 #endif // IDAO_H
```

D.4.4 sip_dao.h

```
1  #ifndef SIPDAO_H
2  #define SIPDAO_H
3
4  #include <QObject>
5  #include "dto/sip_data.h"
6  #include "IDAO.h"
7  #include "db/DBConnection.h"
8
9  class SipDAO : public QObject, public IDAO<SipData>
10 {
11     Q_OBJECT
12 public:
13     SipDAO(QObject *parent = 0);
14
15     virtual bool        create(SipData *to);
16     virtual SipData*    read(int id);
17     virtual bool        update(SipData *to);
18     virtual bool        remove(SipData *to);
19
20
21     QList<SipData*>* getGroupList(int dispName);
22     SipData*         getDispatcher(int name);
23
24
25 };
26
```

```
27 #endif // SIPDAO_H
```

D.4.5 sip_dao.cpp

```
1  #include <QtSql>
2  #include <QList>
3
4  #include "sip_dao.h"
5  #include "dto/sip_data.h"
6  #include "db/DBConnection.h"
7
8  // MySQL Prepared statement
9  #define CREATEQUERY      "INSERT INTO sip_data (name, secret, username, type,
       host, context, mailbox, alias) VALUES (%1, %2, %3, %4, %5; %6, %7, %8)"
10 #define READQUERY        "SELECT * FROM sip_data WHERE name = %1"
11 #define UPDATEQUERY      "UPDATE sip_data SET secret = %1, username = '%2',
       type = '%3', host = '%4', context = '%5', mailbox = '%6', alias = '%7'
       WHERE name = %8"
12 #define REMOVEQUERY      "DELETE FROM sip_data WHERE name = %1"
13 #define FINDUNITSQRY     "SELECT * FROM sip_data WHERE name IN (SELECT sip_name
        FROM sip_data_group WHERE group_id = (SELECT group_id FROM sip_group
       WHERE sip_name = %1))"
14 #define FINDDISPQRY      "SELECT * FROM sip_data WHERE name =  (SELECT sip_name
        FROM sip_group WHERE group_id = (SELECT group_id FROM sip_data_group
       WHERE sip_name = %1))"
15
16 SipDAO::SipDAO(QObject *parent) :
17     QObject(parent)
18 {
19
20 }
21
22 bool SipDAO::create(SipData *to)
23 {
24     qDebug() << __FUNCTION__;
25
26     return DBConnection::instance()->doUpdate(QString(CREATEQUERY)
27                                         .arg(to->name())
28                                         .arg(to->secret())
29                                         .arg(to->username())
30                                         .arg(to->type())
31                                         .arg(to->host())
32                                         .arg(to->context())
33                                         .arg(to->mailbox())
34                                         .arg(to->alias()));
35 }
36
37
38 SipData* SipDAO::read(int id)
39 {
40     qDebug() << __FUNCTION__;
41
42     SipData *sd = new SipData();
43
44     QSqlQuery *qry = DBConnection::instance()->doQuery(QString(READQUERY)
```

```
45                                                          .arg(id));
46
47      if(qry->first())
48      {
49          sd->setSip_id(qry->value(0).toInt());
50          sd->setName(qry->value(1).toString());
51          sd->setSecret(qry->value(2).toString());
52          sd->setUsername(qry->value(3).toString());
53          sd->setType(qry->value(4).toString());
54          sd->setHost(qry->value(5).toString());
55          sd->setIpaddr(qry->value(6).toString());
56          sd->setPort(qry->value(7).toInt());
57          sd->setContext(qry->value(8).toString());
58          sd->setMailbox(qry->value(9).toString());
59          sd->setRegseconds(qry->value(10).toInt());
60          sd->setUseragent(qry->value(11).toString());
61          sd->setLastms(qry->value(12).toInt());
62          sd->setAlias(qry->value(13).toString());
63      }
64      else
65      {
66          delete sd;
67          sd = NULL;
68      }
69      delete qry;
70      qry = NULL;
71      return sd;
72  }
73
74  bool SipDAO::update(SipData *to)
75  {
76      qDebug() << __FUNCTION__;
77
78      return DBConnection::instance()->doUpdate(QString(UPDATEQUERY)
79                                          .arg(to->secret())
80                                          .arg(to->username())
81                                          .arg(to->type())
82                                          .arg(to->host())
83                                          .arg(to->context())
84                                          .arg(to->mailbox())
85                                          .arg(to->alias())
86                                          .arg(to->name()));
87  }
88
89  bool SipDAO::remove(SipData *to)
90  {
91      qDebug() << __FUNCTION__;
92
93      return DBConnection::instance()->doUpdate(QString(REMOVEQUERY)
94                                          .arg(to->name()));
95  }
96
97  /*
98  Returnes list of SipData objects.
99  This list corresponds to the group as an operator manages.
100 */
101 QList<SipData*>* SipDAO::getGroupList(int dispName)
```

```
102 {
103     qDebug() << __FUNCTION__;
104 
105     QList<SipData*> *groupList = new QList<SipData*>();
106 
107     QSqlQuery *qry = DBConnection::instance()->doQuery(QString(FINDUNITSQRY)
108                                                        .arg(dispName));
109 
110     while(qry->next())
111     {
112         SipData *sd = new SipData();
113         sd->setSip_id(qry->value(0).toInt());
114         sd->setName(qry->value(1).toString());
115         sd->setSecret(qry->value(2).toString());
116         sd->setUsername(qry->value(3).toString());
117         sd->setType(qry->value(4).toString());
118         sd->setHost(qry->value(5).toString());
119         sd->setIpaddr(qry->value(6).toString());
120         sd->setPort(qry->value(7).toInt());
121         sd->setContext(qry->value(8).toString());
122         sd->setMailbox(qry->value(9).toString());
123         sd->setRegseconds(qry->value(10).toInt());
124         sd->setUseragent(qry->value(11).toString());
125         sd->setLastms(qry->value(12).toInt());
126         sd->setAlias(qry->value(13).toString());
127         groupList->append(sd);
128     }
129 
130     delete qry;
131     qry = NULL;
132     return groupList;
133 }
134 
135 /*
136 Returnes a SipData object.
137 The object corresponds to the operator who manage the group.
138 */
139 SipData* SipDAO::getDispatcher(int name)
140 {
141     qDebug() << __FUNCTION__;
142 
143     SipData *sd = new SipData();
144 
145     QSqlQuery *qry = DBConnection::instance()->doQuery(QString(FINDDISPQRY)
146                                                        .arg(name));
147 
148     if(qry->first())
149     {
150         sd->setSip_id(qry->value(0).toInt());
151         sd->setName(qry->value(1).toString());
152         sd->setSecret(qry->value(2).toString());
153         sd->setUsername(qry->value(3).toString());
154         sd->setType(qry->value(4).toString());
155         sd->setHost(qry->value(5).toString());
156         sd->setIpaddr(qry->value(6).toString());
157         sd->setPort(qry->value(7).toInt());
158         sd->setContext(qry->value(8).toString());
```

```
159         sd->setMailbox(qry->value(9).toString());
160         sd->setRegseconds(qry->value(10).toInt());
161         sd->setUseragent(qry->value(11).toString());
162         sd->setLastms(qry->value(12).toInt());
163         sd->setAlias(qry->value(13).toString());
164     }
165     else
166     {
167         delete sd;
168         sd = NULL;
169     }
170     delete qry;
171     qry = NULL;
172     return sd;
173 }
```

D.4.6 DBConnection.h

```
1  #ifndef DBCONNECTION_HPP
2  #define DBCONNECTION_HPP
3
4  #include <QObject>
5  #include <QtSql>
6
7  class DBConnection : public QObject
8  {
9      Q_OBJECT
10
11 public:
12     bool open();
13     void close();
14     static DBConnection *instance();
15
16     void        setCredentials(const QString &username, const QString &
           password, const QString &domain, const QString &database);
17     QSqlQuery*  doQuery(const QString &query);
18     bool        doUpdate(const QString &update);
19
20 private:
21     DBConnection(QObject *parent = 0);
22     ~DBConnection();
23     static DBConnection *s_instance;
24     QSqlDatabase db;
25
26     QString m_username;
27     QString m_password;
28     QString m_domain;
29     QString m_database;
30
31     bool isCredentialsSet;
32 };
33 #endif // DBCONNECTION_HPP
```

D.4.7 DBConnection.cpp

```
1  #include <QDebug>
2  #include "db/DBConnection.h"
3
4  DBConnection *DBConnection::s_instance  = 0;
5
6  DBConnection::DBConnection(QObject *parent) :
7      QObject(parent),
8      isCredentialsSet(false)
9  {
10     db  =  QSqlDatabase::addDatabase("QMYSQL");
11 }
12
13 DBConnection::~DBConnection(){}
14
15
16 bool DBConnection::open()
17 {
18     //qDebug() << __FUNCTION__;
19     if (!db.isOpen())
20     {
21         if(isCredentialsSet)
22         {
23             db.setHostName(m_domain);
24             db.setUserName(m_username);
25             db.setPassword(m_password);
26             db.setDatabaseName(m_database);
27         }
28         else
29         {
30             qDebug()<< "No Credentials Set";
31             return false;
32         }
33
34         if(db.open())
35         {
36             qDebug()<< "Database Opne";
37         }
38         else
39         {
40             qDebug()<< "DATABASE ERROR!!!" << db.lastError().text();
41             return false;
42         }
43     }
44     return true;
45 }
46
47 void DBConnection::close()
48 {
49     //qDebug() << __FUNCTION__;
50
51     if(db.isOpen())
52     {
53         db.close();
54         qDebug()<< "Database Close";
55     }
```

```
56 }
57
58 DBConnection *DBConnection::instance()
59 {
60     //qDebug() << __FUNCTION__ << "DBConnection";
61
62     if (!s_instance)
63     {
64         //qDebug()<< "First Instance";
65         s_instance = new DBConnection();
66     }
67     return s_instance;
68 }
69
70 void DBConnection::setCredentials(const QString &username, const QString &
       password, const QString &domain, const QString &database)
71 {
72     //qDebug() << __FUNCTION__;
73     m_username = username;
74     m_password = password;
75     m_domain = domain;
76     m_database = database;
77     isCredentialsSet = true;
78 }
79
80 QSqlQuery* DBConnection::doQuery(const QString &query)
81 {
82     //qDebug() << __FUNCTION__;
83     qDebug() << query;
84     QSqlQuery *qry = new QSqlQuery();
85
86     if(db.isOpen())
87     {
88
89         qry->prepare(query);
90         if( !qry->exec() )
91         {
92             qDebug() << qry->lastError();
93             delete qry;
94             qry = NULL;
95         }
96         else
97         {
98             qDebug( "Query done!" );
99         }
100     }
101     return qry;
102 }
103
104 bool DBConnection::doUpdate(const QString &update)
105 {
106     //qDebug() << __FUNCTION__;
107     qDebug() << update;
108     bool res = false;
109
110     if(db.isOpen())
111     {
```

```
112         QSqlQuery qry;
113         qry.prepare(update);
114         if( !qry.exec())
115         {
116             qDebug() << qry.lastError();
117         }
118         else
119         {
120             qDebug( "Update done!" );
121             res = true;
122         }
123     }
124     return res;
125 }
```

D.4.8 sip_data.h

```
1  #ifndef SIP_DATA_H
2  #define SIP_DATA_H
3
4  #include <QObject>
5
6  class SipData : public QObject
7  {
8      Q_OBJECT
9  public:
10     SipData(QObject *parent = 0);
11
12     void setSip_id(const int &sip_id);
13     void setName(const QString &name);
14     void setSecret(const QString &secret);
15     void setUsername(const QString &username);
16     void setType(const QString &type);
17     void setHost(const QString &host);
18     void setIpaddr(const QString &ipaddr);
19     void setPort(const int &port);
20     void setContext(const QString &context);
21     void setMailbox(const QString &mailbox);
22     void setRegseconds(const int &regseconds);
23     void setUseragent(const QString &useragent);
24     void setLastms(const int &lastms);
25     void setAlias(const QString &alias);
26
27     int     sip_id();
28     QString name();
29     QString secret();
30     QString username();
31     QString type();
32     QString host();
33     QString ipaddr();
34     int     port();
35     QString context();
36     QString mailbox();
37     int     regseconds();
38     QString useragent();
```

```
39      int      lastms();
40      QString alias();
41
42  private:
43      int      m_sip_id;
44      QString m_name;
45      QString m_secret;
46      QString m_username;
47      QString m_type;
48      QString m_host;
49      QString m_ipaddr;
50      int      m_port;
51      QString m_context;
52      QString m_mailbox;
53      int      m_regseconds;
54      QString m_useragent;
55      int      m_lastms;
56      QString m_alias;
57
58
59  };
60
61  #endif // SIP_DATA_H
```

D.4.9 sip_data.cpp

```
1   #include <QDebug>
2   #include "sip_data.h"
3
4   SipData::SipData(QObject *parent) :
5       QObject(parent),
6       m_sip_id(0),
7       m_name(""),
8       m_secret(""),
9       m_username(""),
10      m_type(""),
11      m_host(""),
12      m_ipaddr(""),
13      m_port(0),
14      m_context(""),
15      m_mailbox(""),
16      m_regseconds(0),
17      m_useragent(""),
18      m_lastms(0),
19      m_alias("")
20
21  {
22      //qDebug() << __FUNCTION__;
23  }
24
25
26
27  void SipData::setSip_id(const int &sip_id)
28  {
29      m_sip_id = sip_id;
```

```
30 }
31
32 void SipData::setName(const QString &name)
33 {
34     m_name = name;
35 }
36
37 void SipData::setSecret(const QString &secret)
38 {
39     m_secret = secret;
40 }
41
42 void SipData::setUsername(const QString &username)
43 {
44     m_username = username;
45 }
46
47 void SipData::setType(const QString &type)
48 {
49     m_type = type;
50 }
51
52 void SipData::setHost(const QString &host)
53 {
54     m_host = host;
55 }
56
57 void SipData::setIpaddr(const QString &ipaddr)
58 {
59     m_ipaddr = ipaddr;
60 }
61
62 void SipData::setPort(const int &port)
63 {
64     m_port = port;
65 }
66
67 void SipData::setContext(const QString &context)
68 {
69     m_context = context;
70 }
71
72 void SipData::setMailbox(const QString &mailbox)
73 {
74     m_mailbox = mailbox;
75 }
76
77 void SipData::setRegseconds(const int &regseconds)
78 {
79     m_regseconds = regseconds;
80 }
81
82 void SipData::setUseragent(const QString &useragent)
83 {
84     m_useragent = useragent;
85 }
86
```

```
87  void SipData::setLastms(const int &lastms)
88  {
89      m_lastms = lastms;
90  }
91
92  void SipData::setAlias(const QString &alias)
93  {
94      m_alias = alias;
95  }
96
97
98  int SipData::sip_id()
99  {
100     return m_sip_id;
101 }
102
103 QString SipData::name()
104 {
105     return m_name;
106 }
107
108 QString SipData::secret()
109 {
110     return m_secret;
111 }
112
113 QString SipData::username()
114 {
115     return m_username;
116 }
117
118 QString SipData::type()
119 {
120     return m_type;
121 }
122
123 QString SipData::host()
124 {
125     return m_host;
126 }
127
128 QString SipData::ipaddr()
129 {
130     return m_ipaddr;
131 }
132
133 int SipData::port()
134 {
135     return m_port;
136 }
137
138 QString SipData::context()
139 {
140     return m_context;
141 }
142
143 QString SipData::mailbox()
```

```
144 {
145     return m_mailbox;
146 }
147
148 int SipData::regseconds()
149 {
150     return m_regseconds;
151 }
152
153 QString SipData::useragent()
154 {
155     return m_useragent;
156 }
157
158 int SipData::lastms()
159 {
160     return m_lastms;
161 }
162
163 QString SipData::alias()
164 {
165     return m_alias;
166 }
```

D.5 PTT Dispatcher

D.5.1 main.cpp

```
1  #include <QtGui/QApplication>
2  #include "mainwindow.h"
3
4  int main(int argc, char *argv[])
5  {
6      QApplication::setGraphicsSystem("raster");
7      QApplication a(argc, argv);
8
9      QString *args;
10     args = 0;
11
12     QString *disp;
13     disp = 0;
14
15     if (argc>=2)
16     {
17         for(int i=2;i<argc+1;i++)
18         {
19             qDebug()<<argv[i-1];
20             if(strcmp(argv[i-1],"-MT")==0){
21                 QApplication::setOverrideCursor(QCursor(Qt::BlankCursor));
22                 args = new QString("-MT");
23             }else if(strcmp(argv[i-1],"-DISP")==0){
24                 disp = new QString(argv[i]);
25             }
```

```
26          }
27      }
28
29      MainWindow w(disp,args);
30      w.show();
31
32      return a.exec();
33  }
```

D.5.2 mainwindow.h

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QtDeclarative>
6
7   #include "ptt_controller.h"
8   #include "ptt_user.h"
9
10  #include "qml_widget.h"
11  #include "qml_listmodel.h"
12  #include "qml_buttonindicator.h"
13  #include "ptt_listitem.h"
14
15
16  typedef enum
17  {
18      ALL,
19      NOOFFLINE,
20      INCOMING,
21      MISSED
22  }ListViewState;
23
24  class MainWindow : public QMainWindow
25  {
26      Q_OBJECT
27
28  public:
29      explicit MainWindow(QString *disp = 0, QString *args = 0, QWidget *parent
            = 0);
30      ~MainWindow();
31
32  public slots:
33      void new_log_message(QString text);
34      void showOnline();
35      void showAll();
36      void showIncoming();
37      void showMissed();
38      void removeFromMissedCalls(PTTListItem* pttItem);
39      void exit_app();
40      //TEMP
41      void test();
42
43  private:
```

```
44      QDeclarativeView *ui;
45
46      PTTController *pttController;
47
48      QList<PTTUser*>*    m_allPttUsersList;
49      QList<PTTListItem*> m_allQmlItemsList;
50
51      QList<PTTUser*>     m_onlineUsersList;
52      QList<PTTUser*>     m_incomingCallsList;
53      QList<PTTUser*>     m_missedCallsList;
54
55      QmlListModel*       m_qmlGraphicListModel;
56      PTTListItem*        m_displItme;
57
58      bool offlineShow;
59      ListViewState listStates;
60      QMLButtinIndicator m_buttonIndicator;
61
62
63      void closeEvent(QCloseEvent *event);
64      void keyPressEvent(QKeyEvent * event);
65      void SetFullScreen();
66
67      PTTListItem* findListItem(const QString& name);
68      void updateButtonIndicator();
69
70  private slots:
71      void listIndexChanged(int index);
72      void stateChanged(PTT_States pttState);
73      void missedCall(bool state);
74
75  };
76
77  #endif // MAINWINDOW_H
```

D.5.3 mainwindow.cpp

```
1   #include <QCloseEvent>
2   #include <QEvent>
3   #include <QtGui>
4   #include <QScrollArea>
5   #include <QAbstractScrollArea>
6   #include <QtDeclarative>
7   #include "mainwindow.h"
8   #include "ui_mainwindow.h"
9   #include "ptt_widget.h"
10
11  #include "ptt_listitem.h"
12  #include "ptt_loglistitem.h"
13  #include "qml_listmodel.h"
14  #include "qml_widget.h"
15  #include "testname.h"
16
17  //#include <QGLWidget>
18
```

```
19 MainWindow::MainWindow(QString *disp, QString *args, QWidget *parent) :
20     QMainWindow(parent),
21     offlineShow(true),
22     listStates(ALL)
23 {
24     qmlRegisterType<PTTListItem>("CustomComponents", 1, 0, "PTTListItem");
25     qmlRegisterType<PTTLogListItem>("CustomComponents", 1, 0, "PTTLogListItem"
           );
26
27     if(!disp){
28         qDebug()<< "NO DISP ENTERED";
29         this->exit_app();
30     }
31
32     pttController = new PTTController(*disp);
33     m_allPttUsersList = pttController->getPttUserList();
34
35     m_qmlGraphicListModel = new QmlListModel(new PTTListItem);
36
37     // Dispatcher data-object
38     m_dispItme = new PTTListItem(pttController->getDispUser());
39
40     if(m_allPttUsersList)
41     {
42         for(int i=0;i<m_allPttUsersList->size();i++)
43         {
44             connect(m_allPttUsersList->at(i), SIGNAL(stateChanged(PTT_States))
                   , this, SLOT(stateChanged(PTT_States)));
45             connect(m_allPttUsersList->at(i), SIGNAL(missedCall(bool)), this,
                   SLOT(missedCall(bool)));
46
47             m_allQmlItemsList.append(new PTTListItem(m_allPttUsersList->at(i))
                   );
48             if(m_allPttUsersList->at(i)->getState() == ONLINE)
49             {
50                 m_onlineUsersList.append(m_allPttUsersList->at(i));
51             }
52         }
53     }
54
55
56     for(int i=0;i<m_allQmlItemsList.size();i++)
57     {
58         m_qmlGraphicListModel->appendRow(m_allQmlItemsList.at(i));
59     }
60
61     //Creating til main window
62     this->ui = new QDeclarativeView;
63     this->ui->rootContext()->setContextProperty("mainWindow", this);
64     this->ui->rootContext()->setContextProperty("buttonIndicator", &
           m_buttonIndicator);
65     this->ui->rootContext()->setContextProperty("listModel",
           m_qmlGraphicListModel);
66     this->ui->rootContext()->setContextProperty("detailItem", NULL);
67     this->ui->rootContext()->setContextProperty("dispItem", m_dispItme);
68     this->ui->setSource(QUrl("qrc:qml/window.qml"));
69     this->ui->setResizeMode(QDeclarativeView::SizeRootObjectToView);
```

```
70
71      // Optemize the performance of the app.
72      this->ui->setAttribute(Qt::WA_OpaquePaintEvent);
73      this->ui->setAttribute(Qt::WA_NoSystemBackground);
74      this->ui->viewport()->setAttribute(Qt::WA_OpaquePaintEvent);
75      this->ui->viewport()->setAttribute(Qt::WA_NoSystemBackground);
76
77      if(args && args->compare("-MT")==0)
78      {
79          SetFullScreen();
80      }else{
81          //this->setGeometry()
82      }
83
84      connect(this->ui->rootObject()->findChild<QObject *>("pttListBase"),
            SIGNAL(indexChanged(int)), this, SLOT(listIndexChanged(int)));
85      setCentralWidget(this->ui);
86  }
87
88  MainWindow::~MainWindow()
89  {
90      //delete ui;
91  }
92
93  void MainWindow::SetFullScreen()
94  {
95      //qDebug() << __FUNCTION__;
96      this->showFullScreen();
97  }
98
99  void MainWindow::closeEvent(QCloseEvent */*event*/) {
100     exit_app();
101 }
102
103 void MainWindow::exit_app()
104 {
105     pttController->shutdownPjSIP();
106     exit(0);
107 }
108
109 void MainWindow::keyPressEvent(QKeyEvent * event)
110 {
111     switch (event->key())
112     {
113     case Qt::Key_Escape:
114         if (event->modifiers() & Qt::ControlModifier)
115         {
116             qDebug() << "EXIT";
117             exit_app();
118         }
119
120         break;
121
122     default:
123
124         break;
125     }
```

```
126 }
127
128 void MainWindow::listIndexChanged(int index)
129 {
130     //qDebug()<<__FUNCTION__;
131     qDebug()<< "index: "<<index;
132     PTTListItem* pttItem = qobject_cast<PTTListItem*>(m_qmlGraphicListModel->
        find(index));
133     this->ui->rootContext()->setContextProperty("detailItem", pttItem);
134     this->ui->rootContext()->setContextProperty("loglistModel", pttItem->
        getPTTUser()->getLogListModel());
135     pttItem->update();
136     removeFromMissedCalls(pttItem);
137 }
138
139 void MainWindow::new_log_message(QString text)
140 {
141     qDebug()<<text;
142 }
143
144 void MainWindow::showOnline()
145 {
146     qDebug()<< __FUNCTION__;
147     if(listStates != NOOFFLINE)
148     {
149         listStates = NOOFFLINE;
150         m_qmlGraphicListModel->removeRows(0,m_qmlGraphicListModel->rowCount())
            ;
151
152         for(int i=0; i< m_onlineUsersList.size(); i++)
153         {
154             PTTListItem* pttLi = findListItem(m_onlineUsersList.at(i)->
                getSipData()->name());
155             if(pttLi)
156             {
157                 m_qmlGraphicListModel->appendRow(pttLi);
158             }
159         }
160     }
161
162 }
163
164 void MainWindow::test()
165 {
166     m_qmlGraphicListModel->insertRow(1,m_allQmlItemsList.at(2));
167 }
168
169 void MainWindow::showIncoming()
170 {
171     qDebug()<< __FUNCTION__;
172
173     if(listStates != INCOMING)
174     {
175         listStates = INCOMING;
176         m_qmlGraphicListModel->removeRows(0,m_qmlGraphicListModel->rowCount())
            ;
177
```

```
178         for(int i=0; i< m_incomingCallsList.size(); i++)
179         {
180             PTTListItem* pttLi = findListItem(m_incomingCallsList.at(i)->
                    getSipData()->name());
181             if(pttLi)
182             {
183                 m_qmlGraphicListModel->appendRow(pttLi);
184             }
185         }
186     }
187 }
188
189 void MainWindow::stateChanged(PTT_States pttState)
190 {
191     qDebug()<< __FUNCTION__;
192     PTTUser* pttU = qobject_cast<PTTUser*>(sender());
193
194     switch(pttState)
195     {
196     case UNKNOWN: break;
197     case OFFLINE:
198     {
199         qDebug()<< listStates;
200         m_onlineUsersList.removeOne(pttU);
201         if(listStates == NOOFFLINE)
202         {
203             PTTListItem* pttLi = findListItem(pttU->getSipData()->name());
204             if(pttLi)
205             {
206                 QModelIndex ind = m_qmlGraphicListModel->indexFromItem(pttLi);
207                 m_qmlGraphicListModel->removeRow(ind.row());
208             }
209         }
210     }
211         break;
212     case ONLINE:
213     {
214         if(m_onlineUsersList.indexOf(pttU) == -1)
215         {
216             m_onlineUsersList.append(pttU);
217         }
218         if(listStates == NOOFFLINE)
219         {
220             PTTListItem* pttLi = findListItem(pttU->getSipData()->name());
221             if(pttLi)
222             {
223                 QModelIndex ind = m_qmlGraphicListModel->indexFromItem(pttLi);
224                 if(ind.row() == -1)
225                 {
226                     m_qmlGraphicListModel->appendRow(pttLi);
227                 }
228             }
229         }
230
231         m_incomingCallsList.removeOne(pttU);
232         if(listStates == INCOMING)
233         {
```

```
234             PTTListItem* pttLi = findListItem(pttU->getSipData()->name());
235             if(pttLi)
236             {
237                 QModelIndex ind = m_qmlGraphicListModel->indexFromItem(pttLi);
238                 m_qmlGraphicListModel->removeRow(ind.row());
239             }
240         }
241     }
242         break;
243     case INUSE:
244     {
245         m_incomingCallsList.removeOne(pttU);
246         if(listStates == INCOMING)
247         {
248             PTTListItem* pttLi = findListItem(pttU->getSipData()->name());
249             if(pttLi)
250             {
251                 QModelIndex ind = m_qmlGraphicListModel->indexFromItem(pttLi);
252                 m_qmlGraphicListModel->removeRow(ind.row());
253             }
254         }
255     }
256         break;
257     case RINGING:
258     {
259         m_incomingCallsList.append(pttU);
260         if(listStates == INCOMING)
261         {
262             PTTListItem* pttLi = findListItem(pttU->getSipData()->name());
263             if(pttLi)
264             {
265                 m_qmlGraphicListModel->appendRow(pttLi);
266             }
267         }
268     }
269
270         break;
271     case CALLING:break;
272
273     default: break;
274     }
275     updateButtonIndicator();
276 }
277
278 PTTListItem * MainWindow::findListItem(const QString &name)
279 {
280     for(int j=0; m_allQmlItemsList.size(); j++)
281     {
282         if(name == m_allQmlItemsList.at(j)->getName())
283         {
284             return m_allQmlItemsList.at(j);
285         }
286     }
287     return NULL;
288 }
289
290 void MainWindow::showAll()
```

```
291 {
292     qDebug()<< __FUNCTION__;
293 
294     if(listStates != ALL)
295     {
296         listStates = ALL;
297         m_qmlGraphicListModel->removeRows(0,m_qmlGraphicListModel->rowCount())
                ;
298 
299         for(int i=0; i< m_allPttUsersList->size(); i++)
300         {
301             PTTListItem* pttLi = findListItem(m_allPttUsersList->at(i)->
                    getSipData()->name());
302             if(pttLi)
303             {
304                 m_qmlGraphicListModel->appendRow(pttLi);
305             }
306         }
307     }
308 }
309 
310 void MainWindow::missedCall(bool state)
311 {
312     qDebug()<< __FUNCTION__;
313 
314     if(state)
315     {
316         PTTUser* pttU = qobject_cast<PTTUser*>(sender());
317 
318         if(m_missedCallsList.indexOf(pttU) == -1)
319         {
320             m_missedCallsList.append(pttU);
321         }
322         else
323         {
324             m_missedCallsList.removeOne(pttU);
325             m_missedCallsList.prepend(pttU);
326         }
327 
328         if(listStates == MISSED)
329         {
330             PTTListItem* pttLi = findListItem(pttU->getSipData()->name());
331             if(pttLi)
332             {
333                 QModelIndex ind = m_qmlGraphicListModel->indexFromItem(pttLi);
334                 m_qmlGraphicListModel->removeRow(ind.row());
335                 m_qmlGraphicListModel->prependRow(pttLi);
336             }
337         }
338         updateButtonIndicator();
339     }
340 }
341 
342 void MainWindow::showMissed()
343 {
344     qDebug()<< __FUNCTION__;
345 
```

```
346     if(listStates != MISSED)
347     {
348         listStates = MISSED;
349         m_qmlGraphicListModel->removeRows(0,m_qmlGraphicListModel->rowCount())
                ;
350
351         for(int i=0; i< m_missedCallsList.size(); i++)
352         {
353             PTTListItem* pttLi = findListItem(m_missedCallsList.at(i)->
                    getSipData()->name());
354             if(pttLi)
355             {
356                 m_qmlGraphicListModel->appendRow(pttLi);
357             }
358         }
359     }
360 }
361
362 void MainWindow::updateButtonIndicator()
363 {
364     qDebug() << __FUNCTION__;
365
366     if(m_incomingCallsList.size() > 0)
367     {
368         m_buttonIndicator.setIncomingON(true);
369     }
370     else
371     {
372         m_buttonIndicator.setIncomingON(false);
373     }
374
375     if(m_missedCallsList.size() > 0)
376     {
377         m_buttonIndicator.setMissedON(true);
378     }
379     else
380     {
381         m_buttonIndicator.setMissedON(false);
382     }
383 }
384
385 void MainWindow::removeFromMissedCalls(PTTListItem* pttItem)
386 {
387     PTTUser* pttU = pttItem->getPTTUser();
388     m_missedCallsList.removeOne(pttU);
389
390     pttItem->updateMissedCall(false);
391
392     if(listStates == MISSED)
393     {
394         if(pttItem)
395         {
396             QModelIndex ind = m_qmlGraphicListModel->indexFromItem(pttItem);
397             m_qmlGraphicListModel->removeRow(ind.row());
398         }
399     }
400     updateButtonIndicator();
```

```
401 }
```

D.5.4 ptt_controller.h

```
1  #ifndef PTT_CONTROLLER_H
2  #define PTT_CONTROLLER_H
3
4  #include <QObject>
5  #include "sip_manager.h"
6  #include "ami_manager.h"
7  //#include "ptt_data.h"
8  //#include "ptt_user.h"
9
10 class PTTData;
11 class PTTUser;
12 class SipData;
13
14 class PTTController : public QObject
15 {
16     Q_OBJECT
17 public:
18     PTTController(QString dispName, QObject *parent = 0);
19
20     PTTUser*          getDispUser();
21     QList<PTTUser*>* getPttUserList();
22
23
24 signals:
25
26
27 public slots:
28     void shutdownPjSIP();
29
30 private:
31     SIPManager*       m_sipManager;
32     AMIManager*       m_amiManager;
33     PTTData*          m_pttData;
34     SipData*          m_dispData;
35     PTTUser*          m_dispUser;
36     QList<PTTUser*>* m_pttUserList;
37
38 private slots:
39     void incoming_call(int call_id, QString remote_info);
40     void call_state(int call_id,QString remote_info, QString state_text, int
           state_id);
41     void peerStatusChanged(QString peerStatus, QString peer, QString address,
           QString cause);
42
43 };
44
45 #endif // PTT_CONTROLLER_H
```

D.5.5 ptt_controller.cpp

```
#include "ptt_controller.h"
#include "ptt_user.h"
#include "ptt_data.h"

PTTController::PTTController(QString dispName, QObject *parent) :
    QObject(parent)
{
    m_pttData = new PTTData("asterisk", "astproj", "10.6.20.126", "pttproject");
    m_dispData = m_pttData->getSipData(dispName.toInt());
    m_dispUser = new PTTUser(m_dispData, this);
    m_pttUserList = new QList<PTTUser*>();

    QList<SipData*> *groupList = m_pttData->getGroupList(dispName.toInt());
    for(int i=0; i<groupList->size();i++)
    {
        m_pttUserList->append(new PTTUser(groupList->at(i), this));
    }

    m_sipManager = new SIPManager(m_dispData->name(), m_dispData->secret(), "10.6.20.126");

    m_amiManager = new AMIManager("10.6.20.126");
    m_amiManager->login("ami", "astproj");

    for(int i=0; i<m_pttUserList->size();i++)
    {
        connect(m_pttUserList->at(i), SIGNAL(call_make(QString,int*)), m_sipManager, SLOT(call_make(QString, int*)));
        connect(m_pttUserList->at(i), SIGNAL(call_answer(int,uint)), m_sipManager, SLOT(call_answer(int,uint)));
        connect(m_pttUserList->at(i), SIGNAL(call_hangup(int,uint)), m_sipManager, SLOT(call_hangup(int,uint)));
    }

    connect(m_sipManager, SIGNAL(incoming_call(int, QString)),      this, SLOT(incoming_call(int,QString)));
    connect(m_sipManager, SIGNAL(call_state(int,QString,QString, int)),  this, SLOT(call_state(int,QString,QString, int)));

    connect(m_amiManager, SIGNAL(peerStatusChanged(QString,QString,QString,QString)), this, SLOT(peerStatusChanged(QString,QString,QString,QString)));
}

PTTUser* PTTController::getDispUser()
{
    return m_dispUser;
}

QList<PTTUser*>* PTTController::getPttUserList()
{
    return m_pttUserList;
}

```

```
47
48 void PTTController::shutdownPjSIP()
49 {
50     m_sipManager->shutdownPjSIP();
51 }
52
53 void PTTController::incoming_call(int call_id, QString remote_info)
54 {
55     qDebug() << __FUNCTION__<<remote_info;
56
57     QString temp = remote_info.split("\" ").at(0);
58     remote_info = temp.remove(0,1);
59
60     qDebug() << "call_id: "<<call_id;
61     qDebug()<< "remote_info: "<<remote_info;
62
63     for(int i=0;i<m_pttUserList->size();i++)
64     {
65         qDebug() << m_pttUserList->at(i)->getSipData()->name();
66         if (m_pttUserList->at(i)->getSipData()->name().compare(remote_info)
               ==0)
67         {
68             m_pttUserList->at(i)->incommung_call(call_id);
69         }
70     }
71 }
72
73 void PTTController::call_state(int call_id, QString remote_info, QString
       state_text, int state_id)
74 {
75     qDebug() << __FUNCTION__ <<remote_info<<state_id<<state_text;
76
77     if(remote_info.contains("<"))
78     {
79         QString temp = remote_info.split("\" ").at(0);
80         remote_info = temp.remove(0,1);
81     }
82     else
83     {
84         QString temp = remote_info.split("@").at(0);
85         remote_info = temp.remove("sip:#");
86     }
87
88     qDebug() << "call_id: "<<call_id;
89     qDebug()<< "remote_info: "<<remote_info;
90
91     for(int i=0;i<m_pttUserList->size();i++)
92     {
93         qDebug() << m_pttUserList->at(i)->getSipData()->name();
94         if (m_pttUserList->at(i)->getSipData()->name().compare(remote_info)
               ==0)
95         {
96             m_pttUserList->at(i)->setSIPState(state_id);
97         }
98     }
99 }
100
```

```
101 void PTTController::peerStatusChanged(QString peerStatus, QString peer,
        QString address, QString cause)
102 {
103     peer = peer.remove("sip/");
104 
105     for(int i=0;i<m_pttUserList->size();i++)
106     {
107         if (m_pttUserList->at(i)->getSipData()->name().compare(peer)==0)
108         {
109             if(peerStatus.compare("unregistered")==0)
110             {
111                 m_pttUserList->at(i)->setState(OFFLINE);
112             }
113             else if(peerStatus.compare("registered")==0)
114             {
115                 QStringList addPort = address.split(":");
116                 m_pttUserList->at(i)->getSipData()->setIpaddr(addPort.at(0));
117                 m_pttUserList->at(i)->getSipData()->setPort(addPort.at(1).
                        toInt());
118                 m_pttUserList->at(i)->setState(ONLINE);
119             }
120         }
121     }
122 
123 }
```

D.5.6 ptt_datatypes.h

```
1  #ifndef PTT_DATATYPES_H
2  #define PTT_DATATYPES_H
3  
4  typedef enum
5  {
6      UNKNOWN,
7      OFFLINE,
8      ONLINE,
9      INUSE,
10     RINGING,
11     CALLING
12 }PTT_States;
13 
14 
15 typedef enum
16 {
17     STATE_NULL,
18     STATE_CALLING,
19     STATE_INCOMING,
20     STATE_EARLY,
21     STATE_CONNECTING,
22     STATE_CONFIRMED,
23     STATE_DISCONNECTED
24 }SIP_States;
25 
26 typedef enum
27 {
```

```
28      LOG_OFFLINE,
29      LOG_ONLINE,
30      LOG_RINGING,
31      LOG_CALLING,
32      LOG_MISSED
33  }PTT_Log_States;
34
35  #endif // PTT_DATATYPES_H
```

D.5.7 ptt_listitem.h

```
1   #ifndef PTT_LISTITEM_H
2   #define PTT_LISTITEM_H
3
4   #include <QtDeclarative>
5   #include <QtCore>
6   #include "ptt_datatypes.h"
7
8   #include "qml_listitem.h"
9   //class QmlListItem;
10  class PTTUser;
11
12  class PTTListItem : public QmlListItem
13  {
14      Q_OBJECT
15      Q_PROPERTY(QString status   READ getStatus     NOTIFY statusChanged)
16      Q_PROPERTY(QString name     READ getName       NOTIFY itemChanged)
17      Q_PROPERTY(QString ip       READ getIp         NOTIFY itemChanged)
18      Q_PROPERTY(QString alias    READ getAlias      NOTIFY itemChanged)
19      Q_PROPERTY(bool missedCall  READ getMissedCall NOTIFY itemChanged)
20
21  public:
22      enum Roles {
23          ObjectRole = Qt::UserRole+1,
24          NameRole,
25          StatusRole,
26          IpRole,
27          AliasRole
28      };
29
30  public:
31      PTTListItem(QDeclarativeItem *parent = 0): QmlListItem(parent) {}
32      PTTListItem(PTTUser *pttUser, QDeclarativeItem *parent = 0);
33
34      QVariant data(int role) const;
35      QHash<int, QByteArray> roleNames() const;
36
37      QString getStatus() const;
38      QString getName() const;
39      QString getIp() const;
40      QString getAlias() const;
41
42      PTTUser* getPTTUser();
43      bool getMissedCall();
44
```

```
45      void update();
46
47  signals:
48      void statusChanged(QString status);
49      void itemChanged();
50
51  public slots:
52      void operate();
53      void updateMissedCall(bool state);
54
55  private slots:
56      void stateChanged(PTT_States states);
57
58
59  private:
60      PTTUser *m_pttUser;
61      QHash<int, QByteArray> statusNames;
62      bool m_missedCall;
63
64  };
65
66
67
68
69  #endif // PTT_LISTITEM_H
```

D.5.8 ptt_listitem.cpp

```
1   #include "ptt_listitem.h"
2   #include "ptt_user.h"
3   #include "qml_listitem.h"
4
5
6
7   PTTListItem::PTTListItem(PTTUser *pttUser, QDeclarativeItem *parent) :
8       QmlListItem(parent)
9   {
10      m_pttUser = pttUser;
11      connect(m_pttUser, SIGNAL(stateChanged(PTT_States)), this, SLOT(
            stateChanged(PTT_States)));
12      connect(m_pttUser, SIGNAL(missedCall(bool)), this, SLOT(updateMissedCall(
            bool)));
13      statusNames[UNKNOWN] = "unknown";
14      statusNames[OFFLINE] = "offline";
15      statusNames[ONLINE]  = "online";
16      statusNames[INUSE]   = "inuse";
17      statusNames[RINGING] = "ringing";
18      statusNames[CALLING] = "calling";
19  }
20
21  QVariant PTTListItem::data(int role) const
22  {
23      switch(role) {
24      case ObjectRole:
25          return QVariant::fromValue((QObject*)(this));;
```

```
26      case NameRole:
27          return m_pttUser->getSipData()->name();
28      case StatusRole:
29          return statusNames[m_pttUser->getState()];
30      case IpRole:
31          return m_pttUser->getSipData()->ipaddr();
32      case AliasRole:
33          return m_pttUser->getSipData()->alias();
34      default:
35          return QVariant();
36      }
37  }
38
39  QHash<int, QByteArray> PTTListItem::roleNames() const
40  {
41      QHash<int, QByteArray> names;
42      names[ObjectRole]   = "object";
43      names[NameRole]     = "name";
44      names[StatusRole]   = "status";
45      names[IpRole]       = "ip";
46      names[AliasRole]    = "alias";
47      return names;
48  }
49
50  void PTTListItem::stateChanged(PTT_States states)
51  {
52      emit statusChanged(statusNames[states]);
53      update();
54  }
55
56  void PTTListItem::operate()
57  {
58      m_pttUser->operate();
59  }
60
61  void PTTListItem::update()
62  {
63      qDebug()<< __FUNCTION__;
64      emit dataChanged();
65      emit itemChanged();
66  }
67
68  QString PTTListItem::getStatus() const
69  {
70      return statusNames[m_pttUser->getState()];
71  }
72
73  QString PTTListItem::getName() const
74  {
75      return m_pttUser->getSipData()->name();
76  }
77
78  QString PTTListItem::getIp() const
79  {
80      return m_pttUser->getSipData()->ipaddr();
81  }
82
```

```
83 QString PTTListItem::getAlias() const
84 {
85     return m_pttUser->getSipData()->alias();
86 }
87
88 PTTUser * PTTListItem::getPTTUser()
89 {
90     return m_pttUser;
91 }
92
93 bool PTTListItem::getMissedCall()
94 {
95     return m_missedCall;
96 }
97
98 void PTTListItem::updateMissedCall(bool state)
99 {
100     m_missedCall = state;
101     update();
102 }
```

D.5.9 ptt_loglistitem.h

```
1  #ifndef PTT_LOGLISTITEM_H
2  #define PTT_LOGLISTITEM_H
3
4  #include <QObject>
5  #include <QDateTime>
6  #include "ptt_datatypes.h"
7
8  #include "qml_listitem.h"
9  //class QmlListItem;
10
11 class PTTLogListItem : public QmlListItem
12 {
13     Q_OBJECT
14     Q_PROPERTY(QDateTime time   READ getTime  NOTIFY itemChanged)
15     Q_PROPERTY(QString   alias  READ getAlias NOTIFY itemChanged)
16     Q_PROPERTY(QString   event  READ getEvent NOTIFY itemChanged)
17
18
19 public:
20     enum Roles {
21         ObjectRole = Qt::UserRole+1,
22         TimeRole,
23         AliasRole,
24         EventRole
25     };
26
27 public:
28     PTTLogListItem(QDeclarativeItem *parent = 0): QmlListItem(parent) {}
29     PTTLogListItem(QDateTime time, PTT_Log_States event, QString alias,
        QDeclarativeItem *parent = 0);
30
31     QVariant data(int role) const;
```

```
32      QHash<int, QByteArray> roleNames() const;
33
34      QDateTime getTime() const;
35      QString   getAlias() const;
36      QString   getEvent() const;
37
38      void update();
39
40  signals:
41      void itemChanged();
42
43  private:
44      QHash<int, QByteArray> statusNames;
45
46      QDateTime       m_time;
47      QString         m_alias;
48      PTT_Log_States  m_event;
49
50  };
51
52  #endif // PTT_LOGLISTITEM_H
```

D.5.10 ptt_loglistitem.cpp

```
1   #include "ptt_loglistitem.h"
2   #include "qml_listitem.h"
3
4   PTTLogListItem::PTTLogListItem(QDateTime time, PTT_Log_States event, QString
        alias, QDeclarativeItem *parent) :
5       QmlListItem(parent),
6       m_time(time),
7       m_event(event),
8       m_alias(alias)
9   {
10
11  }
12
13  QVariant PTTLogListItem::data(int role) const
14  {
15      switch(role) {
16      case ObjectRole:
17          return QVariant::fromValue((QObject*)(this));;
18      case TimeRole:
19          return m_time;
20      case AliasRole:
21          return m_alias;
22      case EventRole:
23          return getEvent();
24      default:
25          return QVariant();
26      }
27  }
28
29  QHash<int, QByteArray> PTTLogListItem::roleNames() const
30  {
```

```
31      QHash<int, QByteArray> names;
32      names[ObjectRole]   = "object";
33      names[TimeRole]     = "time";
34      names[AliasRole]    = "alias";
35      names[EventRole]    = "event";
36
37      return names;
38  }
39
40  void PTTLogListItem::update()
41  {
42      qDebug()<< __FUNCTION__;
43      emit dataChanged();
44      emit itemChanged();
45  }
46
47  QDateTime PTTLogListItem::getTime() const
48  {
49      return m_time;
50  }
51
52  QString PTTLogListItem::getAlias() const
53  {
54      return m_alias;
55  }
56
57  QString PTTLogListItem::getEvent() const
58  {
59      qDebug()<< __FUNCTION__;
60
61      switch(m_event)
62      {
63      case LOG_OFFLINE: return "Offline";
64      case LOG_ONLINE:  return "Online";
65      case LOG_RINGING: return "Incoming";
66      case LOG_CALLING: return "Outgoing";
67      case LOG_MISSED:  return "Missed";
68      default:          return "Unknown";
69      }
70
71  }
```

D.5.11 ptt_user.h

```
1  #ifndef PTT_USER_H
2  #define PTT_USER_H
3
4  #include <QObject>
5  #include <QDateTime>
6  #include "ptt_data.h"
7
8  #include "ptt_datatypes.h"
9
10 class QmlListModel;
11
```

```
12 class PTTUser : public QObject
13 {
14     Q_OBJECT
15 public:
16     PTTUser(SipData *sipData, QObject *parent = 0);
17
18     SipData*    getSipData();
19     void        setState(PTT_States pttState);
20     PTT_States  getState();
21     void        setSIPState(int sipState);
22     // controler
23     void        incommung_call(int call_id);
24
25     QmlListModel* getLogListModel();
26
27
28
29 signals:
30     //view
31     void stateChanged(PTT_States pttState);
32     // controler
33     void call_hangup(int call_id, uint code);
34     void call_answer(int call_id, uint code);
35     void call_make(QString pjurl, int *call_id);
36
37     void missedCall(bool state);
38     void logSignal(QDateTime now, PTT_Log_States pttState, QString alias);
39
40
41 public slots:
42     //view
43     void operate();
44
45 private slots:
46     void recordLogEvent(QDateTime now, PTT_Log_States pttState, QString alias)
       ;
47
48
49 private:
50     SipData     *m_sipData;
51     PTT_States  m_pttStates;
52     PTT_States  m_pttStatesOld;
53     int         m_current_call_id;
54     int         *p_current_call_id;
55     bool          m_incoming;
56     QmlListModel*  m_logListModel;
57 };
58
59 #endif // PTT_USER_H
```

D.5.12 ptt_user.cpp

```
1 #include <QDebug>
2 #include "ptt_user.h"
3 #include "ptt_data.h"
```

```
4  #include "qml_listmodel.h"
5  #include "ptt_loglistitem.h"
6
7  #define PROTOCOL "sip"
8
9  PTTUser::PTTUser(SipData *sipData, QObject *parent) :
10     QObject(parent),
11     m_incoming(false)
12 {
13     m_logListModel = new QmlListModel(new PTTLogListItem);
14     m_sipData = sipData;
15     m_current_call_id = 0;
16     p_current_call_id = &m_current_call_id;
17
18     m_pttStates = OFFLINE;
19     if(m_sipData->port() != 0)
20     {
21         m_pttStates = ONLINE;
22     }
23
24     connect(this, SIGNAL(logSignal(QDateTime,PTT_Log_States,QString)), this,
           SLOT(recordLogEvent(QDateTime,PTT_Log_States,QString)));
25 }
26
27 SipData* PTTUser::getSipData()
28 {
29     return m_sipData;
30 }
31
32 void PTTUser::setState(PTT_States pttState)
33 {
34     m_pttStatesOld = m_pttStates;
35     m_pttStates = pttState;
36     emit stateChanged(m_pttStates);
37
38     switch(m_pttStates)
39     {
40     case UNKNOWN: break;
41     case OFFLINE:
42     {
43         this->getSipData()->setIpaddr("");
44         emit logSignal(QDateTime::currentDateTime(), LOG_OFFLINE, m_sipData->
               alias());
45     }
46         break;
47     case ONLINE:
48     {
49         if(m_incoming)
50         {
51             m_incoming = false;
52             emit missedCall(true);
53             emit logSignal(QDateTime::currentDateTime(), LOG_MISSED, m_sipData
                   ->alias());
54         }
55         if(m_pttStatesOld == OFFLINE)
56         {
```

```
57              emit logSignal(QDateTime::currentDateTime(), LOG_ONLINE, m_sipData
                    ->alias());
58          }
59      }
60          break;
61      case INUSE:
62      {
63          m_incoming = false;
64          if(m_pttStatesOld == CALLING)
65          {
66              emit logSignal(QDateTime::currentDateTime(), LOG_CALLING,
                    m_sipData->alias());
67          }
68          else if (m_pttStatesOld == RINGING)
69          {
70              emit logSignal(QDateTime::currentDateTime(), LOG_RINGING,
                    m_sipData->alias());
71          }
72      }
73          break;
74      case RINGING:
75      {
76           m_incoming = true;
77      }
78          break;
79      case CALLING:break;
80
81      default: break;
82      }
83  }
84
85  PTT_States PTTUser::getState()
86  {
87      return m_pttStates;
88  }
89
90  void PTTUser::setSIPState(int sipState)
91  {
92      switch(sipState)
93      {
94      case STATE_NULL:
95          setState(UNKNOWN);
96          break;
97      case STATE_CALLING:
98          setState(CALLING);
99          break;
100     case STATE_INCOMING:
101         setState(RINGING);
102         break;
103     case STATE_EARLY: break;
104     case STATE_CONNECTING: break;
105     case STATE_CONFIRMED:
106         setState(INUSE);
107         break;
108     case STATE_DISCONNECTED:
109         setState(ONLINE);
110         break;
```

```
111     default: break;
112     }
113 }
114
115
116
117 void PTTUser::operate()
118 {
119     qDebug() << __FUNCTION__ << getSipData()->name();
120
121     switch(m_pttStates)
122     {
123     case UNKNOWN: break;
124     case OFFLINE: break;
125
126     case ONLINE:
127     {
128         emit call_make(QString("%1:#%2").arg(PROTOCOL).arg(m_sipData->name()),
                p_current_call_id);
129         //this->setState(INUSE);
130     }
131         break;
132
133     case INUSE:
134     {
135         int id = *p_current_call_id;
136         emit call_hangup(id,200);
137     }
138         break;
139     case RINGING:
140         emit call_answer(*p_current_call_id,200);
141         break;
142     case CALLING:
143     {
144         int id = *p_current_call_id;
145         emit call_hangup(id,200);
146     }
147         break;
148
149     default: break;
150     }
151
152 }
153
154 void PTTUser::incommung_call(int call_id)
155 {
156     switch(m_pttStates)
157     {
158     case UNKNOWN: break;
159     case OFFLINE: break;
160
161     case ONLINE:
162     {
163         this->setState(RINGING);
164         *p_current_call_id = call_id;
165     }
166         break;
```

```
167
168     case INUSE:
169     {
170         //emit call_hangup(*p_current_call_id,200);
171         //*p_current_call_id = 0;
172     }
173         break;
174     case RINGING:
175         //this->setState(RINGING);
176         break;
177
178     default: break;
179     }
180
181 }
182
183 void PTTUser::recordLogEvent(QDateTime now, PTT_Log_States pttState, QString
        alias)
184 {
185     m_logListModel->prependRow(new PTTLogListItem(now, pttState, alias));
186 }
187
188 QmlListModel * PTTUser::getLogListModel()
189 {
190     return m_logListModel;
191 }
```

D.5.13 qml_buttonindicator.h

```
1  #ifndef QML_BUTTONINDIKATOR_H
2  #define QML_BUTTONINDIKATOR_H
3
4  #include <QObject>
5
6  class QMLButtinIndicator : public QObject
7  {
8      Q_OBJECT
9      Q_PROPERTY(bool isIncomingON READ isIncomingON WRITE setIncomingON NOTIFY
           isIncomingONChanged)
10     Q_PROPERTY(bool isMissedON   READ isMissedON   WRITE setMissedON   NOTIFY
           isMissedONChanged)
11
12 public:
13     QMLButtinIndicator(QObject *parent = 0);
14
15     void setIncomingON(bool state);
16     void setMissedON(bool state);
17
18     bool isIncomingON();
19     bool isMissedON();
20
21 signals:
22     void isIncomingONChanged();
23     void isMissedONChanged();
24
```

```
25 private:
26     bool m_incomingON;
27     bool m_missedON;
28
29 };
30
31 #endif // QML_BUTTONINDIKATOR_H
```

D.5.14 qml_buttonindicator.cpp

```
1  #include <QDebug>
2  #include "qml_buttonindicator.h"
3
4  QMLButtinIndicator::QMLButtinIndicator(QObject *parent) :
5      QObject(parent),
6      m_incomingON(false),
7      m_missedON(false)
8  {
9  }
10
11 void QMLButtinIndicator::setIncomingON(bool state)
12 {
13     //qDebug()<< __FUNCTION__ << state;
14     if(state != m_incomingON)
15     {
16         m_incomingON = state;
17         emit isIncomingONChanged();
18     }
19 }
20
21 void QMLButtinIndicator::setMissedON(bool state)
22 {
23     if(state != m_missedON)
24     {
25         m_missedON = state;
26         emit isMissedONChanged();
27     }
28 }
29
30 bool QMLButtinIndicator::isIncomingON()
31 {
32     return m_incomingON;
33 }
34
35 bool QMLButtinIndicator::isMissedON()
36 {
37     return m_missedON;
38 }
```

D.5.15 qml_listitem.h

```
#ifndef QML_LISTITEM_H
#define QML_LISTITEM_H

#include <QDeclarativeItem>

class QmlListItem: public QDeclarativeItem
{
    Q_OBJECT

public:
    QmlListItem(QDeclarativeItem* parent = 0);
    virtual ~QmlListItem() {}
    virtual QVariant data(int role) const = 0;
    virtual QHash<int, QByteArray> roleNames() const = 0;


signals:
    void dataChanged();
};

#endif // QML_LISTITEM_H
```

D.5.16 qml_listitem.cpp

```
#include <QDeclarativeItem>
#include "qml_listitem.h"

QmlListItem::QmlListItem(QDeclarativeItem *parent) :
    QDeclarativeItem(parent)
{

}
```

D.5.17 qml_listmodel.h

```
#ifndef PTT_LISTMODEL_H
#define PTT_LISTMODEL_H

#include <QAbstractListModel>
#include <QVariant>

class QmlListItem;

class QmlListModel : public QAbstractListModel
{
    Q_OBJECT

public:
    explicit QmlListModel(QmlListItem* prototype, QObject *parent = 0);
    ~QmlListModel();
    int rowCount(const QModelIndex &parent = QModelIndex()) const;
```

```
17      QVariant data(const QModelIndex &index, int role = Qt::DisplayRole) const;
18      void appendRow(QmlListItem* item);
19      void appendRows(const QList<QmlListItem*> &items);
20      void prependRow(QmlListItem* item);
21      void insertRow(int row, QmlListItem* item);
22      bool removeRow(int row, const QModelIndex &parent = QModelIndex());
23      bool removeRows(int row, int count, const QModelIndex &parent =
            QModelIndex());
24      QmlListItem* takeRow(int row);
25      Q_INVOKABLE QmlListItem* find(int index) const;
26      //MyListItem* find(const QString &id) const;
27      QModelIndex indexFromItem( const QmlListItem* item) const;
28      void clear();
29
30  private slots:
31      void handleItemChange();
32
33  private:
34      QmlListItem* m_prototype;
35      QList<QmlListItem*> m_list;
36  };
37
38
39  #endif // PTT_LISTMODEL_H
```

D.5.18 qml_listmodel.cpp

```
1   #include "qml_listmodel.h"
2   #include "qml_listitem.h"
3
4
5   QmlListModel::QmlListModel(QmlListItem* prototype, QObject *parent) :
6       QAbstractListModel(parent), m_prototype(prototype)
7   {
8       setRoleNames(m_prototype->roleNames());
9   }
10
11  QmlListModel::~QmlListModel() {
12      delete m_prototype;
13      clear();
14  }
15
16  int QmlListModel::rowCount(const QModelIndex &parent) const
17  {
18      Q_UNUSED(parent);
19      return m_list.size();
20  }
21
22  QVariant QmlListModel::data(const QModelIndex &index, int role) const
23  {
24      if(index.row() < 0 || index.row() >= m_list.size())
25          return QVariant();
26      return m_list.at(index.row())->data(role);
27  }
28
```

```
29 void QmlListModel::appendRow(QmlListItem *item)
30 {
31     appendRows(QList<QmlListItem*>() << item);
32 }
33
34 void QmlListModel::appendRows(const QList<QmlListItem *> &items)
35 {
36     beginInsertRows(QModelIndex(), rowCount(), rowCount()+items.size()-1);
37     foreach(QmlListItem *item, items) {
38         connect(item, SIGNAL(dataChanged()), SLOT(handleItemChange()));
39         m_list.append(item);
40     }
41     endInsertRows();
42 }
43
44 void QmlListModel::prependRow(QmlListItem *item)
45 {
46     beginInsertRows(QModelIndex(),0,0);
47     connect(item, SIGNAL(dataChanged()), SLOT(handleItemChange()));
48     m_list.insert(0,item);
49     endInsertRows();
50 }
51
52
53 void QmlListModel::insertRow(int row, QmlListItem *item)
54 {
55     beginInsertRows(QModelIndex(), row, row);
56     connect(item, SIGNAL(dataChanged()), SLOT(handleItemChange()));
57     m_list.insert(row, item);
58     endInsertRows();
59 }
60
61 bool QmlListModel::removeRow(int row, const QModelIndex &parent)
62 {
63     Q_UNUSED(parent);
64     if(row < 0 || row >= m_list.size()) return false;
65     beginRemoveRows(QModelIndex(), row, row);
66     //delete m_list.takeAt(row);
67     m_list.removeAt(row);
68     endRemoveRows();
69     return true;
70 }
71
72 bool QmlListModel::removeRows(int row, int count, const QModelIndex &parent)
73 {
74     Q_UNUSED(parent);
75     if(row < 0 || (row+count) > m_list.size()) return false;
76
77     beginRemoveRows(QModelIndex(), row, row+count-1);
78     for(int i=0; i<count; ++i)
79     {
80         //delete m_list.takeAt(row);
81         m_list.removeAt(row);
82     }
83     endRemoveRows();
84     return true;
85 }
```

```
86
87 QmlListItem * QmlListModel::takeRow(int row)
88 {
89     beginRemoveRows(QModelIndex(), row, row);
90     QmlListItem* item = m_list.takeAt(row);
91     endRemoveRows();
92     return item;
93 }
94
95 QmlListItem * QmlListModel::find(int index) const
96 {
97     return m_list.at(index);
98 }
99
100 QModelIndex QmlListModel::indexFromItem(const QmlListItem *item) const
101 {
102     Q_ASSERT(item);
103     for(int row=0; row<m_list.size(); ++row) {
104         if(m_list.at(row) == item) return index(row);
105     }
106     return QModelIndex();
107 }
108
109 void QmlListModel::clear()
110 {
111     //qDeleteAll(m_list);
112     m_list.clear();
113 }
114
115 void QmlListModel::handleItemChange()
116 {
117     QmlListItem* item = static_cast<QmlListItem*>(sender());
118     QModelIndex index = indexFromItem(item);
119     if(index.isValid())
120         emit dataChanged(index, index);
121 }
```

D.6 QML Scripts

D.6.1 window.qml

```
import QtQuick 1.0
import CustomComponents 1.0
import "content"

Rectangle {
    id: window
    objectName: "window"
    width: 800;
    height: 600;
    color: "#ffffff"
    //smooth: true
    //border.color: "#f50606"

    property bool incomingON: buttonIndicator.isIncomingON
    property bool missedON: buttonIndicator.isMissedON

    Image {
        width: parent.width; height: parent.height
        //fillMode: Image.PreserveAspectFit
        smooth: true
        source: "content/tma_back_network_800_520.png"
        //source: "content/tma_back_star2_800_520.png"
    }

    Image {
        source: "content/logo_final.png"
        anchors.top: parent.top
        anchors.left: parent.left
        anchors.topMargin: 8
        anchors.leftMargin: 10
    }

    Clock {
        anchors.bottom: toptopBorder.top
        anchors.right: parent.right
    }

    Rectangle {
        id: toptopBorder
        width: parent.width +10; height: 35
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: parent.top
        anchors.topMargin: 45
        border.width: 1
        border.color: "white"
        smooth: true
        gradient:Gradient {
            GradientStop { position: 0.0; color: "#808080" }
            GradientStop { position: 1.0; color: "#333333" }
        }

        Text {
```

```
53              text: "PTT Dispatcher"
54              //text: dispItem.name
55              anchors.fill: parent
56              horizontalAlignment: Text.AlignHCenter
57              font.bold: true;
58              font.pointSize: 25
59              color: "white"
60              style: Text.Raised
61              styleColor: "black"
62          }
63
64          TextButton{
65              text: "EXIT"
66              anchors.verticalCenter: parent.verticalCenter
67              anchors.left: parent.left
68              anchors.leftMargin: 20
69              onClicked: mainWindow.exit_app()
70              pointSize: 9
71          }
72      }
73
74      Rectangle {
75          id: topBorder
76          width: parent.width +10; height: 20
77          anchors.horizontalCenter: parent.horizontalCenter
78          anchors.top: toptopBorder.bottom
79          anchors.topMargin: 5
80          border.width: 1
81          border.color: "white"
82          smooth: true
83          gradient:Gradient {
84              GradientStop { position: 0.0; color: "#808080" }
85              GradientStop { position: 1.0; color: "#333333" }
86          }
87
88          Text {
89              text: "| "+dispItem.alias+" | "+dispItem.name+" |"
90              anchors.fill: parent
91              horizontalAlignment: Text.AlignHCenter
92              font.bold: true;
93              font.pointSize: 10
94              color: "white"
95              style: Text.Raised
96              styleColor: "black"
97          }
98      }
99
100     Column{
101         id: column1
102         //anchors.fill: parent
103         anchors.horizontalCenter: parent.horizontalCenter
104         anchors.verticalCenter: parent.verticalCenter
105         height: 345
106         width: 600
107         spacing: 5
108
109         Item{
```

```
110             id: item1
111             width: parent.width - 5 - 100
112             height: 30
113             anchors.left: parent.left
114             anchors.top: parent.top
115 
116             Rectangle{
117                 id: pttContentTop
118                 width: parent.width
119                 height: parent.height
120                 color: "black"
121                 opacity: 0.5
122                 radius: 3
123                 clip: false
124                 anchors.left: parent.left
125                 anchors.top: parent.top
126                 smooth: true
127                 property string listText: "All"
128             }
129 
130             Text {
131                 id: titleTop
132                 height: 16
133                 //height: 0
134                 text: pttContentTop.listText
135                 anchors.left: parent.left
136                 anchors.leftMargin: 25
137                 anchors.verticalCenter: parent.verticalCenter
138                 font.family: "Ubuntu"
139                 font.bold: true;
140                 font.pointSize: 12
141                 color: "white"
142                 style: Text.Raised
143                 styleColor: "black"
144             }
145         }
146 
147         Row {
148             id: row1
149             width: parent.width
150             height: parent.height - pttContentTop.height - 5
151             spacing: 5
152 
153             VisualItemModel{
154                 id: container
155 
156                 Item {
157                     id: pttListBack
158                     width: view.width; height: view.height
159                     Rectangle{
160                         anchors.fill: parent
161                         color: "black"
162                         opacity: 0.5
163                         radius: 3
164                         smooth: true
165                     }
166 
```

```
167                     PTTList{
168                         id: pttListFront
169                         anchors.fill: parent
170                         anchors.margins: 5
171                         anchors.centerIn: parent
172                         color: "transparent"
173                         smooth: true
174                         delegate: "PTTListDelegate"
175                     }
176                 }
177 
178                 Item {
179                     id: detailBack
180                     width: view.width; height: view.height
181                     Rectangle{
182                         id: pttContentBackRect
183                         anchors.fill: parent
184                         color: "black"
185                         opacity: 0.5
186                         radius: 3
187                         smooth: true
188                     }
189 
190                     PTTListItem{
191                         id: detail
192 
193                         Item{
194                             id: dItem
195                             width: pttContentBackRect.width; height:
                                    pttContentBackRect.height
196                             property real onlineOpacity : 0
197                             property real offlineOpacity : 0
198                             property real inuseOpacity : 0
199                             property real ringingOpacity : 0
200                             property real callingOpacity : 0
201 
202 
203                             Column{
204 
205                                 anchors.fill: parent
206                                 anchors.centerIn: parent
207                                 anchors.margins: 5
208                                 spacing: 5
209 
210                                 Rectangle {
211                                     id: rectangle1
212                                     //anchors.fill: pttContentBack
213                                     width: parent.width
214                                     height: 100
215                                     anchors.horizontalCenter: parent.
                                            horizontalCenter
216                                     //anchors.top: parent.top
217                                     //anchors.topMargin: 5
218                                     radius: 3
219                                     color: "dimgray"
220                                     smooth: true
221 
```

```
222
223                                 Column {
224                                     id: infoBox
225                                     anchors.fill: parent
226                                     anchors.margins: 10
227                                     spacing: 10
228
229                                     Item {
230                                         width: parent.width
231                                         height: 10
232                                         Text {
233                                             text: 'Name:'
234                                             width: parent.width
235                                             //anchors.bottom: parent.
                                                    bottom
236                                             verticalAlignment: Text.
                                                    AlignBottom
237                                             font.pointSize: 12
238                                             color: "white"
239                                             style: Text.Raised
240                                             styleColor: "black"
241                                         }
242
243                                         Text {
244                                             text: detailItem.alias
245                                             width: parent.width
246                                             horizontalAlignment: Text.
                                                    AlignRight
247                                             verticalAlignment: Text.
                                                    AlignBottom
248                                             font.bold: true;
249                                             font.pointSize: 16
250                                             color: "white"
251                                             style: Text.Raised
252                                             styleColor: "black"
253                                         }
254                                     }
255
256                                     Item {
257                                         width: parent.width
258                                         height: 10
259                                         Text {
260                                             text: 'Number:'
261                                             //anchors.bottom: parent.
                                                    bottom
262                                             verticalAlignment: Text.
                                                    AlignBottom
263                                             font.pointSize: 12
264                                             color: "white"
265                                             style: Text.Raised
266                                             styleColor: "black"
267                                         }
268
269                                         Text {
270                                             text: detailItem.name
271                                             width: parent.width
```

```
272                                     horizontalAlignment: Text.
                                            AlignRight
273                                     verticalAlignment: Text.
                                            AlignBottom
274                                     font.bold: true;
275                                     font.pointSize: 16
276                                     color: "white"
277                                     style: Text.Raised
278                                     styleColor: "black"
279                                 }
280                             }
281
282                             Item {
283                                 width: parent.width
284                                 height: 10
285                                 Text {
286                                     text: 'IP:'
287                                    //anchors.bottom: parent.bottom
288                                     verticalAlignment: Text.
                                            AlignBottom
289                                     font.pointSize: 12
290                                     color: "white"
291                                     style: Text.Raised
292                                     styleColor: "black"
293                                 }
294
295                                 Text {
296                                     text: detailItem.ip
297                                     width: parent.width
298                                     horizontalAlignment: Text.
                                            AlignRight
299                                     verticalAlignment: Text.
                                            AlignBottom
300                                     font.bold: true;
301                                     font.pointSize: 16
302                                     color: "white"
303                                     style: Text.Raised
304                                     styleColor: "black"
305                                 }
306                             }
307                         }
308
309                         MouseArea {
310                             id: butAera
311                             enabled: true
312                             anchors.fill: parent
313                             onClicked:{ view.currentIndex = 0;
                                    titleTop.text = pttContentTop.
                                    listText }
314                         }
315                     }
316
317                     Text {
318                         text: "Call Log"
319                         anchors.leftMargin: 25
320                         anchors.left: parent.left
321                         font.bold: true;
```

```
322                             font.pointSize: 9
323                             color: "white"
324                             style: Text.Raised
325                             styleColor: "black"
326                         }
327
328                         PTTLogList {
329                             height: 100
330                             width: parent.width
331                             //anchors.top: rectangle1.bottom
332                             anchors.horizontalCenter: parent.
                                    horizontalCenter
333                             //anchors.topMargin: 5
334                             radius: 3
335                             color: "dimgray"
336                             smooth: true
337                         }
338                     }
339
340                     Rectangle{
341                         id: pttDetailHeider
342                         anchors.fill: parent
343                         color: "black"
344                         opacity: 0
345                         radius: 3
346                         smooth: true
347                     }
348
349                     Item {
350                         id: bottomRow
351                         width: parent.width - 10;  height: 50
352                         anchors.horizontalCenter: parent.
                                horizontalCenter
353                         anchors.bottom: parent.bottom
354                         anchors.bottomMargin: 5
355                         //spacing: 1
356
357                         Rectangle {
358                             id: activeButton
359                             height: parent.height
360                             width: 250
361                             smooth: true
362                             radius: 10
363                             anchors.horizontalCenter: parent.
                                    horizontalCenter
364                             color: "dimgray"
365                         }
366
367                         Rectangle {
368                             id: activeButtonActive
369                             height: parent.height
370                             width: 250
371                             smooth: true
372                             radius: 10
373                             anchors.horizontalCenter: parent.
                                    horizontalCenter
374                             color: "dimgray"
```

```
375                 opacity: 0
376
377                 SequentialAnimation on color {
378                     loops: Animation.Infinite
379                     ColorAnimation { from: "green"; to: "
                            lightgreen"; duration: 200 }
380                     ColorAnimation { from: "lightgreen";
                            to: "green"; duration: 100 }
381                     ColorAnimation { from: "green"; to: "
                            green"; duration: 200 }
382                     ColorAnimation { from: "green"; to: "
                            lightgreen"; duration: 100 }
383                     ColorAnimation { from: "lightgreen";
                            to: "green"; duration: 200 }
384                     ColorAnimation { from: "green"; to: "
                            green"; duration: 200 }
385                 }
386             }
387
388             MouseArea {
389                 id: bottomRowMouseArea
390                 anchors.fill: parent
391                 onClicked: { detailItem.operate() }
392             }
393
394             Text {
395                 id: bottomRowOfflineText
396                 text: "offline"
397                 anchors.centerIn: parent
398                 opacity: dItem.offlineOpacity
399             }
400
401             Text {
402                 id: bottomRowNormalText
403                 text: "Call"
404                 anchors.centerIn: parent
405                 opacity: dItem.onlineOpacity
406             }
407
408             Text {
409                 id: bottomRowInuseText
410                 text: "Hang up"
411                 anchors.centerIn: parent
412                 opacity: dItem.inuseOpacity
413             }
414             Text {
415                 id: bottomRowAnswerText
416                 text: "Answer"
417                 anchors.centerIn: parent
418                 opacity: dItem.ringingOpacity
419             }
420             Text {
421                 id: bottomRowEndText
422                 text: "End"
423                 anchors.centerIn: parent
424                 opacity: dItem.callingOpacity
425             }
```

```
426                         }
427
428                         Rectangle {
429                             id: activeInfoBox
430                             height: 1
431                             width: 1
432                             anchors.centerIn: parent
433                             border.width: 1
434                             border.color: "white"
435                             radius: 3
436                             color: "gray"
437                             opacity: 0
438
439                             property string activeState: ""
440
441                             Text {
442                                 anchors.centerIn: parent
443                                 text: activeInfoBox.activeState
444                                 font.bold: true;
445                                 font.pointSize: 12
446                                 color: "white"
447                                 style: Text.Raised
448                                 styleColor: "black"
449
450                             }
451                         }
452
453                         state: detailItem.status
454                         states:[
455                             State{
456                                 name: "offline"
457                                 PropertyChanges { target: activeButton;
                                        color: "dimgray" }
458                                 PropertyChanges { target: dItem;
                                        offlineOpacity: 1;}
459                             },
460                             State{
461                                 name: "online"
462                                 PropertyChanges { target: activeButton;
                                        color: "green" }
463                                 PropertyChanges { target: dItem;
                                        onlineOpacity: 1;}
464                             },
465                             State{
466                                 name: "inuse"
467                                 PropertyChanges { target: activeButton;
                                        color: "red" }
468                                 PropertyChanges { target: dItem;
                                        inuseOpacity: 1;}
469                                 PropertyChanges { target: butAera; enabled
                                        : false;}
470                                 PropertyChanges { target: butColumHider;
                                        opacity: 0.8;}
471                                 PropertyChanges { target: pttDetailHeider;
                                         opacity: 0.8;}
472                                 PropertyChanges { target: butColum;
                                        buttomsEnabled: false;}
```

```
473
474                                 PropertyChanges { target: activeInfoBox;
                                        height: 80;}
475                                 PropertyChanges { target: activeInfoBox;
                                        width: 120;}
476                                 PropertyChanges { target: activeInfoBox;
                                        opacity: 1;}
477                                 PropertyChanges { target: activeInfoBox;
                                        activeState: "Active";}
478
479                             },
480                             State{
481                                 name: "ringing"
482                                 PropertyChanges { target:
                                        activeButtonActive; opacity: 1}
483                                 PropertyChanges { target: dItem;
                                        ringingOpacity: 1;}
484                             },
485                             State{
486                                 name: "calling"
487                                 PropertyChanges { target: activeButton;
                                        color: "red" }
488                                 PropertyChanges { target: dItem;
                                        callingOpacity: 1;}
489                                 PropertyChanges { target: butAera; enabled
                                        : false;}
490                                 PropertyChanges { target: butColumHider;
                                        opacity: 0.8;}
491                                 PropertyChanges { target: pttDetailHeider;
                                         opacity: 0.8;}
492                                 PropertyChanges { target: butColum;
                                        buttomsEnabled: false;}
493
494                                 PropertyChanges { target: activeInfoBox;
                                        height: 80;}
495                                 PropertyChanges { target: activeInfoBox;
                                        width: 120;}
496                                 PropertyChanges { target: activeInfoBox;
                                        opacity: 1;}
497                                 PropertyChanges { target: activeInfoBox;
                                        activeState: "Dialing";}
498
499                             }
500                         ]
501                         transitions: Transition {
502                             ParallelAnimation {
503                                 ColorAnimation { property: "color";
                                        duration: 200 }
504                                 NumberAnimation { target: pttDetailHeider;
                                         property: "opacity"; duration: 200 }
505                                 NumberAnimation { target: butColumHider;
                                        property: "opacity"; duration: 200 }
506                                 NumberAnimation { target: activeInfoBox;
                                        property: "height"; duration: 200 }
507                                 NumberAnimation { target: activeInfoBox;
                                        property: "width"; duration: 200 }
```

```
508                                     NumberAnimation { target: activeInfoBox;
                                            property: "opacity"; duration: 200 }
509                                 }
510                             }
511                         }
512                     }
513                 }
514             }
515
516             ListView{
517                 id: view
518                 height: parent.height
519                 width: parent.width -listButtons.width - 5
520                 //anchors{ fill: parent;}
521                 model: container
522                 preferredHighlightBegin: 0; preferredHighlightEnd: 0
523                 highlightRangeMode: ListView.StrictlyEnforceRange
524                 orientation: ListView.Horizontal
525                 snapMode: ListView.SnapOneItem; flickDeceleration: 2000
526                 cacheBuffer: 200
527                 clip: true
528                 interactive: false
529                 boundsBehavior: Flickable.StopAtBounds
530                 highlightMoveSpeed: 1500
531             }
532
533             Item {
534                 id: listButtons
535                 anchors.verticalCenter: parent.verticalCenter
536                 height: parent.height
537                 width: 100
538
539                 Rectangle{
540                     anchors.fill: parent
541                     color: "black"
542                     opacity: 0.5
543                     radius: 3
544                 }
545
546                 Column {
547                     id: butColum
548                     y:5
549                     width: parent.width
550                     anchors.margins: 10
551                     anchors.horizontalCenter: parent.horizontalCenter
552
553                     spacing: 5
554                     property bool buttomsEnabled: true
555
556                     TextButton{
557                         anchors.horizontalCenter: parent.horizontalCenter
558                         width: parent.width-10
559                         text: "All"
560                         onClicked: {
561                             view.currentIndex = 0
562                             mainWindow.showAll()
563                             pttContentTop.listText = text
```

```
564                         titleTop.text = pttContentTop.listText
565                     }
566                     enabled: butColum.buttomsEnabled
567                 }
568 
569                 TextButton{
570                     anchors.horizontalCenter: parent.horizontalCenter
571                     width: parent.width-10
572                     text: "Online"
573                     onClicked:{
574                         view.currentIndex = 0
575                         mainWindow.showOnline()
576                         pttContentTop.listText = text
577                         titleTop.text = pttContentTop.listText
578                     }
579                     enabled: butColum.buttomsEnabled
580                 }
581 
582                 TextButton{
583                     id: incBut
584                     anchors.horizontalCenter: parent.horizontalCenter
585                     width: parent.width-10
586                     text: "Incoming"
587                     onClicked:{
588                         view.currentIndex = 0
589                         mainWindow.showIncoming()
590                         pttContentTop.listText = text
591                         titleTop.text = pttContentTop.listText
592                     }
593                     active: incomingON
594                     activeColor: "green"
595                     enabled: butColum.buttomsEnabled
596                 }
597 
598                 TextButton{
599                     anchors.horizontalCenter: parent.horizontalCenter
600                     width: parent.width-10
601                     text: "Missed"
602                     onClicked: {
603                         view.currentIndex = 0
604                         mainWindow.showMissed()
605                         pttContentTop.listText = text
606                         titleTop.text = pttContentTop.listText
607                     }
608                     active: missedON
609                     activeColor: "yellow"
610                     enabled: butColum.buttomsEnabled
611                 }
612             }
613 
614             Rectangle{
615                 id: butColumHider
616                 anchors.fill: parent
617                 color: "black"
618                 opacity: 0
619                 radius: 3
620             }
```

```
621             }
622         }
623     }// Row
624 }
```

D.6.2 PTTList.qml

```
1  import QtQuick 1.0
2
3  Rectangle {
4      id: pttListBase
5      objectName: "pttListBase"
6      property string delegate: ""
7      property bool interactive: true
8
9      signal indexChanged(int index)
10
11     ListView {
12         id: listView
13         objectName: "listView"
14         anchors.fill: parent
15         model: listModel
16         delegate: PTTListDelegate{}
17
18         interactive: pttListBase.interactive
19
20         width: parent.width
21         height: parent.height
22         clip: true
23         smooth: true
24         cacheBuffer: 10
25         currentIndex: -1
26
27         boundsBehavior: Flickable.DragOverBounds
28
29         ScrollBar{
30             flickable: listView
31             vertical: true
32         }
33
34         function updateIndex(indexx) {
35             currentIndex = indexx
36             pttListBase.indexChanged(currentIndex)
37         }
38     }
39 }
```

D.6.3 PTTListDelegate.qml

```
1  import QtQuick 1.0
2
3  Item {
```

```
4       id: pttItem
5
6       width: listView.width
7       height: 80
8
9       property real onlineOpacity : 1
10      property real offlineOpacity : 0.5
11      property real incomingActiveOpacity : 0
12      property string action: ""
13      property bool missedON: object.missedCall
14
15      Component.onCompleted: {
16          //state = status
17      }
18
19      Connections{
20          target: object//listModel.find(index)
21          //onStatusChanged: state = status
22          onStatusChanged: console.log('--------------------> STATUS: '+status)
23      }
24
25      Rectangle {
26          id: butBg
27          x: 1; y: 1; width: parent.width - x*2; height: parent.height - y*2
28          radius: 3
29          gradient: Gradient {
30              GradientStop {
31                  position: 0.0
32                  color: "white"
33              }
34
35              GradientStop {
36                  position: 0.8
37                  color: "lightgray"
38              }
39          }
40          smooth: true
41      }
42
43      MouseArea {
44          anchors.fill: parent
45          onClicked:{
46              listView.updateIndex(index)
47              view.currentIndex = 1
48              titleTop.text = "Detail"
49
50          }
51      }
52
53      Row {
54          id: topLayoutLeft
55          anchors.verticalCenter: parent.verticalCenter
56          anchors.left: parent.left
57          anchors.margins: 20
58          spacing: 10
59          Column {
60              //width: parent.width - 0; height: butBg.height - 0
```

```
61              anchors.verticalCenter: parent.verticalCenter
62              spacing: 5
63
64              Text {
65                  text: alias
66                  font.bold: true;
67                  font.pointSize: 16
68                  opacity: pttItem.onlineOpacity
69                  color: "black"
70                  style: Text.Raised
71                  styleColor: "gray"
72              }
73          }
74      }
75
76      Row {
77          id: topLayoutRight
78          anchors.verticalCenter: parent.verticalCenter
79          anchors.right: parent.right
80          anchors.margins: 10
81          spacing: 5
82
83          Rectangle{
84              anchors.verticalCenter: parent.verticalCenter
85              height: butBg.height - 4
86              width: 2
87              color: "black"
88              opacity: 0.5
89          }
90
91          Rectangle {
92              anchors.verticalCenter: parent.verticalCenter
93              height: 15
94              width: 15
95              color: "black"
96
97              Rectangle {
98                  id: missedLight
99                  anchors.fill: parent
100                 anchors.margins: 2
101                 color: "gray"
102
103                 states:State{
104                     name: "active"
105                     when: pttItem.missedON
106                     PropertyChanges{target: missedLight; color: "yellow"}
107                 }
108             }
109         }
110
111         Rectangle {
112             anchors.verticalCenter: parent.verticalCenter
113             height: 15
114             width: 15
115             color: "black"
116
117             Rectangle {
```

```
118                 id: incomingLight
119                 anchors.fill: parent
120                 anchors.margins: 2
121                 color: "gray"
122             }
123 
124             Rectangle {
125                 id: incomingLightActive
126                 anchors.fill: parent
127                 anchors.margins: 2
128                 color: "gray"
129                 opacity: incomingActiveOpacity
130 
131                 SequentialAnimation on color{
132                     loops: Animation.Infinite
133                     ColorAnimation { from: "green"; to: "lightgreen"; duration
                        : 200 }
134                     ColorAnimation { from: "lightgreen"; to: "green"; duration
                        : 100 }
135                     ColorAnimation { from: "green"; to: "green"; duration: 200
                         }
136                     ColorAnimation { from: "green"; to: "lightgreen"; duration
                        : 100 }
137                     ColorAnimation { from: "lightgreen"; to: "green"; duration
                        : 200 }
138                     ColorAnimation { from: "green"; to: "green"; duration: 200
                         }
139                 }
140             }
141         }
142     }
143 
144     state: status
145     states:[
146         State {
147             name: "offline"
148             PropertyChanges { target: missedLight; color: "gray" }
149             PropertyChanges { target: incomingLight; color: "gray" }
150             //PropertyChanges { target: pttItem; offlineOpacity: 0.5;}
151             PropertyChanges { target: pttItem; onlineOpacity: offlineOpacity;}
152             PropertyChanges { target: pttItem; incomingActiveOpacity: 0;}
153         },
154         State {
155             name: "online"
156             //PropertyChanges { target: missedLight; color: "gray" }
157             //PropertyChanges { target: incomingLight; color: "gray" }
158             //PropertyChanges { target: pttItem; offlineOpacity: 0;}
159             PropertyChanges { target: pttItem; onlineOpacity: 1;}
160             PropertyChanges { target: pttItem; incomingActiveOpacity: 0;}
161         },
162         State {
163             name: "ringing"
164             //PropertyChanges { target: missedLight; color: "gray" }
165             //PropertyChanges { target: incomingLight; color: "green" }
166             //PropertyChanges { target: pttItem; offlineOpacity: 0;}
167             PropertyChanges { target: pttItem; onlineOpacity: 1;}
168             PropertyChanges { target: pttItem; incomingActiveOpacity: 1;}
```

```
169         }
170     ]
171
172     transitions:Transition {
173         ParallelAnimation {
174             ColorAnimation { property: "color"; duration: 500 }
175             //NumberAnimation { duration: 300; properties: "ringingOpacity,
                    onlineOpacity,x,contentY,height,width" }
176         }
177     }
178
179
180     ListView.onAdd: SequentialAnimation {
181         //PropertyAction { target: pttItem; property: "onlineOpacity"; value:
                0 }
182         PropertyAction { target: pttItem; property: "state"; value: status }
183         PropertyAction { target: pttItem; property: "height"; value: 0 }
184         NumberAnimation { target: pttItem; property: "height"; to: 80;
                duration: 250; easing.type: Easing.InOutQuad }
185         //NumberAnimation { target: pttItem; property: "onlineOpacity"; to: 1;
                 duration: 150; easing.type: Easing.InOutQuad }
186     }
187
188     ListView.onRemove: SequentialAnimation {
189         PropertyAction { target: pttItem; property: "ListView.delayRemove";
                value: true }
190
191         NumberAnimation { target: pttItem; property: "onlineOpacity"; to: 0;
                duration: 150; easing.type: Easing.InOutQuad }
192         NumberAnimation { target: pttItem; property: "height"; to: 0; duration
                : 350; easing.type: Easing.InOutQuad }
193
194         PropertyAction { target: pttItem; property: "ListView.delayRemove";
                value: false }
195     }
196 }
```

D.6.4 PTTLogList.qml

```
1  import QtQuick 1.0
2
3  Rectangle {
4      id: pttLogListBase
5
6      ListView {
7          id: loglistView
8          objectName: "loglistView"
9          anchors.centerIn: parent
10         model: loglistModel
11         delegate: PTTLogListDelegate{}
12
13         //interactive: pttListBase.interactive
14
15         width: parent.width
16         height: parent.height - 10
```

```
        clip: true
        smooth: true
        cacheBuffer: 10
        currentIndex: -1

        boundsBehavior: Flickable.DragOverBounds

        ScrollBar{
            flickable: loglistView
            vertical: true
        }
    }
}
```

D.6.5 PTTLogListDelegate.qml

```
import QtQuick 1.0


Item {
    id: pttLogListItem
    width: parent.width
    height: 24
    clip: true

    Rectangle{
        width: parent.width-10
        height: 20
        radius: 3
        anchors.centerIn: parent
        color: "gray"
    }

    function getDate() {
        return Qt.formatDateTime(time, "yyyy-MM-dd [hh:mm.ss]");
    }

    Text {
        text: getDate()
        width: parent.width - (10*2)
        anchors.centerIn: parent
        font.bold: true;
        font.pointSize: 10
        color: "white"
        style: Text.Raised
        styleColor: "black"
    }

    Text {
        text: alias
        horizontalAlignment: Text.AlignHCenter
        width: parent.width - (10*2)
        anchors.centerIn: parent
        font.bold: true;
        font.pointSize: 10
```

```
40          color: "white"
41          style: Text.Raised
42          styleColor: "black"
43      }
44
45      Text {
46          text: event
47          horizontalAlignment: Text.AlignRight
48          width: parent.width - (10*2)
49          anchors.centerIn: parent
50          font.bold: true;
51          font.pointSize: 10
52          color: "white"
53          style: Text.Raised
54          styleColor: "black"
55      }
56  }
```

D.6.6 ScrollBar.qml

```
1   import QtQuick 1.0
2
3   Rectangle {
4       property variant flickable
5       property bool vertical: true
6       property bool hideScrollBarsWhenStopped: true
7       property int scrollbarWidth: 15
8
9       color: "black"
10      radius: vertical ? width/2 : height/2
11
12      function sbOpacity()
13      {
14          if (!hideScrollBarsWhenStopped) {
15              return 0.5;
16          }
17
18          return (flickable.flicking || flickable.moving) ? (vertical ? (height
                >= parent.height ? 0 : 0.5) : (width >= parent.width ? 0 : 0.5)) :
                 0;
19      }
20
21      opacity: sbOpacity()
22
23      width: vertical ? scrollbarWidth : flickable.visibleArea.widthRatio *
            parent.width
24      height: vertical ? flickable.visibleArea.heightRatio * parent.height :
            scrollbarWidth
25      x: vertical ? parent.width - width : flickable.visibleArea.xPosition *
            parent.width
26      y: vertical ? flickable.visibleArea.yPosition * parent.height : parent.
            height - height
27
28      Behavior on opacity { NumberAnimation { duration: 300 }}
29  }
```

D.6.7 TextButton.qml

```
import QtQuick 1.0

Rectangle {
    id: container

    property alias text: label.text
    property alias pointSize: label.font.pointSize
    property bool active: false
    property string activeColor: "red"

    signal clicked

    width: label.width + 6;
    height: label.height + 20
    smooth: true
    radius: 10

    gradient: Gradient {
        GradientStop { id: gradientStop; position: 0.0; color: palette.light }
        GradientStop { position: 1.0; color: palette.button }
    }

    SystemPalette { id: palette }

    MouseArea {
        id: mouseArea
        anchors.fill: parent
        onClicked: { container.clicked() }
    }

    Text {
        id: label
        anchors.centerIn: parent
    }

    states:[
        State {
            name: "pressed"
            when: mouseArea.pressed
            PropertyChanges { target: gradientStop; color: palette.dark }
        },
        State {
            name: "active"
            when: container.active
            PropertyChanges { target: gradientStop; color: activeColor}
        }
    ]
}
```

D.6.8 Clock.qml

```
import QtQuick 1.0

Item {
    id: clock
    width: 200; height: 25
    property string time: ""

    function timeChanged() {
        var date = new Date;
        return Qt.formatDateTime(date, "ddd, dd MMM. yyyy hh:mm:ss");
    }

    Timer {
        interval: 100; running: true; repeat: true;
        onTriggered: time = clock.timeChanged()
    }

    Text {
        id: cityLabel
        y: 200; anchors.horizontalCenter: parent.horizontalCenter
        text: time
        color: "white"
        font.bold: true; font.pixelSize: 14
        style: Text.Raised; styleColor: "black"
        anchors.fill: parent
        horizontalAlignment: Text.AlignLeft
    }
}
```