uMSG

Dokumentasjon

Effektiv kode med C og C++ Prosjektoppgave våren 2014
Fakultet for teknologi og design, Høgskolen i Oslo og Akershus

Tittel: uMsg

Gruppemedlemmer:
Jonas Moltzau (s176250)
Martin W. Løkkeberg (s176251)

Sammendrag

Programmet er et peer-to-peer chatte-program i likhet med msn/skype/irc, etc., men serverløst. Så lenge to maskiner har programmet kjørende kan man sende meldinger via Internett kun ved å skrive inn IP'en, eller man kan sende via LAN hvor man kan søke etter andre klienter med et «rop og svar» system (sammenliknbart med ARP protokollens virkemåte). Programmet fungerer i skrivende stund både på LAN og via Internett, men Internett-kommunikasjon krever at brukerne åpner port 7755 via «Port Forwarding».

Løsningen

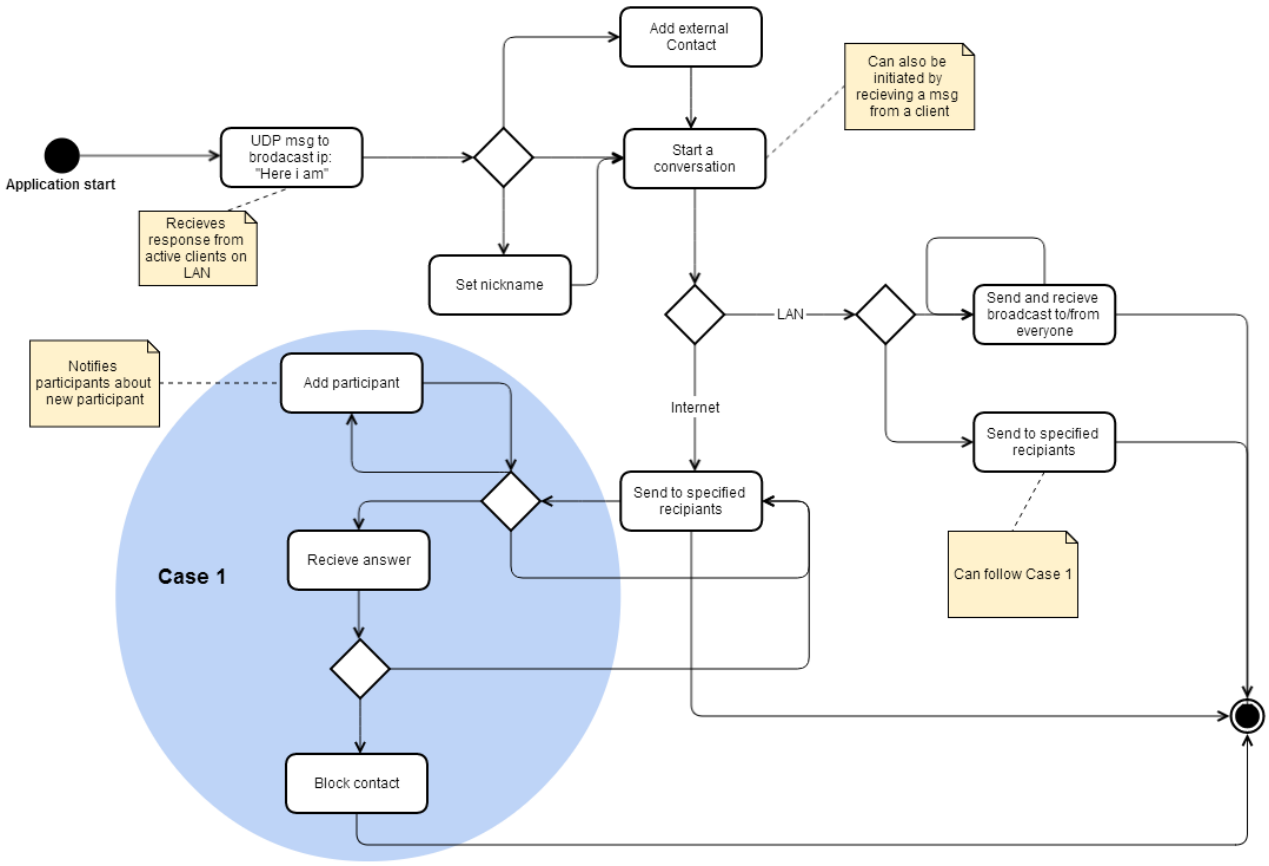
Beskrivelse av funksjonalitet

Her finner du en kortfattet punktliste over all den viktigste funksjonaliteten i programmet. Her er tanken at denne listen kan brukes for å få en enkel oversikt over alt du kan gjøre og teste. Vi har laget et aktivitetsdiagram som beskriver den normale gangen i programmet. Dette finner du under listen, vi har også lagt det ved som .png i GIT.

Man kan:

- starte en samtale ved å dobbeltklikke på en kontakt
- legge til en kontakt ved å skrive inn IPen til en klient og gi den et navn, disse lagres i en fil og huskes av programmet.
- sende meldinger med enten én klient eller flere klienter, da ved å legge til klienter i allerede eksisterende samtaler.
- snakke med alle lokale klienter som har programmet kjørende ved å bruke «broadcast».
- sette et kallenavn som vises når du sender en melding via File-menyen, dette huskes av programmet.
- blokke klienter ved å høyre-klikke på disse, dette skjer per «session» og lagres ikke
- fjerne kontakter ved å høyreklikke på disse
- minimalisere programmet til «tray» og hente det opp igjen derfra. Når programmet er minimalisert får man ballong-notifikasjoner hvis noen sender deg en melding (disse kan skrus av i File-menyen).
- sende smileys ved å dobbelt-klikke på smiley-ikonene til høyre i programmet. Disse setter da inn en <img> tag som brukes for å vise denne på andre klienter.
- se hvem du er i en samtale med i «Participants» menyen til venstre i programmet. Her dukker alle medlemmer i den nåværende samtalen opp.
- viske ut all tekst i en samtale ved å gå i Edit-menyen.

- Skanne LANet etter klienter med programmet kjørende. Dette finner man i File-menyen. Skal strengt tatt ikke være nødvendig da programmet gjør dette automatisk ved oppstart.



Om Programstrukturen

For prosjektet vårt har vi valgt å bruke Qt Creator som utviklingsverktøy og bruke deler av Qts rammeverk. Nesten alle klassene vi har laget utvider baseklassen QObject slik at vi kunne ta nytte av en del av funksjonalitet dette medfører, f.eks signal og slots.

Vi har strukturert programmet etter MVC modellen ved å separere visning(MainWindow), logikk(Controller), modell (Conversation, Peer) og data aksess (Connection, UdpConnection).

Som man kan se i bildet under utgjør vindusklassen vår, *MainWindow*, view'et, og her opprettes all GUI og logikken i denne er i all hovedsak kun relatert til brukergrensesnittet.
Klassen "*Controller"* tar imot forespørsler fra vinduet om å utføre ulike handlinger samt gir beskjed til vinduet om ny data som skal vises. For å fasilitere kommunikasjonen bruker Controlleren klassen UdpConnection (via Connection peker) som håndterer all sending og mottak av data gjennom nettverket.

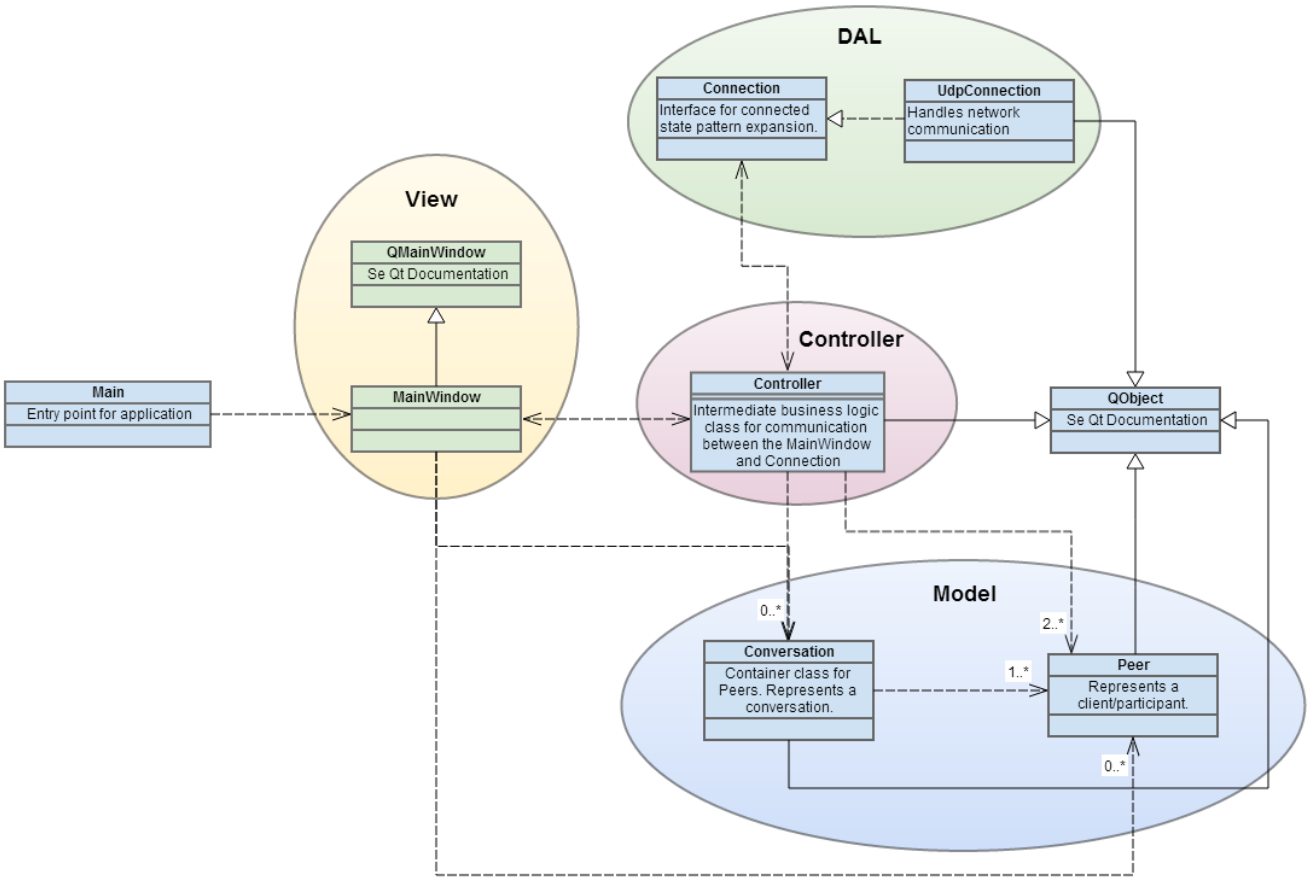
Et eksempel på en slik interaksjon, kan være:
- Bruker skriver inn en melding og trykker "Enter".

- Vinduet fanger opp "Enter"-tastetrykket, og ber kontrolleren om å sende gitte melding.

- *Controlleren* henter den nye meldingen, formaterer den korrekt, slik at den blir riktig oppfattet av andre klienter, f.eks "*uMsg|{FLAG}|{my_ip}|{my_name}|{convo_id}|{super secret message}".* Deretter itererer den gjennom samtalens(Conversation) mottagere(Peers) og gir beskjed til

"*Connection*" om å sende melding x til mottager y, evt. melding x til broadcast IP.

- UdpConncetion tar imot en string, konverterer den til en byte array og sender på forhåndsbestemt port til angitt IP.



Samtaler

Samtaler er lagret som "Conversation" objekter som inneholder mottagere i form av "Peer" objekter, disse to klassene utgjør modellen. Med unntak av den første tab'en i vinduet, korresponderer alle de ulike tab'sene som opprettes i hovedvinduet med Conversation-objekter i Controllerens samtaleliste, på denne måten vet både kontrolleren og vinduet hvilken samtale som er den gjeldene basert på valgt tab.

Connection

UdpConnection implementerer den abstrakte klassen *Connection* som overlater til implementerende klasser hvordan den vil sende meldinger, osv. UdpConnection er laget som en Singleton fordi den binder seg til en port og i essens låser denne resurssen, skal man da bruke klassen må man derfor hente den ene instansen som er opprettet, det er heller ikke nødvendig å ha flere instanser av den. *UdpConnection* er på dette tidspunkt den eneste klassen som implementerer "Connection" klassen, men Connection kan senere brukes til f.eks "TcpConnection" eller "MockConnection" til enhetstesting.

Valg og utfordringer

I forhold til minimumskrav

I forhold til kravene for minimumsfunksjonalitet har vi implementert nærmest alt vi satt oss fore. Det eneste punktet som ikke er fullt ut implementert er punktet som sier: «** Velge om man skal kommunisere via UDP eller TCP».* Her valgte vi å heller fokusere på UDP og dermed gi oss selv mer tid til å implementere funksjonalitet. Vi så det slik at å kunne velge mellom TCP og UDP for pakke-sending var mindre viktig enn mer avansert brukerfunksjonalitet, internettkommunikasjon og generell funksjonalitet. Derfor kan man ikke i skrivende stund bruke TCP i programmet, men vi har lagt til rette for det ved å lage baseklassen *Connection* som nå implementeres av kun UdpConnection, men som ved senere ekspansjon enkelt kan utvides med en subklasse for TCP. Her er tanken at denne delen av programmet vil bruker «State pattern» når TCP evt. implementeres. Vi valgte også å forgå TCP fordi det strengt tatt ikke er nødvendig for at programmet skal kjøre tilfredsstillende. UDP er ikke en spesielt robust protokoll, men når programmet i all hovedsak skal brukes på ekstremt korte avstander over mindre LAN nettverk er ikke dette et stort problem. Det ville selvsagt vært fordeler med TCP når vi nå også har implementert Internett-kommunikasjon, men fordi dette må gjøres via «Port Forwarding» eller via en «Hole Punching Service» ser vi ikke på Internett-brukere som hovedmålgruppe for dette programmet.

Ang. Internett-kommunikasjon

Vi lette lenge etter en mulighet for å fasilitere P2P kommunikasjon over Internett. Vi fant tidlig ut at routere, pga. NAT, ville blokke alle innkommende signaler som ikke var forventet til en klient. Dermed kunne vi ikke «bare» sende meldinger av gårde og forvente at disse kom frem. Vi studerte andre P2P løsninger som «Skype» og «BitTorrent» og fant ut at det finnes to basismodeller for å løse dette problemet. Disse er, som nevnt, enten «Hole Punshing Service» (HPS) eller «Port Forwarding». Fordi det viktigste kravet for oss var at programmet skulle kjøre 100% serverløst var HPS uaktuelt da det krever en server. Valget falt da på «Port Forwarding», denne metoden fungerer utmerket på vanlige hjemme-LAN hvor brukeren kan «Port Forwarde» ved å konfigurere sin router, men på større nettverk eller WAN (slik som HIOA), vil ikke dette fungere fordi man ikke kan konfigurere disse routerne selv. Slik innså vi at for å kommunisere over internett må man være en «ekspertbruker» og ha tilganger man gjerne bare har hjemme på sitt eget utsyr. Vi så på flere muligheter for å løse dette, men endte til slutt opp med konklusjonen at alt vi kom på ville ta for lang tid å gjennomføre innenfor dette prosjektets rammer. Dermed er dette programmet for oss primært et LAN-program med ekstrafunksjonalitet ut mot nett og denne funksjonaliteten kan dermed være ustabil og brukes på eget ansvar.

Testrapport

Vi har ikke eksportert programmet som en exe fil eller liknende, så det må kjøres som et Qt-prosjekt. Her kan du se hva vi har testet programmet på:

- Linux Mint med Qt-Creator med «Vanilla» installasjon. Her funger alle funksjoner som antatt. Var mye herk å installere Qt, men vårt program fungerte med en gang.
- Windows 7. Her testet vi på 5 forskjellige maskiner, alle med Qt-Creator «Vanilla» installasjon, her oppførte programmet seg likt på alle maskiner.
- Programmet er ikke testet på VM.

Installasjonsveiledning

1. For å få brukt smileys må man kopiere mappen *images* fra kildemappen til *build* mappen.
2. På Windows måtte vi gi tilgang i brannmuren, dette trengte vi ikke på Linux Mint.
3. Ellers skal det fungere bare å bygge og kjøre prosjektet via Qt-Creator

Konklusjon

Vi er godt fornøyde med resultatet av oppgaven og alle funksjonene fungerer tilfredsstillende på de maskinene vi har testet på (se testrapport). Det er vel alltid slik i programmering at man gjerne skulle

ha implementert mer funksjonalitet, men i skopet av denne oppgaven mener vi løsningen er tilfredsstillende. Skulle vi derimot ha kommet med et ønske om hva vi gjerne skulle ha hatt med ville det vært; TCP funksjonalitet og en smartere måte å fasilitere Internett-kommunikasjon på, slik som mulighet for å sette opp sin egen HPS bare ved å legge inn en IP og installere et script vi har laget.

Håper du synes oppgaven ser bra ut, så ser vi frem mot presentasjonen før sommeren.