Diplomoppgave i Informatikk

Kontekstsensitiv tilpasning av interaksjonsmodi mot mobile terminaler.

André Henriksen

Tromsø, 2003
For Telenor FoU

Det matematisk-Naturvitenskapelige Fakultet
Institutt for Informatikk
Universitetet i Tromsø, N-9037 Tromsø

Kontekstsensitiv tilpasning av interaksjonsmodi mot mobile terminaler.

André Henriksen

Tromsø, 2003
For Telenor FoU

Universitetet i Tromsø
2003

Forord

Denne oppgaven representerer siste del av min sivilingeniørgrad i informatikk ved Universitetet i Tromsø. Oppgaven ble utført i tilknytning til prosjektet ”Personalisering og kontekst”[34] ved Telenor FoU i Tromsø. Oppgaven ble skrevet i perioden august – desember 2003.

Formålet med oppgaven har vært å undersøke muligheten for å kunne tilpasse interaksjonsmodi mot mobile terminaler for å skreddersy informasjon til kunder av et transportselskap. Kontekstinformasjon brukes for å oppnå skreddersydd informasjon. Det er implementert en demo-applikasjon for å illustrere mulighetene.

Takk

Jeg vil gjerne takke Wiggo Finnset ved Telenor FoU som har vært min veileder under hele prosessen, og som har kommet med hjelp og innspill under utvikling av oppgaven.

Videre vil jeg takke min samboer Ingunn Nilsen for all støtten, samt alle medstudenter som har hjulpet meg underveis i oppgaven.

Sammendrag

Denne oppgaven ser på mulighetene for å bruke kontekstinformasjon for å kunne tilpasse interaksjonsmodi mot mobile terminaler, spesielt mobiltelefoner. Hensikten er å illustrere hvordan brukere av et transportselskap automatisk kan motta beskjeder om forsinkelser på bussavganger han/hun er interessert i, samt andre beskjeder fra transportselskapet. Brukere skal altså slippe å kjenne til vanlige rutetider. På denne måten blir tjenesten skreddersydd for hver enkelt bruker. Kontekst som identitiet, lokasjon, situasjon og tid brukes for å avgjøre hvilket interaksjonsmodus som skal benyttes. Interaksjonsmodi som ses på i denne oppgaven er SMS og J2ME.

Transportsystemet spesifiseres og designes i sin helhet, men kun en demonstrator utvikles. Demonstratoren er utviklet for å fungere på en Sony Ericsson P800, men andre telefoner vil også fungere. Demonstratoren utvikles for å vise at det er mulig å bruke kontekstinformasjon for å avgjøre hvilket interaksjonsmodus som skal benyttes. I demonstratoren er kun J2ME brukt som interaksjonsmodus, mens SMS som interaksjonsmodus ikke er implementert.

Innholdsfortegnelse

Figurliste

Tabelliste

1 Innledning

Dette kapittelet handler om hvorfor og hvordan denne oppgaven blir skrevet. Eksemplene i dette kapittlet er generelle eksempler og henviser ikke til situasjoner og modi som faktisk blir brukt i denne oppgaven.

1.1 Bakgrunn

Utgangspunktet for denne oppgaven er at flere og flere får gjennom sine mobiltelefoner tilgang til informasjon og tjenester de ikke har hatt tilgang til før. Internett har lenge eksistert, men kun i den senere tid blitt tilgjengelig mobilt. Vi kan altså finne informasjon på nettet uansett hvor vi befinner oss. På grunn av dette forventer folk skreddersydde løsninger i de systemer de ønsker å benytte. Ved å bruke kontekstinformasjon i applikasjoner er det mulig å gi dem nettopp slike løsninger.

Kontekst brukes for på en mer effektiv måte å gi brukere den informasjon de trenger når de trenger den. Det finnes fire hovedtyper kontekst som kan brukes for å lage en personlig tjeneste, disse er lokasjon, tid, identitet og aktivitet [1]. Ved å bruke kontekstinformasjon for å kunne tilpasse interaksjonsmodi, kan man på en mer effektiv måte kommunisere mellom brukere og tjenestetilbydere.

1.2 Problemstilling

I denne oppgaven skal det brukes kontekstinformasjon for å kunne tilpasse interaksjonsmodi mot mobile terminaler, med den hensikt å spesifisere en tjeneste mellom kunder og et transportselskap, hvor kunden på en tilpasset måte får informasjon om forsinkelser i bussavganger som er interessant for han/henne.

Hovedproblemstilling: **Hvordan kan kontekstinformasjon benyttes til tilpasning av interaksjonsmodi mot mobile terminaler.**

1.3 Mål

Hovedmålet med oppgaven er å spesifisere, designe og modellere et kontekstsensitivt system for kommunikasjon mellom kunde og kollektivtransportselskap, samt å implementere en demonstrator av dette systemet. En klientdel skal implementeres på en mobil enhet ved hjelp av Java (J2ME).

Delmål for oppgaven blir å utforme brukerkrav, designe systemet, og implementere og teste en demomonstrator.

1.5 Avgrensninger

Systemet spesifiseres og designes i sin helhet, men demonstratoren som implementeres vil kun gjenspeile deler av designet. En demonapplikasjon for transportkunder lages for å kunne fungere på en Sony Ericsson P800. SMS-delen av systemet designes ikke.

1.6 Personalisering og kontekst

Denne diplomoppgaven er tilknyttet prosjektet "Personalisering og kontekst" [34]. Dette prosjektet har som mål å:

- utvikle forretningsmessig interessante demonstratorer og pilottjenester som tilbyr individuell brukertilpasning (personalisering), og som dynamisk tilpasser seg endringer i brukerens kontekst (kontekstsensitivitet) og brukerens preferanser og behov (dynamisk brukerprofil).
- utvikle kunnskap om og erfaring med koordinering og tjenesteyting ut fra bruk av kontekst informasjon der kunder og tjenestetilbydere er mobile og opererer i organisatoriske nettverk.
- utvikle kunnskap om brukerperspektiver og sosialpsykologiske faktorer knyttet til kontekstsensitive tjenester.
- lage kravspesifikasjon, arkitektur og design av en kontekstmotor som på vegne av tjenester vil samle inn, modellere og forvalte kontekst informasjon. Kontekstmotoren skal på dertil egnede måter gjøre slik informasjon tilgjengelig for tjenesteutvikling.
- bidra til utvikling av fleksible og åpne mellomvareteknologier for håndtering av informasjon om brukerens kontekst

Denne oppgaven har relevans spesielt for første mål ovenfor, i tilknytning til utvikling av en demonstrator av transporttjenester for differensiert kollektivtransport.

1.8 Teoretisk rammeverk

Kontekstsensitivitet

Kontekstinformasjon kan brukes for å skreddersy applikasjoner for hver enkelt bruker. Denne oppgaven handler om nettopp dette. Kontekstinformasjon er i denne oppgaven forstått som *tid, lokasjon, aktivitet og identitet* [1].

Se kapittel 2 for en utfyllende beskrivelse av kontekst og kontekstsensitivitet.

All informasjon som kan gjøre det enklere for kunden å benytte den offentlige transporten blir i denne oppgaven kontekstinformasjon. Denne oppgaven fokuserer blant annet på hvilken situasjon brukeren er i og hvordan systemet kan tilpasse seg

brukerens situasjon gjennom endring av interaksjonsmodus. I denne oppgaven blir derfor kontekstinformasjon:

- Hvem kunden er (identitet) – Kunden gjenkjennes gjennom telefonnummer.
- Hvor kunden befinner seg (lokasjon) – Er kunden hjemme, på jobb, ute å går.
- Hvilken situasjon kunden befinner seg i (aktivitet) – Er kunden hun på jobb eller har han/hun fri.
- Hvilken tidspunkt på døgnet/uken/året det er (tid) – Er det helg, helligdag eller hverdag.

Kontekstinformasjonen brukes i denne oppgaven som et redskap for å avgjøre om en bruker skal motta informasjon fra systemet, og hvilket interaksjonsmodus som skal brukes. Systemet bruker altså kontekst for at brukeren skal slippe å fortelle systemet hvordan det skal oppføre seg hver gang han/hun endrer situasjon.

Kontekstsensitiv tilpasning

Kontekstsensitiv tilpasning betyr her at systemet har evne til å endre sin oppførsel etter hvilken kontekst brukeren er i. Når en brukers kontekst endres må systemet avgjøre hvordan det skal reagere på dette. Denne tilpasningen skal skje automatisk slik at brukeren selv skal slippe å tenke på dette. Når en bruker endrer lokasjon eller situasjon er det opp til systemet og vite hvordan det skal reagere. Det samme gjelder når tiden endrer seg.

Noen ganger endrer brukeren sin egen situasjon (f.eks. han har ferie), mens andre ganger skjønner automatisk systemet at brukeren har endret situasjon, og da skal dette også endres i systemet.

Eksempler på kontekstsensitiv tilpasning der situasjon endres:

- En bruker kjører ut av byen og vil etterhvert ikke lengre befinne seg i Tromsø. Systemet skjønner dette og brukeren settes over i situasjon *borte*. Brukeren vil da ikke motta informasjon fra systemet.
- En brukeren beveger seg fra hjemmet sitt til bussholdeplassen. Brukerens situasjon endres til *ute,* og tilbake til *inne* når brukeren kommer inn på bussen. Hvis brukeren mottar en beskjed mens han/hun er ute, økes lyden for å øke merkbarheten.

Grunnen til at vi ønsker å vite hvilken situasjon en bruker er i, er at interaksjonsmodus kan endres når en brukers situasjon endres.

Interaksjonsmodus

Interaksjonsmodus er ulike metoder systemet kan kommunisere med kunden på. Tilgjengelige modi i denne oppgaven er *SMS* og via *J2ME-applikasjon*. Kommunikasjon gjennom SMS gir muligheten for bruk av tekst og begrenset lyd. Kommunikasjon gjennom *J2ME-applikasjonen* gir muligheten for bruk av rikere tekst, grafikk og lyd. Ulike typer interaksjonsmodi skaper ulik forståelse hos brukeren. Noen modus passer best i enkelte situasjoner mens noen passer bedre i andre situasjoner.

Eksempler på bruk av interaksjonsmodi:

- *Bruk av grafikk* (sammen med tekst) – En holdeplass endrer lokasjon midlertidig på grunn av gravearbeid. Kunder som benytter denne holdeplassen får tilsendt et kart som viser hvor den nye og den gamle holdeplassens lokasjon er. Dette ville vært vanskeligere å beskrive kun ved bruk av tekst.
- *Bruk av lyd* – Telefonen lager en eller annen lyd når den mottar info fra kommunikasjonssystemet. Dette for at brukeren skal merke at han/hun har mottatt ny informasjon.
- *Bruk av vibrasjon* – Telefonen vibrerer når ny informasjon er mottatt. På den måten økes sannsynligheten for at en bruker merker at han/hun har mottatt ny informasjon.
- *Bruk av tekst* – Kunden mottar informasjon via SMS som består kun av tekst. Eller kunden mottar informasjon i en *J2ME-applikasjon*, som vanligvis består av kun tekst.

Eksempler der interaksjonsmodi endres på grunn av tilgjengelig kontekst:

- *Fra vanlig lyd til vibrasjon og sterk lyd* – En brukeren beveger seg mellom hjemmet sitt og holdeplassen. Systemet legger merke til at brukeren beveger seg utendørs og mobiltelefonen starter å vibrere når den får oppdateringer. I tillegg lager den mer lyd enn den gjorde innendørs.
- *Fra vibrasjon og sterk lyd til vanlig lyd* - En brukeren går fra holdeplassen og hjem. Når han er kommet inn i huset oppdager systemet at han er innendørs og begynner å bruke vanlig lyd som interaksjonsmodus.
- *Fra vibrasjon og sterk lyd til vanlig lyd* – En bruker sitter på bussen og beveger seg mellom holdeplasser. Systemet skjønner at han er inni i et kjøretøy og vibrasjon er ikke lengre nødvendig. Systemet bruker vanlig lyd som interaksjonsmodus.
- *Fra vanlig lyd til stille* – En bruker sitter i et møte og ønsker ikke å bli forstyrret. Systemet sender ikke beskjeder, eller det sendes ut beskjeder, men lager ingen lyd når beskjeden mottas.
- *Fra J2ME til SMS* – En bruker har J2ME-applikasjonen på for å endre sin kommunikasjonskanal. Mens han er logget på sendes en beskjed ut om at bussen han/hun venter på er forsinket. Kunden slår av kundeapplikasjonen. Dette fanges opp av transportsystemet og beskjeden sendes til denne brukeren via SMS.

Ulike terminaler har ulike muligheter for mottak av informasjon. Systemet som skal utvikles gjennom denne oppgaven har som utgangspunkt at brukere har en mobil terminal for kommunikasjon med systemet.

1.9 Øvrige sentrale begreper

Mobil terminal

Mobil terminal vil i denne oppgaven være en mobiltelefon GPRS. Terminalen må i tillegg støtte Java for å kunne benytte tjenesten fullt ut. Muligheten for å bruke systemet på telefoner som ikke støtter Java er altså tilstede, men da vil alt gå over SMS. Noe som vil begrense nytten av systemet.

Kommunikasjonstjeneste / informasjonstjeneste

Systemet som spesifiseres og designes i denne oppgaven er i det vesentlige en informasjonstjeneste. Mobile tjenester deles ofte opp i tre grupper:

- Kommunikasjonstjenester som lar brukere kommunisere med hverandre. Eksempler på kommunikasjonstjenester er SMS, MMS, e-post og telefoni.
- Transaksjonstjenester der penger overføres. Eksempler her er banktjenester og kjøp/salg på Internett
- Informasjonstjenester der brukere mottar informasjon fra et system, enten automatisk etter gitte regler (push) eller manuelt (pull). Eksempel på "push" er aksjepriser og trafikksituasjon der en bruker automatisk mottar oppdateringer. Eksempel på "pull" er underholdning, nyheter og rutetabeller som må hentes manuelt av brukeren.

Transportselskap

Transportselskap kan være hvilket som helst offentlig transportselskap (buss, tog), men i denne oppgaven vil det være et busselskap. Tromsbuss er et eksempel på et busselskap.

Bruker

Enhver person som tar buss (og er interessert i informasjon om forsinkelser ol) og har en mobil terminal med nettilgang er en potensiell bruker. En bruker er en som har registrert seg i systemet som transportkunde og ønsker å motta informasjon om forsinkelser på bussavganger som han/hun er interessert i.

I dette kapittlet har vi sett på problemstilling, mål og teori rundt oppgaven.I neste kapittel ser vi på hva kontekst og kontekstsensitivitet er. I tillegg ser vi på ulike typer interaksjonsmodi.

2 Kontekst og interaksjonsmodi

Dette kapittelet handler om teorien bak oppgaven. Vi kommer til å diskutere kontekst, kontekstsensitivitet og mobile interaksjonsmodi.

2.1 Kontekst

Det finnes mange meninger om hva kontekst er. Hvilken metode man velger å bruke avhenger av hva som er viktig for det systemet som vi skal lage. Det finnes også ulike metoder for å forklare hva kontekst er.

Tidligere definisjoner av kontekst ble ofte beskrevet som eksempler.
Schilit og Theimer [5] beskriver kontekst som lokasjon, identitet til tilstedeværende folk og objekter, og endringer til disse objektene. Brown et al. [6] beskriver kontekst som lokasjon, identiteten til folk rundt brukeren, klokkeslett, sesong på året, temperatur, etc. Ryan et al.[7] beskriver kontekst som brukerens lokasjon, omgivelse, identitet og tid. Dey [8] beskriver kontekst som brukerens emosjonelle tilstand, hva han fokuserer på, lokasjon, tid, objekter og folk i brukerens omgivelse. Slike definisjoner blir vanskelig å bruke når vi skal avgjøre om ting som ikke er nevnt i definisjonen er kontekst.

Det er også mulig å forklare hva kontekst er ved hjelp av synonymer. Slike definisjoner er også vanskelig å bruke i praksis. Eksempler på bruk av synonymer er: Brown [9] beskriver kontekst som de elementer i brukerens omgivelse som brukerens datamaskin vet om. Franklin & Flaschbart [10] ser på kontekst som brukerens situasjon.

Schilit et al.[11], Dey et al. [12] og Pascoe [13] mener at de viktigste aspektene ved kontekst er: hvor du er, hvem du er sammen med, og hvilke resurser som er i nærheten. De ser på kontekst ut fra tre forskjellige omgivelser.

- **Systemomgivelse** – Tilgjengelige prosessorer, nettverkskapasitet, kostnader, enheter som får input fra brukeren
- **Brukeromgivelse** – Lokasjon, personer i nærheten, brukerens sosiale situasjon
- **Fysisk omgivelse** – Lys- og lydnivå.

Disse definisjonene kan fort bli for spesifikke, så i denne oppgaven brukes Dey & Abowd [1] definisjon på hva kontekst er. (fritt oversatt) **"Kontekst er alle typer informasjon som kan brukes til å karakterisere situasjonen til en entitet. En entitet er en person, plass eller objekt som er relevant med hensyn på interaksjon mellom en bruker og en applikasjon, inkludert brukeren og applikasjonen selv [1]".**

Følgende typer kontekst er i følge Dey & Abowd viktigst:

- **Lokasjon** – Handler om hvor en bruker befinner seg. En brukers lokasjon kan man for eksempel finne ut ved bruk av GPS, brukeren oppgir sin lokasjon, sensorer eller (som i denne oppgaven) GSM posisjonering. Andre metoder finnes også.
- **Tid** – Handler om tid på døgnet, dag i uken og når på året.

- **Aktivitet** – Handler om hva brukeren gjør. Dette er kanskje ikke altid like enkelt å måle. Sensorer kan brukes for å avgjøre aktiviteten til en bruker. Et eksempel på en slik sensor er å plassere en sensor i en bil. På den måten kan et system vite når brukeren bruker bilen.
- **Identitet** – Handler om hvem brukeren er. En bruker kan identifiseres gjennom flere typer informasjon. Navn, telefonnummer, fødselsdato eller e-post er eksempler på informasjon som kan identifisere en bruker.

2.2 Kontekstsensitivitet

Fritt oversatt fra Dey og Abowd [1] er kontekstsensitivitet følgende:
"Et system er kontekstsensitivt hvis det bruker kontekst for å gi relevant informasjon og/eller service til en bruker, hvor relevant avhenger av hva brukeren gjør."

Kontekstsensitive applikasjoner kan deles opp i følgende fire kategorier [1]:

- **Proximate selection** – Applikasjoner som henter informasjon for brukeren manuelt basert på tilgjengelig kontekst kalles *proximate selection*. Det er en teknikk hvor objekter og enheter som er relevant for en brukers kontekst er uthevet og lettere tilgjengelig.
- **Automatic contextual reconfiguration** – Applikasjoner som henter informasjon for brukeren automatisk basert på tilgjengelig kontekst kalles a*utomatic contextual reconfiguration*. Det er en teknikk som bruker kontekst for å endre konfigurasjonen til et system.
- **Contextual command application**– Applikasjoner som utfører kommandoer manuelt for en bruker basert på tilgjengelig kontekst kalles *contextual command applications*. Dette er kjørbare tjenester som har blitt gjort tilgjengelig, eller blir modifisert pga. brukers kontekst.
- **Context-triggered actions** – Applikasjoner som kjører kommandoer automatisk for en bruker basert på tilgjengelig kontekst kalles *context-triggered actions*. Dette er tjenester som utføres automatisk når en riktig kombinasjon av kontekst eksisterer. Er basert på enkle if-then regler.

Systemet som spesifiseres og designes i denne oppgaven bruker *context-triggered actions* på den måten at den sier fra til brukere om forsinkelser på bussavganger som de er interessert i. Kombinasjonen av kontekst blir da, bussavgang X er forsinket og person Y er interessert i bussavgang X. Når denne kombinasjonen av kontekst oppdages, utfører systemet automatisk en predefinerte hendelse, som i dette tilfellet er å sende ut beskjed til person Y om at bussavgang X er forsinket.

I og med at systemet varsler brukerene gjennom en kundeapplikasjonen, brukes også a*utomatic contextual reconfiguration* ved at applikasjonen viser forsinkelsesmeldingene automatisk. Da endres altså applikasjonens konfigurasjon ut fra tilgjengelig kontekst. I tillegg brukes også kontekst for å avgjøre hvilken interaksjonsmodus som skal benyttes. Systemets konfigurasjon endres når ulike interaksjonsmodi skal benyttes for å kommunisere med brukeren.

2.3 Mobile interaksjonsmodi

Det finnes flere metoder som kan brukes for å kommunisere med mobiltelefoner og andre mobile terminaler (eks: PDA). Her vil vi diskutere de forskjellige teknologiene og de forskjellige interaksjonsmodi som kan brukes for å kommunisere med brukere.

2.3.1 SMS

SMS – *Short Message Service* [15] er en protokoll som er en del av GSM standarden. SMS brukes for å sende tekstbeskjeder mellom mobiltelefoner. Hver beskjed kan være opp til 160 tegn. Nyere mobiltelefoner har muligheten til å slå sammen 3 SMS'er som i realitet gjør at en SMS kan være opp til 480 tegn.

SMS brukes i dag ikke bare til å sende beskjeder mellom personer, den brukes også i andre tjenester. Det finnes mange tjenester i Norge (og resten av verden) som bruker SMS som eneste inntektskilde.

Den første SMS beskjeden ble sendt i desember 1992 [14] og siden da har SMS bruken eksplodert. Bare i Norge sendes det 10 millioner beskjeder pr. dag [18]. 97% av befolkningen der de bor i Norge har GSM dekning [23] og kan sende og motta SMS. SMS en derfor en bra teknologi å bruke i et slikt system som beskrives i denne oppgaven.

SMS har altså kun mulighet for å overføre tekst. Dette setter visse begrensninger på kommunikasjonen mellom bruker og systemet. Det finnes muligheter for tjenester som er egnet for mer avansert overføring av informasjon. MMS er en slik protokoll.

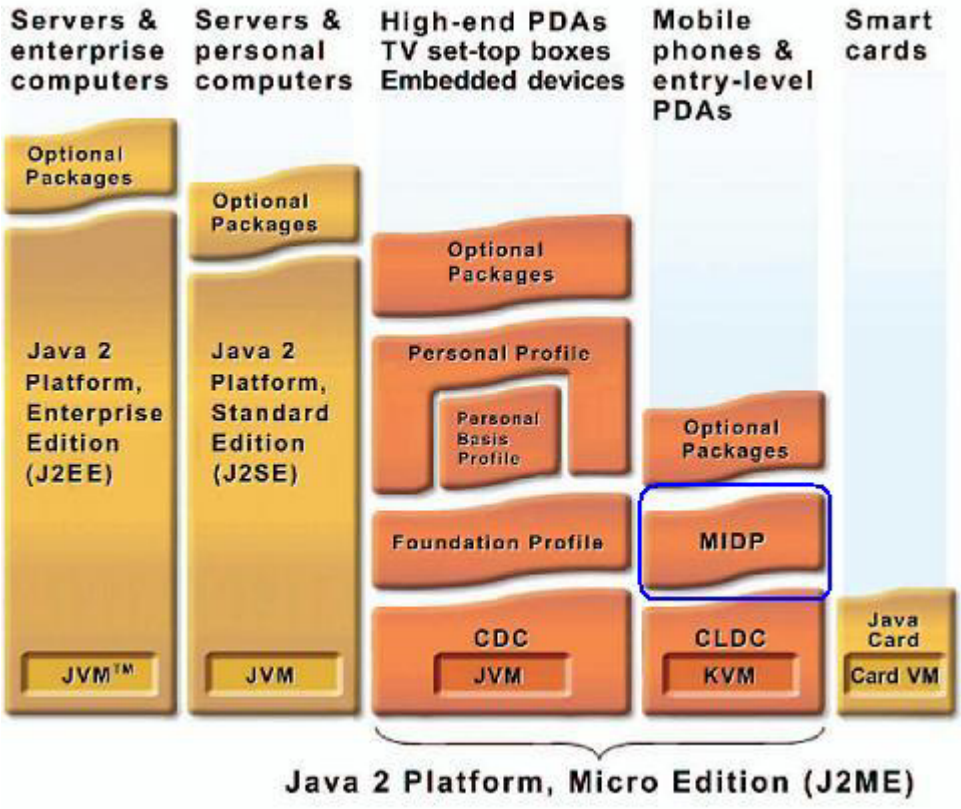
2.3.2 MMS

Denne oppgaven fokuserer ikke på bruk av MMS som interaksjonsmodus, men vi nevner likevel MMS som en mulighet ved videreutvikling av systemet. MMS – *Multimedia Messaging Service* [16] er en ganske ny protokoll i GSM standarden som kan ses på som en videreutvikling av SMS. I motsetning til SMS, som kan sende 160 tegn, kan MMS sende 100KB i en melding[16]. MMS kan i tillegg inneholde flere sider med tekst, lyd, video og bilder, noe som gir mange flere muligheter sammenlignet med SMS.

MMS er en ganske ny teknologi og er fortsatt på prøvestadiet. Det er langt færre mobilbrukere som bruker MMS enn SMS. Det finnes flere grunner til dette. De to viktigste er nok prisen og tilgjengeligheten. Prisen på en MMS er i pr. november 2003 2 kroner og 50 øre pr. Melding [26]. Hvis vi sammenlignet med SMS som koster under 1 kroner pr melding, ser vi at SMS kan sendes til under halve prisen. Tilgjengeligheten er også noe begrenset da man må ha en telefon som støtter MMS for å sende og motta MMS-beskjeder. Det har i det siste kommet mange nye MMS kompatible telefoner på markedet, men det er fortsatt mange som ikke har en slik telefon.

2.3.3 Java

De fleste nye mobiltelefoner har støtte for J2ME – Java 2 Micro Edition [19]. Sony Ericsson P800 som vi skal bruke i denne oppgaven er en slik teleon. J2ME er Sun sin versjon av Java som er beregnet på små enheter. J2ME kan deles opp i to. CLDC – *Connected Limitede Devices Configuration* og CDC – *Connected Device Configuration*[19]. CLDC er teknologi som benyttes i små enheter med lite ressurser som er koblet til et tregt nett, for eksempel mobiltelefon gjennom GSM nettet eller GPRS nettet. CDC derimot brukes i kraftigere enheter som ofte er koblet opp mot raskere nett, for eksempel kraftige PDA'er gjennom WLAN. MIDP – *Mobile Information Device Profile* [19] ligger over CLDC og er den mest brukte profilen for små enheter . MIDP finnes i to versjoner, v1.0 og v2.0. Det finnes pr desember 2003 kun to mobiltelefoner som støtter v2.0. Versjon 2.0 utvider MIDP spesifikasjonen med blant annet avansert brukergrensesnitt, multimedia og sikker HTTP nettverk kommunikasjon[20].



Figur 1 J2ME oversikt [22]

Figuren 1 illustrerer hvordan Java-teknologiene ser ut fra et overordnet logisk synspunkt. Figuren fokuserer på J2ME. Innringet område "MIDP" er den profilen som brukes i J2ME-applikasjoner for mobiltelefoner.

Kommunikasjonen mellom *J2ME-applikasjonen* og transportsystemet går over GSM eller GPRS. Forskjellene på GSM og GPRS er i hovedsak pris og overføringshastighet. GSM har en overføringshastighet på 9.6 kbps mens GPRS har en teoretisk maksimum overføringshastighet på 171.2 kbps [17][21]. I praksis vil

GPRS ha en overføringshastighet på 40 kbps [17]. GPRS bruker *packet switching*[1] i motsetning til GSM som er *circuit switched*[2]. Dette betyr at GPRS sender pakker og bruker radioressurser kun når data sendes eller mottas. Vi betaler altså kun for den dataen som overføres. GSM på den andre siden bruker en dedikert radiokanal. Da må vi betale for hele tidsperioden vi bruker denne kanalen. Derfor er GPRS billigere enn GSM.

Fordelene med å bruke et Java-program for utveksling av informasjon er de ulike modi som kan benyttes. SMS kan ofte bruke de samme interaksjonsmodi, men Java vil ofte kunne utnytte disse bedre. Tilgjengelige modi er *lyd, tekst, vibrasjon og grafikk*. Vi vil under gå mer i detalj om de ulike modi som er tilgjengelig ved bruk av SMS og Java.

2.3.4 Lyd

Lyd er en veldig god metode for å få oppmerksomhet. Det betyr at vi ønsker å bruke lyd for å få en bruker til å forstå at ny informasjon er mottatt.

Når brukeren mottar en SMS vil brukeren oppdage dette ved at telefonens innstillinger bestemmer hvilken lyd som skal genereres. Ofte er dette en enkelt signal eller en form for meldi.

Når *J2ME-applikasjonen* er på og en beskjed mottas, er det fortsatt innstillingene på brukerens telefon som bestemmer hvilken lyd som brukes, hvis lyd brukes i det hele tatt. Transportsystemet kan altså ikke påvirke lydvalget eller lydnivået på noe måte. Mer om bruk av lyd i J2ME-applikasjoner i kapittel 8.3(Diskusjon).

2.3.5 Tekst

SMS kan som sagt kun bruke tekst. Ofte er dette nok for å formidle informasjonen til brukeren. Problemet med tekst gjennom SMS meldinger er at det ikke finnes noen støtte for teksthåndtering. Det er altså ikke mulig å bestemme farge, størrelse, stil eller font i en SMS-beskjed. Hvordan teksten vises bestemmes av mobiltelefonen. Dagens telefoner er ofte utstyrt med fargeskjermer som støtter tusenvis av farger. Her er med andre ord et stort potensiale for å øke lesbarheten i meldinger ved å ha forskjellig farge og størrelse på tekst som sendes ut.

I J2ME-applikasjoner er det mulig å formatere teksten på en enkel måte. Her er det mulig å endre stil (fet, skjev og understreket), størrelse(liten, medium og stor) og skrifttype (tre tilgjengelig) [19]. Det blir da mulig å utheve viktig informasjon slik at det blir enklere for brukere å lese beskjeden. Foreløpig er det ikke støtte for bruk av farge på teksten, men dette er i ferd med å endres. Mer om bruk av tekst og farget tekst i J2ME-applikasjoner i kapittel 8.3 (Diskusjon).

I tillegg til at lesbarheten økes kan vi også, som tidligere nevnt, ha mer tekst pr melding. Vi er altså ikke begrenset til SMS'ens 160 tegn.

[1] I "packet switching" nettverk sendes hver datapakke i små blokker. Tilkobling opprettholdes ikke.
[2] I "circuit switched" nettverk krever en kontinuerlig tilkobling

2.3.6 Vibrasjon

Ved å bruke vibrasjon i kombinasjon med lyd øker man sannsynligheten for at en bruker merker når ny informasjon er tilgjengelig. Dette kan være veldig aktuelt når brukeren beveger seg utendørs og ikke hører like godt når telefonen begynner å lage lyder. Det finnes også situasjoner der det kanskje er aktuelt å bare bruke vibrasjon. Et eksempel kan være hvis man sitter i et møte og ikke ønsker at mobiltelefonen skal begynne å lage lyder. Ved å bruke vibrasjon kan vi altså informere en bruker om ny informasjon på en alternativ måte.

Ikke alle mobiltelefoner støtter vibrasjon, men de fleste nye telefoner som produseres har slik støtte. Om vibrasjon brukes er opp til brukeren å bestemme. Vibrasjonsinnstillinger settes i telefonen, og det er ikke mulig for systemet og overstyre valgene brukeren har gjort

2.3.7 Grafikk

Bruk av grafikk er kanskje en av de mest interessante måtene et system kan kommunisere med en bruker på. Grafikk kan være alt fra veikart til fotografier og tegninger. Ved å bruke grafikk for å illustrere noe, økes lesbarheten. Ting som ikke er så enkelt å forklare med ord kan ofte forklares bedre med bruk av grafikk. J2ME støtter bruk av grafikk i programmer skrevet for mobile enheter[19].

Det er viktig å merke seg at selv om en bruker har en telefon som støtter Java, er det ikke sikkert telefonen har fargeskjerm. All grafikk i *J2ME-applikasjonen* bør derfor også kunne ses på en skjerm uten farge, uten at viktig informasjon forsvinner.

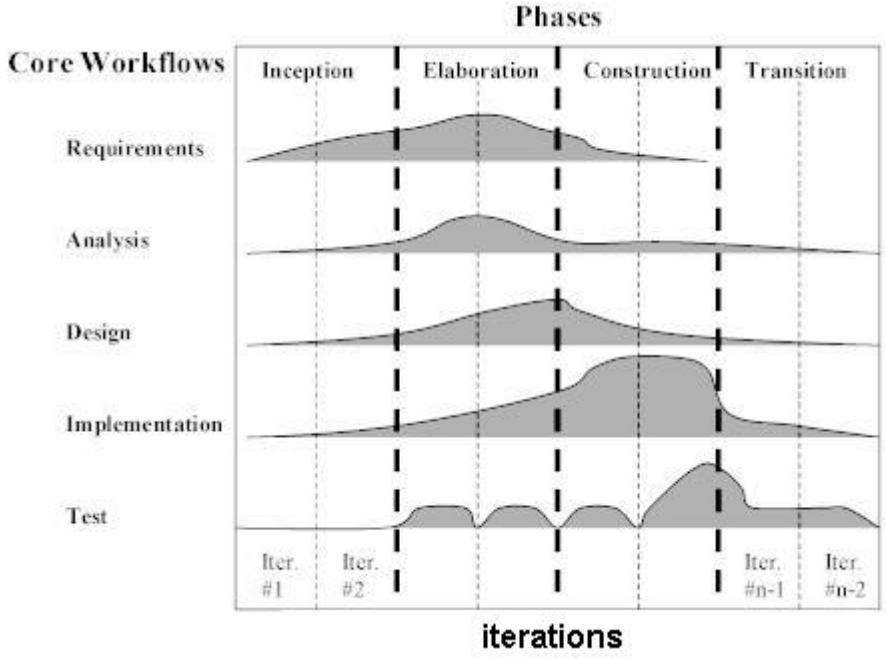
I tillegg bør ikke grafikk brukes når det ikke er nødvendig, da dette øker størrelsen på beskjeden som overføres. Se diskusjon kapittel 7.3.

I dette kapittlet har vi sett på begreper som kontekst, kontekstsensitivitet og mobile interaksjonsmodi. I neste kapittel ser vi på metoden som er brukt i denne oppgaven. ”The Unified Software Development Process”.

3 Metode

Oppgaven er skrevet rundt metoden ”The Unified Software Development Process”[3]. I tillegg er ”Voleremalen”[14] brukt som en veiledning under kravhåndteringen.

Teorien i dette kapittelet er hentet fra boken ”The Unified Software Development Process” [3]



Figur 2 The Unified Software Development Process oversikt [3]

The Unified Software Development Process har fem arbeidsflyter (workflows), disse er kravhåndteringsfasen (*requirements)*, *analyse (analysis)*, *design*, *implementasjon (implementation)* og *test*. Disse arbeidsflytene deles opp over fire faser hvor hver fase består av flere iterasjoner. Disse fasene kalles *startfasen* (*inception phase), utdypningsfasen (elaboration phase), konstruksjonsfasen (construction phase)* og *overgangsfasen (transition phase)*. Se figur 2 for illustrasjon.

3.1 Arbeidsflyter

Metoden består av fem flyter. Disse flytene beskrives her.

Kravhåndtering

I kravhåndteringsfasen prøver vi å forstå hva som skal lages. Vi beskriver hva systemet skal gjøre og hva det ikke skal gjøre. Som regel ender man opp med et kravdokumentet. Kravdokumentet skal i hovedsak leses av *kunden* (de som kjøper systemet), og derfor må man skrive dette dokumentet i et språk som kunden forstår.

Generelt kan man si at i løpet av denne arbeidsflyten skal man

- Få en oversikt over mulige krav – Denne oversikten består av ideer som kunden, brukerene og utviklerne kommer opp med. Listen øker når nye ideer legges til og krymper når ideer blir gjort om til krav. Ikke alle ideene blir gjort om til krav.
- Forstå systemets kontekst – For å finne de riktige kravene og for å bygge riktig system, må man forstå hvilken kontekst systemet skal eksistere i.
- Finne funksjonelle krav – funksjonelle krav er krav som skal bli til brukstilfeller. Et brukstilfelle representerer en måte å bruke systemet på. Det er altså her vi beskriver alle måter en bruker kan bruke systemet på.
- Finn ikke-funksjonelle krav – ikke-funksjonelle krav er egenskaper systemet skal ha. Slike egenskaper kan være: ytelse, pålitelighet, plattformuavhengighet og brukergrensesnitt.

Analyse

Når alle krav har blitt identifisert og beskrevet er det vanlig å analysere dem.

- Analysen lages for å få en mer presis spesifikasjon av kravene.
- Der kravdokumentet er skrevet på en slik måte at kunden forstår det, skal analysedokumentet skrives på en slik måte at utviklerne forstår den.
- Analysemodellen strukturerer kravene på en slik måte at det blir enklere å forstå, klargjøre, endre og vedlikeholde dem.
- Analysemodellen er et førsteutkast av designmodellen.

Denne oppgaven inneholder ingen analyse. Vi valgte på grunn av tiden ikke å ta den med. I tillegg er kravspessifikasjonen såpass klar at det ikke føltes nødvendig å lage en analyse før designet ble laget.

Design

Målet med designet er å beskrive i detalj hvordan systemet skal implementeres. Mer detaljert skal vi:

- Få en grundig forståelse av begrensninger som følger av bl.a. programmeringsspråk, komponentgjenbruk, operativsystem og databaseteknologier.
- Dele opp kravene i individuelle subsystemer, grensesnitt og klasser.
- Dele opp implementasjonen slik at ulike utviklingsteam kan implementere forskjellige deler av systemet.
- Finne grensesnitt mellom subsystemer tidlig i systemets livssykel.
- Kunne se for oss og diskutere designet ved å bruke en felles notasjon.
- Lage en sømløs abstraksjon av systemets implementasjon. Altså, vise hvordan strukturen skal være, slik at det blir enkelt å fylle inn koden rundt denne.

Implementasjon

Etter at designet av systemet er ferdig må det implementeres. Det gjøres i implementasjonsfasen. Denne fasen består av å:

- Planlegge hvordan systemet skal integreres i hver iterasjon

- Implementere desigklassene og subsystemene som ble laget under designfasen.
- Test hver komponent før de integreres i systemet.

Test

Etter implementasjonen må systemet testes for å finne alle feil og mangler. Denne fasen består av å:

- Planlegge hvordan hver iterasjon skal testes.
- Lag *testtilfeller* som spesifiserer hva som skal testes, *test prosedyrer* som forklarer hvordan ting skal testes, og *test komponenter* som skal automatisere mest mulig av testingen.
- Utfør testene og håndtere dem hver for seg. De deler av systemet som inneholder feil blir implementert på nytt, eventuelt designet på nytt hvis det er feil som kanskje gjør at dette er nødvendig.

Denne oppgaven inneholder ikke et eget testkapittel da implementasjonskapittelet og testkapittelet er slått sammen.

3.2 Faser

Metoden består av fire faser. Disse fasene beskrives her.

Startfasen

Målet for denne fasen er å finne ut hva som egentlig skal gjøres. Normalt undersøkes, gjennom flere iterasjoner, flere mulige løsninger og flere mulige arkitekturer. Når denne fasen er over har vi vanligvis fått:

- En enkel forståelse av de viktigste kravene, muligens i form av brukstilfeller.
- Et grovt bilde av programmets arkitektur
- En beskrivelse av målene med prosjektet
- En foreløpig prosjektplan

Utdypningsfasen

Iterasjonene i denne fasen er som regel vanskeligere å ”kaste” enn de i startfasen. Hver iterasjon tilføyer nye deler og nye tester til systemet. Hver iterasjon i denne fasen består av følgende steg:

- Få en god forståelse av problemet som skal løses
- Få en oversikt over de arkitekurmessige grunndelene i programmet
- Lage en detaljert plan for etterfølgende iterasjoner
- Identifisere og eliminere høyrisiko problemer

Denne fasen ender som regel i:

- En eller flere prototyper
- Testprosedyrer for de ulike iterasjonene
- Detaljert prosjektplan som beskriver iterasjonene i prosjektet
- Brukstilfeller som beskriver mesteparten av systemet

- Foreløpig brukermanual
- Software-arkitektur beskrivelse

Konstruksjonsfasen

Iterasjoner i denne fasen tilføyer nye programdeler til systemet, deler som faktisk blir en del av det endelige systemet. Selv om vi er i konstruksjonsfasen, kan fortsatt endringer i brukstilfellene forekomme. Gjennom denne fasen legges flere og flere brukstilfeller til systemet, helt til systemet er ferdig. Brukstilfeller testes etterhvert som de er ferdig. På den måten ender man forhåpentligvis opp med en system med få feil.

Denne fasen ender i:
- Et ferdig system
- Testprossedyrer
- Brukermanual

Overgangsfasen

I den siste fasen leveres produktet ut til de som skal bruke det. Iterasjoner tilføyer fortsatt deler til systemet, men nå som oppgraderinger.

3.3 Volere

Voleremalen ”*Requirements Specification Template*”[14], er en mal for kravhåndtering. Voleremalen beskriver hvordan vi finner krav og hvordan de skal presenteres.

Malen blir ikke fulgt slavisk, men den kommer til å brukes den som en veiledning under kravhåndteringen.

I dette kapittlet har vi sett på metoden oppgaven er skrevet rundt. I neste kapittel beskrives kravspesifikasjonen.

4 Kravspesifikasjon

I dette kapittlet vil vi først beskrive alle funksjonelle krav og alle ikke-funksjonelle krav. Til slutt vil *transportsystemet* beskrives både fra et overordnet logisk synspunkt og et mer detaljert synspunkt.

4.1 Introduksjon

Scenarioene under illustrerer hvordan *transportsystemet* kan fungere i gitte situasjoner.

Scenario 1: Manuelt registrerte bussavganger

En bruker står opp tirsdag morgen og skal ta 20-bussen på jobb. Han har lagt inn i transportsystemet at han begynner på jobb klokken 08.00, adressen til jobben og hvor han bor. Han ønsker å motta informasjon om eventuelle forsinkelser gjennom *kundeapplikasjonen*, så han slår på denne. Det viser seg at bussen er 7 minutter forsinket og transportsystemet finner ut at det må sende ut informasjon om dette til denne brukeren. Før transportsystemet sender ut beskjeden, sjekker det om brukeren er logget på transportsystemet, hvilken situasjon han er i og hvor han befinner seg. Det viser seg at brukeren er logget på, situasjonen er *vanlig* og han er hjemme. Transportsystemet ”bestemmer” seg da for å sende ut beskjed til *kundeapplikasjonen*.

Scenario 2: Endring av situasjon ut fra synkroniseringen med kalenderapplikasjon

EN bruker har et møte onsdag fra 13.00 til 14.00. Dette har han/hun lagt inn i sin kalenderapplikasjon. I dette tidsrommet settes brukerens situasjon automatisk til *møte* og han/hun vil ikke motta noe informasjon fra transportsystemet. Etter møtet vil brukerens situasjon settes tilbake til *vanlig*, og han vil på vanlig måte få meldinger om forsinkelser på busser som han/hun er interessert i.

Scenario 3: Endring av interaksjonsmodus

En bruker beveger seg mellom hjemmet sitt og holdeplassen. *Kundeapplikasjonen* er på. Transportsystemet legger merke til at brukeren beveger seg utendørs og mobiltelefonen begynner å vibrere når den får oppdateringer. I tillegg lager den mer lyd enn den gjorde innendørs.

Når brukeren er kommet fram til jobben (og er innendørs igjen) endres interaksjonsmodus tilbake til vanlig lydnivå og ingen vibrasjon. Transportsystemet vet hvor brukeren jobber og skjønner derfor at hans lokasjon er den samme som jobbeadressen.

Brukeren slår av *kundeapplikasjonen* og transportsystemet går over til å bruke SMS som interaksjonsmodus. Eventuelle meldinger vil nå leveres via SMS.

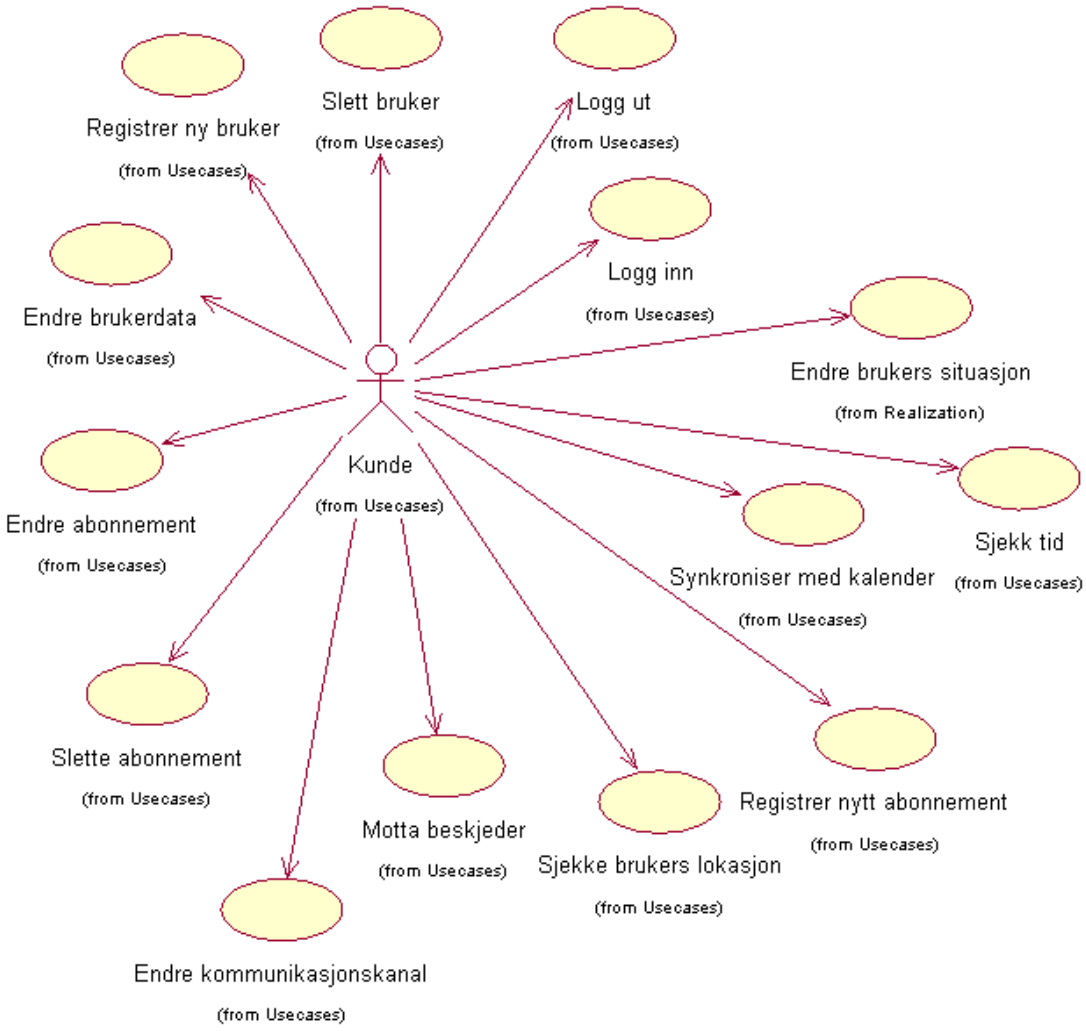
4.2 Funksjonelle krav

Rolleliste

Bruker: En bruker er en som ønsker å benytte transportsystemet for å slippe å kjenne til vanlige rutetider.

Administrator: Administrator har som oppgave å legge inn manuelle beskjeder i transportsystemet.

Brukstilfeller for vanlige brukere



Figur 3 Brukstilfeller for brukere

Figur 3 illustrerer alle brukstilfellene for brukere.

Krav 1: Registrer ny bruker

En bruker må registrere seg før han/hun kan benytte transportsystemet. Følgende skal/bør registreres:

- Navn – Fornavn og etternavn skal registreres
- Adresse – Bostedsadresse bør registreres for å utnytte transportsystemet maksimalt
- Telefonnummer – Telefonnummer skal registreres
- E-post – E-post bør registreres
- Arbeidsadresse – Arbeidsadressen bør registreres for å utnytte transportsystemet maksimalt
- Passord – Passord skal registreres

En bruker skal kunne registrere seg gjennom *kundeapplikasjonen* på mobiltelefonen, via en webside eller via SMS.

Krav 2: Slett bruker

Det skal være mulig for en bruker å slette seg selv fra transportsystemet, hvorpå all informasjon om brukeren og hans kontekst skal slettes. Brukeren sletter sin konto via *kundeapplikasjonen* på mobiltelefonen eller via SMS.

Krav 3: Endre brukerdata

Det skal være mulig for en bruker å endre personlige data. Ting som skal kunne endres er telefonnummer, e-post adresse, passord og jobbadresse. Brukerdata endres gjennom *kundeapplikasjonen* på mobiltelefonen eller via SMS.

Krav 4: Logg inn

Det skal være mulig for en bruker å logge seg inn i transportsystemet.

Brukere logger inn via *kundeapplikasjonen* på mobiltelefonen.

Administrator logger ikke inn via mobiltelefon, men via en datamaskin som er direkte tilknyttet transportsystemet.

Krav 5: Logg ut

Det skal være mulig for en bruker å logge seg ut av transportsystemet.

Brukere logges automatisk ut når de avslutter *kundeapplikasjonen* på mobiltelefonen, men de kan også manuelt logge ut uten å avslutte denne applikasjonen.

Administrator logges automatisk ut når han avslutter administrator-applikasjonen.

Krav 6: Registrer nytt abonnement

Brukere skal kunne registrere abonnement på nye bussavganger

For de brukerne som ikke kan (eller ikke ønsker) å synkronisere kalenderen i sin telefon med transportsystemet, skal det være mulig å legge inn manuelt alle bussavganger de ønsker å motta informasjon om. Transportsystemet må derfor ha en oversikt over alle bussavganger brukeren abonnerer på. Dette gjøres ved at brukeren

registrerer de bussavgangene han/hun ønsker oppdateringer om. En bruker kan abonnere på ubegrenset antall bussavganger.

Følgende ting skal kunne lagres om en bussavgang:

- **Start** – Adressen brukeren befinner seg på når han/hun begynner reisen mot ønsket destinasjon. Brukeren kan også registrere start som *hjemme* eller *jobb*, og transportsystemet vil da automatisk putte inn riktig adresse hvis disse er registrert i transportsystemet. Transportsystemet finner automatisk ut hvilken holdeplass som er nærmest startpunktet.
- **Destinasjon** – Adressen brukeren ønsker å komme seg til. Brukeren kan også registrere destinasjon som *hjemme* eller *jobb*, og transportsystemet vil da automatisk putte inn riktig adresse hvis disse er registrert i transportsystemet. Transportsystemet finner automatisk ut hvilken holdeplass som er nærmest ønsket destinasjon.
- **Rutenummer** – Kan legges inn av brukeren hvis han/hun vet hvilket rutenummer det er på bussen. Hvis han/hun ikke vet rutenummeret skal transportsystemet avgjøre hvilket rutenummer som skal brukes.
- **Tid for ankomst** – Brukeren oppgir når han/hun ønsker å være framme på spesifisert destinasjon.
- **Tid for bussavgang** – Transportsystemet beregner hvilken buss brukeren må ta for å rekke oppgitt ankomsttid. Dette feltet er det altså transportsystemet som finner og lagrer.
- **Dager** – Brukeren oppgir hvilke dager dette abonnementet skal gjelde. Følgende valg er mulig.
    - Daglig – Alle dager i uken
    - Hverdager – Mandag til fredag
    - Enkle dager – Mandag, tirsdag, onsdag, torsdag, fredag, lørdag eller søndag
    - Enkelttilfelle – Abonnementer gjelder kun en enkel avgang. Dato oppgis.
- **Unntak** – Brukeren kan oppgi hvilke unntak som skal gjelde for hvert abonnement. Følgende unntak skal være mulig å spesifisere:
    - Helligdager - Send informasjon på helligdager også
    - Ferie - Send informasjon også når brukeren er i situasjonen *ferie*

Nye abonnementer registreres gjennom *kundeapplikasjonen* på mobiltelefonen eller via SMS.

Krav 7: Slett abonnement

En bruker skal kunne slette bussavganger som han/hun allerede abonnerer på. Abonnementer slettes gjennom *kundeapplikasjonen* på mobiltelefonen eller SMS.

Krav 8: Endre abonnement

En bruker skal kunne endre på bussavganger som han/hun allerede abonnerer på. Det skal være mulig å endre på alle punkter som nevnt i krav 6. Abonnementer endres gjennom *kundeapplikasjonen* på mobiltelefonen eller via SMS.

Krav 9: Informasjon som leveres skal være tilpasset/relevant i forhold til brukerens lokasjon

Transportsystemet må sjekke brukerens lokasjon når det skal sende ut meldinger. Hvis brukerens lokasjon *"samsvarer"* med den lokasjonen han/hun normalt ville hatt når han/hun skal ta en buss, skal informasjon sendes ut. Hvis ikke skal informasjon ikke sendes ut. At lokasjonen *"samsvarer"* betyr at brukeren er så nært bussholdeplassen at han/hun ville ha rukket bussen hvis den ikke var forsinket. Det er da naturlig å anta at brukeren fortsatt skal ta bussen, og ønsker oppdateringer om akkurat denne avgangen.

Eksempel der transportsystemet sjekker brukerens lokasjon før beskjeder sendes ut:

- En bruker tar vanligvis bussen til trening klokken 14.00 på onsdager. Men denne onsdagen er brukeren allerede kommet fram til treningen da han/hun startet en time tidligere denne gangen. Transportsystemet sjekker brukerens lokasjon og finner ut at det ikke er nødvendig å sende ut informasjon om forsinkelser på den bussavgangen han/hun vanligvis ville tatt, da brukeren ikke kommer til å ta den bussen uansett.

Krav 10: Informasjon som leveres skal være tilpasset/relevant i forhold til tid på døgnet og dato.

Selv om en brukers lokasjon tilsier at hans/hennes situasjonen bør være *vanlig,* må tiden også tas hensyn til.

Eksempler der situasjonen endres på grunn av tid

- Det er helligdag og brukerens situasjon endres til *ferie*
- Helligdagen er over og brukerens situasjon endres til *vanlig*

Med tid menes klokkeslett på dagen og dato. Spesielt er det viktig at transportsystemet vet når brukeren har ferie eller sitter i et møte, slik at det ikke sendes ut unødvendig informasjon.

Krav 11: Endre en brukers situasjon

En bruker skal kunne endre sin egen situasjon.

Følgende situasjoner finnes:

- **Ferie** – Når en bruker er i situasjonen *ferie* skal han/hun ikke har noen oppdateringer fra transportsystemet. Dette gjelder både ruteoppdateringer og beskjeder fra administrator. Det finnes flere situasjoner der en bruker skal være i situasjonen *ferie*. Disse er:
    - Helligdag – Hvis ikke brukeren spesifiserer noe annet skal han/hun settes over i situasjonen *ferie* når det er helligdag.
    - Ferie – Når brukeren har fri fra jobb skal han/hun settes over i situasjonen *ferie.*
    - Fridager – Når brukeren har fridager skal han/hun settes over i situasjonen *ferie*.

- **Borte** – Når en bruker er i situasjonen *borte* skal han/hun ikke ha ruteoppdateringer fra transportsystemet. Beskjeder fra administrator skal mottas. Eksempler på når en bruker er *borte*:
    - Brukeren befinner seg ikke lengre i byen hvor hans/hennes bostedsadresse er..
    - Brukeren trenger ikke oppdateringer og ønsker å framstå som *borte.*
- **Møte** – Når en bruker er opptatt og sitter i et møte skal han /hun ikke ha noen ruteoppdateringer fra transportsystemet. Brukeren skal heller ikke ha beskjeder fra administrator mens han/hun sitter i møtet. Beskjeder fra administrator skal sendes når brukeren går over i situasjonen *vanlig.*
- **Vanlig** – Når en bruker er i situasjonen *vanlig* skal han/hun ha oppdateringer fra transportsystemet. Dette gjelder både ruteoppdateringer og beskjeder fra administrator. *Vanlig* er standardsituasjonen, altså den situasjonen brukeren befinner seg i når han/hun ikke befinner seg i noen annen situasjon.

Hvis en bruker ønsker å endrer sin situasjon, skal han/hun kunne fortelle transportsystemet dette på en enkel måte. Transportsystemet har ikke anledning til å endre situasjonen til en bruker hvis han/hun har endret den manuelt. Unntaket er hvis brukeren setter seg selv i situasjon *vanlig*, da kan transportsystemet endre situasjonen hvis konteksten tilsier dette. Brukeren har anledning til å endre til alle situasjoner som eksisterer i transportsystemet. Situasjonen endres gjennom *kundeapplikasjonen* på mobiltelefonen eller via SMS.

Krav 12: Endre kommunikasjonskanal

Brukeren må kunne endre kommunikasjonskanalen som skal brukes.

Brukeren må kunne spesifisere hvordan han/hun skal få informasjon fra transportsystemet. Valgene er *SMS, Java-applikasjonen* eller *begge*. Ikke alle har mobiltelefoner som støtter Java og det er derfor viktig at det finnes andre muligheter for kommunikasjon mellom system og slike brukere.

- Hvis brukeren velger *SMS* som kommunikasjonskanal skal han/hun motta all informasjon fra transportsystemet på sin mobiltelefon via SMS. Transportsystemet må likevel ta hensyn til hvilken situasjon brukeren er i og ikke sende informasjon til brukeren når han/hun ikke ønsker dette.
- Hvis brukeren velger *Java-applikasjonen* som kommunikasjonskanal skal informasjon fra transportsystemet kun sendes til *J2ME-applikasjonen*. Det skal altså ikke sendes ut informasjon via SMS. Applikasjonen må da sørge for at brukeren oppdager den nye informasjonen. Dette kan gjøres via lyd og/eller vibrasjon. Dette betyr at hvis kundeapplikasjonen ikke er på vil ikke brukeren motta informasjonen.
- Hvis brukeren velger *Begge* som kommunikasjonskanal skal både SMS og *J2ME-applikasjonen* brukes. Det fungerer på den måten at hvis *J2ME-applikasjonen* er på, skal informasjonen sendes dit. Hvis *J2ME-applikasjonen* ikke er på, skal informasjonen sendes direkte til mobiltelefonen ved bruk av SMS. Dette er *default* valget for alle brukere.

Kommunikasjonskanalen endres gjennom applikasjonen på mobiltelefonen eller via SMS.

Krav 13: Brukere skal automatisk motta beskjed om forsinkelser og andre viktige hendelser

Oppdateringer fra transportsystemet kan deles opp i to typer. Ruteoppdateringer og beskjeder fra administrator.

Ruteoppdateringer

Brukeren skal motta beskjeder om alle forsinkelser han/hun er interessert i å vite om.

Hvis en buss er forsinket skal transportsystemet sende ut beskjed til alle brukere som abonnerer på denne bussen på dette tidspunktet. I tillegg kan informasjon fra brukerens mobiltelefonkalender brukes for å finne ut om han/hun er interessert i denne avgangen. *Se krav 14.*

Transportsystemet må sjekke hvilken kommunikasjonskanal brukeren ønsker at beskjeden skal sendes på. Dette er oppgitt av brukeren selv. *Se krav 12*.

Transportsystemet må sjekke hvilken situasjon en bruker er i før beskjeder sendes ut. Se *krav 11*. Dette betyr også at tiden og brukerens lokasjon må sjekkes. *Se krav 9 og 10.*

Administratorens brukstilfeller

Brukerene skal også motta beskjed om viktige hendelser endringer i tranportselskapet. Dette er beskjeder som ikke handler om forsinkelser på busser, men om endringer som ofte påvirker mange eller alle brukere.

Eksempel på beskjeder fra administrator:

- Endring fra vinterruter til sommerruter
- Endring av holdeplasslokasjon
- Endring av billettpris

Beskjeder fra administrator legges manuelt inn.

Krav 14: Brukeren skal kunne bruke sin mobiltelefonkalender for å kommunisere med transportsystemet

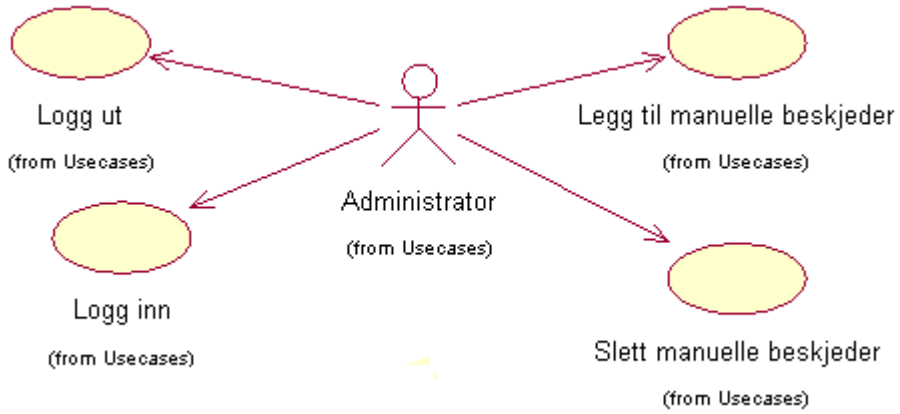
Transportsystemet må kunne endre en brukers situasjon automatisk ut fra informasjon hentet fra brukerens mobiltelefonkalender.

Brukere som benytter kalenderen i sin mobiltelefon skal kunne synkronisere denne med transportsystemet. Hensikten her er at transportsystemet skal kunne bruke informasjon i kalenderen til å finne ut hvilke busser brukeren ønsker å få melding om, hvilken situasjon brukeren befinner seg i og hvilket interaksjonsmodus som skal brukes.

Eksempler på utnyttelse av informasjon i kalenderen:

- Hvis en bruker har lagt inn ferie i sin kalenderapplikasjon, skal transportsystemet automatisk endre brukerens situasjon til *ferie* i dette tidsrommet. Brukerens situasjon endres altså ut fra informasjon i kalenderen.
- Hvis en bruker har lagt inn et møte i kalenderen sin, og dette møtet befinner seg på et hotell i byen. Kan det være interessant for brukeren å få informasjon om bussavganger som går fra brukerens nåværende lokasjon til der møtet skal avholdes. I tillegg kan brukeren settes i situasjonen *møte* mens møtet avholdes. Bussønske og situasjon hentes altså fra brukerens kalender.

Administrator brukstilfeller



Figur 4 Brukstilfeller for administrator

Figur 4 illustrerer brukstilfellene for administrator.

Krav 15: Administrator skal kunne legg til manuelle beskjeder

Administrator skal kunne legge inn manuelle beskjeder. Det skal være mulig for administrator å legge inn nok informasjon til at kun de brukerer som beskjeden påvirker får disse oppdateringene.

En beskjed skal kunne bestå av:

- Hvilke bussruter som påvirkes
- Selve beskjeden
- Bildefil
- Lydfil
- Gyldighetstid

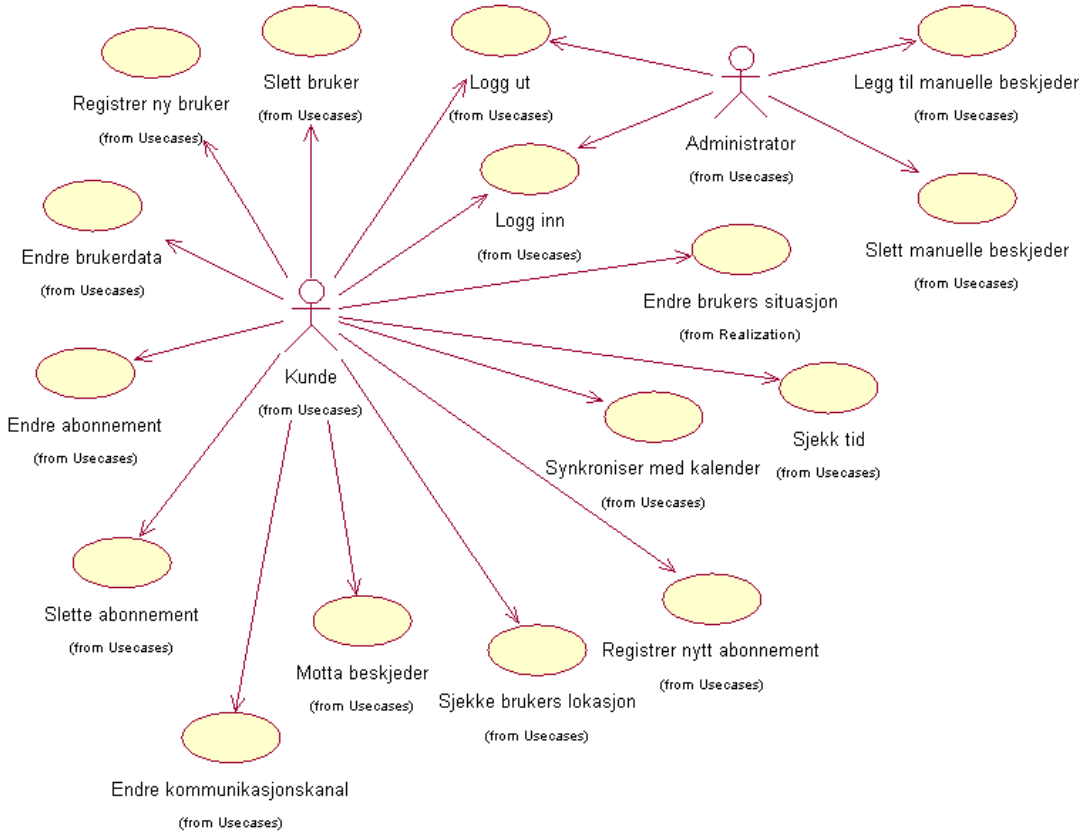
Beskjeder som legges inn, sendes til relevante brukere når de er i situasjon *vanlig* eller *borte.*

Eksempel på bruka av beskjeder fra administrator:

- Billettprisen endres fra kroner 20 til 21.
- Transportsystemet går over fra vinterruter til sommerruter.

Krav 16: Administrator skal kunne slett manuelle beskjeder

Administrator må kunne slette beskjeder som er lagt inn i transportsystemet.



Figur 5 Brukstilfeller hele transportsystemet

Figur 5 viser alle brukstilfellene for hele transportsystemet.

4.3 Ikke-funksjonelle krav

SMS-grensesnitt

Transportsystemet må ta hensyn til om en bruker er logget på transportsystemet før beskjeder sendes ut.

Når en bruker ikke er logget på via *kundeapplikasjonen*, kan SMS som kommunikasjonskanal brukes. I dette tilfellet brukes kun tekst som interaksjonsmodus for å informere brukerene om forsinkelser. Om telefonen lager lyd eller vibrerer er opp til brukeren og hvordan han har konfigurert sin telefon. Siden en SMS kun kan bestå av 160 tegn er det begrenset hvor mye informasjon som kan sendes på en gang. Noen telefoner kan sette sammen 3 tekstmeldinger slik at en tekstmelding kan bli opp til 480 tegn, men det er fortsatt mange ”eldre” mobiltelefoner som ikke har denne funksjonen og den bør derfor ikke brukes. SMS’er må derfor inneholde så lite tekst som mulig, men det må være nok til at brukeren enkelt forstår den.

Hvis det skal sendes ut beskjeder som bør eller må mottas i *kundeapplikasjonen*, kan transportsystemet sende ut en beskjed der brukeren oppfordres til å starte *kundeapplikasjonen* slik at informasjonen kan sendes direkte dit. Hvis for eksempel noe skal illustreres ved bruk av kart må det gjøres gjennom *kundeapplikasjonen*.

J2ME-grensesnitt

Når informasjon mottas gjennom *J2ME-applikasjonen* skal brukeren varsles med et lydsignal. Hvordan dette signalet blir avhenger av brukerens innstillinger i mobiltelefonen. På samme måte vil mobilen vibrere hvis den er stilt inn på å vibrere når den ringer eller mottar SMS.

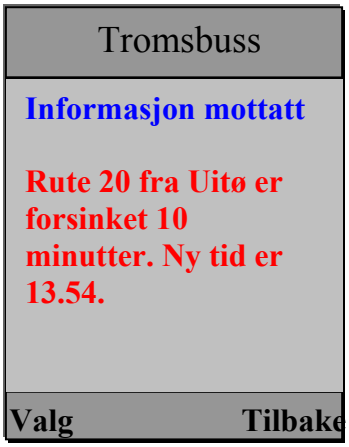
J2ME-applikasjonen tilbyr et bedre grensesnitt mot transportsystemet og tilbyr mye mer enn bare ren tekst. Grafikk og rik tekst er her de viktigste elementene. *J2ME-applikasjonen* må derfor være laget på en slik måte at brukeren foretrekker denne formen for interaksjonsmodus framfor SMS.

- Det er billigere å overføre informasjon til *J2ME-applikasjonen* enn via SMS. Dette fordi informasjon til *J2ME-applikasjonen* går via GPRS, som er billigere enn SMS [25][27]. Se diskusjon.
- Grensesnittet er bedre. Noe som gjør at det er enklere å få fram viktig informasjon, samt at det ser bedre ut og gir et bedre inntrykk

Det er med andre ord viktig at grensesnittet ser bra ut, slik at det blir intuitivt for brukeren hvordan han/hun skal bruke *kundeapplikasjonen.*

Utseende

Figur 6 illustrerer hvordan skjermbildet skal deles opp.



Figur 6 Kundeapplikasjonen illustrasjon

Skjermbildene i *kundeapplikasjonen* skal være del i tre. I den øverst delen i applikasjonen skal det stå hvilket transportsystem brukeren er koblet opp til. Eks: ”Tromsbuss Transportsystem” eller bare ”Tromsbuss”.

Nederst skal det være to valg. Valget til venstre skal være *”valg”* som gir brukeren en liste over de valgene han/hun har. Hvilke valg som kommer her gjenspeiles i brukerkravene. Valget til høyre skal være ”*tilbake*” som tar tilbake til hovedsiden.

Delen i midten skal gjenspeile valget fra ”*valg*”-menyen. Her skal det altså være mulig å gjøre endringer i transportsystemet. I tillegg skal informasjon fra transportsystemet (eks: forsinkelser) komme her. Viktig informasjon fra transportsystemet kan skrives med farget skrift for å øke oppmerksomheten.

Brukervennlighet

Det er viktig at brukervennligheten er så bra som mulig. Med *bra* menes det at det skal være intuitivt for brukeren hvordan *kundeapplikasjonen* skal brukes. Applikasjonen må altså være oversiktlig, med ikke for mange valg samtidig.

Ytelse og kapasitet

Overføringshastigheten til *J2ME-applikasjonen* bestemmes av GPRS standarden. Pr. i dag har GPRS en overføringshastighet på 40 kbps [17]. Med andre ord må transportsystemet begrense mengden data som overføres slik at overføringshastigheten ikke blir før høy. Hvis den blir for høy vil enkelte brukere ikke ha tålmodighet til å vente på informasjonen.

For SMS er kapasiteten 160 tegn pr beskjed. I tillegg er det ingen ventetid ved bruk av SMS. Når brukeren får en SMS kan den leses med en gang.

Operasjonelle krav

Datamaskinen som skal drive transportsystemet må være kraftig nok til å kunne sende og motta informasjon fortløpende hele dagen. Maskinen må ha nettilgang.
I tillegg til en datamaskin trengs en maskin som kan sende og motta SMS.

Vedlikeholdskrav og krav til systemuavhengighet

Transportsystemet trenger ingen operatør.

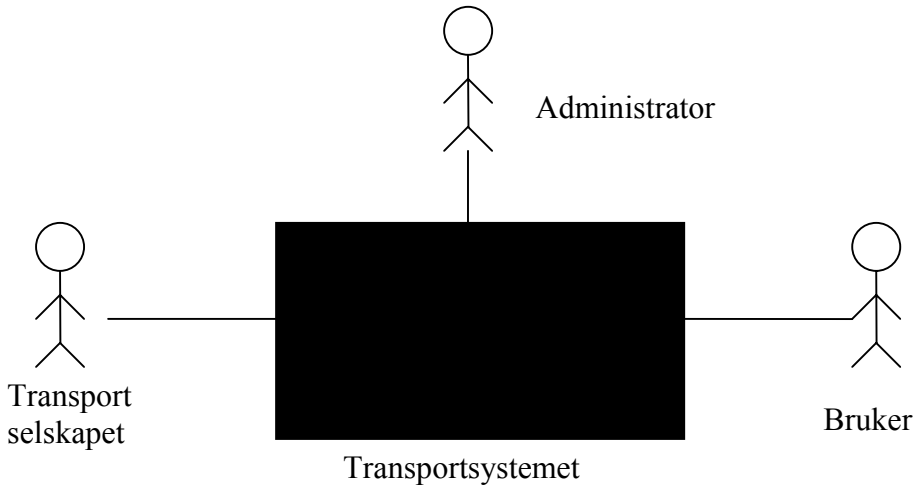
Sikkerhetskrav

Brukere må logge inn for at de skal kunne bruke transportsystemet. Brukere logges automatisk ut når de avslutter *kundeapplikasjonen*. Hvis *kundeapplikasjonen* på telefonen en crasher, skal dette oppdages av transportsystemet og brukeren logges automatisk ut.

4.4 Transportsystemet

Transportsystemet holder oversikt over all relevant informasjon om brukere og deres kontekst. Det holder orden på hvilke beskjeder som skal sendes ut og hvem de skal sendes til. Informasjon om forsinkelser og rutetider hentes fra *Transportselskapet.*

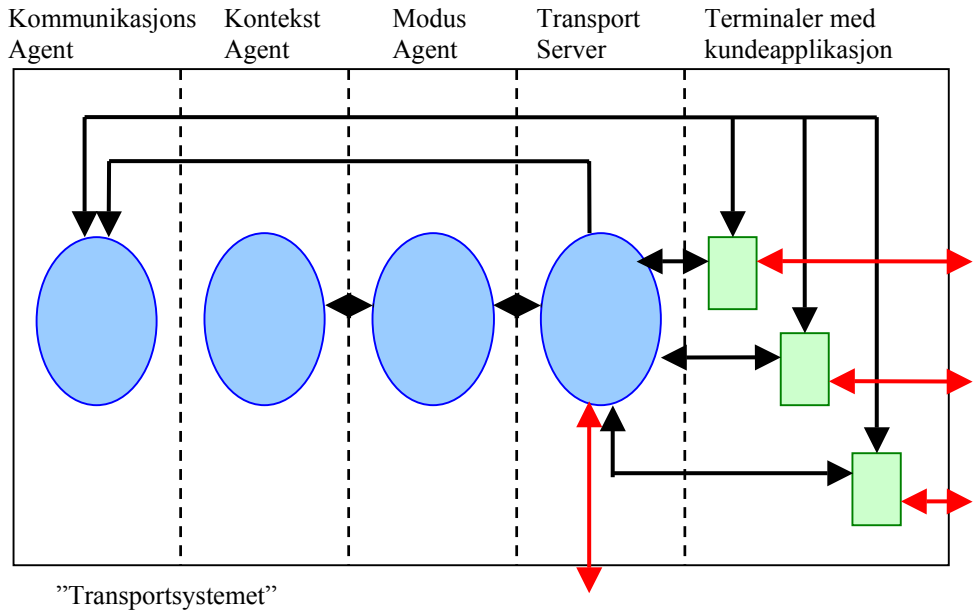
Figur 7 illustrerer hvordan transportsystemet ser ut fra et overordnet logisk synspunkt. Transportsystemet binder sammen de tre aktørene, administrator, bruker og transportselskapet. Administratoren passer på at transportsystemet fungerer tilfredsstillende, transportselskapet har informasjon om rutetider, busser, priser, tilbud, etc., og bruker vil (tilpasset ut fra for eksempel preferanser og abonnement) ha informasjon om forsinkelser på busser han/hun er interessert i.



Figur 7 Transportsystemet oversikt

Figur 8 viser hvordan selve *transportsystemet* er bygd opp. Transportsystemet består av fem deler; kommunikasjonsagent, kontekstagent, modusagent, transportserver og terminaler (som kjører *kundeapplikasjonen*).

- **Kommunikasjonsagent** – Kommunikasjonsagenten er ansvarlig for kommunikasjon fra transportsystemet til de mobile terminalene. Kommunikasjon mot brukerne skjer i hovedsak ved hjelp av GPRS eller SMS.
- **Kontekstagent** – Kontekstagenten har som oppgave å finne og lagre kontekst om relevante entiteter i transportsystemet.
- **Modusagent** – Modusagenten er ansvarlig for å finne ut hvilket modus de forskjellige brukerne er i. Modus brukes for å avgjøre hvilke brukere som skal motta beskjeder som sendes fra transportsystemet. For å klare dette benytter den seg av kontekstagenten.
- **TransportServer** - Transportserveren binder sammen agentene og klienten og utfører alle klientkallene. I tillegg er den ansvarlig for å oppdage forsinkelser og sende ut disse ved hjelp av agentene.
- **Terminaler** – Terminaler brukes for å vise informasjon til brukerne. En terminal kan være hvilken som helst enhet med nettilkobling, men i hovedsak er det snakk om mobiltelefoner, smarttelefoner eller PDA'er.



Figur 8 Transportsystemet detaljert

Røde linjer i figur 8 viser kommunikasjon ut av systemet. Rød linje til og fra terminalene betyr kommunikasjon mellom terminal og bruker. Rød linjen til og fra TransportServer betyr kommunikasjon mellom TransportServer og et eksternt system som inneholder data om rutetider og forsinkelser i disse. Svarte linjer viser kommunikasjon mellom ulike deler av systemet.

Sammen utgjør disse fem delene *transportsystemet*. For at transportsystemet skal fungere tilfredsstillende må alle delene være tilstede. Som det går fram av figuren går kommunikasjonen begge veier. Dette er fordi brukere både mottar informasjon fra transportsystemet samt sender informasjon til det. Informasjon som hentes fra transportsystemet er for eksempel meldinger om forsinkelser, mens informasjon som sendes til transportsystemet er for eksempel kontekstinformasjon.

I dette kapittlet har vi sett på kravspesifikasjonen til transportsystemet. I tillegg har vi sett på hvordan dette systemet er bygget opp. I neste kapittel ser vi på designet av oppgaven og på oppbyggingen av den interne databasen.

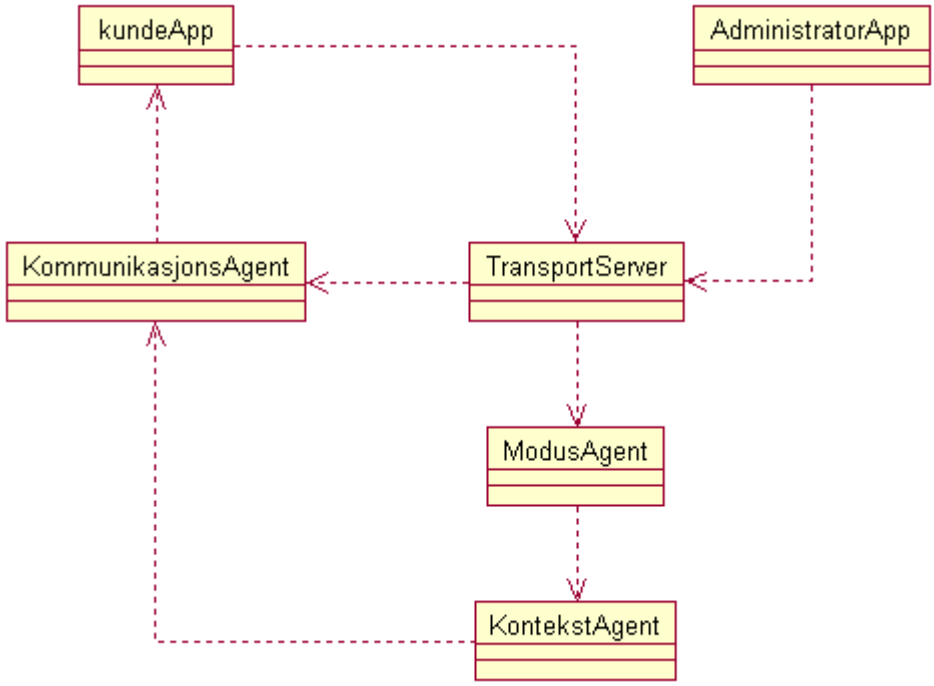
5 Design

5.1 Introduksjon

Designmodellen beskriver interaksjonen mellom de ulike designklassene. Det vil først bli gitt en detaljert beskrivelse av de forskjellige designklassene som eksisterer i transportsystemet. Deretter blir hvert brukstilfelle gjennomgått og beskrives med hensyn på hvilke designklasser de benytter. Til slutt blir databasen som ligger i bunnen av transportsystemet beskrevet.

5.2 Designklasser

Vi vil her gi en presis beskrivelse av hver klasse som er involvert i alle brukstilfelle-realiseringene.

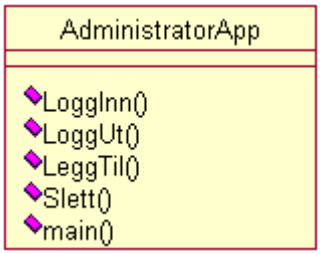


Figur 9 Designklasser oversikt

Figur 9 viser alle designklassene og hvordan de er avhengig av hverandre. Som figuren viser, eksisterer følgende designklasser: *KundeApp, AdministratorApp, TransportServer, ModusAgent, KontekstAgent og KommunikasjonsAgent*. Hver gang ordet *kundeapplikasjonen* brukes i teksten, henviser dette til *KundeApp,* som er den applikasjonen som ligger på brukerens telefon.

5.2.1 Designklasse AdministratorApp

AdministratorApp er den applikasjonen som benyttes av administrator for å legge inn og slette manuelle beskjeder. Applikasjonen skal implementeres i Java (J2SE). Figur 10 illustrerer denne klassen og dens metoder.



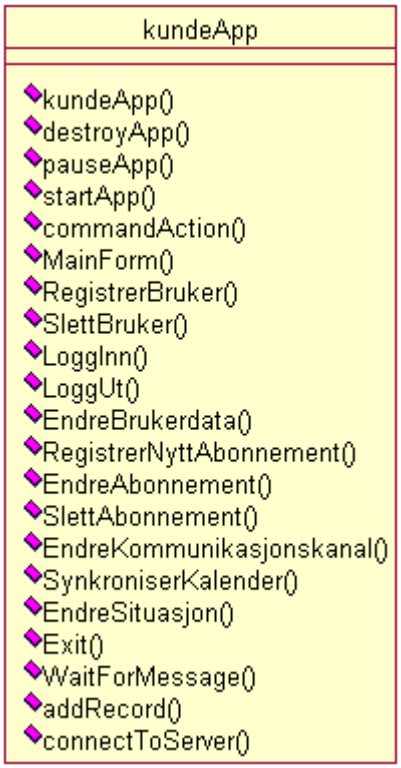
Figur 10 Designklasse AdministratorApp

Følgende metoder eksisterer i *AdministratorApp*:

- **LoggInn()** – Administrator må logge inn i transportsystemet før han/hun kan slette eller legge til manuelle beskjeder. Administrator vil her måtte skrive inn brukernavn og passord. Denne metoden kaller på metoden *LoggInn()* i *TransportServer*.
- **LoggUt()** – Administratoren skal logge ut når han/hun er ferdig. Denne metoden kalles når administratoren ønsker å logge ut. Etter 10 minutter uten aktivitet logges administrator automatisk ut.
- **LeggTil()** – Metoden kalles når administratoren ønsker å legge til beskjeder i transportsystemet. Administratoren vil da få mulighet til å skrive inn selve beskjeden, legge til et bilde, legge til en lydfil, bestemme hvilken bussrute beskjeden gjelder for og spesifisere gyldighetstiden for beskjeden. Denne metoden kaller på metoden *LeggTil()* i *TransportServer*.
- **Slett()** – Metoden kalles når administrator ønsker å slette en beskjed. En liste over tilgjengelige beskjeder vises. Administratoren velger de beskjeder som skal slettes fra denne listen. Denne metoden kaller på metoden *Slett()* i *TransportServer*.
- **Main()** – Denne metoden inneholder kode for å starte administrator-applikasjonen. Når administrator-applikasjonen er startet, kalles metoden *LoggInn()* som venter på at en administrator skal logge inn.

5.2.2 Designklasse KundeApp

KundeApp er den applikasjonen som brukes av brukerene for å kommunisere med transportsystemet. Denne applikasjonen skal implementeres i J2ME. Figur 11 illustrerer denne klassen og dens metoder.



Figur 11 Designklasse KundeApp

Følgende metoder eksisterer i *KundeApp*:

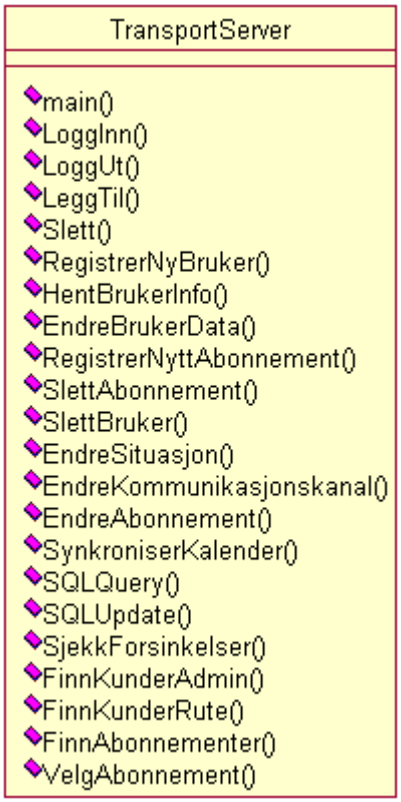
- **KundeApp() –** Constructor.
- **destroyApp()** – Obligatorisk metode som kalles når applikasjonen avsluttes. Rydder opp i minnet til telefonen.
- **pauseApp()** – Obligatorisk metode som kalles når en bruker setter applikasjonen på pause. Den er tom da dette aldri skjer i denne applikasjonen.
- **startApp()** – Obligatorisk metode som startes automatisk når applikasjonen startes. Sjekker om logg inn informasjon er lagret i telefonen og starter metoden *LoggInn()* hvis dette er tilfellet. Hvis ikke kalles metoden *MainForm()*.
- **commandAction()** – Når en bruker gjør et menyvalg, fanges dette valget opp av denne metoden som har ansvar for at den rette metoden kalles.
- **MainForm(***informasjon***) –** Dette er hovedformen i programmet som legger til menyvalg og skriver ut informasjon til skjermen. Metoden tar en streng som argument. Denne strengen skrives til skjermen når metoden kalles.
- **RegistrerBruker()** – Metode som lar en bruker registrere seg selv i transportsystemet. Følgende informasjon må skrives inn: Fornavn, etternavn, telefon og passord. I tillegg kan også følgende lagres: E-post, adresse, postnummer, poststed, jobbadresse, jobbpostnummer og jobbpoststed. Denne metoden kaller på metoden *RegistrerNyBruker()* i *TransportServer*.

- **SlettBruker()** – Metoden lar en bruker slettes seg selv fra transportsystemet. Ved sletting må brukeren oppgi sitt telefonnummer og sitt passord. Metoden *SlettBruker()* i *TransportServer* kalles.
- **LoggInn(***LoggInnInnformasjonLagret***)** – Argumentet *LoggInnInnformasjon Lagret* indikerer om brukerens logg inn informasjon er lagret i telefonen. Hvis dette er lagret, logges brukeren automatisk inn. Hvis ikke må brukeren logge manuelt inn. Telefonnummer og passord lagres i telefonen når brukeren logger inn. Brukeren slipper altså å logge inn manuelt hver gang. Denne metoden kaller på metoden *LoggInn()* i *TransportServer*.
- **LoggUt()** – Når *kundeapplikasjonen* avsluttes logges brukeren automatisk ut. Brukeren kan også logge ut manuelt. Ved manuell utloging slettes innloggingsinformasjonen fra telefonen. Denne metoden kaller på metoden *LoggUt()* i *TransportServer*.
- **EndreBrukerdata()** –Metoden lar en bruker endre følgende personlige opplysninger: Adresse, postnummer, poststed, e-post, passord, jobbadresse, jobbpostnummer og jobbpoststed. Metoden *HentBrukerInfo()* kalles for å få returnert all data om en bruker. Disse data skrives ut til skjermen. Når brukeren har spesifisert de endringene han/hun ønsker å gjøre trykkes Ok som kaller på metoden *EndreBrukerData()* i *TransportServer*.
- **RegistrerNyttAbonnement()** – Metoden lar en bruker legge til nye abonnementer på bussavganger. Følgende data registreres om bussavganger: destinasjonsadresse, destinasjonspoststed, destinasjonspostnummer, senest tid for ankomst, startadresse, startpostnummer, startpoststed og eventuelt kan bussrute velges. I tillegg kan brukeren velge hvilke dager i uken- og hvilke unntak som skal gjelde. Denne metoden kaller på metoden *RegistrerNyttAbonnement()* i *TransportServer*.
- **EndreAbonnement()** – Metoden lar en bruker endre sine abonnementer. Denne metoden kaller på metoden *FinnAbonnement*() i *TransportServer* som returnerer en liste over abonnementer som brukeren kan endre på. Brukeren velger så fra denne listen hvilket abonnement han/hun ønsker å endre. Når brukeren har valgt hvilken abonnement som skal endres, kalles metoden *VelgAbonnement*() i *TransportServer*, som returnerer alle data om valgt abonnement. Følgende informasjon kan så endres: destinasjonsadresse, destinasjonspoststed, destinasjonspostnummer, senest tid for ankomst, startadresse, startpostnummer, startpoststed, bussrute, dager abonnementet gjelder og unntak som gjelder. Endringer utføres ved at metoden *EndreAbonnement()* i *TransportServer* kalles.
- **SlettAbonnement()** – Metoden lar brukeren slette sine abonnementer. Denne metoden kaller på metoden *FinnAbonnement*() i *TransportServer*. En liste over tilgjengelige abonnementer returneres. Kunden velger så de abonnementene som skal slettes. Abonnementer slettes ved at metoden *SlettAbonnement()* i *TransportServer* kalles en gang for hver abonnement som skal slettes.
- **EndreKommunikasjonskanal()** – Metoden lar en bruker endre sin kommunikasjonskanal. Brukeren kan velge tilgjengelige kommunikasjonskanaler fra en liste. Denne metoden kaller på metoden *EndreKommunikasjonskanal()* i *TransportServer*.
- **SynkroniserKalender()** – Metoden synkroniserer telefonens interne kalender med transportsystemet, med den hennsikt å lage abonnementer av denne informasjonen. Denne metoden kaller på metoden *SynkroniserKalender()* i *TransportServer*. Kalenderinformasjonen overføres som argumenter.

- **EndreSituasjon()** – Metoden lar brukeren endre sin situasjon. Brukeren kan velge tilgjengelige situasjoner fra en liste. Denne metoden kaller på metoden *EndreSituasjon()* i *TransportServer*.
- **Exit()** – Når brukeren ønsker å avslutte *kundeapplikasjonen* kalles denne metoden. Denne metoden kaller metoden *LoggUt()* før applikasjonen avsluttes.
- **WaitForMessage()** – Når *kundeapplikasjonen* startes, startes en ny tråd som kjører denne metoden. Metoden tar kontakt med metoden *WaitForMessage()* i *KommunikasjonsAgent* og venter på nye beskjeder fra transportsystemet. Hvert femte minutt returneres kallet fra *KommunikasjonsAgent* selv om ingen beskjeder er funnet. Metoden kaller på nytt metoden *WaitForMessage()* i *KommunikasjonsAgent.*
- **addRecord()** – Denne metoden lagrer brukerens telefonnummer og passord i telefonen når han/hun logger inn i transportsystemet.
- **ConnectToServer() –** Hver gang *kundeapplikasjonen* skal kommunisere med transportsystemet kalles denne metoden som tar kontakt med *TransportServer* og kaller på riktig metode.

5.2.3 Designklasse TransportServer

TransportServer er kjernen i transportsystemet. Det er den som binder de ulike delene av transportsystemet sammen. Figur 12 illustrerer denne klassen og dens metoder.



Figur 12 Designklasse TransportServer

Følgende metoder finnes i *TransportServer*:

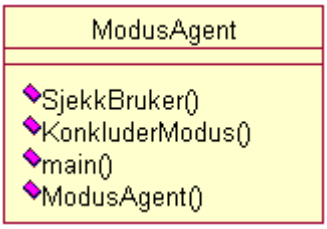
- **main()** –Starter *TransportServer* og lytter etter kall fra *AdministratorApp* og *KundeApp*. Starter også *SjekkForsinkelser*()-metoden i en egen tråd som sjekker om forsinkelser oppstår.
- **LoggInn(***brukernavn, passord***)** – Denne metoden kalles av administratoren og brukerene når de logges inn. For brukere vil brukernavnet være deres telefonnummer. Denne metoden sjekker i databasen om informasjonen stemmer og returnerer *fornavn* og *etternavn* hvis den stemmer og *false* hvis den ikke stemmer. Brukere som logger inn blir registrert som *online* i databasen.
- **LoggUt(***telefonnummer***)** – Kalles av en bruker når han/hun ønsker å logge av transportsystemet. Telefonnummer brukes som argument for å identifisere den brukeren som ønsker å logge ut. Brukeren registreres i databasen som *offline*
- **LeggTil(***beskjed, ruteID, bildefil, lydfil, gyldighetTid***)** – *AdministratorApp* kaller denne metoden når en ny beskjed skal registreres. Selve beskjeden, bildefilen, lydfilen identifikatoren til den bussruten som påvirkes og beskjedenes gyldighetstid overføres som argumenter. Beskjeden lagres i databasen slik at den kan slettes ved en senere anledning. I tillegg skal brukere som logger på systemet før gyldighetstiden utgår også mottar beskjeden. Metoden *FinnKunderAdmin*() kalles.
- **Slett (***beskjedID***)** – *AdministratorApp* kaller denne metoden når en beskjed skal slettes. Argumentet *beskjedID* identifiserer den beskjeden som skal slettes.
- **RegistrerNyBruker(***fornavn, etternavn, adresse, postnummer, poststed, telefonnummer, passord, e-post, jobbadresse, jobbpostnummer, jobbpoststed***)** – Metoden sjekker argumentene mot databasen og kontrollerer at brukeren ikke eksisterer fra før. Hvis brukeren ikke eksisterer fra før, lagres han/hun i databasen. Hvis brukeren allerede eksisterer, returneres *false* og brukeren registreres ikke.
- **HentBrukerInfo(***telefonnummer***) –** Brukes av metoden *EndreBrukerdata()* i *KundeApp* for å få returnert all personlig data om en bruker. Argumentet *telefonnummer* identifiserer den brukeren som skal sjekkes.
- **EndreBrukerdata(***telefonnummer, adresse, postnr, poststed, email, passord, jobbadresse, jobbpostnummer, jobbpoststed***)** – Kalles av *KundeApp* når en bruker ønsker å endre noen av sine brukerdata. Argumentet *telefonnummer* identifiserer hvilken bruker som ønsker å utføre endringene. De resterende argumentene indikerer hvilken informasjon en bruker ønsker å endre og hva de skal endres til. Argumenter som ikke inneholder informasjon skal heller ikke lagres.
- **RegistrerNyttAbonnement(***telefonnummer, tid, adresse, postnr, poststed, adresseStop, postnrStopp, poststedStop, rute, dag, Unntak***)** – *KundeApp* kaller denne metoden når et nytt abonnement skal lagres. Metoden finner ut hvilken buss som skal tas av brukeren ved å kontrollere *TransportTider-databasen* (den eksterne databasen) mot argumentene. Det nye abonnementet lagres så i *transportsystem-databasen*. (den interne databasen)
- **SlettAbonnement(***abonnementID***)** – *KundeApp* kaller denne metoden når et abonnement skal slettes. Argumentet *abonnementID* identifiserer abonnementet som skal slettes.

- **SlettBruker(***telefonnummer, passord***)** – Kalles av *KundeApp* når en bruker ønsker å slette sin bruker fra transportsystemet. Argumentene *telefonnummer og passord* kontrolleres mot databasen før slettingen utføres. Hvis argumentene stemmer slettes brukeren og all informasjon som tilhører han/henne fra databasen.
- **EndreSituasjon(***telefonnummer, situasjon***)** – Kalles av *KundeApp* når en bruker ønsker å endre sin situasjon. Argumentet *telefonnummer* identifiserer brukeren, og argumentet *situasjon* identifiserer den nye situasjonen som skal gjelde. Hvis den nye situasjonen er noe annet enn *vanlig,* kan ikke transportsystemet endre brukerens situasjon automatisk.
- **EndreKommunikasjonskanal(***telefonnummer, KomKanal***)** – Kalles av *KundeApp* når en bruker ønsker å endre sin kommunikasjonskanal. Argumentet *telefonnummer* identifiserer brukeren og argumentet *KomKanal* identifiserer den nye kommunikasjonskanalen.
- **EndreAbonnement(***abonnement, startholdeplass, stopptid, stoppholdeplass, dag, unntak***)** – Kalles av *KundeApp* når en bruker ønsker å endre et av sine abonnementer. Argumentet *abonnement* identifiserer abonnementet som skal endres. Hvis noe av argumentene *startholdeplass, stopptid, stoppholdeplass, unntak* eller *dag* inneholder noe informasjon betyr dette at abonnementet skal oppdateres med de nye verdiene.
- **SynkroniserKalender(***telefonnummer, kalenderData***)** – Kalles av *KundeApp* når en bruker ønsker å synkronisere sin kalender mot transportsystemet. Argumentet *telefonnummer* identifiserer hvilken bruker som synkroniserer. Argumentet *kalenderData* inneholder informasjonen som skal gjøres om til et eller flere abonnement.
- **SQLQuery(***sqlStatement***)** – Denne metoden kjører SQL-spørringer av typen *SELECT*. Argumentet *sqlStatement* inneholder selve spørringen som skal utføres. Metoden returnerer resultatet fra spørringen.
- **SQLUpdate(***sqlStatement***) -** Denne metoden kjører SQL-spørringer av typen *UPDATE, INSERT* og *DELETE.* Argumentet *sqlStatement* inneholder selve spørringen som skal utføres
- **SjekkForsinkelser()** – Denne metoden har som oppgave å oppdage forsinkelser i transporttidene. Denne informasjonen kommer fra et eksternt system med en ekstern databasen over rutetider og forsinkelser. Når forsinkelser oppdages kalles metoden *FinnKunderRute*().
- **FinnKunderAdmin(***beskjed, ruteID***)** – Sjekker hvilke brukere som påvirkes av en beskjed som legges inn av administrator. Argumentet *ruteID,* sjekkes mot alle abonnementer for å finne de brukerene som ønsker denne beskjeden. For hver bruker som finnes, kalles metoden *SjekkBruker*() i *ModusAgent* for å avgjøre om en beskjed skal sendes ut. *ModusAgent* returnerer om en bruker ønsker å motta beskjeden. *SendBeskjed*() i *KommunikasjonsAgent* kalles så for hver bruker som skal ha beskjeden.
- **FinnKunderRute(***ruteID, gammelTid, NyTid***)** – Sjekker hvilke brukere som påvirkes av en ruteending. Denne metoden sjekker alle abonnementer, og ut fra de argumentene som overføres genereres en liste over alle brukerene som påvirkes. For hver bruker i denne listen kalles metoden *SjekkBruker*() i *ModusAgent*. *ModusAgent* returnerer om en bruker skal motta en beskjed. *SendBeskjed*() i *KommunikasjonsAgent* kalles så for hver bruker som skal ha beskjeden.

- **FinnAbonnementer(***telefonnummer***)** – Metoden returnerer en liste over alle abonnementene som en bruker har. Argumentet *telefonnummer* identifiserer brukeren.
- **VelgAbonnement(***abonnementID***)** – Returnerer all data om abonnementet som identifiserer av argumentet *abonnementID*.

5.2.4 Designklasse ModusAgent

ModusAgent er den delen av transportsystemet som avgjør hvilken modus en bruker befinner seg i, og om en bruker skal motta en beskjed. Figur 13 illustrerer denne klassen og dens metoder.



Figur 13 Designklasse ModusAgent

Følgende metoder finnes i *ModusAgent*:

- **SjekkBruker(***telefonnummer, beskedtype, unntak***)** – Denne metoden kalles av *TranportServer* for å sjekke hvilken modus en bruker befinner seg i. Argumentet *telefonnummer* brukes for å identifisere brukeren som skal kontrolleres. Argumentet *beskjedtype* indikerer hvilken type beskjed dette er, ruteoppdatering eller beskjed fra administrator. Argumentet unntak, inneholder eventuelle unntak som gjelder. Metoden *SjekkKontekst()* i *KontekstAgent,* som returnerer en brukers kontekst, kalles. Når kontekst er returnert, kalles metoden *KonkluderModus()* som returnerer om brukeren skal motta beskjeden eller ikke. Denne informasjonen returneres til *TransportServer*.
- **KonkluderModus(***situasjonID, lokasjon, beskjedType, unntakID***)** – Når *KontekstAgent* har returnert kontekst om en bruker, kalles denne metoden som konkluderer hvilken modus brukeren befinner seg i ut fra argumentene: *situasjonID, lokasjon, unntakID og beskjedtype.* Metoden returnerer om en bruker ønsker å motta beskjeden.
- **main()** – Starter *ModusAgent* og lytter etter kall fra *TransportServer*.
- **ModusAgent()** - Constructor

5.2.5 Designklasse KontekstAgent

KontekstAgent er den delen av transportsystemet som brukes av *ModusAgenten* for å finne kontekst om en bruker. Figur 14 illustrerer denne klassen og dens metoder.

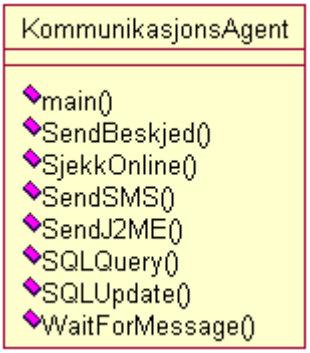
Figur 14 Designklasse KontekstAgent

Følgende metoder finnes i *KontekstAgent*:

- **KontekstAgent() –** Constructor
- **main()** – Startet opp *KontekstAgent* og lytter etter kall fra *ModusAgent*.
- **SQLQuery(***sqlStatement***)** – Denne metoden kjører SQL-spørringer av typen *SELECT*. Argumentet *sqlStatement* inneholder selve spørringen som skal utføres. Metoden returnerer resultatet fra spørringen.
- **SQLUpdate(***sqlStatement***)** – Denne metoden kjører SQL-spørringer av typen *UPDATE*, *INSERT* og *DELETE*. Argumentet *sqlStatement* inneholder selve spørringen som skal utføres
- **SjekkKontekst(***telefonnummer***)** –*ModusAgent* kaller denne metoden for å finne kontekst om en bruker. Argumentet *telefonnummer* brukes for å identifisere brukeren. Denne metoden kaller på metodene *SjekkTid*(), *SjekkLokasjon*(), *SjekkSituasjon*() og *SjekkInteraksjonsmodi*() for å finne de forskjellige typene kontekst en bruker har. Denne informasjonen returneres til *ModusAgent*.
- **SjekkTid(***telefonnummer***)** – Sjekker dato og klokkeslett mot kalenderinformasjonen som er lagret om brukeren. Endrer brukerens situasjon hvis tiden tilsier dette.
- **SjekkLokasjon(***telefonnummer***)** – Sjekker lokasjonen til en bruker. Bruker argumentet *telefonnummer* for å identifisere hvilken bruker som skal sjekkes. Bruker GSM-posisjonering for å kunne returnere brukerens geografiske posisjon.
- **SjekkSituasjon(***telefonnummer***)** – Sjekker situasjonen til en bruker. Bruker argumentet *telefonnummer* for å identifisere hvilken bruker som skal sjekkes. Returnerer brukerens situasjon.
- **SjekkInteraksjonsmodi(***telefonnummer***)** – Sjekker interaksjonsmodus til en bruker. Bruker argumentet *telefonnummer* for å identifisere hvilken bruker som skal sjekkes. Returnerer brukerens interaksjonsmodus.

5.2.6 Designklasse KommunikasjonsAgent

KommunikasjonsAgenten er den delen av transportsystemet som brukes ved utsending av beskjeder til brukerne. *Kommunikasjonsagenten* er delvis ansvarlig for å avgjøre hvordan beskjeden skal sendes ut. Figur 15 illustrerer denne klassen og dens metoder.



Figur 15 Designklasse KommunikasjonsAgent

Følgende metoder finnes i *KommunikasjonsAgent*:

- **main()** – Starter *KommunikasjonsAgent* og lytter etter kall fra *TransportServer*.
- **SendBeskjed(***telefonnummer, beskjed, komkanal***)** – Transportsystemet kaller på denne metoden når beskjeder skal sendes ut. Argumentet *telefonnummer* identifiserer brukeren, *beskjed* inneholder selve beskjeden og *komkanal* forteller hvilken kommunikasjonskanal brukeren ønsker å benytte.
    - Hvis *komkanal* er *SMS* kalles metoden *SendSMS*().
    - Hvis *komkanal* er *Java-applikasjonen* eller *begge* kalles metoden *SjekkOnline*().
        - Hvis brukeren er *online* kalles metoden SendJ2ME() .
        - Hvis brukeren er *offline* og *komkanal* er *begge* kalles metoden *SendSMS*().
        - Hvis brukeren er *offline* og *komkanal* er *Java-applikasjonen*, skjer ingen ting. Beskjeden sendes altså ikke ut.
- **SjekkOnline(***telefonnummer***)** – Denne metoden kalles på av metoden *SendBeskjed()*. Den har som oppgave å avgjøre om en bruker er logget på transportsystemet eller ikke. Argumentet *telefonnummer* identifiserer brukeren som skal kontrolleres. Returnerer *true* eller *false. True* hvis en bruker er logget på gjennom *kundeapplikasjonen* og *false* hvis brukeren ikke er logget på.
- **SendSMS(***telefonnummer, beskjed***)** – Metoden sendes ut en beskjed til en bruker via SMS. Beskjeden som sendes spesifiseres i argumentet *beskjed.* Brukeren som beskjeden sendes til spesifiseres i argumentet *telefonnummer.*
- **SendJ2ME(***telefonnummer,beskjed***)** – Beskjeden lagres i databasen til en brukeren som eier den henter den ut. Beskjeden som sendes spesifiseres i argumentet *beskjed.* Brukeren som beskjeden sendes til spesifiseres i argumentet *telefonnummer*
- **SQLQuery(***sqlStatement***)** - Denne metoden kjører SQL-spørringer av typen *SELECT*. Argumentet *sqlStatement* inneholder selve spørringen som skal utføres. Metoden returnerer resultatet fra spørringen.

- **SQLUpdate(***sqlStatement***) –** Denne metoden kjører SQL-spørringer av typen *DELETE*, *UPDATE* og *SELECT*. Argumentet *sqlStatement* inneholder selve spørringen som skal utføres.
- **WaitForMessage(***telefonnummer***) -** Kalles av *kundeapplikasjonen* med gjevne mellomrom. Metoden leser i databasen en gang i minuttet etter beskjeder fra transportsystemet som tilhører brukeren som identifiseres av argumentet *telefonnummer*. Beskjeder som oppdages returneres til *kundeapplikasjonen*, som starter en ny tråd å kaller denne metoden på nytt.

5.3 Usecase-realisering

Vi vil her beskrive alle brukstilfellene. Vi vil bruke sekvensdiagram for å illustrere hvordan flyten mellom designklassene er. I og med at mange av sekvensdiagrammene er ganske like, vil alle diagrammene bli plassert i appendix C for å øke lesbarheten i teksten.

Bortsett fra brukstilfelle ”5.3.1 Registrer ny bruker” vil vi i alle brukstilfellene anta at brukeren allerede har slått på henholdsvis *kundeapplikasjonen* eller administrator-applikasjonen.
Bortsett fra i brukstilfelle ”5.3.1 Registrer ny bruker” og ”5.3.4 Logg inn” vil vi anta at brukeren allerede er logget inn i transportsystemet.

SMS-delen av systemet designes ikke.

5.3.1 Registrer ny bruker

Alle brukere må registrere seg før de kan begynne å bruke transportsystemet. Figur 21 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Brukeren begynner med å starte *kundeapplikasjonen* som han/hun har lastet ned fra transportselskapets hjemmesider. Ved oppstart vil metoden *startaApp()* automatisk startes. Brukeren vil nå, ved å velge fra menyen, kunne registrere seg i transportsystemet. Brukeren velger menyvalget *Registrer bruker* og metoden *RegistrerNyBruker()* startes. Brukeren vil her kunne legge inn følgende informasjon: Fornavn, etternavn, adresse, postnummer, poststed, telefonnummer, e-post, passord to ganger og eventuelt jobbadressen, jobbpostnummeret og jobbpoststedet.
Når brukeren velger *OK* vil metoden *RegistrerNyBruker()* kalles i *TransportServer*, med oppgitt informasjon som argumenter. Brukerinformasjonen sjekkes mot databasen. Hvis informasjonen blir godkjent, registreres den nye brukeren i transportsystemet og han/hun vil få beskjed om at det nå er mulig logge inn.

Følgende lagres i tabellen *Kunde*:
- *KundeID*: Tellende verdi. Neste verdi settes inn.
- *Fornavn*: Brukerens oppgitte fornavn.
- *Etternavn*: Brukerens oppgitte etternavn.
- *Adresse*: Brukerens oppgitte adresse.
- *Postnummer*: Brukerens postnummer.
- *Poststed*: Brukerens poststed.
- *Telefonnummer*: Brukerens oppgitte telefonnummer.

- *Passord*: Brukerens oppgitte passord.
- *E-post*: Brukerens oppgitte e-post.
- *Jobbadresse*: Brukerens jobbadresse.
- *Jobbpostnummer*: Brukerens jobbpostnummer.
- *Jobbpoststed*: Brukerens jobbpoststed.
- *SituasjonID*: Henter *SituasjonID* fra tabellen *situasjon* som tilsvarer situasjonen *vanlig*.
- *KommunikasjonskanalID*: Henter *KomKanalID* fra tabellen *komkanal* som tilsvarer kommunikasjonskanalen *begge*
- *Online*: Settes til 0.
- *IkkeEndreSituasjon*: Settes til *0*.

5.3.2 Slett bruker

Alle brukere skal kunne slette seg selv fra transportsystemet. Figur 22 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når en bruker ønsker å slette seg selv fra transportsystemet velger han/hun *Slett bruker* fra menyen. Brukeren må da oppgi telefonnummer og passordet før han/hun trykker *OK*. Dette er for at transportsystemet skal være sikkert på at det er rette brukeren som bestiller slettingen.
Når brukeren trykker *OK*, kalles metoden *SlettBruker()* i *TransportServer*. Denne metoden tar telefonnummeret og passordet som ble oppgitt som argumenter. Denne informasjonen sjekkes med databasen, og hvis den stemmer slettes brukeren fra transportsystemet. Brukeren vil få beskjed fra transportsystemet om at han/hun er slettet og blir automatisk logget ut. Dette gjøres ved at *LoggUt()* metoden i *KundeApp* kalles.

Følgende informasjon slettes i gitt rekkefølgen:

- I tabell *DagAbo*: Alle poster som tilhører alle abonnementene som tilhører brukeren med telefonnummeret som ble oppgitt slettes
- I tabell *UnntakAbo*: Alle poster som tilhører alle abonnementene som tilhører brukeren med telefonnummeret som ble oppgitt selttes
- I tabell Abonnement: Alle poster som tilhører brukeren med telefonnummeret som ble oppgitt slettes.
- I tabell Kunde: All informasjon som tilhører brukeren med telefonnummeret som ble oppgitt slettes.

5.3.3 Endre brukerdata

Alle brukere skal kunne endre sin egen brukerdata. Figur 23 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

For å endre noen av sine personlige data velger brukeren *Endre brukerdata* fra menyen i *kundeapplikasjonen*. Metoden *EndreBrukerdata()* kalles da i *KundeApp*. Brukeren får mulighet til å fylle inn følgende felter: Adresse, postnummer, postadresse, telefonnummer, passord, e-post, jobbadresse, jobbpostnummer og jobbpoststed.

Når brukeren trykker *OK* vil metoden *EndreBrukerdata* i *TransportServer* kalles. Alle feltene hvor informasjon ble skrevet vil overføres som argumenter til denne metoden. Feltene som ikke fylles ut, overses. De nye dataene blir så lagret i databasen. Brukeren får beskjed om hva som ble lagret når lagringen er utført.

Følgende informasjon lagres i tabellen *kunde*:

- Adresse: Lagres hvis ny verdi ble oppgitt.
- Postnummer: Lagres hvis ny verdi ble oppgitt.
- Poststed: Lagres hvis ny verdi ble oppgitt.
- Telefonnummer: Lagres hvis ny verdi ble oppgitt.
- Passord: Lagres hvis ny verdi ble oppgitt.
- E-post: Lagres hvis ny verdi ble oppgitt.
- Jobbadresse: Lagres hvis ny verdi ble oppgitt.
- Jobbpostnummer: Lagres hvis ny verdi ble oppgitt.
- Jobbpostadresse: Lagres hvis ny verdi ble oppgitt.

5.3.4 Logg inn

Brukere
Alle brukerer skal kunne logge inn i transportsystemet. Figur 24 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Brukere som ikke er logget inn vil kun få følgende menyvalg i *kundeapplikasjonen*: *Logg inn* og *Registrer ny bruker*. Brukere som er logget inn vil få tilgang til alle valgene som støttes av applikasjonen.

Brukere må logge inn i transportsystemet ved første gangs bruk. Ved alle påloggінger etter dette vil brukeren bli logget automatisk inn.
Ved første gangs bruk må brukeren manuelt velge *Logg inn* fra menyen. Metoden *LoggInn*() i *KundeApp* startes. Her vil brukeren måtte skrive inn telefonnummer og passord før han/hun velger *OK*.
Når brukeren velger *OK* kalles metoden *LoggInn()* i *TransportServer*, med brukerens telefonnummer og passord som argumenter. Informasjonen sjekkes mot databasen. Hvis informasjonen stemmer, logges brukeren inn i transportsystemet og han/hun får tilgang på alle menyvalgene som transportsystemet støtter. Brukeren registreres som *online* i transportsystemet.

Følgende lagres i tabellen *kunde*:

- Online: Settes til 1

Følgende lagres i kundeapplikasjonen ved første gangs logg inn:

- Telefonnummer: Oppgitt telefonnummer lagres i kundeapplikasjonen.
- Passord: Oppgitt passord lagres i kundeapplikasjonen.

Administrator
Administrator skal kunne logge inn i transportsystemet. Figur 25 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Metoden *LoggInn()* i *AdministratorApp* kalles når administrator-applikasjonen startes. Administrator må oppgi brukernavn og passord for å logge inn.
Når administrator trykker *OK*-knappen kalles metoden *LoggInn()* i *TransportServer*, med administratorens brukernavn og passord som argumenter. Denne metoden sjekker om informasjonen stemmer i forhold til databasen. Hvis den stemmer, logges administratoren inn og han/hun kan legge til og/eller slette beskjeder fra transportsystemet.

5.3.5 Logg ut

Bruker
Alle brukere skal kunne logge ut av transportsystemet. Figur 26 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når en bruker ønsker å avslutte *kundeapplikasjonen* velger han/hun *Avslutt* fra menyen. Denne metoden kaller på metoden *LoggUt()* i *KundeApp*. Denne metoden kaller så på metoden *LoggUt()* i *TransportServer* med brukerens telefonnummer som argument. Brukeren registreres som *offline.* Brukeren logges ut av transportsystemet.

Følgende lagres i tabellen *kunde*

- Online: Settes til 0

Administrator
Administrator skal kunne logge ut av transportsystemet. Figur 27 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når administrator ønsker å avslutte administrator-applikasjonen, velger han *Avslutt.* Dette valget kaller metoden *LoggUt*() i *AdministratorApp*, som logger ut administrator og avslutter administrator-applikasjonen.

5.3.6 Registrer nytt abonnement

Alle brukere skal kunne registrere nye abonnementer. Figur 28 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når en bruker ønsker å legge inn et nytt abonnement velger han/hun *Legg til abonnement* fra menyen. Metoden *RegistrerNyttAbonnement*() i *KundeApp* startes. Brukeren skriver inn adressen han/hun ønsker å dra til og klokkeslettet han/hun ønsker å være der. Det blir opp til transportsystemet å finne ut hvilken buss og hvilke holdeplasser som skal benyttes. Brukeren kan også oppgi hvor han/hun befinner seg i minuttene før han/hun ønsker å dra. Hvis ingenting oppgis der, brukes brukerens nåværende lokasjon som verdi her. Brukeren kan også velge hvilke dager abonnementet skal gjelde og hvilke unntak som skal gjelde for dette abonnementet. Når brukeren velger *OK*, kalles metoden *RegistrerNyttAbonnement()* i *TransportServer*. Denne metoden har følgende argumenter: *telefonnummer, tid, destinasjonAdresse, destinasjonPostnummer, destinasjonPoststed, utgangspunktAdresse, utgangspunktPostnummer, utgangspunktPoststed, ruteID, dagID* og *unntakID*. Argumentet *telefonnummer* er brukerens telefonnummer, *tid* er brukerens krav til når han/hun skal være framme på ønsker adresse, *destinasjonAdresse, destinasjonPostnummer og destinasjonPoststed* er brukerens

destinasjonsønske, *utgangspunktAdresse, utgangspunktPostnummer og utgangspunktPoststed* er adressen han/hun befinner seg på i minuttene før avreise, *dagID* er identifikatoren til den/de dagene i uken dette abonnementet skal gjelde, *ruteID* er identifikatoren til den bussruten som gjelder for dette abonnementet og *unntakID* er identifikatoren til det/de unntakene som skal gjelde.

Følgende lagres i tabellen *Abonnement*:

- AbonnementID: Tellende verdi. Neste verdi settes inn.
- RuteID: Identifikatoren til den bussruten som skal brukes
- StartTid: Tid for avgang fra startholdeplass.
- StartHoldeplass: Adressen til startholdeplassen
- StoppTid: Tid for ankomst til ønsket holdeplass
- StoppHoldeplass: Adressen til endeholdeplassen
- KundeID: Identifikatoren til den brukeren som eier dette abonnementet. Bruker telefonnummeret for å finne denne verdien.

Følgende lagres i tabellen *DagAbo*:

- DagID: Identifikatoren til den dagen dette abonnementet skal gjelde
- AbonnementID: AbonnementID'en hentes fra tabellen *Abonnement*. Indikerer hvilket abonnement denne dagen tilhører.

Følgende lagres i tabellen *UnntakAbo*:

- UnntakID: Identifikatoren dette unntaket som gjelder for dette abonnementet.
- AbonnementID: AbonnementID'en hentes fra tabellen *Abonnement*. Indikerer hvilken abonnement dette unntaket tilhører.

5.3.7 Slett abonnement

Alle brukere skal kunne slette sine abonnementer. Figur 29 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når en bruker ønsker å slette et abonnement, velger han/hun *Slett abonnement* fra menyen. Metoden *SlettAbonnement*() i *KundeApp* startes. Når denne metoden startes kalles metoden *FinnAbonnementer()* i *TransportServer*. Denne metoden tar bruker3ns telefonnummer som argument. Ut fra dette argumentet returneres alle abonnementene som denne brukeren eier tilbake til *kundeapplikasjonen*. Fra denne listen kan brukeren velge hvilken abonnement som skal slettes. Brukeren velger det abonnementet som skal slettes og velger *slett*. Metoden *SlettAbonnement*() i *TransportServer* kalles. Argumentet *abonnementID* identifiserer abonnementet som skal slettes. I tillegg slettes også all informasjon som tilhører dette abonnementet.

Følgende informasjon slettes i gitt rekkefølge:

- I tabell DagAbo: Alle poster som har *AbonnementID* som er lik det argumentet som ble overført slettes.
- I tabell UnntakAbo: Alle posten som har *AbonnementID* som er lik det argumentet som ble overført slettes.
- I tabell Abonnement: Posten som har *AbonnementID* som er lik det argumentet som ble overført slettes.

5.3.8 Endre abonnement

Alle brukere skal kunne endre sine abonnementer. Figur 30 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når en bruker ønsker å endre et av sine abonnementer, velger han/hun *Endre abonnement* fra menyen. Metoden *EndreAbonnement()* i *KundeApp* startes. Når denne metoden startes kalles metoden *FinnAbonnement*() i *TransportServer*. Denne metoden tar brukerens telefonnummer som argument. Ut fra dette argumentet returneres alle abonnementene som denne brukeren eier tilbake til *kundeapplikasjonen*. Fra denne listen kan brukeren velge hvilken abonnement som skal endres. Brukeren velger det abonnementet som skal endres og velger *endre.*Metoden *VelgAbonnement()* i *TransportServer* kalles når dette skjer. Argumentet *abonnementID* indikerer hvilket abonnement som skal endres. Metoden leter fram all informasjon om dette abonnementet og returnerer dette til *kundeapplikasjonen*.

Følgende returneres og vises til brukeren:

- Rutenummer: Rutenummeret til bussen
- StartTid: Tidspunktet bussen går
- StoppTid: Tidspunktet bussen er framme
- StartHoldeplass: Adressen til holdeplassen bussen går fra
- StoppHoldeplass: Adressen til holdeplassen bussen skal til
- Dager: Dager i uken abonnementet gjelder
- Unntakene: Unntakene som gjelder for dette abonnementet.

Brukeren kan endre følgende felter:

- *StoppTid* kan endres til det nye tidspunktet han/hun ønsker å være framme på ønsker destinasjon.
- *StartHoldeplass* kan endres til den adressen brukeren kommer til å være i minuttene før han/hun tar bussen.
- *StoppHoldeplass* kan endres til et nytt destinasjonsønske.
- Dag kan endres etter ønsker.
- Unntak kan endres etter ønske.

For å aktivere endringene velger brukeren *OK*. Metoden *EndreAbonnement()* i *TransportServer* kalles. Denne metoden tar følgende argumenter fra kundeapplikasjonen: *AbonnementID, StoppTid, startHoldeplass* og *StoppHoldeplass*. Kun de feltene som ble endret av brukeren inneholder verdier. Ut fra den nye informasjonen kalkuleres kalkulerer ny rutetid og hvilken buss som skal brukes. Databasen oppdateres med de nye verdiene. Den nye tiden for avgang kalkuleres og lagres i feltet *StartTid.*

Følgende informasjon endres i tabellen *Abonnement*:

- *StartTid*: Den nye tiden for avgang.
- *StartHoldeplass*: Den nye adressen til holdeplassen som bussen går fra.
- *StoppTid:* Den nye ankomsttiden.
- *StoppHoldeplass*: Den nye adressen til holdeplassen nærmest ønsket destinasjon.

Følgende informasjon endres i tabellen *DagAbo*:

- *DagID*: Den nye dagen som abonnementet gjelder.

Følgende informasjon endres i tabellen *UnntakAbo:*

- U*nntakID*: Det nye unntaket som gjelder.

5.3.9 Informasjonen som leveres skal være relevant i forhold til brukerens lokasjon

Alle beskjeder som sendes til en bruker skal være relevant i forhold til brukerens lokasjon. Figur 31 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når en bruker får en beskjed fra transportsystemet, må det først sjekkes hvor brukeren befinner seg. Dette gjøres ved at metoden *SjekkLokasjon()* i *KontekstAgent* kalles. Argumentet *telefonnummer* brukes for å identifisere hvilken bruker det skal sjekkes lokasjon på. *SjekkLokasjon()* benytter GSM-posisjonering for å finne ut hvor en bruker befinner seg. Metoden returnerer brukerens lokasjon. Returnert data brukes av *ModusAgent* for å avgjøre om en bruker skal få en beskjed.

5.3.10 Informasjonen som leveres skal være relevant i forhold til tid på døgnet

Alle beskjeder som sendes til en bruker skal være relevant i forhold til tid på døgnet. Figur 32 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når en bruker skal få en beskjed fra transportsystemet, må transportsystemet først sjekke tiden. Både klokkeslettet og dato skal sjekkes. Dette gjøres ved at metoden *SjekkTid()* i *KontekstAgent* kalles. Argumentet *Telefonnummer* identifiserer brukeren som skal sjekkes. Denne metoden sjekker dato og klokkeslett og endrer en brukeres situasjon hvis tiden tilsier dette

5.3.11 Endre brukerens situasjon

Alle brukere skal kunne endre sin egen situasjon. Figur 33 og 34 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Det er to måter å endre en brukers situasjon. Brukeren kan gjøre det manuelt gjennom *kundeapplikasjonen*, og *transportsystemet* kan gjøre det automatisk ut fra tilgjengelig kontekst.

Manuelt
Figur 33 i appendix C viser flyten i henne handlingen.

Når en bruker ønsker å endre sin situasjon, velger han *Endre situasjon* fra menyen i *kundeapplikasjonen*. Metoden *EndreSituasjon()* i *KundeApp* kalles. Brukeren kan nå velge hvilken situasjon han/hun ønsker å være i. For å registrere endringene må brukeren velge *Ok* som kaller metoden *EndreSituasjon()* i *TransportServer*. Brukerens

telefonnummer og identifikatoren til den nye situasjonen overføres som argumenter. Den nye situasjonen lagres i databasen.

Endringer i tabellen *Kunde*:

- SituasjonsID: SituasjonsID-argumentet som ble overført lagres her
- I*kkeEndreSituasjon*: Settes til 1 hvis den nye situasjonen IKKE er *vanlig.* Settes til 0 hvis den nye situasjonen er *vanlig.*

Når brukeren endre situasjon manuelt kan ikke transportsystemet endre den før brukeren endrer den tilbake til *vanlig.* Dette betyr at transportsystemet ikke kan endre en brukers situasjon hvis attributtet *IkkeEndreSituasjon* = 1.

Automatisk
Figur 34 i appendix C viser flyten i denne handlingen.

Transportsystemet kan også endre en brukers situasjon ut fra tilgjengelig kontekst. Dette gjøres bland annet ut fra tid på døgnet og brukerens lokasjon. Når en beskjed sendes ut sjekkes tiden og brukerens lokasjon. Brukerens situasjon endres hvis tiden tilsier dette. Brukerens situasjon endres også hvis lokasjonen tilsier en situasjonsendring.

5.3.12 Endre kommunikasjonskanal

Alle brukerer skal kunne endre sin egen kommunikasjonskanal. Figur 35 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Når en bruker ønsker å endre sin kommunikasjonskanal, velger han *Endre Kommunikasjonskanal* fra menyen i *kundeapplikasjonen*. Metoden *EndreKommunikasjonskanal()* i *KundeApp* kalles. Brukeren kan nå velge hvilken kommunikasjonskanal han/hun ønsker å benytte. For å registrere endringene velger brukeren *OK*. Når dette skjer, kalles *EndreKommunikasjonskanal()* i *TransportServer*, med brukerens telefonnummer og identifikatoren til den nye kommunikasjonskanalen som argument. Den nye situasjonen lagres i databasen.

Endringer i tabellen *Kunde*:

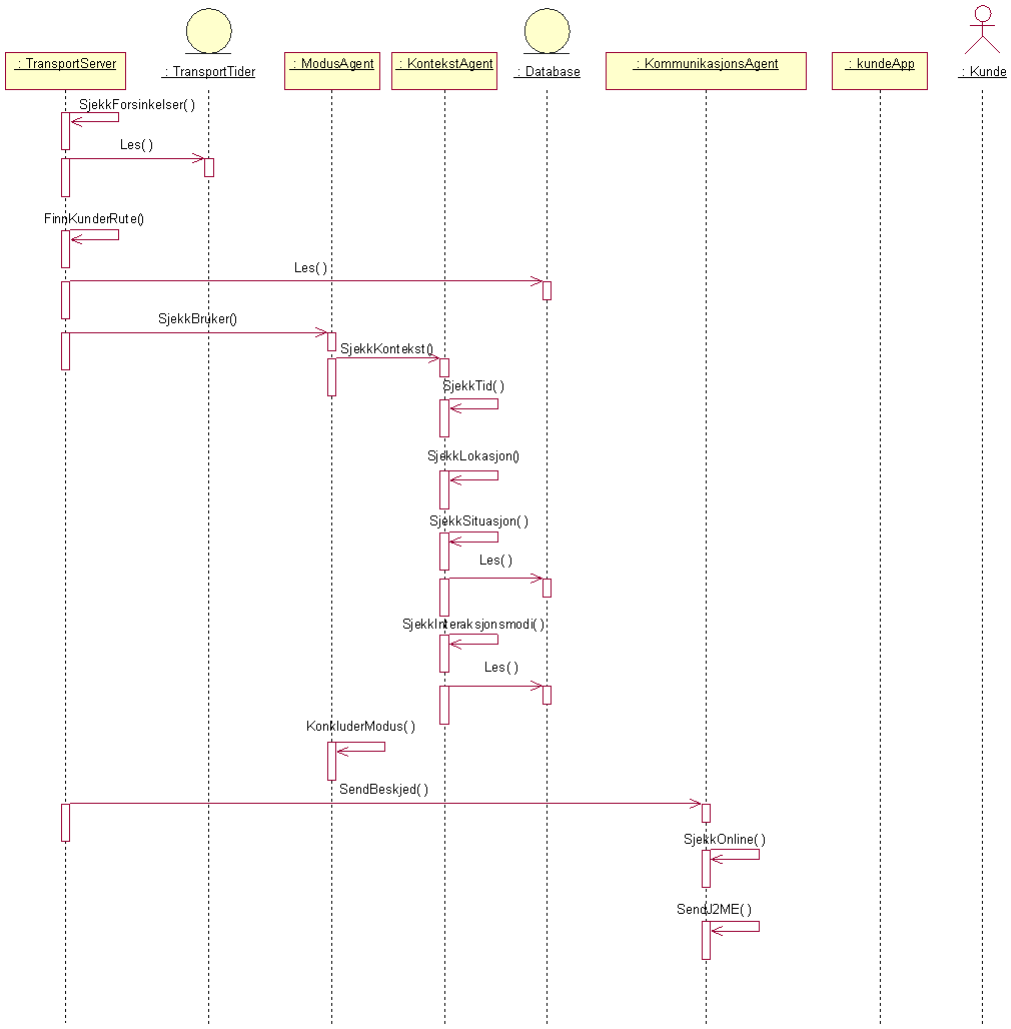
- *KomKanalID*: KomKanalID argumentet som ble overført lagres her

5.3.13 Brukere skal automatisk motta beskjed om forsinkelser og andre viktige hendelser

Alle brukere skal kunne få informasjon om forsinkelser og viktige hendelser fra transportsystemet. Figur 16 illustrerer hvordan flyten i transportsystemet er for denne handlingen.

TransportServer sjekker her *transporttid-databasen* (den eksterne databasen) for forsinkelser. Når en forsinkelse oppdages sjekkes det hvilke brukere som påvirkes av denne forsinkelsen. Dette sjekkes ved at metoden *FinnKunder()* i *TransportServer*

kjøres. Denne metoden sjekker hvilke brukere som påvirkes av forsinkelsen som ble oppdaget og lager en liste over disse. Denne listen inneholder brukerens telefonnummer og abonnementID til det abonnementet det gjelder. For hver bruker i listen kalles metoden *SjekkBruker()* i *ModusAgent* med brukerens telefonnummer som argument. Denne metoden kaller på metoden *SjekkKontekst()* i *KontekstAgent* med telefonnummer som argument. Denne metoden sjekker alle typer kontekst som denne brukeren har. Denne klassen benytter fire metoder for å finne ulik kontekst. Metoden *SjekkTid()* sjekker tiden og endrer en brukers situasjon hvis nødvendig. Metoden *SjekkLokasjon()* sjekker og returnerer brukerens lokasjon. Metoden *SjekkSituasjon()* sjekker og returnerer brukerens situasjon. *SjekkInteraksjonsmodi()* sjekker og returnerer brukerens interaksjonsmodus. Situasjon og interaksjonsmodus leses direkte fra databasen.



Figur 16 Sekvensdiagram send beskjeder

Når all kontekst er funnet returneres den til *ModusAgent*. Metoden *KonkluderModus()* i *ModusAgent* startes med hver type kontekst som argument. Denne metoden konkluderer hvilken modus en bruker befinner seg i ut fra tilgjengelig kontekst og dette returneres til *TransportServer*. Alle brukere som befinner seg i et modus som tilsier at de skal ha meldingen, vil motta denne. Metoden *SendBeskjed()* i

KommunikasjonsAgent kalles en gang for hver bruker som skal ha beskjeden. Metoden tar brukerens telefonnummer, selve beskjeden og brukerens kommunikajsonskanal som argument. Denne metoden sjekker hvordan beskjeden skal sendes ut. Metoden *SjekkOnline()* kjøres for å avgjøre om en bruker er logget på transportsystemet gjennom *kundeapplikasjonen*. Argumentet telefonnummer identifiserer brukeren som skal sjekkes.

Brukerens kommunikajsonskanal og om han/hun er *online* bestemmer hvordan beskjeden sendes ut. Her er det fire muligheter:

- Brukeren er *online* og kommunikasjonskanal er *Java* eller *begge*: Beskjeden sendes via *J2ME-applikasjonen*
- Brukeren er *online* og kommunikasjonskanalen er *SMS*: Beskjeden sendes via *SMS*
- Brukeren er *offline* og kommunikasjonskanalen er *Java*: Beskjeden sendes ikke ut.
- Brukeren er *offline* og kommunikajsonskanalen er *SMS* eller *begge*: Beskjeden sendes via *SMS*.

Hvis beskjeden skal sendes via SMS kalles metoden *SendSMS()*. Hvis beskjeden skal sendes via J2ME-applikasjonen kalles metoden *SendJ2ME()*. Begge metodene tar brukerens telefonnummer og selve beskjeden som argumenter.

SendSMS() sender beskjeden til brukeren via SMS. *SendJ2ME()* lagrer beskjeden i en tabell over beskjeder. Metoden *WaitForMessage()* kjører i *KommunikasjonsAgent* med en tråd for hver telefon som er logget på. Denne metoden sjekker regelmessig i databasen over beskjeder som tilhører hver enkelt bruker. Når en slik beskjed oppdages, returneres denne til *kundeapplikasjonen* som viser beskjeden på telefonen.

Følgende lagres i tabellen *kundebeskjed*:

- BeskjedID: Tellende verdi. Neste verdi settes inn.
- Telefonnummer: Eieren av beskjeden sitt telefonnummer.
- Beskjed: Selve beskjeden
- KundeID: Kundeidentifikator.

5.3.14 Brukere skal kunne bruke sin kalenderapplikasjon til å kommunisere med transportsystemet

Alle brukere skal kunne synkronisere sin kalenderapplikasjon med transportsystemet. Figur 37 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Hvis en bruker har en kalender på sin mobiltelefon kan denne synkroniseres med transportsystemet. Denne funksjonen kan manuelt aktiveres av brukeren, eller den kan utføres hver gang en bruker logger på. Brukeren synkroniserer ved å velge *Synkroniser* fra menyen i *kundeapplikasjonen*. Når dette skjer kalles metoden *SynkroniserKalender()* i *kundeapplikasjonen*. Denne metoden vil da finne kalenderen i telefonen og overføre informasjonen som finnes der til transportsystemet. Metoden *SynkroniserKalender()* i *TransportServer* kalles med kalenderinformasjon og

brukerens telefonnummer som argument. Transportsystemet bruker denne informasjonen til å lage abonnementer som brukeren kan abonnere på.

Følgende lagres i tabellen *Abonnement*:

- *AbonnementID*: Tellende verdi. Neste verdi settes inn.
- *RuteID*: Identifikatoren til den bussruten som skal brukes
- *StartTid*: Tid for avgang fra start holdeplassen.
- *StartHoldeplass*: Adressen til startholdeplassen
- *StoppTid*: Tid for ankomst til ønsket holdeplass
- *StoppHoldeplass*: Adressen til endeholdeplassen
- *KundeID*: Identifikatoren til den brukeren som eier dette abonnementet. Bruker telefonnummeret for å finne denne verdien.

Følgende lagres i tabellen *DagAbo*:

- *DagID*: Identifikatoren til den dagen dette abonnementet skal gjelde
- *AbonnementID*: *AbonnementID*'en hentes fra tabellen *Abonnement*. Indikerer hvilket abonnement denne dagen tilhører.

5.3.15 Administrator skal kunne legge til manuelle beskjeder

Administrator skal kunne legge til manuelle beskjeder i transportsystemet. Figur 38 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Administrator kan legge til manuelle beskjeder fra aministrator-applikasjonen. Når administrator velger *Legg til manuell beskjed* fra menyen startes metoden *LeggTilManuellBeskjed()*.

Denne metoden gir administrator mulighet til følgende ting:

- Skrive inn selve beskjeden
- Skrive inn hvilken bussrute som beskjeden gjelder for
- Legge ved et bilde
- Legg ved en lydfil
- Skrive inn gyldighetstiden til beskjeden

Når administrator velger *OK* kalles metoden *LeggTilManuellBeskjed*() i *TransportServer*. Denne metoden tar følgende fem argumenter som lagres i databasen: Beskjed, ruteID, bildefil, lydfil og beskjedens gyldighets tid

Endringer i tabellen *beskjed*:

- BeskjedID: Tellende verdi. Neste verdi settes inn.
- Beskjed: Selve beskjeden.
- Bildefil: Navnet på bildefilen.
- Lydfil: Navnet på lydfilen.
- Gyldighetstid: Indikerer hvor lenge denne beskjeden gjelder.

Endringer i tabellen *beskjedrute*:

- BeskjedID: Identifikatoren til beskjeden
- RuteID*:* Identifikatoren til den ruten denne beskjeden gjelder.

Når en beskjed legges til i databasen sjekkes det hvilke brukere som påvirkes. Alle som påvirkes får tilsendt beskjeden. Når alle som skal motta beskjeden har mottatt den, slettes den fra databasen.

5.216 Administrator skal kunne slette manuelle beskjeder

Administrator skal kunne slette beskjeder fra transportsystemet. Figur 39 i appendix C illustrerer hvordan flyten i transportsystemet er for denne handlingen.

Alle beskjeder som er lagt inn i transportsystemet kan slettes av administrator. Administrator kan velge *Slett beskjed* fra administrator-applikasjonen. Når dette skjer startes metoden *SlettManuellBeskjed()* i *AdministratorApp*. Denne metoden viser en liste over alle beskjeder. Administrator velger den/de metodene han/hun vil slette ved å velge dem med *checkboxes.* Når administrator trykker på *slett*-knappen kalles metoden *SlettManuellBeskjed()* en gang for hver beskjed som skal slettes. Denne metoden tar *BeskjedID* til argument.

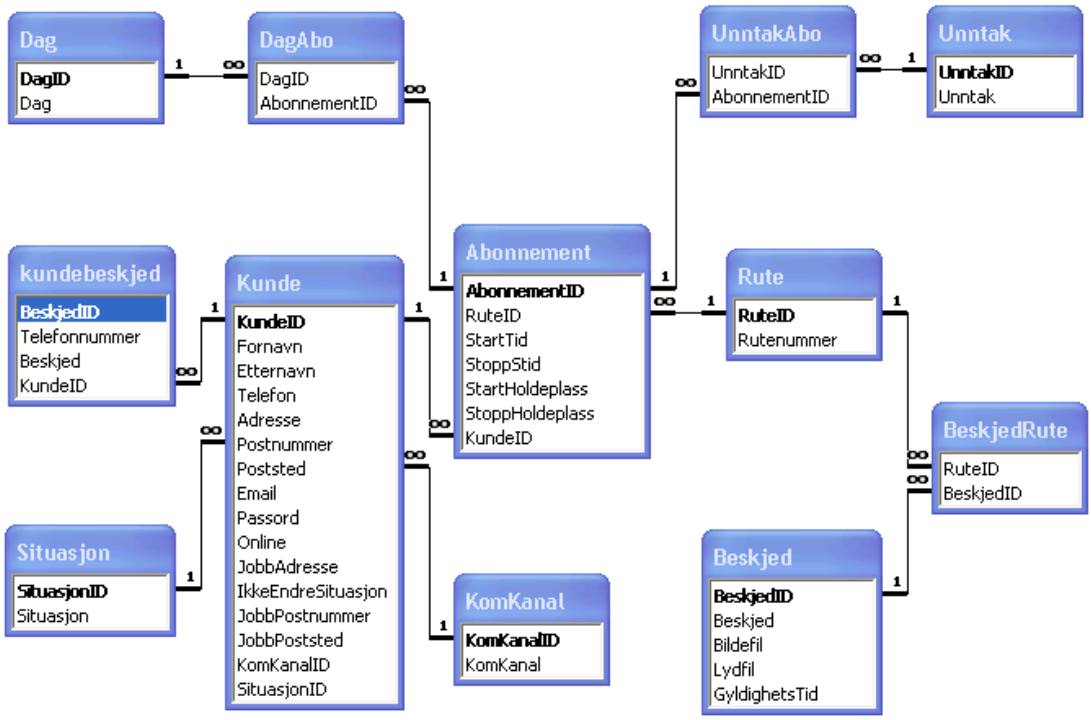
Følgende informasjon slettes i gitt rekkefølge:

- I tabell *BeskjedRute*: Alle poster som har *BeskjedID* som er lik det argumentet som ble overført.
- I tabell Beskjed: Posten som har *BeskjedID* som er lik det argumentet som ble overført.

5.3 TransportSystemdatabasen

Databasen som beskrives her er den interne databasen som inneholder all informasjon om alle brukere, alle abonnementet, og all kontekst om hver bruker. For at transportsystemet skal kunne tas i bruk, må tilgang til en ekstern database også eksistere. Den eksterne databasen vil tilhører transportselskapet som skal benytte transportsystemet og må inneholde informasjon om rutetider og forsinkelser.

Figur 17 illustrerer hvordan den interne databasen for transportsystemet er bygd opp. Denne databasen er en MySql database og vedlikeholdes av MySql versjon 2.3 eller nyere.



Figur 17 **Transportsystemdatabasen**

Under følger en nøye beskrivelse av de ulike tabellene i databasen og deres attributter og egenskaper.

5.4.1 kunde

Attributt	KundeID	Fornavn	Etternavn	Adresse	Postnummer	Poststed
Type	*Integer/Teller*	*String*	*String*	*String*	*Integer*	*String*

Email	Telefon	Jobbadresse	Jobbpostnummer	Jobbpoststed
String	*String*	*String*	*Integer*	*String*

Passord	Online	EndreSituasjon	SituasjonID	KomKanalID
String	*Integer*	*Integer*	*Integer*	*Integer*

Tabellen *kunde* inneholder personlig informasjon om hver enkelt bruker. Kundetabellen består av følgende attributter:

KundeID: Hver bruker har et kundenummer som brukes i hele databasen som identifikator. Tellende verdi.
Fornavn: Brukerens fornavn.
Etternavn: Brukerens etternavn.
Adresse: Brukerens bostedsadresse.
Postadresse: Brukerens postnummer.
Poststed: Brukerens poststed.
Email: Brukerens e-post.

Telefon: Brukerens telefonnummer. Mobiltelefonnummeret brukes som identifikator hver gang brukeren logger på.
Jobbadresse: Brukerens jobbadresse.
Jobbpostnummer: Brukerens jobbpostnummer.
Jobbpoststed: Brukerens jobbpoststed.
Passord: Passord som brukes ved pålogging.
Online: Indikerer om brukeren er logget på transportsystemet eller ikke. Har verdi *'1'* hvis brukeren er pålogget og *'0'* hvis brukeren ikke er pålogget.
EndreSituasjon: Indikerer om transportsystemet har lov til å endre en brukers situasjon. Verdien *'1'* indikerer at transportsystemet *ikke* har lov til å endre en brukers situasjonen. Verdien *'0'* indikerer at transportsystemet har lov til å endre en brukers situasjon.
SituasjonID: Identifikator fra tabellen *situasjon* som bestemmer brukerens situasjon
KomKanalID: Identifikator fra tabellen *komkanal* som bestemmer brukerens kommunikasjonskanalvalg.

5.4.2 situasjon

Attributt	SituasjonID	Situasjon
Type	*Integer/Teller*	*String*

Tabellen *situasjon* inneholder de ulike situasjonene en bruker kan befinne seg i. Tabellen består av følgende attributter:

SituasjonID: Situasjonsidentifikator. Tellende verdi.
Situasjon: Navnet på situasjonen.

Følgende situasjoner finnes:

1. **Vanlig** – Ruteoppdateringer sendes. Beskjed fra administrator sendes.
2. **Møte** – Ruteoppdateringer sendes ikke. Beskjeder fra administrator sendes når bruker går over i situasjon *vanlig*.
3. **Ferie** – Ruteoppdateringer sendes ikke. Beskjeder fra administrator sendes ikke.
4. **Borte** – Ruteoppdateringer sendes ikke. Beskjeder fra administrator sendes.

5.4.3 abonnement

Attributt	AbonnementID	StartTid	StartHoldeplass	StoppTid
Type	*Integer/Teller*	*String*	*String*	*String*

StoppHoldeplass	KundeID	RuteID
String	*Integer*	*Integer*

Tabellen *abonnement* inneholder informasjon om alle abonnementene som er lagret i transportsystemet. Tabellen består av følgende attributter:

AbonnementID: Abonnementsidentifikator/nummer. Tellende verdi. Identifiserer abonnementet.
StartTid: Tiden bussen går fra holdeplassen
StartHoldeplass: Holdeplassen bussen går fra

StoppTid: Tiden bussen er framme på destinasjonsholdeplassen
StoppHoldeplass: Holdeplassen bussen stopper på
KundeID: Kundeidentfikatoren til den brukeren dette abonnementet gjelder
RuteID: Identifikatoren til den ruten dette abonnementet gjelder.

5.4.4 komkanal

Attributt	KomKanalID	KomKanal
Type	*Integer/Teller*	*String*

Tabellen *komkanal* inneholder de ulike kommunikasjonskanalene en bruker kan benytte. Tabellen består av følgende attributter:

KomKanalID: Kommunikasjonskanalidentifikatoren. Tellende verdi.
KomKanal: Navnet på kommunikasjonskanalen

Følgende kommunikasjonskanaler finnes:

1. **Begge** – Informasjon sendes til *kundeapplikasjonen* hvis denne er på. Hvis *kundeapplikasjonen* ikke er på sendes informasjonen via SMS.
2. **Java**– All informasjon sendes kun til *kundeapplikasjonen*, og kun hvis denne er på.
3. **SMS** – All informasjon sendes via SMS uavhengig om *kundeapplikasjonen* er på.

5.4.5 rute

Attributt	RuteID	Rutenummer
Type	*Integer/Teller*	*Integer*

Tabellen *rute* inneholder alle rutenummerene som eksisterer i et transportselskap. Tabellen består av følgende attributter:

RuteID: Rutenummeridentifikatoren. Tellende verdi.
Rutenummer: Rutenummeret

5.4.6 beskjedrute

Attributt	RuteID	BeskjedID
Type	*Integer*	*Integer*

Tabellen *beskjedrute* binder sammen en beskjed og den ruten beskjeden gjelder. Tabellen består av følgende attributter:

RuteID*:* Rutenummeridentifikatoren. Tellende verdi.
BeskjedID*:* Beskjedidentifikatoren

5.4.7 beskjed

Attributt	BeskjedID	Beskjed	Bildefil	Lydfil	GyldighetsTid
Type	*Integer/Teller*	*String*	*String*	*String*	*Integer*

Tabellen *beskjed* inneholder alle manuelle beskjeder en administrator legger inn. Tabellen består av følgende attributter:

BeskjedID: Beskjedidentifikatoren. Tellende verdi.
Beskjed: Selve beskjeden.
Bildefil: Bildefil oppgis hvis administrator ønsker å legge ved et bilde.
Lydfil: Lydfil oppgis hvis administrator ønsker å legge ved lyd.
GyldighetsTid: Indikerer når gyldighetstiden for denne beskjeden går ut.

5.4.8 dagabo

Attributt	DagID	AbonnementID
Type	*Integer*	*Integer*

Tabellen *dagabo* kobler sammen abonnementer og hvilke dager de gjelder. Tabellen består av følgende attributter:

DagID: Dagidentifikator.
AbonnementID*:* Abonnementidentifikator

5.4.9 dag

Attributt	DagID	Dagnavn
Type	*Integer/Teller*	*String*

Tabellen *dag* inneholder all dagvalgene som kan gjelde for et abonnement. Tabellen inneholder følgende attributter.

DagID: Dag identifikator. Tellende verdi.
Dag: Navnet på dagen.

Følgende dager finnes:

1. Søndag
2. Mandag
3. Tirsdag
4. Onsdag
5. Torsdag
6. Fredag
7. Lørdag
8. Hverdager – mandag til og med fredag
9. Daglig – Alle dager i uken

5.4.10 unntakabo

Attributt	UnntakID	AbonnementID
Type	*Integer*	*Integer*

Tabellen *unntakabo* kobler sammen de unntakene som gjelder for et abonnement. Tabellen består av følgende attributter:

UnntakID: Unntakidentifikator.
AbonnementID: Abonnementindetifikator.

5.4.11 unntak

Attributt	UnntakID	Unntak
Type	*Integer/Teller*	*String*

Tabellen *unntak* inneholder alle unntakene som kan gjelder for et abonnement. Tabellen består av følgende attributter:

UnntakID: Unntakidentifikator. Tellende verdi.
Unntak: Navnet på unntaket.

Følgende unntak finnes:

1. Send beskjed også på helligdager
2. Send beskjed også når brukeren er i situasjon *ferie*

5.4.11 kundebeskjed

Attributt	BeskjedID	Beskjed	Telefonnummer	KundeID
Type	*Integer/Teller*	*String*	*String*	*Integer*

Når en bruker er *online* og skal motta en beskjed, lagres den i tabellen k*undebeskjed*. Tabellen består av følgende attributter:

BeskjedID: Beskjedidentifikatoren. Tellende verdi.
Telefonnummer: Telefonnummeret til den brukeren som skal ha beskjeden.
Beskjed: Selve beskjeden.
KundeID: Identifiserer brukeren som eier beskjeden.

I dette kapittelet har vi gått gjennom designet til transportsystemet, samt sett på hvordan den interne databasen er bygget opp. I neste kapittel ser vi på hva som er implementert av *transportsystemet.*

6 Implementasjon og testing

6.1 Introduksjon

Dette kapittelet beskriver hva som er implementert, hva som ikke er implementert og hvordan transportsystemet er implementert og testet.

6.2 Hva er implementert

Transportsystemet deles som nevnt opp i følgende moduler: *administrator-applikasjonen, kundeapplikasjonen, transportserver, modusagent, kommunikasjonsagent* og *kontekstagent*. Deler av alle modulene er implementert, bortsett fra administratorapplikasjonen.

Følgende krav realiseres i demoapplikasjonen:

- Krav 1: Registrer Ny Bruker
- Krav 2: Slett Bruker
- Krav 3: Endre brukerdata
- Krav 4: Logg inn
- Krav 5: Logg ut
- Krav 8: Registrer nytt abonnement
- Krav 11: Endre situasjon (manuelt)
- Krav 12: Endre kommunikasjonskanal
- Krav 13: Brukere skal automatisk motta beskjed om forsinkelser og andre viktige hendelser (delvis implementert)

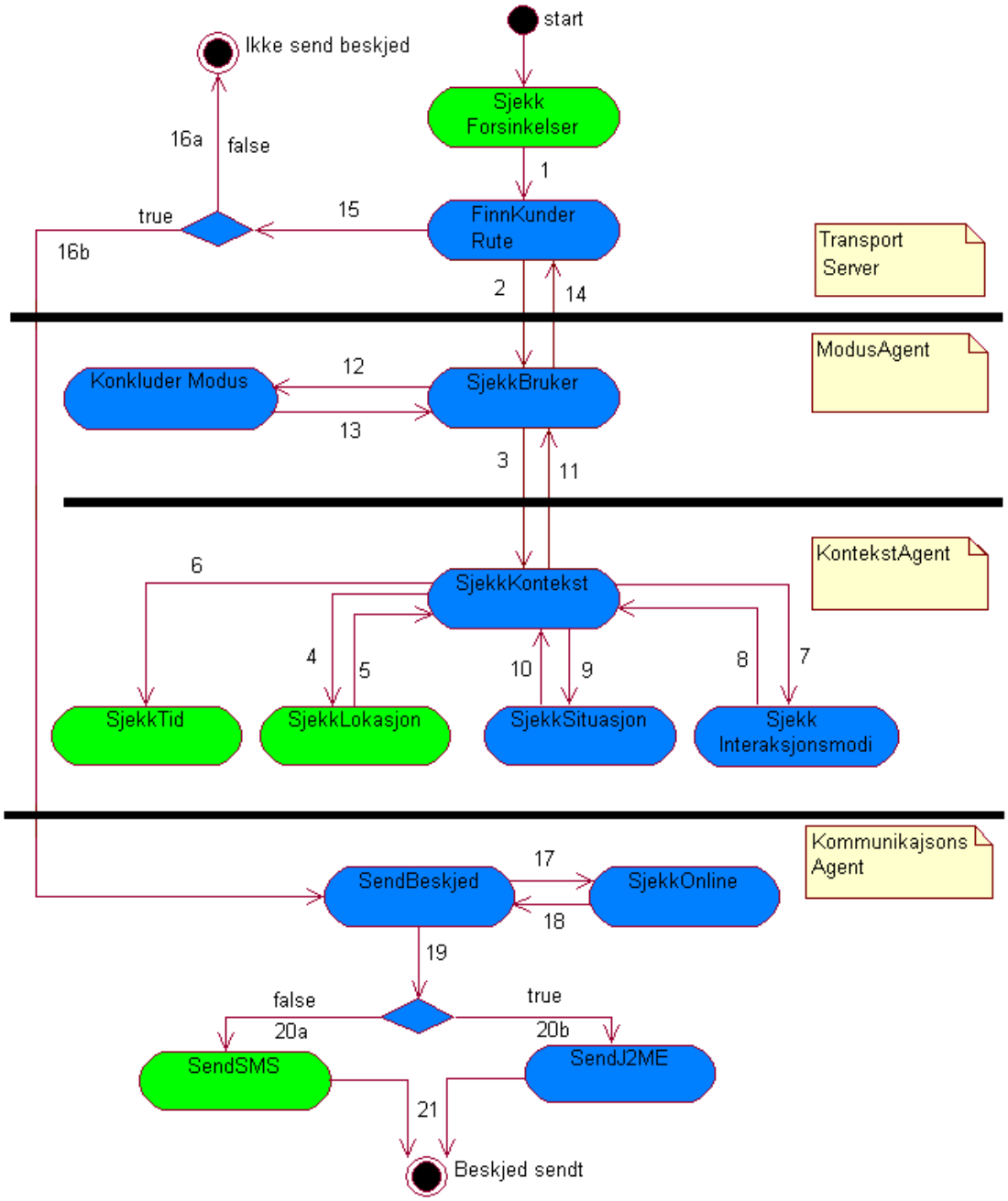
Dette betyr at de metodene i de ulike modulene/agentene som trengs for å tilfredsstille disse kravene, er implementert. Av disse, er krav 13 det mest interessante kravet, mens krav 1,2,3,4,5,8,11 og 12 er et minimum som bør være tilstede for at applikasjonen skal kunne demonstreres. Krav 8 er bare delvis utviklet, men det fungerer tilfredsstillende for å kunne demonstrere oppgaven.

Kundeapplikasjonen kontakter *TransportServer* direkte når krav 1,2,3,4,5,8,11 eller 12 skal utføres. Ingen av de andre modulene/agentene brukes når disse kravene utføres.

Krav 13 er den viktigste mekanismen i transportsystemet. Den brukes for å avgjøre at en forsinkelse eksisterer, den finner ut hvem som ønsker informasjon om denne forsinkelsen, samt hvordan den skal sendes ut. Mekanismen er kompleks og benytter funksjonalitet i alle agentene i transportsystemet. Selve metoden som finner forsinkelser, er ikke implementert, men mye av funksjonaliteten som avgjør til hvem og hvordan en beskjed sendes ut er implementert.

Figur 17 er et aktivitetsdiagram som viser hvilke metoder som kalles i de ulike modulene/agentene når en forsinkelse oppdages. Alt som er blått indikerer elementer/metoder som er implementert, mens alt som er grønt indikerer elementer/metoder som ikke er implementert eller bare er delvis implementert. Flyter som går over en svart linje, indikerer kommunikasjon mellom ulike moduler i

transportsystemet. Hvordan denne kommunikasjonen foregår beskrives i neste delkapittel (6.3 Kommunikasjon).



Figur 18 Hendelsesflyt når forsinkelse oppdages

Figur 18 viser flyten grafisk. Under følger en forklaring.

1. Forsinkelse oppdages og FinnKunderRute() kalles for å finne brukere som påvirkes av forsinkelsen
2. SjekkBruker() kalles for hver bruker som påvirkes av forsinkelsen
3. SjekkKontekst() kalles for å finne kontekst om brukeren
4. SjekkLokasjon() kalles for å finner brukerens lokasjon
5. Brukerens lokasjon returneres

6. SjekkTid() kalles for å avgjøre om tiden påvirker brukerens situasjon slik at beskjeder ikke skal sendes ut
7. SjekkInteraksjonsmodi() kalles for å finne brukerens interaksjonsmodus
8. Brukerens interaksjonsmodus returneres
9. SjekkSituasjon() kalles for å sjekke brukerens situasjon
10. Brukerens situasjon returneres
11. Brukerens kontekst returneres
12. KonkluderModus() kalles for å avgjøre om brukeren skal motta beskjed om forsinkelsen ut fra tilgjengelig kontekst
13. Brukerens modus returneres
14. Brukerens modus returneres. Indikerer om brukeren skal ha beskjeden eller ikke
15. Sjekker brukerens modus og ser om han/hun skal ha beskjeden
16. a. Brukeren ønsker ikke beskjeden. Avslutt.
16. b. Brukeren ønsker beskjeden. SendBeskjed() kalles.
17. SjekkOnline() kalles for å sjekke om brukeren er koblet til systemet via J2ME applikasjonen.
18. Returnerer *true* hvis brukeren er *online*, og *false* hvis brukeren er *offline*
19. Sjekker om SjekkOnline() returnerte *true* eller *false*
20. a. Brukeren er *offline.* SendSMS() kalles for å sende beskjeden om forsinkelse via SMS.
20. b. Brukeren er *online.* SendJ2ME() kalles for å sende beskjeden om forsinkelse via J2ME-applikasjonen.
21. Avslutt

En nærmere beskrivelse av hva som skjer i de ulike metodene i figuren beskrives i kapittel 5.2

Fra figuren ser vi at følgende metoder ikke er implementert:

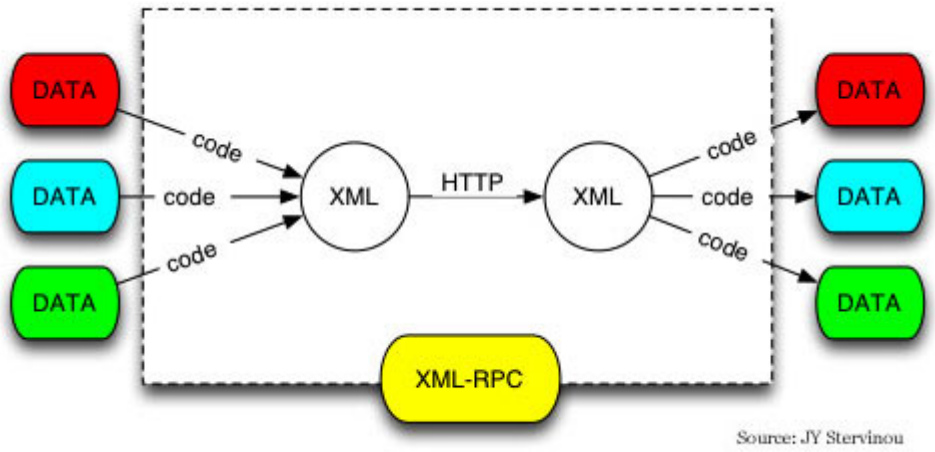
- *SjekkForsinkelser*: For å kunne sjekke etter forsinkelser i rutetider, må vi ta kontakt med en ekstern database som inneholder ruteinformasjon og informasjon om forsinkelser i rutene. Da kun en demoapplikasjon implementeres, er det ikke satt opp tilgang til databasen til et eksisterende kollektivtransportfirma.
- *SendSMS*: For å kunne sende SMS til en bruker, måtte vi hatt tilgang til en *SMS-server*. En slik maskin var ikke tilgjengelig under denne oppgaven.
- *SjekkLokasjon*: For å sjekke en brukers lokasjon, villle vi måtte hatt tilgang til GSM-posisjonering.
- *SjekkTid*: For å avgjøre om en bruker skal endre situasjon ut fra tiden, måtte vi hatt tilgang til brukerens kalender. Siden dette ikke er implementert, blir heller ikke denne metoden implementert.

6.3 Kommunikasjon

Kommunikasjon mellom de ulike modulene (ikke *kundeapplikasjonen*) går via Apache sin XML-RPC implementasjon. Kommunikasjon mellom *kundeapplikasjonen* og transportsystemet går via Enhydras kXML-RPC implementasjon på kundesiden og Apache XML-RPC på serversiden.

XML-RPC og Apache XML-RPC

Apache XML-RPC er en Java-implementasjon av XML-RPC. XML-RPC er en populær protokoll som bruker XML over HTTP for å implementere fjerne metodekall[26]. XML-RPC er et sett av implementasjoner som tillater applikasjoner som kjører på ulike operativsystem i ulike miljø å gjøre prosedyrekall over Internett [28]. HTTP fungerer her altså som transportprotokoll, mens metodekallet er skrevet i XML. XML-RPC er skrevet for å være enkelt, samtidig som det lar komplekse datastrukturer bli overført[28].



Figur 19 XML-RPC oversikt[28]

Figur 19 er hentet fra hjemmesiden [28] til XML-RPC og viser hvordan XML-RPC fungerer sett fra et overordnet logisk nivå. Data fra den ene siden sendes som XML via HTTP til den andre siden.

Enhydra kXML-RPC

Enhydras kXML-RPC er en J2ME implementasjon av XML-RPC protokollen og er bygget over Enhydras kXML parseren [38]. kXML parseren er en XML *pull parser* beregnet for mobile enheter[29]. kXML-RPC gir J2ME enheter en mekanisme for å utveksle data mellom en mobil enhet og for eksempel en server. J2ME har i utgangspunktet ingen støtte for RPC, og kXML-RPC gir derfor på en enkel måte tilgang til dette. kXML-RPC er utmerket for å utveksle enkle beskjeder over et smalbåndsnettverk som GPRS. kXML-RPC har kun støtte for klientsiden.

```
<methodCall>
  <methodName>TransportServer.LoggInn</methodName>
  <params>
    <param>
      <value>
        <string>91368382</string>
      </value>
    </param>
    <param>
      <value>
        <string>passord</string>
      </value>
    </param>
  </params>
</methodCall>
```

Over ser vi et eksempel på hvordan XML'en ser ut når *kundeapplikasjonen* kaller LoggInn() metoden i *TransportServer*. Det som overføres er: navnet på serveren, metodenavnet og argumentene (telefonnummer og passord). I tillegg til XML'en, vil et XML-RPC kall også inneholder informasjon som er spesifikk for HTTP protokollen (vises ikke her).

6.4 Utviklingsverktøy / plattform

Plattform

TransportServer og alle *Agentene* i transportsystemet er implementert i Java, og derfor har det ikke noe å si hvilken plattform transportsystemet kjører på. Så lenge Java VM er installert på maskinene som skal kjøre transportsystemet, er det i teorien uviktig hvilket operativsystem disse maskinene kjører. Transportsystemet er bare testet på windowsbaserte (Windows XP og Windows Server 2000) maskiner, og bør likevel derfor kjøres på en slik maskin for å unngå eventuelt uforutsette problemer.

Kundeapplikasjonen er implementert i Java2 (J2ME), og alle andre deler av transportsystemet er implementert i Java2 (J2SE).

Støtteverktøy

Følgende verktøy er benyttet under utviklingen

Microsoft Word 2002 – Oppgaven er skrevet i MS Word

Microsoft Access 2002 – Databasen er illustrert ved hjelp av mS Access

XEmacs for Windows – Alle delene av transportsystemet er implementert i XEmacs

MySQL Front 2.1 – Manuell databasehåndtering underveis i oppgaven er utført ved hjelp av MySQL Front 2.1

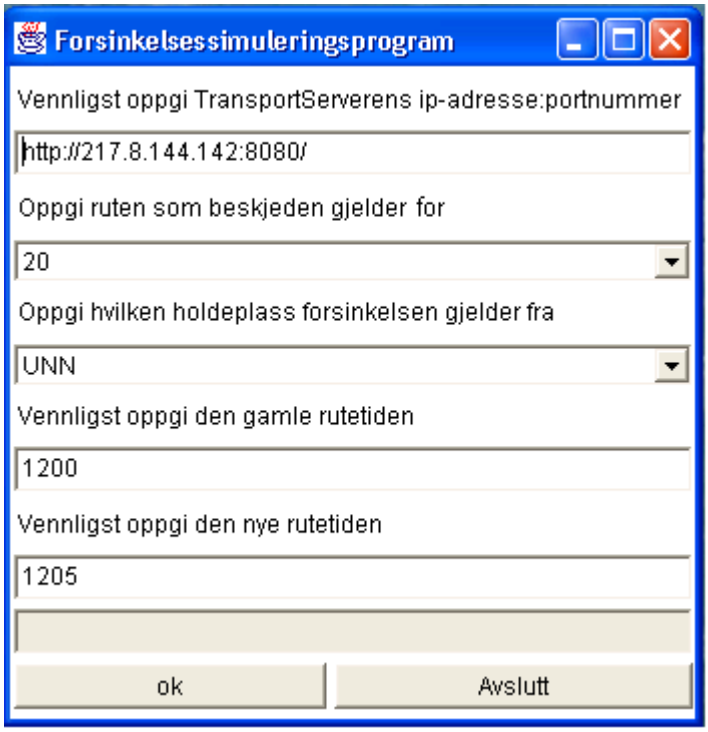
Rational Rose – Transportsystemet er modellert ved hjelp at Rational Rose

J2ME Wireless Toolkit 1.0.4 – Transportsystemet er simulert og delvis testet ved bruk av Wireless Toolkit

MySql 2.23 – Databasen i transportsystemet håndteres av MySql 2.23

6.5 Testing av demoapplikasjonen

For lettere å illustrere hvordan demoen fungerer, er det implementert et testprogram (testClient.java) . Dette programmet brukes for å simulere at en forsinkelse oppdages. Figur 20 viser hvordan dette testprogrammet ser ut.



Figur 20 Screenshot: testprogram

Følgende felter er tilgjengelig:

- TransportServerens ip-adresse og portnummer. Adressen som kommer opp når testprogrammet startes vil fungere slik at systemet kan testes.
- Rutenummeret som forsinkelsen gjelder for.
- Holdeplassen som forsinkelsen gjelder fra.
- Transportmidlets egentlige rutetid fra oppgitt holdeplass.
- Den nye tiden som gjelder fra oppgitt holdeplass.
- Grått felt over knapperaden gir tilbakemeldinger og feilmeldinger.
- Ok-knapp som utfører simuleringen.
- Avslutt-knapp avslutter testprogrammet.

Når en simulering utføres, tar testprogrammet kontakt med *TransportServeren* og ”*sier*” at en forsinkelse er oppdaget. Hendelsen som beskrives i figur 17 utføres. Dette betyr at alle de riktige metodene i de ulike agentene kalles. Kort fortalt sjekkes det om det finnes noen abonnementer som påvirkes av denne forsinkelsen. Hvis noen abonnementer finnes, vil en forsinkelsesmelding sendes ut til de brukere som har nytte av en slik melding. Siden *SendSMS()* metoden ikke er utviklet, vil meldingen kun sendes ut hvis kunden er logget på systemet via *kundeapplikasjonen*.

Det er altså viktig at det først registreres et abonnement på de verdiene som det skal simuleres en forsinkelse på, og at brukeren som abonnementet gjelder for er logget inn via kundeapplikasjonen. Det er også viktig å passe at abonnementet er registrert med riktig dag. Dette betyr at hvis en forsinkelse skal simuleres på en tirsdag, må abonnementet som simuleringen er ment å skal påvirke gjelde for en tirsdag. Hvis ikke vil ingen melding sendes ut.

Vi har i dette kapittelet sett på hva som er implementer og hvordan dette er gjort. I neste kapittel vil vi diskutere hva som er gjort, samt konkludere hva vi har lært.

7 Konklusjon

I dette kapittlet ser vi på hva vi har oppnådd med oppgaven og prøver å konkludere hva vi har lært underveis. Vi avslutter med å se på hva mer som kan gjøres på implementasjonssiden.

7.1 Oppsummering

Kontekstinformasjon og kontekstsensitivitet har vært viktige begreper i denne oppgaven. Vi har sett på hvordan ulike kontekst kan brukes for å lage et system mellom kunde og et kollektivtransportselskap, som tilpasser systemets interaksjonsmodi i ulike situasjoner.

Problemstilling i oppgaven var: **Hvordan kan kontekstinformasjon benyttes til tilpasning av interaksjonsmodi mot mobile terminaler.**

De første kapitlene gikk gjennom bakgrunnen for oppgaven, problemstilling, mål og teori. Dette gir et grunnlag for å forstå grunnen til å utvikle systemet som er spesifisert og designet i denne oppgaven.

En demonstrator av systemet ble implementert for å vise at prosjektet er gjennomførbart, samt for å illustrere mulig nytte av å benytte seg av et slikt system mellom kunde og kollektivtransportselskap.

Til sammen er begge målene med oppgaven innfridd. Målene var å

1. spessifisere, designe og modellere et kontekstsensitivt system for kommunikasjon mellom kunde og kollektivtransportselskap
2. implementere en demonstrator av dette systemet.

7.2 Relatert arbeid

7.2.1 Informasjonssystemer for transport

WINLAND Transport (Telenor FoU)
WINLAND Transports formål er å bidra til økt tilgjengelighet til kollektivtransport og mer differensierte transporttjenester. Kunnskap om anvendelse av allment tilgjengelige mobile informasjonstjenester som er tilpasset den enkelte kunde, skal bidra til framtidige løsninger for kollektivtransport. Hovedmålet i prosjektet er å utvikle mobile IKT-løsninger som muliggjør differensierte informasjonstjenester innen kollektivtransport [35].

IBIS "Integrerte Betalings – og Informasjonssystemer for persontrafikk" (SINTEF)
Målsettingen med IBIS er å undersøke hvordan man kan bruke ny teknologi for å øke bruken av kollektivtransport [4]. Prosjektet tester ut bruken av sanntids ruteinformasjon, altså informasjon om når bussen faktisk kommer. 3 metoder ble brukt for å formidle denne informasjonen.

- Sanntids ruteinformasjon på monitor på utvalgte holdeplasser

- Sanntids ruteinformasjon på Internett
- Sanntids ruteinformasjon til mobiltelefon ved bruk av SMS

7.2.2 Kontekstsensitive tjenester

Netcom Buddy (Netcom)
Netcom buddy er en mobil tjeneste som lar brukere spore opp sine venner via deres mobiltelefoner. Vennenes lokasjon finnes ved bruk av GSM-posisjonering. Tjenesten må godkjennes av de kundene som ønsker å dele sin lokasjon. Tjenesten er kun tilgjengelig for Netcoms kunder [39].

GAID (Telenor)
GAID er en kontekstsensitiv demonstratortjeneste for mobile brukere, som bl.a. tar utgangspunkt i dagens papirbaserte tilgjengelighetsguider beregnet for funksjonshemmede. Slike guider rangerer interessepunkter (POI'er) ut fra kriterier for tilgjengelighet mht ulike funksjonshemminger (syn, hørsel, bevegelse, allergi). Rangeringen av POI'en bestemmes av en totalvurdering av de ulike kriteriene som gjelder innenfor en funksjonshemming. GAID har fokus på hvordan informasjon i dagens tilgjengelighetsguider kan brukes i en kontekstsensitiv byguide, slik at skreddersømmen blir bedre tilpasset den individuelle brukeren. En personlig skreddersøm gjør at informasjon om tilgjengelighet også blir mer relevant for brukere som ikke er permanent funksjonshemmet.

7.2.3 J2ME-applikasjoner

Det finnes mange steder på Internett der man kan laste ned J2ME-applikasjoner. Mest utbrett er spill. Et par eksempler på slike steder er http://www.myphonegames.co.uk/ og http://www.mobile-phone-games.ws/.

7.3 Diskusjon

Valg av programmeringsspråk

Alle deler av systemet er implementert i Java. Kundeapplikasjonen er implementert i J2ME og alle agentene og transportserveren er implementert i J2SE.

Selv om kundeapplikasjonen ble implementert i J2ME, kunne resten av systemet vært implementeres i et annet språk enn Java, uten at dette i teorien ville by på spesielle problemer. Grunnen til dette er at XML-RPC ble brukt for å kommunisere mellom de ulike delene av systemet. Det finnes implementasjoner for XML-RPC i følgende programmeringsspråk: Perl, Python, Java, Frontier, C/C++, Lisp, PHP, Microsoft .NET, Rebol, Real Basic, Tcl, Delphi, WebObjects og Zope [28]. Denne listen utvides stadig. Dette betyr at vi kunne brukt kXML-RPC for J2ME på klientsiden og for eksempel XML-RPC for C++ på serversiden.

Vi kunne også valgt å ikke implementere kundeapplikasjonen i J2ME. P800 støtter nemlig Mophun [33], som er en alternativ måte å programmere applikasjoner for mobiltelefoner på. Det er en del forskjeller mellom Mophun og J2ME. Blant disse er:

1. Mophun programmeres i C eller C++, og eksekverer derfor raskere enn J2ME-applikasjoner.
2. Mophun er beregnet for programmering av spill, og gir derfor bedre støtte for lyd
3. Det er foreløpig bedre støtte i mobiltelefoner for J2ME
4. Alle programmer som lages i Mophun må sendes til Mophuns sertifiserings-prosess før de kan installeres på telefoner.

Som punkt 2 viser kunne vi valg Mophun for å få tilgang til lyd i kundeapplikasjonen. Det som var hovedgrunnene til å velge J2ME framfor Mophun var punkt 4. Mophun er beregnet for kommersiell bruk, og tillater ikke at applikasjoner blir fritt distribuert. Applikasjonen må sertifiseres for hver telefon som skal bruke den. I tillegg er det ikke mulig å la en bruker laste ned applikasjonen uten å betale for dette.

Resultatet av å bruke J2ME blir da at støtten for lyd ikke blir så bra som vi kunne ønsket, og at kundeprogrammet ikke eksekverer så raskt som det gjør i Mophun. På den andre siden slipper vi unna sertifiseringsprosessen.

Valg av kommunikasjonsmetode

kXML-RPC ble valgt som kommunikasjonsmetode på *kundeapplikasjonen*, men kSOAP var et annet alternativ som kunne brukes. Under følger en pro/con for kXML-RPC og kSOAP. Oversikten er hentet fra FAQ til kXML-RPC på enhydra.org [30].

kXML-RPC brukes når
- Hastighet er viktigere en fleksibilitet
- Minnebruk er en viktig faktor
- Programstørrelsen må være liten
- Dataen som overføres er enkel
- Du trenger en nøytral, standardisert, lettvektsmekanisme for overføring av data, eller når du kaller et fjernt nettverkservice.

kSOAP brukes når
- Fleksibilitet og robusthet er viktig
- Programstørrelse er mindre viktig enn avanserte nettverkskarakteristika
- Dataen som overføres er komplisert, forholdet mellom dataen er komplisert, eller det er viktig å kunne definere egne datatyper
- Du er usikker på hvilken protokoll som vil bli brukt av potensielle klienter og partnere. SOAP er mer populær og det er derfor større sjanse for at det blir brukt.

Som vi ser av oversikten er kXML-RPC bedre egnet til vår bruk framfor kSOAP fordi:
- Vi bruker GPRS som er et smalbåndsnett for å overføre informasjon. Her betaler brukeren pr kilobyte og derfor bør størrelsen på det som overføres være minst mulig.
- Minnebruken bør være minst mulig da systemet skal kunne brukes av mobile terminaler med variabelt tilgjengelig minne

- Programstørrelsen bør være minst mulig da systemet skal kunne brukes av mobile terminaler med variabelt tilgjengelig lagringskapasitet.
- Dataen som overføres er enkel. Vi overfører kun enkle tekststrenger.
- Vi vet hvilken protokoll som befinner seg på serversiden av systemet

Her kan vi konkludere at kXML-RPC er mer egnet for nettverkskommunikasjon på mobile terminaler av typen mobiltelefon, mens kSOAP er mer egnet for nettverkskommunikasjon på mobile terminaler av typen PDA.

Interaksjonsmodi

Tabellen under viser et sammendrag av de ulike interaksjonsmodi som vi har sett på.

Tabell 1 Interaksjonsmodi sammendrag

Teknologi/interaksjonsmodi	Muligheter
SMS	Kun enkel tekst, kun 160 tegn
MMS	Enkel tekst, lyd (musikk og tale), bilde, video
Java MIDP 1.0	Grafikk, rik tekst
Lyd	Skaper oppmerksomhet
Grafikk	Øker lesbarheten
Tekst	Ulik størrelse, type og stil gir økt lesbarhet
Vibrasjon	Skaper oppmerksomhet

Som beskrevet tidligere i oppgaven er det flere muligheter for hvilke interaksjonsmodi som kan brukes, alt etter brukerens situasjon og ønsker. Denne oppgaven har ikke fokusert mye på å bruke MMS som kommunikasjonskanal, men dette betyr ikke at MMS ikke er en interesant form for kommunikasjonskanal. Pr desember 2003 koster en MMS 2,50 [26]. Hvis vi velger å utvide systemet med MMS, kan det veie opp for et av problemet med MIDP 1.0, nemlig bruk av ordentlig lyd.

Kostnader

Det ble skrevet i kapittel 2.3.3 side 16, at det er billigere å bruke J2ME-applikasjonen framfor å sende SMS hver gang en forsinkelse oppdages.

Slik J2ME er i dag er dette ikke nødvendigvis riktig. J2ME har ikke mulighet for å ta i mot informasjon uten at den vet at informasjonen kommer. Dette betyr at for at applikasjonen skal kunne ta i mot informasjon, må den lytte etter informasjon, og dette skaper et par utfordringer.

1. Vi bruker to tråder for å kunne lytte etter informasjon samtidig som brukeren kan navigere i applikasjonen. Noe som resulterer i tregere oppstart av kundeapplikasjonen.
2. Kommunikasjon mellom kundeapplikasjonen og serveren øker, da kundeapplikasjonen poller etter informasjon med jevne mellomrom.

Punkt 1 er ikke kritisk, men punkt 2 er verre. Mye kommunikasjon over nettet betyr nødvendigvis økte kostnader. Hvis kun beskjeder fra transportsystemet gikk over

nettet, ville det blitt billigere å bruke J2ME-applikasjonen, men siden det blir mye ekstraoverføring (overhead), kan SMS bli billigere i lengden. Dette bestemmes av hvor lenge J2ME-applikasjonen står på.

Regneeksempel:

Denne utregningen gjelder for applikasjonen som er utviklet i denne oppgaven. Denne applikasjonen bruker MIDP 1.0. Hadde MIDP 2.0 blitt brukt villedette regnestykket blitt helt anneledes.

En SMS koster 0,69[3] [25] .
GPRS koster 0,10 pr kilobyte [27], som er 0,00097 øre pr byte.

Hver beskjed fra kundeapplikasjonen er ca 350 bytes[4]. Hver returbeskjed er omtrent like stor. Til sammen blir dette 700 bytes hver gang kundeapplikasjonen sender en beskjed og får svar. Dette betyr at hver slik beskjed koster 6,8 øre (700 bytes * 0,00097), som igjen betyr at vi kan sende 10 slike beskjeder før det blir like dyrt som å sende en SMS.

Kundeapplikasjonen sender en ny beskjed hvert 5 minutt. Dette betyr at applikasjonen kan stå på i 50 minutter før kostnadene blir like stor som ved å sende en SMS. Her er det viktig å merke seg at J2ME applikasjonen har konstant kostnad, uavhengig av hvor mange beskjeder som mottas. Dette er ikke tilfellet for bruk av SMS. Hvis en bruker har mange abonnementer i et tidsrom, sendes det kanskje ut mange beskjeder og da vil det bli billigere å bruke J2ME-applikasjonen i dette tidsrommet. Sendes få eller ingen beskeder blir det billigere å bruke SMS. Hvor mange beskeder som sendes ut kan vi ikke vite på forhånd, men det kan være lurt for en bruker med mange abonnementer i et tidsrom å ha på J2ME-applikasjonen i dette tidsrommet.

Dette er et problem som løses ved bruk av MIDP 2.0.

MIDP 1.0 vs MIDP 2.0

Demoapplikasjonen er utviklet for å kunne fungerer på en SonyEricsson P800. P800 støtter Java og MIDP 1.0. MIDP 2.0, som allerede finnes støtte for i noen mobiltelefoner, utvider MIDP 1.0 på mange områder[20]. Pr 26.november 2003 finnes det to telefoner med støtte for denne versjonen av MIDP, Nokia 6600 [31] og Sony Ericsson P900 [32].

Utvidelser i MIDP 2.0 er blant annet et *push-register*[20] som tillater MIDlets[5] å starte når en innkommende nettverkskobling oppdages. Dette betyr at transportsystemet ikke trenger å sjekke om *J2ME-applikasjonen* er på når en beskjed skal sendes ut, den kan slås på direkte av transportsystemet hver gang det er nødvendig. I tillegg slipper vi å kjøre to tråder i kundeapplikasjonen, da det ikke er nødvendig å polle etter informasjon.

[3] Faktisk pris avhenger av abonnement.
[4] TCP/IP header 50 bytes, HTTP header 110 bytes og XML nyttelast (payload) 200 byte.
[5] J2ME applikasjoner kalles ofte MIDlets.

En annen viktig utvidelse i MIDP 2.0 er dens støtte for multimedia[20]. MIDP 1.0 har lite støtte for avspilling av lyd. Dette endres i MIDP 2.0. MIDlets kan der spille av enkle toner, sekvenser av toner, og wav-filer. Dette kan brukes for å øke oppmerksomheten når nye beskjeder mottas. *MIDP* 1.0 støtter kun bruk av telefonens standardlyder gjennom *Alerts*[6]. Det er med andre ord ikke mulig for programmereren å bestemme hvilke lyder som skal brukes.

Situasjoner

I denne oppgaven har vi antatt at en bruker befinner seg i en av følgende situasjoner:

- Vanlig
- Møte
- Borte
- Ferie

Det kan tenkes at det finnes andre situasjoner en bruker kan befinne seg i. Dette blir da noe man må ta mer stilling til i videreutviklingen av transportsystemet.

Situasjonen *vanlig* kan med fordel deles opp. For eksempel kan en person som er på jobb kanskje ha andre krav enn en person som er hjemme, men de vil befinne seg i situasjonen *vanl*ig i begge tilfeller. Derfor kan situasjonen *vanlig* deles opp i *hjemme* og *jobb.*

Situasjonen *ferie* kan også deles opp. En person som har ferie kan være hjemme og ønsker fortsatt ruteoppdateringer som ikke er jobbrelatert. Han/hun kan også være bortreist, og da er det ikke interessant med noen typer oppdateringer. Situasjonen *ferie* kan derfor for eksempel deles opp i *ferie-hjemme* og *ferie-borte.*

Problemstilling

Hvordan kan kontekstinformasjon benyttes til tilpasning av interaksjonsmodi mot mobile terminaler.

Vi har gjennom demoapplikasjonen illustrert at vi kan bruke kontekstinformasjon for å avgjøre hvilke interaksjonsmodus som skal benyttes i ulike situasjoner mot brukerens mobiltelefoner. Demoapplikasjonen benytter likevel bare en type interaksjonsmodus, som er J2ME. Så selv om applikasjonen avgjører hvilket interaksjonsmodus som skal benyttes, er kun meldinger via J2ME-applikasjonen tilgjengelig i demoapplikasjonen. Dette endres hvis applikasjonen får SMS-støtte. Altså muligheten for å sende ut ruteoppdateringer via SMS når kundens modus tilsier dette.

I og med at demoapplikasjonen ikke tar hensyn til *lokasjon og tid*, blir ikke tilpasningen av interaksjonsmodus så kontekstsensitiv som vi kunne ønske. Dette er også noe som endres når transportsystemet blir ferdigutviklet og får støtte for dette. Selv om tilpasninger av interaksjonsmodus kunne vært mer kontekstsensitiv, er det likevel noe kontekstsensitive tilpasning av interaksjonsmodus ved at

[6] Alert brukes i J2ME for å varsle brukeren om farer, hendelser, spørsmål mm.

transportsystemet sjekker om en bruker en *online* før en beskjed sendes ut. Hvis brukeren er *offline* og egentlig skal motta beskjeden via SMS, sendes ikke beskjeden ut.

I tillegg brukes kontekst for å sjekke om en bruker ikke ønsker beskjeder fra systemet. Dette tas hensyn til gjennom endring av brukerens situasjon. I tilfeller hvor en brukers situasjon tilsier at han/hun ikke ønsker å motta noen beskjeder, har kontekst blitt brukt for å avgjøre at *ingen* interaksjonsmodus skal benyttes.

7.4 Videre arbeid

Gjenstående krav

Følgende krav er ikke implementert:

- Krav 7: Slett abonnement
- Krav 8: Endre abonnement
- Krav 9: Informasjon som leveres skal være tilpasset/relevant i forhold til brukerens lokasjon
- Krav 10: Informasjon som leveres skal være tilpasset/relevant i forhold til tid på døgnet og dato.
- Krav 14: Brukeren skal kunne bruke sin mobiltelefonkalender for å kommunisere med transportsystemet
- Krav 15: Administrator skal kunne legge til manuelle beskjeder
- Krav 16: Administrator skal kunne slette manuelle beskjeder

Krav 7: Slett abonnement ble ikke implementert da det ikke er viktig for å kunne illustrere hvordan transportsystemet fungerer.

Krav 8: Endre abonnement ble ikke implementert da det ikke er viktig for å kunne illustrere hvordan transportsystemet fungerer.

Krav 9: Brukerens lokasjon er viktig for å kunne lettere avgjøre om en beskjed skal sendes eller ikke. For å kunne implementere krav 9, må applikasjonen få tilgang til GSM-posisjonering. For å få slik tilgang må eksterne systemer tas i bruk, og dette tas ikke høyde for i demoapplikasjonen.

Krav 10: Tid er viktig for ikke å sende ut informasjon til brukere som ikke trenger det. Dette kravet ble ikke implementert da det henger mye sammen med synkroniseringen av kalender som heller ikke ble implementert. Kalenderinformasjon brukes sammen med tid for å endre en brukeres situasjon automatisk.

Krav 14: . Krav 14 er pr. dags dato ikke mulig å implementere i J2ME på en enkel måte. Den eneste måten man i dag kan få dette til på, vil være å lage en utvidelse av *kundeapplikasjonen* som inneholder en komplett kalenderapplikasjon. Når neste versjon *MIDP* kommer ut, vil det være mulig å dele *RecordStore*[7] mellom *J2ME-applikasjoner*. Dette betyr at da blir mulig å bruke en tredje parts J2ME-kalenderapplikasjon og hente informasjon fra denne.

Krav 15 og 16: Administrator-applikasjonen er ikke implementert. For å kunne bruke transportsystemet optimalt, bør denne delen implementeres først. Den er ikke påkrevd for at transportsystemet skal kunne tas i bruk, men det er viktig å kunne sende ut manuelle beskjeder hvis man ønsker å kunne bruke transportsystemet til andre ting enn bare å sende ut ruteoppdateringer. For eksempel er det viktig å kunne sende ut beskjeder til alle brukerene om at nye rutetider gjelder, eller at prisen på billetten endres.

[7] RecordStore er metoden j2me bruker for å lagre informasjon i telefonen

Sikkerhet

Sikkerhet er ikke tatt hensyn til i denne oppgaven. Det som minst burde gjøres i videreutviklingen av implementasjonen er å kryptere telefonnummer og passord når brukere logger inn i systemet. Kryptering støttes ikke i J2ME, men det finnes tredje parts løsninger tilgjengelig. ”The Legion of the Bouncy Castle” [36] er et krypteringsprosjekt for Java som også har støtte for J2ME.

Transaksjonstjenester og kommunikasjonstjenester

Transportsystemet kan også utvides med ulike typer tjenester. Ved å legge til transaksjonstjenester i systemet, kan det bli mulig å bestille og betale billetten før man reiser. Dette gjør at systemet blir enklere å bruke for reisende, som slipper å lete etter småpenger hver gang de reiser. Når en betaling er registrer kan brukeren motta en SMS som fungerer som kvittering på bussen hvis det skulle bli kontroll.

Alternativt kan det være mulig å bestille en billett, men betalingen utføres kun hvis brukeren faktisk benyttet seg av denne. Dette kunne løses ved å GSM-posisjonering brukes for å avgjøre om brukeren faktisk har tatt bussen. Ved å sjekke om brukeren er på startholdeplassen når bussen kommer, og på stoppholdeplassen når bussen er framme, kan man avgjøre om brukeren faktisk tok bussen. Betaling for bussturen utføres så automatisk.

Systemet kan også utvides med bruk at kommunikasjonstjenester. Det kan for eksempel være interesant for en passasjer å kunne gi tilbakemelding til transportselskapet om positive og negative erfaringer med transportsystemet. Dette er en fin måte å sjekke om systemet fungerer slik det var ment. Dette gjør det også lettere og avgjøre hva som må endres ved en eventuell oppdatering av systemet.

8 Referanser

[1] Dey, A.K. & Adowd, G.D. "Towards a Better Understanding of Context and Context-Awareness". In the 2000 Conference on Human factors in Computing Systems (CHI 2000): workshop on the What, Who, Where, When and How of Context-Awareness. The Hague, Netherlands (2000.)

[2] Finnset, W. Telenor FoU, Tromsø: Sm@rtBuss. Prosjektbeskrivelse. Tromsø, Norge (2002)

[3] Jacobsen, I., Booch, G., Rumbaugh, J. "The Unified Software Development Process", (1999)

[4] Kjørstad, K. og Lodden, U. "Sammendrag av IBIS Logitrans Brukernes, vurdering av sanntids ruteinformason I Trondheim", 2003, TØI rapport 638/2003

[5] Schilit, B., Theimer, M. "Disseminating Active Map Information to Mobile Hosts". IEEE Network, 8(5) (1994)

[6] Brown, P.J., Bovey, J.D., Chen, X. Context-Aware Applications: "From the Laboratory to the Marketplace", IEEE Personal Communications, 4(5) (1997)

[7] Ryan, N., Pascoe, J., Morse, D. Enhanced Reality Fieldwork: "the Context-Aware Archaeological Assistant". Gaffney V., van Leusen M. Exxon S. Computer Applications in Archaeology, (1997)

[8] Dey, A.K. Context-Aware Computing: "The CyberDesk Project", AAAI 1998 Spring Symposium on Intelligent Environments, Technical Report SS-98-02, (1998)

[9] Brown, P.J. "The Stick-e Document: a Framework for creating Context-Aware Application. Electronic Publishing, (1996)

[10] Franklin, D., Flaschbart, J. "All Gadget and No Representation Makes Jack a Dull Environment". AAAI 1998 Spring Sumposium on Werable Computers (1997)

[11] Schilit, B., Adams, N., Want, R. "Context-Aware Computing Applications". 1st Interational Workshop on Mobile Computing Systems and Application. (1994)

[12] Dey, A.K., Abowd G.F., Wood, A. CyberDesk: "A Framework for Providing Self-Integrating Context-Aware Services. Knowledge-Based Systems" (1999)

[13] Pascoe, J. "Adding Generic Contextual Capabilities to Wearable Computers". 2nd International Symposium on Wearable Computers, (1998)

[14] Robertson, J. & Robertson, S. Volere – Requirements Spesification Template, Edition 9, (2003)

[15] Buckingham, S. “What is SMS”, GSM Association, (2002)
http://www.gsmworld.com/technology/sms/intro.shtml

[16] “What is MMS”, GSM Association
http://www.gsmworld.com/technology/mms/whatis_mms.shtml

[17] Buckingham, S. “What is GPRS”, GSM Association, (2002)
http://www.gsmworld.com/technology/gprs/intro.shtml

[18] Ottestad, G. “Instant Messages”, ITU, av. 4, lj 8,
http://www.itu.no/Dokumenter/Artikler/t1037878092_34/view

[19] Sun Microsystems, “J2ME Technologies”, http://Java.sun.com/j2me/

[20] Knudsen, J. Sun Microsystems, “Whats new in MIDP 2.0” (2002),
http://wireless.Java.sun.com/midp/articles/midp20/

[21] GSM Association, GSM World – “The wireless evolution”,
http://www.gsmworld.com/index.shtml

[22] Sun Microsystems, “J2ME Technologies Overview”,
http://Java.sun.com/j2me/docs/j2me-ds.pdf

[23] Telenor, “Fakta om Telenor Mobil”,
http://telenormobil.no/omtelenormobil/fakta/fakta.tnm

[24] Quatrani, T. “Visual Modeling With Rational Rose 2000 and UML”, Addison-Wesley, (2000)

[25] Telenor ASA, Priser SMS, http://telenormobil.no/priser/tjenester/gsmtekst/

[26] Telenor ASA, Priser MMS http://telenormobil.no/priser/tjenester/mms/

[27] Telenor ASA, Priser GPRS, http://telenormobil.no/priser/tjenester/gprs/

[28] XML-RPC Home Page, http://www.xml-rpc.com

[29] The home of kXML at Enhydra.org, http://kxml.enhydra.org/

[30] kXML-RPC faq. http://kxmlrpc.enhydra.org/project/faq/index.html

[31] Nokia 6600 http://www.nokia.no/phones/6600/

[32] SonyEricsson P900
http://www.sonyericsson.com/developer/site/global/products/phones/p900/p900.jsp

[33] Mophuns hjemmeside http://www.mophun.com/

[34] Telenor FoU, Tromsø: Personalisering og Kontekst. Prosjektbeskrivelse. Tromsø, Norge (2003)

[35] Telenor FoU, Tromsø: WINLAND "Wireless Information Network Landscape" Transport. Prosjektbeskrivelse. Norge (2003)

[36] The Legion of the Bouncy Castle: http://www.bouncycastle.org.

[37] Apache XML-RPC, http://ws.apache.org/xmlrpc/

[38] The home of kXML-RPC at Enhydra.org, http://kxmlrpc.enhydra.org/

[39] Netcom Buddy, http://www.netcom.no/

Appendix A – Hvordan starte transportsystemet

Appendix A beskriver hva som skal til for å kunne bruke transportsystemet.

Software
Følgende software/drivere må være installert for å kunne bruke demoen av transportsystemet:

- MySQL 3.23 eller nyere
  http://www.mysql.com/downloads/
- Java 2 Java Runtime Environment 1.4.2 eller nyere
  http://java.sun.com/downloads/
- Apache XML-RPC 1.2 eller nyere
  http://ws.apache.org/xmlrpc/download.html
- JDBC for MySql
  http://www.mysql.com/downloads/

For å kunne starte transportserveren og agentene må Java 2 Standard Edition (J2SE) jsdk1.4.2 eller nyere og MySQL 3.23 eller nyere være installert. I tillegg må Apache XML-RPC 1.2 eller nyere og JDBC drivere for MySql lastes ned og gjøres tilgjengelig.

Følgende software/drivere må være installert for å kunne implementere videre på klientprogrammet (KundeApp.java)

- Java 2 J2ME *MIDP* 1.0
  http://java.sun.com/downloads/
- Enhydras kXML-RPC versjon 06 eller nyere
  http://kxmlrpc.enhydra.org/software/downloads/
- Enhydras kXML versjon 1.2 eller nyere
  http://kxml.enhydra.org/software/downloads/

Hardware
På klientsiden må brukeren ha tilgang til en SonyEricsson P800. Det finnes andre telefoner som vil kunne kjøre demoen, men den er kun testet på en P800.

PC'en som kjører og er brukt til å teste serverdelen av demoen er en Intel Pentium III 450 Mhz med 128 MB Ram.

Server
For å kunne bruke transportsystemet må *TransportServer*, *ModusAgent*, *KommunikasjonsAgent* og *KontekstAgent* være påslått. Disse programmene kan kjøre lokalt på samme maskin eller distribuert på ulike maskiner.

Når *TransportServer* startes må ip-adressen til *ModusAgent* og *KommunikasjonsAgent* oppgis.
Når *ModusAgent* startes må ip-adressen til *KontekstAgent* oppgis.

Klient
For å kunne bruke demoen må brukeren ha en mobiltelefon som støtter J2ME. *KundeApp.jar* må installeres/legges inn på mobiltelefonen.

Når *KundeApp.jar* er installert på telefonen og serversiden av transportsystemet kjører er demosystemet klart til bruk.

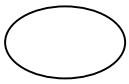
For å bruke transportsystemet må en bruker først registrere seg. Registrering skjer gjennom *KundeApp* applikasjonen. Når registreringer en fullført må brukeren logge inn. Brukeren vil da få tilgang til alle menyvalgene.

All software og filer som trengs for å kunne bruke demoen ligger på vedlagt CD. Se Appendix E for mer informasjon om dette.

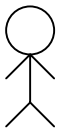
Appendix B – UML notasjon

Appendix B beskriver de forskjellige UML elementene som er brukt i denne oppgaven. Figurene og teorien er hentet fra ”Visual Modeling With Rational Rose 2000 and UML”[24].

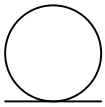
Usecase – Et usecase spesifiserer et krav i systemet.



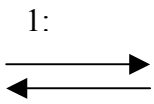
Actor – En Actor er en som påvirker systemet. Dette kan være en bruker eller systemet selv.



Entity (Entitetsklasse) – Indikerer endringer i systemet. For eksempel oppdateringer i en database.



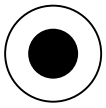
Assosiation – Viser relasjoner mellom ulike objekter



Decision point – Indikerer at et valg blir gjort

Starting point – Viser hvor en flyt starter

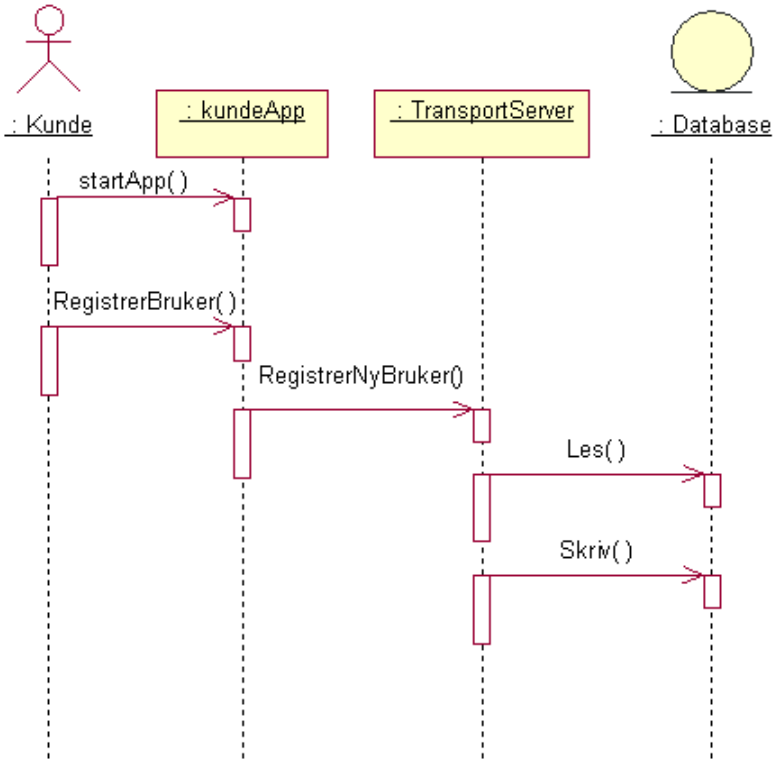
End point – Viser hvor en flyt slutter



Appendix C – Sekvensdiagrammer

Appendix C viser sekvensdiagrammene til alle funksjonelle krav i transportsystemet.

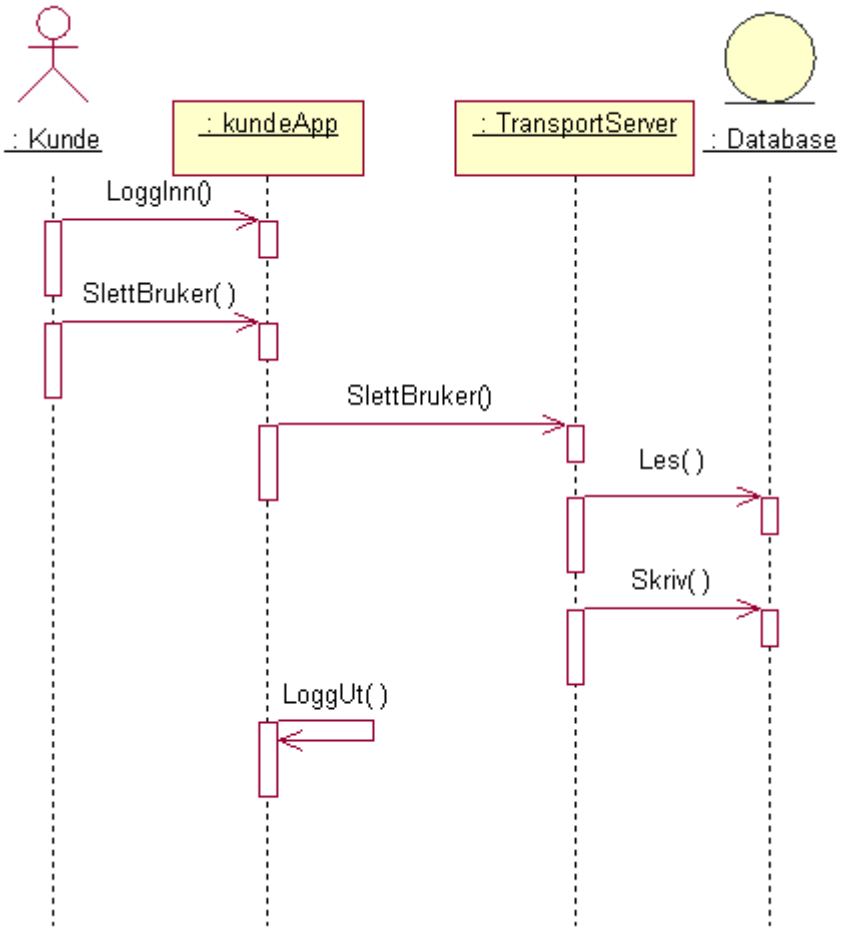
Krav 1: Registrer ny bruker



Figur 21 Sekvensdiagram registrer ny bruker

Figur 21 viser sekvensdiagrammet til krav 1: ”Registrer ny bruker”

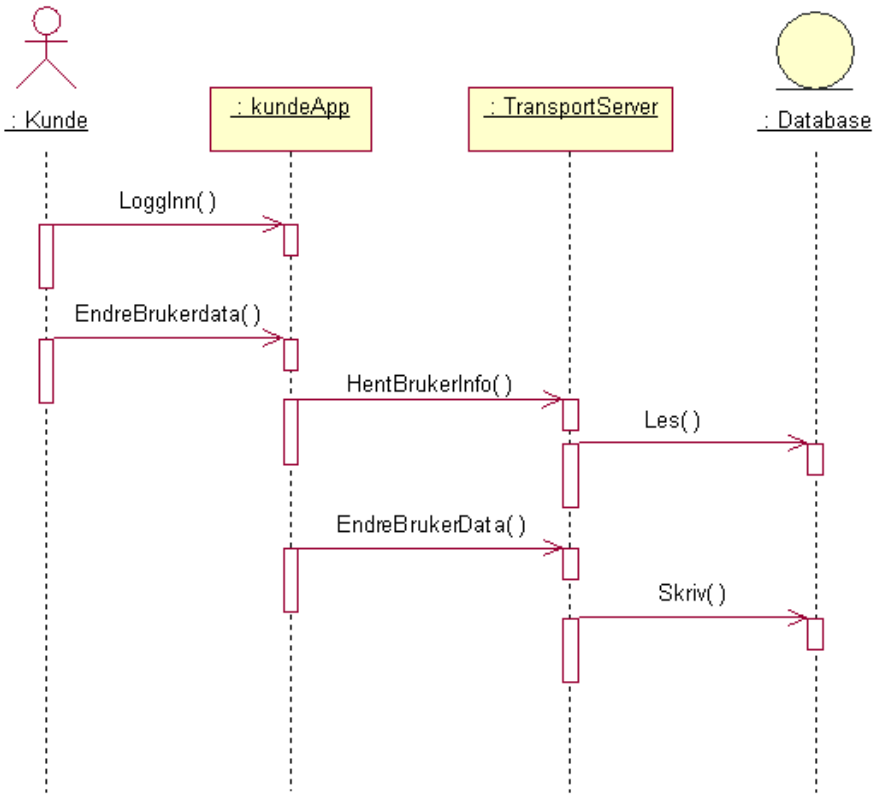
Krav 2: Slett bruker



Figur 22 Sekvensdiagram slett bruker

Figur 22 viser sekvensdiagrammet til krav 2: ”Slett bruker”

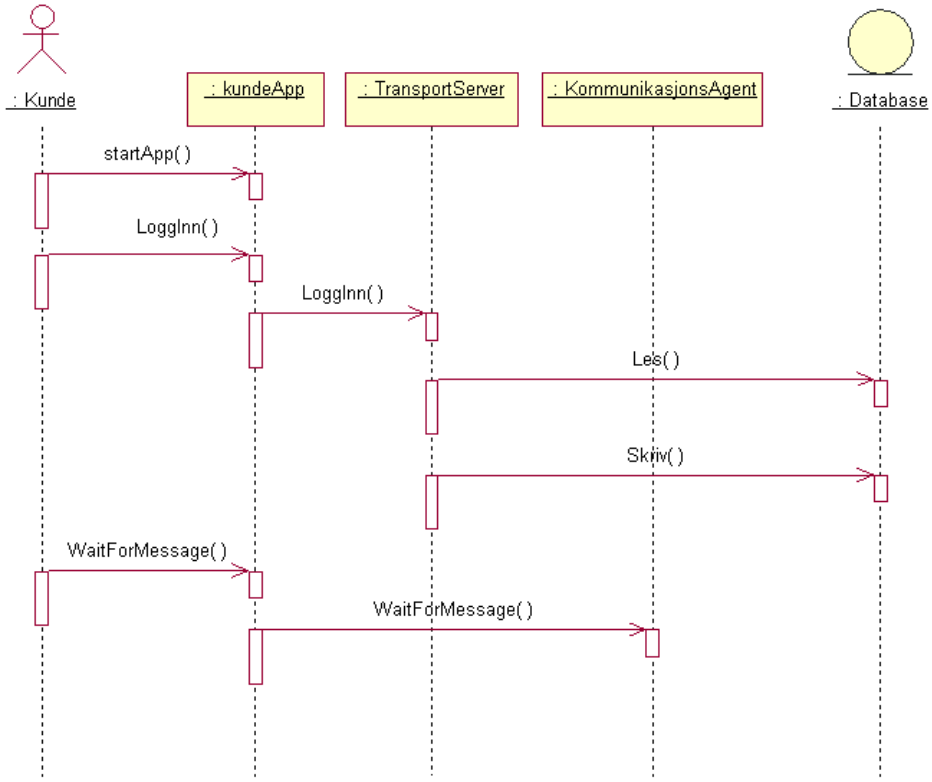
Krav 3: Endre brukerdata



Figur 23 Sekvensdiagram endre brukerdata

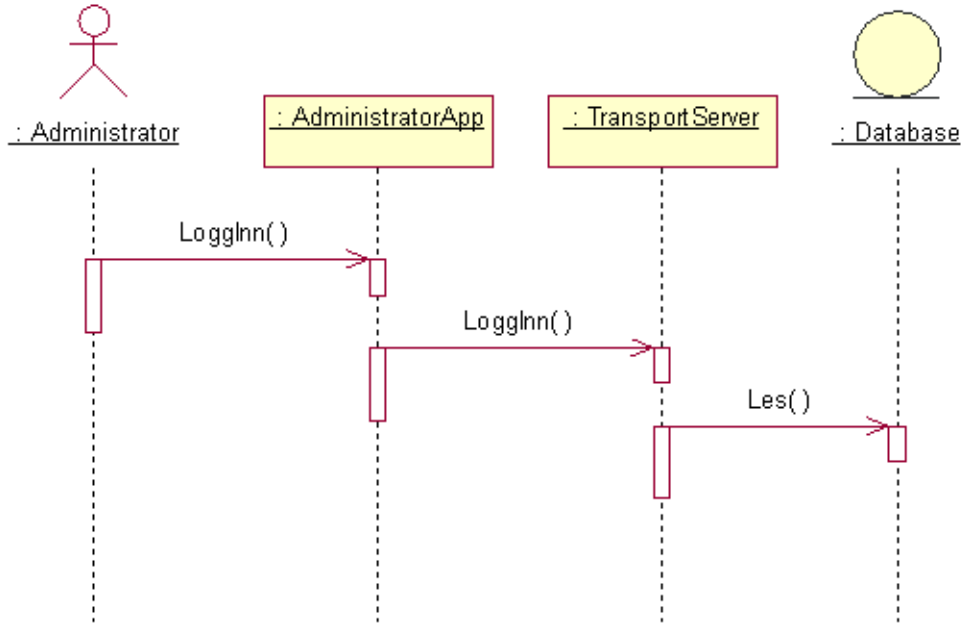
Figur 23 viser sekvensdiagrammet til krav 3: ”Endre brukerdata”

Krav 4: Logg inn



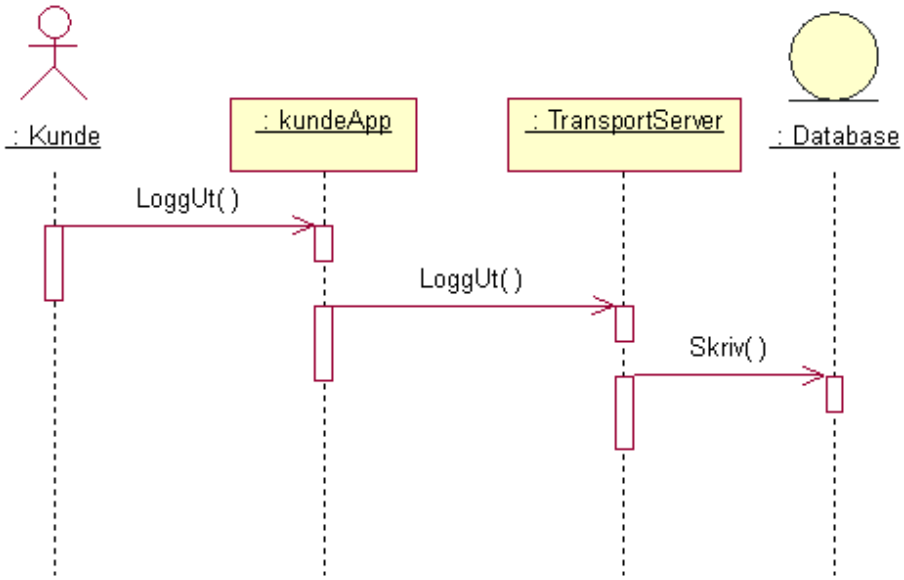
Figur 24 Sekvensdiagram logg inn bruker

Figur 24 viser sekvensdiagrammet til krav 4: ”Logg inn” for brukere. Figur 25 viser sekvensdiagrammet til krav 4: ”Logg inn” for administrator.



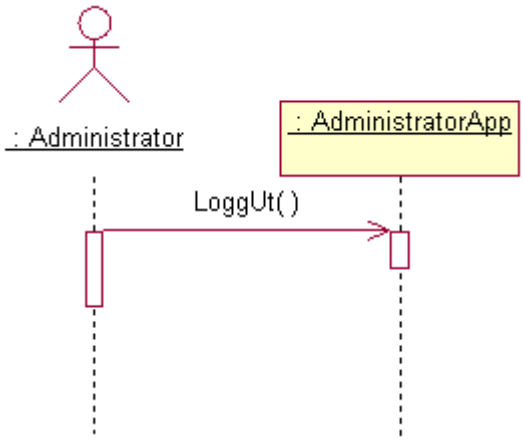
Figur 25 Sekvensdiagram logg inn administrator

Krav 5: Logg ut



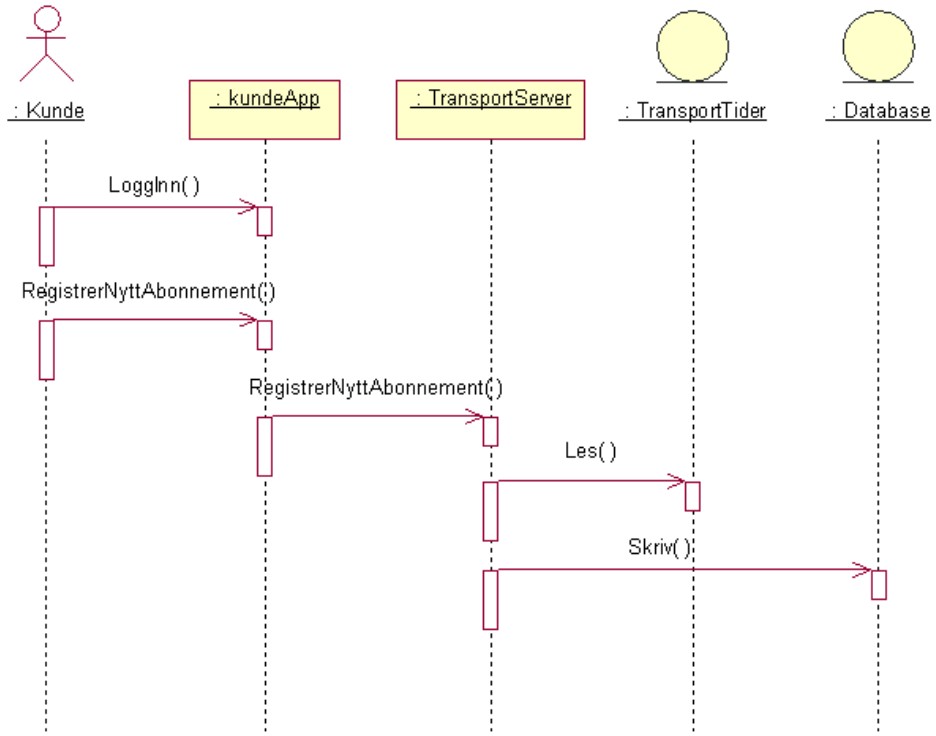
Figur 26 **Sekvensdiagram logg ut bruker**

Figur 26 viser sekvensdiagrammet til krav 5: ”Logg ut” for brukere. Figur 27 viser sekvensdiagrammet til krav 5: ”Logg ut” for administrator.



Figur 27 **Sekvensdiagram logg ut administrator**

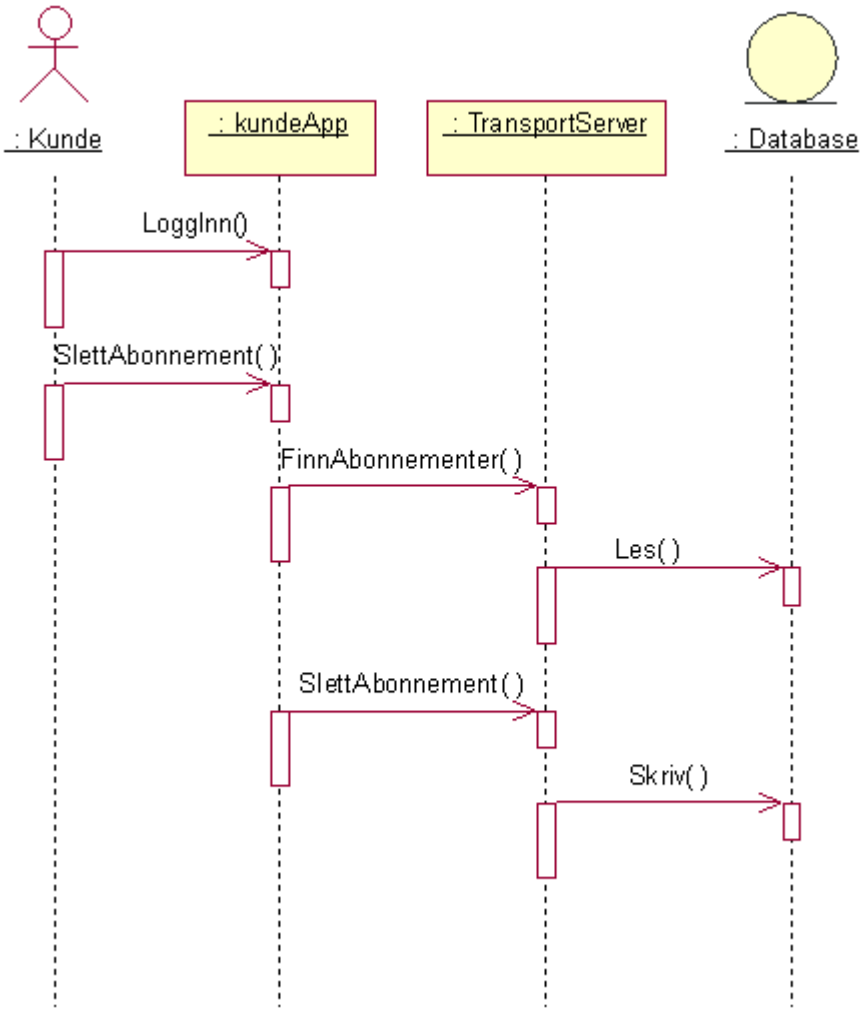
Krav 6: Registrer nytt abonnement



Figur 28 Sekvensdiagram registrer nytt abonnement

Figur 28 viser sekvensdiagrammet til krav 6: ”Registrer nytt abonnement”.

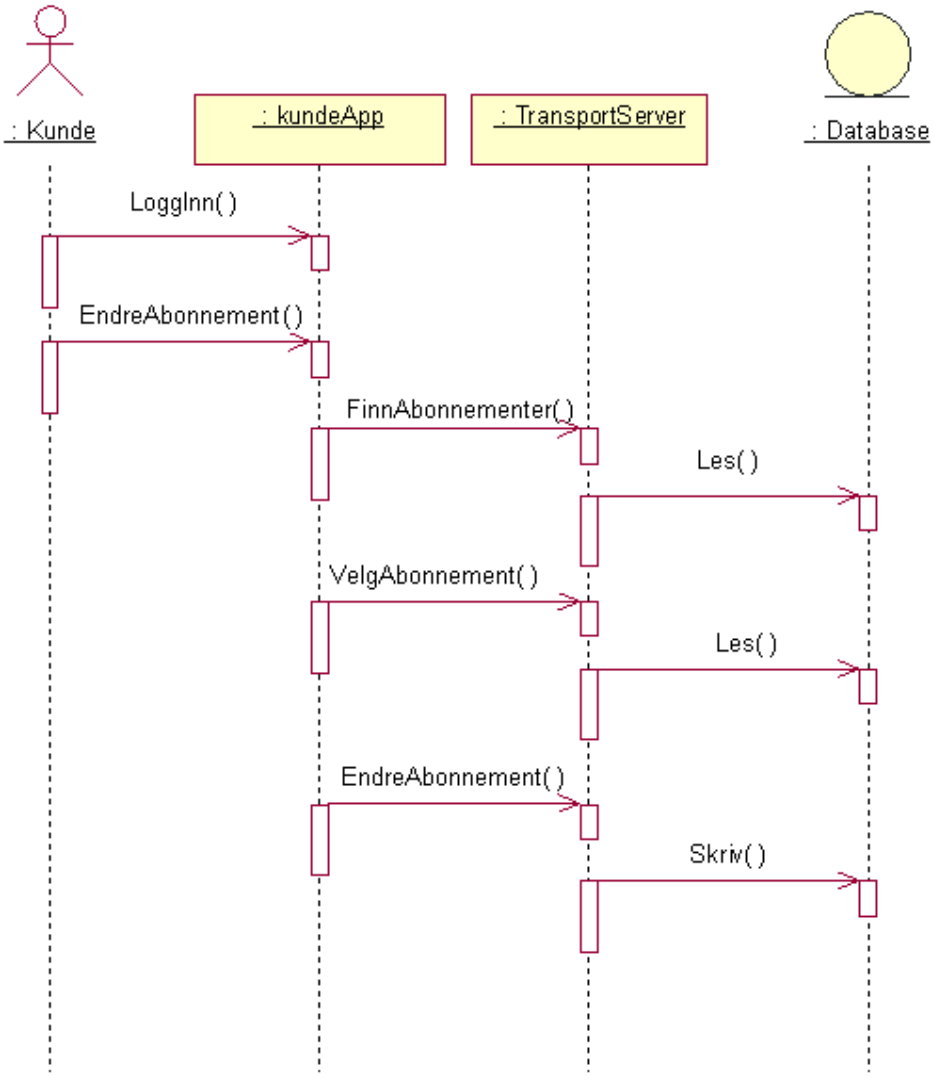
Krav 7: Slett abonnement



Figur 29 Sekvensdiagram slett abonnement

Figur 29 viser sekvensdiagrammet til krav 7: ”Slett abonnement”.

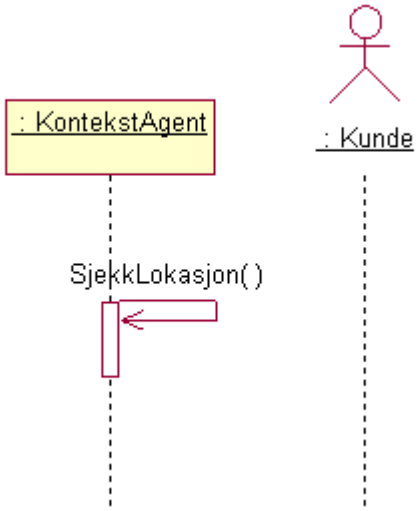
Krav 8: Endre abonnement



Figur 30 **Sekvensdiagram endre abonnement**

Figur 30 viser sekvensdiagrammet til krav 8: ”Endre abonnement”.

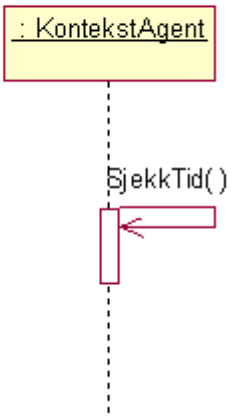
Krav 9: Informasjonen som leveres skal være relevant i forhold til brukerens lokasjon



Figur 31 **Sekvensdiagram sjekk lokasjon**

Figur 31 viser sekvensdiagrammet til krav 9: ”Informasjonen som leveres skal være relevan i forhold til brukerens lokasjon”.

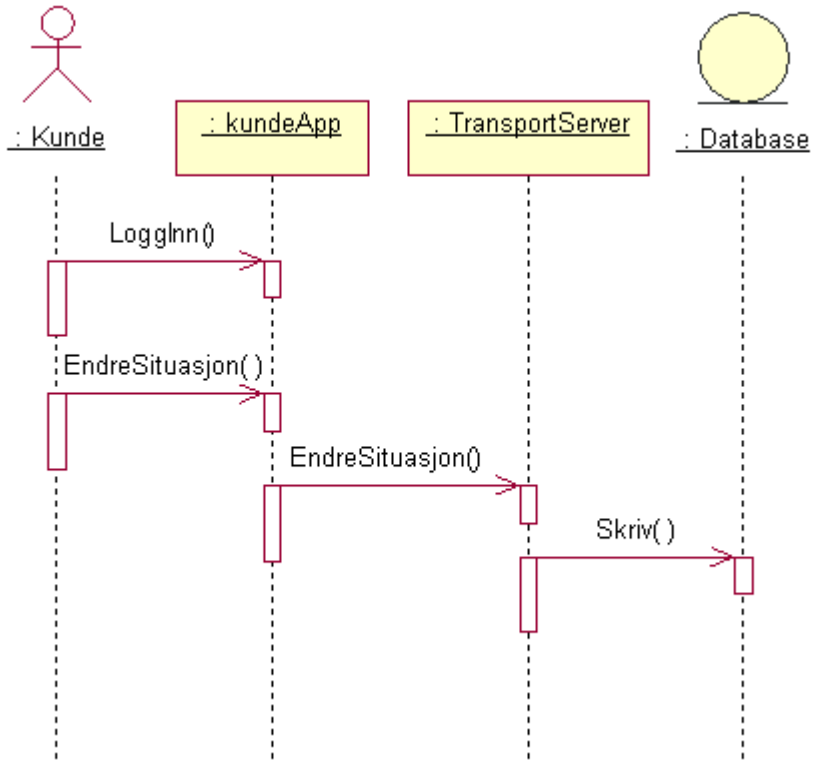
Krav 10: Informasjonen som leveres skal være relevant i forhold til tid på døgnet og dato



Figur 32 **Sekvensdiagram sjekk tid**

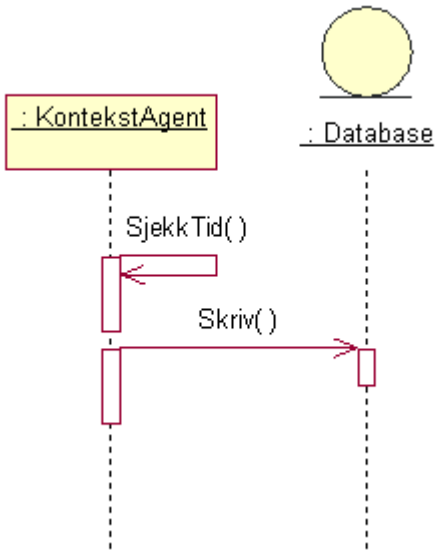
Figur 32 viser sekvensdiagrammet til krav 10: ”Informasjon som leveres skal være relevant i forhold til tid på døgnet og dato”.

Krav 11: Endre brukerens situasjon



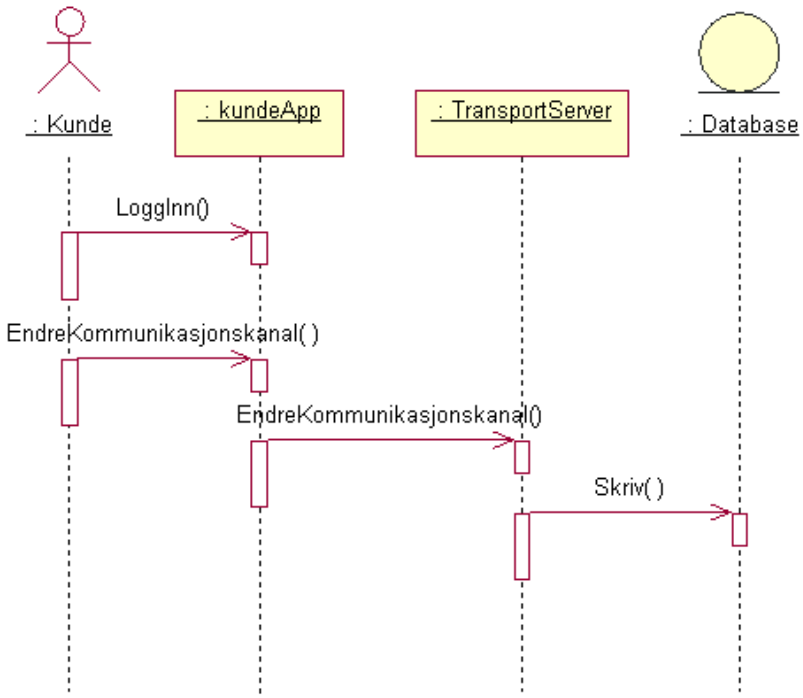
Figur 33 **Sekvensdiagram endre situasjon (manuelt)**

Figur 33 viser sekvensdiagrammet til krav 11: ”Endre brukerens situasjon” når en bruker endrer situasjonen manuelt. Figur 34 viser sekvensdiagrammet til krav 11: ”Endre brukerens situasjon” når transportsystemet automatisk endrer en brukers situasjon.



Figur 34 **Sekvensdiagram endre situasjon (automatisk)**

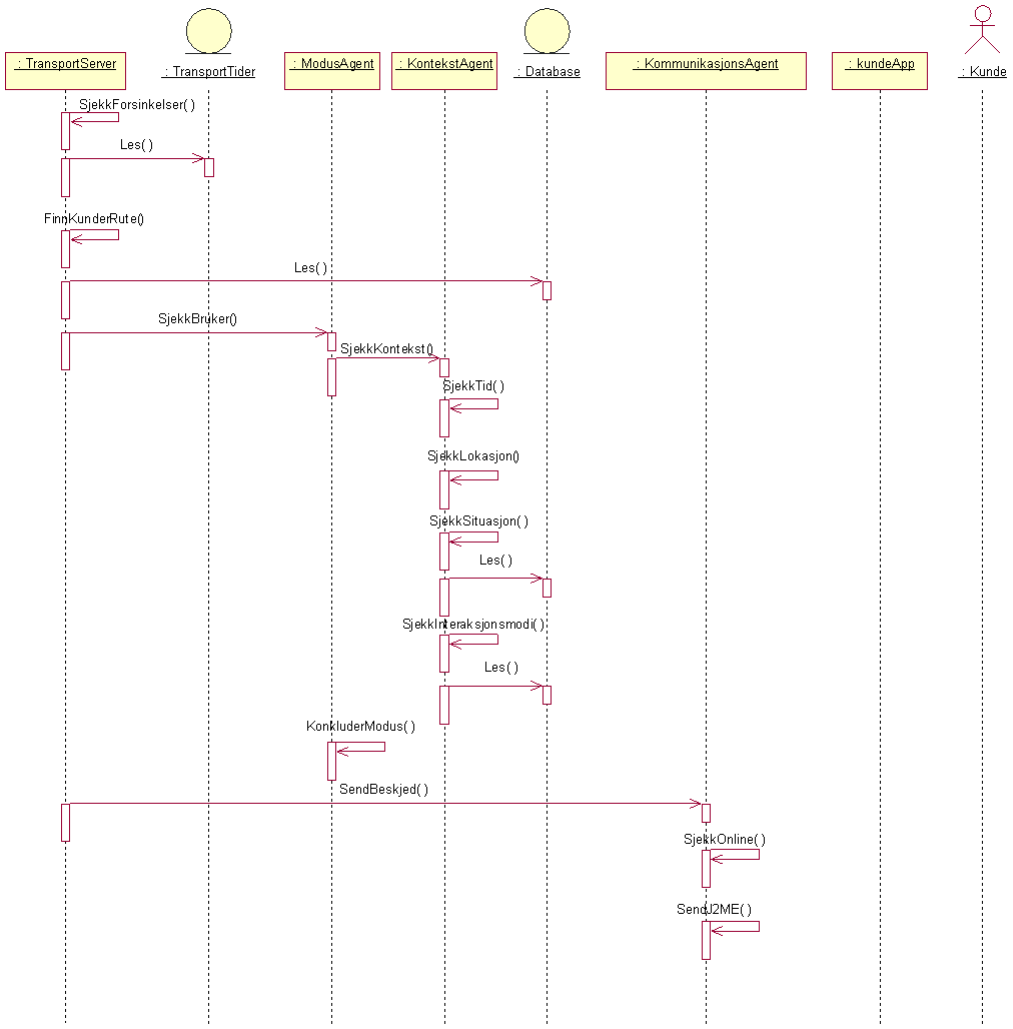
Krav 12: Endre kommunikasjonskanal



Figur 35 Sekvensdiagram endre kommunikasjonskanal

Figur 35 viser sekvensdiagrammet til krav 12: ”Endre kommunikasjonskanal”.

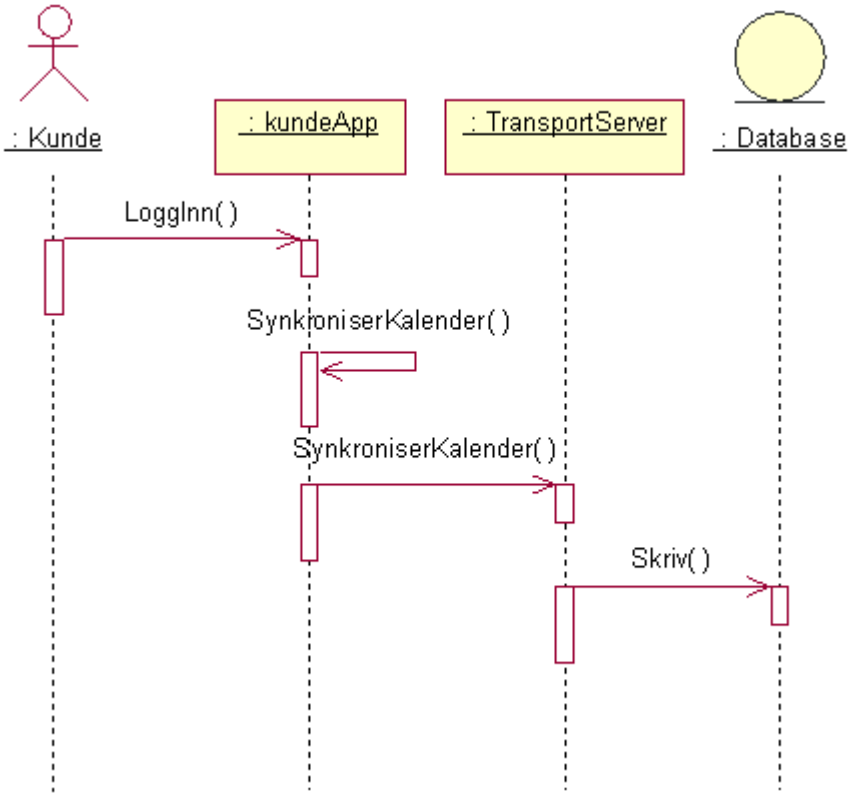
Krav 13: Brukere skal automatisk motta beskjed om forsinkelser og andre viktige hendelser



Figur 36 Sekvensdiagram send beskjeder

Figur 36 viser sekvensdiagrammet til krav 13: ” Brukere skal automatisk motta beskjed om forsinkelser og andre viktige hendelser”.

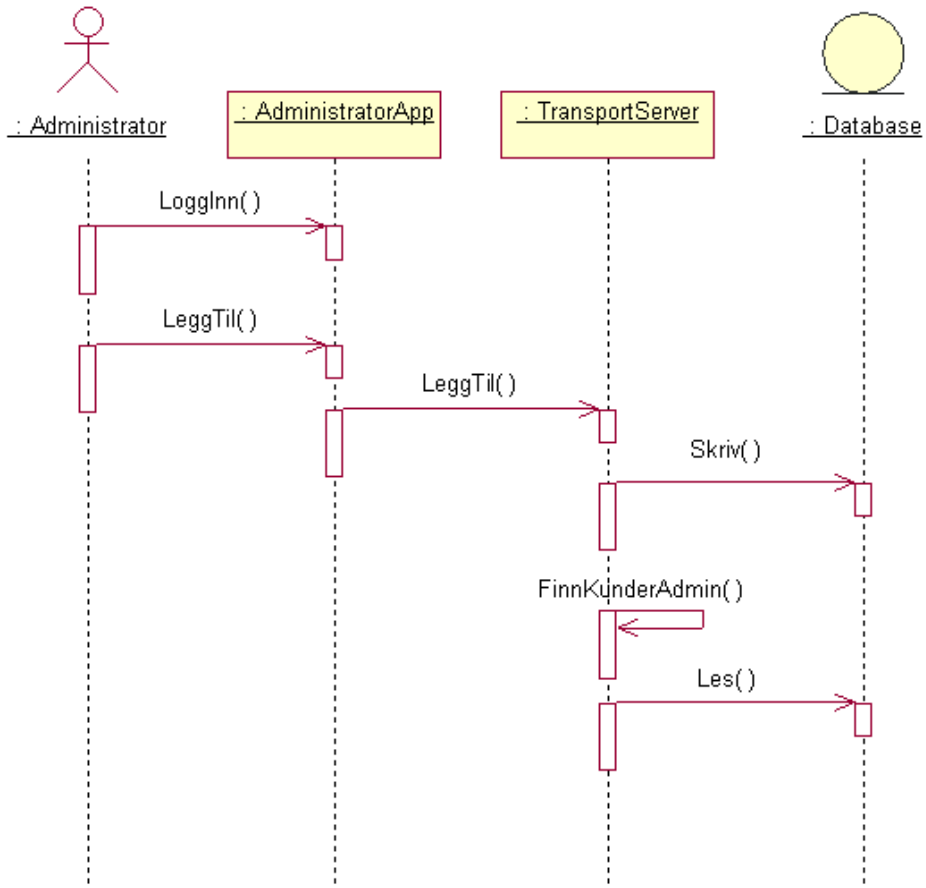
Krav 14: Brukeren skal kunne bruke sin kalenderapplikasjon til å synkronisere med transportsystemet



Figur 37 Sekvensdiagram synkroniser med kalender

Figur 37 viser sekvensdiagrammet til krav 14: ” Brukeren skal kunne bruke sin kalenderapplikasjon til å synkronisere med transportsystemet”.

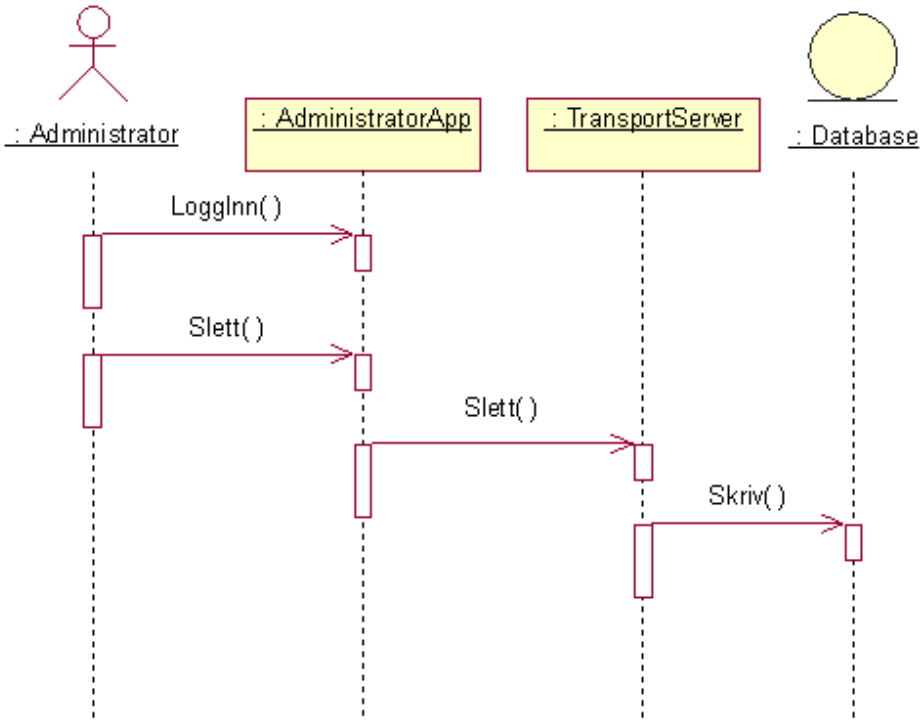
Krav 15: Administrator skal kunne legge til manuelle beskjeder



Figur 38 Sekvensdiagram legg til manuell beskjed

Figur 38 viser sekvensdiagrammet til krav 15: ”Administrator skal kunne legge til manuelle beskjeder”.

Krav 16: Administrator skal kunne slette manuelle beskjeder



Figur 39 Sekvensdiagram slett manuell beskjed

Figur 39 viser sekvensdiagrammet til krav 16: ”Administrator skal kunne slette manuelle beskjeder”.

Appendix D – Screenshots

Klient

Figur 40-50 viser screenshots fra J2ME-applikasjonen fra de ulike menyvalgene.

Figur 40 **Screenshot: Registrer ny bruker**

Figur 40 viser J2ME-applikasjonen når ”Registrer ny bruker” velges fra menyen.

Figur 41 **Screenshot: Slett bruker**

Figur 41 viser J2ME-applikasjonen når ”Slett kunde” velges fra menyen.

Figur 42 **Screenshot: Endre situasjon**

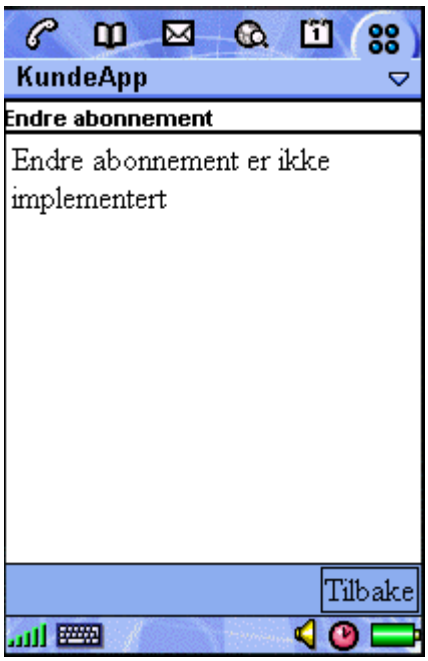
Figur 42 viser J2ME-applikasjonen når ”Endre situasjon” velges fra menyen.

Figur 43 **Screenshot: Endre brukerdata**

Figur 43 viser J2ME-applikasjonen når ”Endre brukerdata” velges fra menyen.

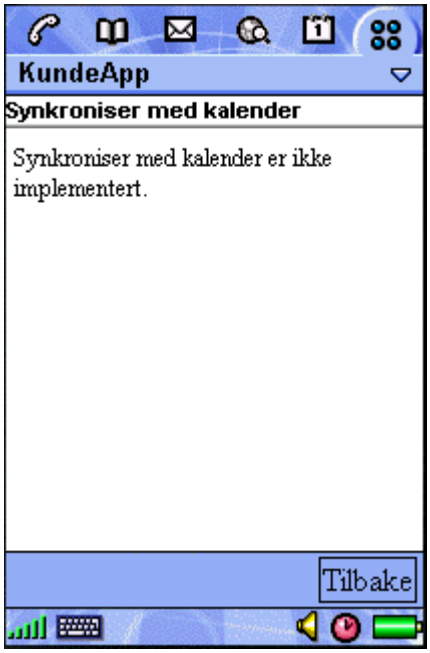
Figur 44 **Screenshot: Endre kommunikasjonskanal**

Figur 44 viser J2ME-applikasjonen når ”Endre kommunikasjonskanal” velges fra menyen.



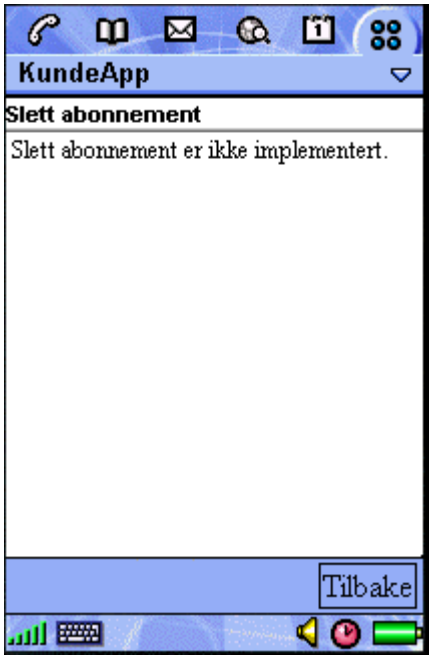
Figur 45 **Screenshot: Endre Abonnement**

Figur 45 viser J2ME-applikasjonen når ”Endre abonnement” velges fra menyen.



Figur 46 **Screenshot: Synkroniser med kalender**

Figur 46 viser J2ME-applikasjonen når ”Synkroniser kalender” velges fra menyen.

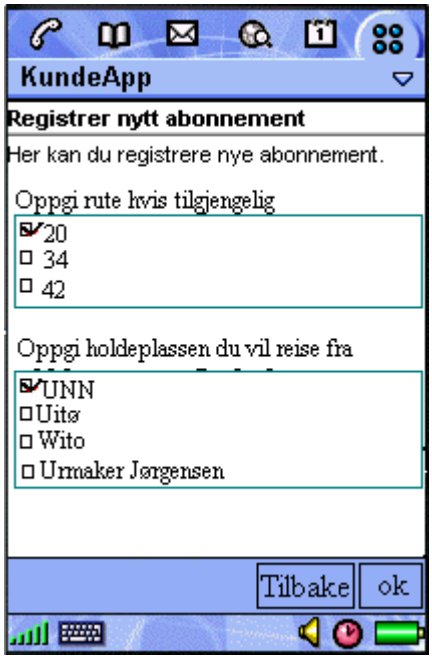


Figur 47 **Screenshot: Slett abonnement**

Figur 47 viser J2ME-applikasjonen når ”Slett abonnement” velges fra menyen.

Figur 48 **Screenshot: Logg ut**

Figur 48 viser J2ME-applikasjonen når ”Logg ut” velges fra menyen.

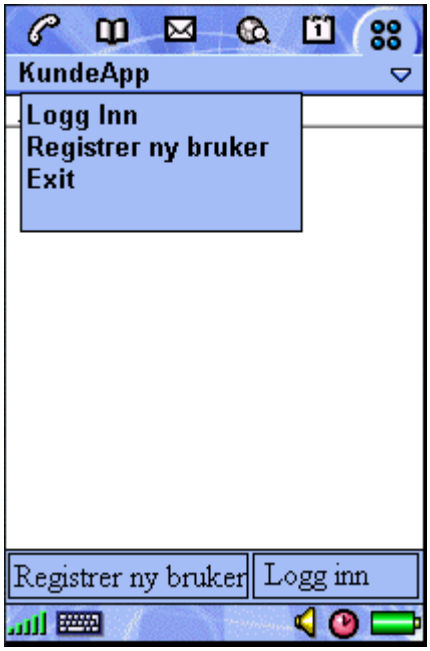


Figur 49 **Screenshot: Registrer nytt abonnement**

Figur 49 viser J2ME-applikasjonen når ”Registrer nytt abonnement” velges fra menyen.

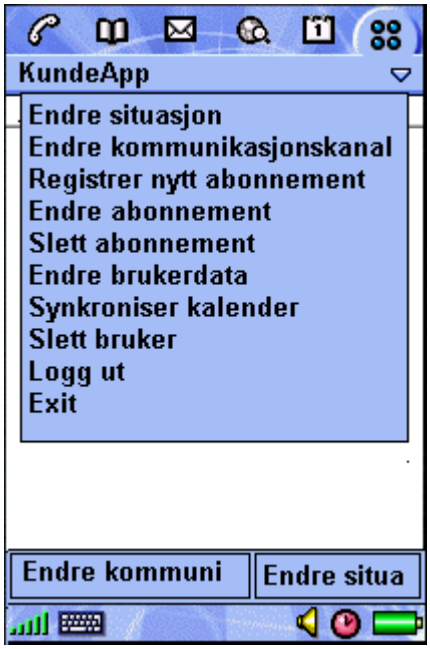
Figur 50 **Screenshot: Logg inn**

Figur 50 viser J2ME-applikasjonen når ”Logg inn” velges fra menyen.



Figur 51 **Screenshot: Menyvalg (ikke logget inn)**

Figur 51 viser menyvalgene når brukeren ikke er logget inn

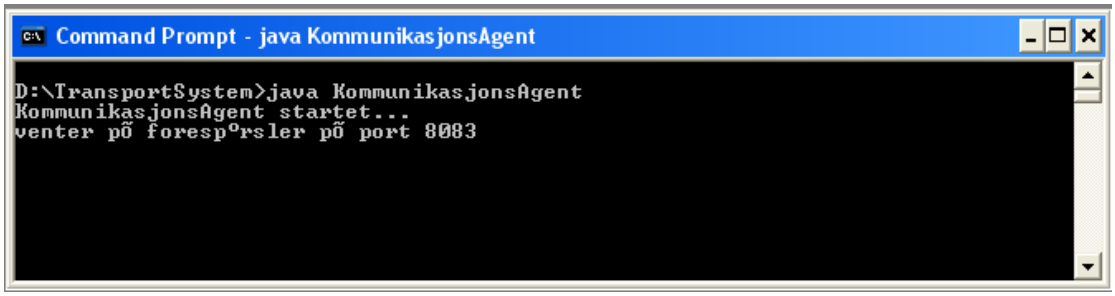


Figur 52 **Screenshot: Menyvalg (logget inn)**

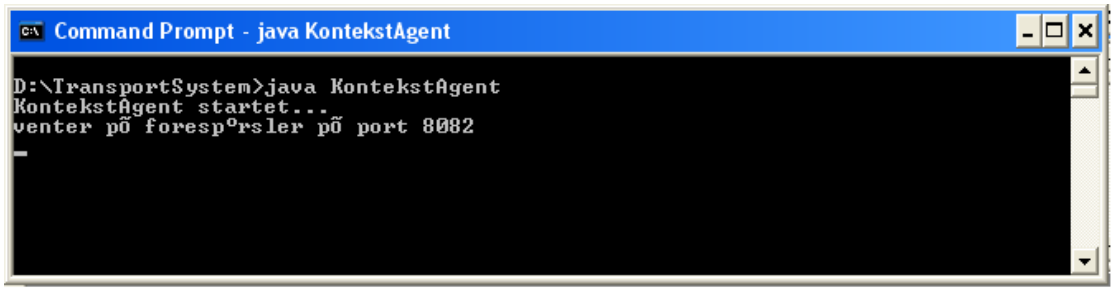
Figur 52 viser menyvalgene når en bruker er logget inn

Server

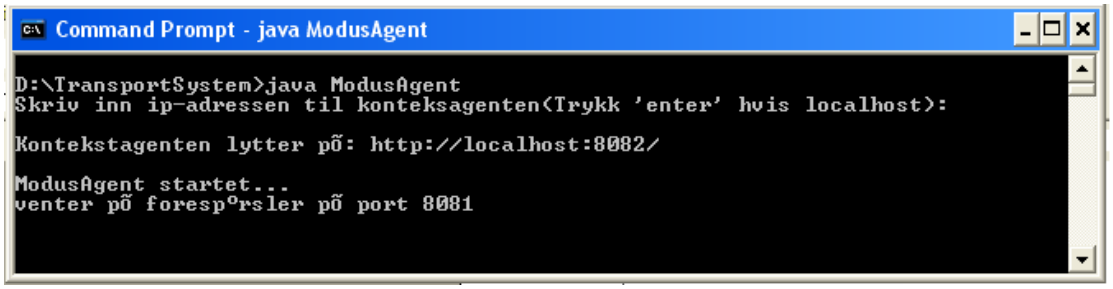
Figur 53,52,53 og 56 viser KommunikasjonsAgent, KontekstAgent, ModusAgent og TransportServer etter de har blitt startet opp.



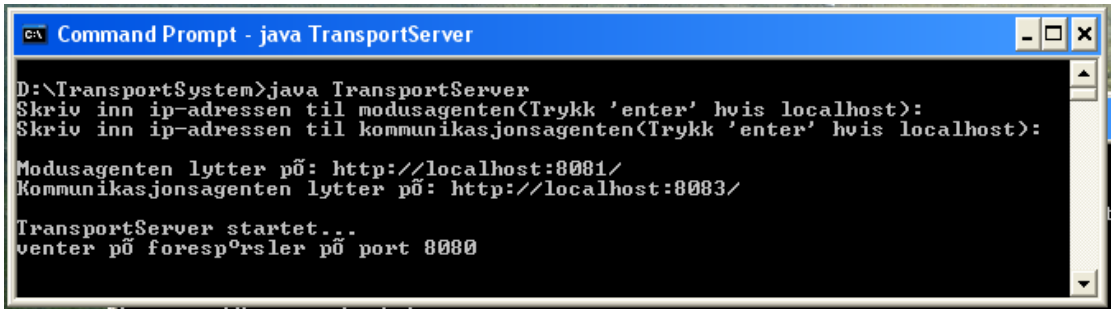
Figur 53 **Screenshot: KommunikasjonsAgent**



Figur 54 **Screenshot: KontekstAgent**



Figur 55 **Screenshot: ModusAgent**



Figur 56 **Screenshot: TransportServer**

Appendix E – Vedlagt CD-plate

Appendix E forklarer hvordan vedlagt CD-plate er strukturert

Katalogstrukturen er slik:

- **eksterne klasser** – inneholder eksterne klasser som må være tilgjengelig for å kunne bruke transportsystemet
    - Apache XML-RPC 1.2
    - JDBC for MySql
- **jar** – inneholder *Kundeapp.jad* og *KundeApp*.jar. Jar-filen skal kopieres over til telefonen.
- **java** – Inneholder alle serverjavafilene og testClienten
    - TransportServer.java
    - KommunikasjonsAgent.java
    - ModusAgent.java
    - KontekstAgent.java
    - testClient.java
- **class** – Inneholder alle serverclassfilene som genereres når filene i Java-katalogen kompileres.
    - TransportServer.class
    - KommunikasjonsAgent.class
    - ModusAgent.class
    - KonteksAgent.class
    - testClient.class
- **software –** Inneholder all software som trengs for å kunne bruke transportsystemet
    - MySQL 2.32
    - Java 2 J2SE 1.4.2
- **doc** – Inneholder den skriftlige delen av diplomoppgaven. ”diplomoppgave.doc”
- **bilder** – Inneholder alle bildene som er med i diplomoppgaven
- **html** – Inneholder alle javafilene konvertert til html.
- **sql** – Inneholder ”transportsystem.sql” som viser sql’en som bygger opp databasen. Denne sql’en oppretter databasen og alle dens tabeller.
- **Rational rose** – Rational Rose modeleringen av oppgaven