UiB | Dept. of Information Science and Media Studies
UiTø | Dept. of Computer Science
NTNU | Dept. of Computer and Information Science
Telenor | Telenor R&I

CAIM
CONTEXT-AWARE IMAGE MANAGEMENT

The MMIR prototype for Mobile Image Retrieval

August 2007

Roe Fyllingsnes

Christian Hartvedt

Utvikling av en prototyp for mobil bildefremhenting ved bruk av Nokia N95 og Oracle *inter*Media

CAIM-UiB

SOMMERPROSJEKT 2007

CAIM-TR-2

Innhold

Innhold i
1 Introduksjon 1
1.1 Mål for arbeidet 1
1.2 Gjennomføring 1
1.3 Vurdering av resultat fra sommerprosjektet 3
2 Dokumentasjon av systemet 3
2.1 Databasesystemet 3
2.1.1 Databasestruktur 4
2.1.2 Databaseprosedyrer 4
2.1.3 Reduksjon av bildestørrelsen i databasen 5
2.1.4 Konklusjon 6
2.2 Mobilapplikasjonen og serverfunksjonalitet 7
2.2.1 Teknisk arkitektur 7
2.2.2 Apache Tomcat 8
2.2.3 Verktøy 8
2.2.4 Logisk arkitektur 9
2.2.5 Funksjonalitet 10
2.2.6 Oppsummering av implementasjon av mobilapplikasjon 15
2.2.7 Serversiden 15
2.2.8 Problemer under utviklingen 16
2.2.9 Veien videre 17
3 Kilder 18
Appendiks 19

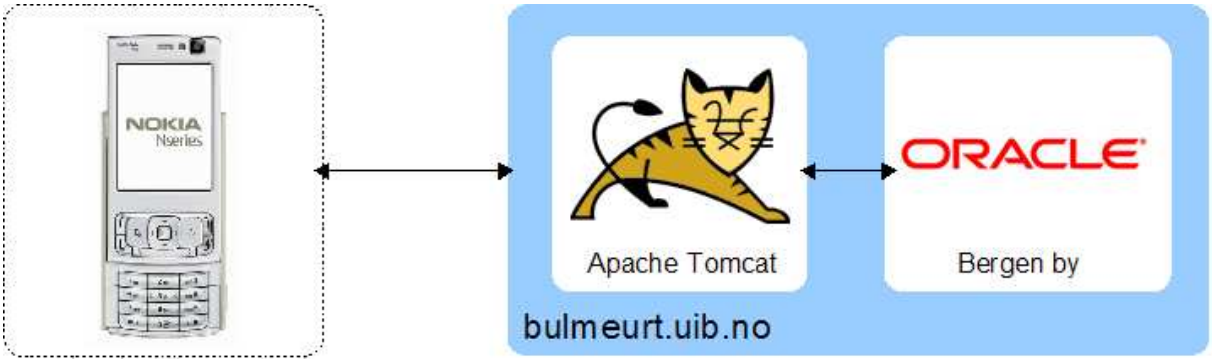
Christian Hartvedt
Roe Fyllingsnes

1 Introduksjon

Denne rapporten er skrevet for å dokumentere et mobilt image retrieval system, MMIR (Mobile Multimedia Information Retrieval), utviklet for Context-Aware Image Management – CAIM prosjektet[1] sommeren 2007. I tillegg kan det tjene som et utgangspunkt for videre arbeid med prosjektet. Rapporten dekker arbeidet utført av Roe Fyllingsnes og Christian Hartvedt, ansatt i prosjektet sommeren 2007.

1.1 Mål for arbeidet

Det overordnede målet for arbeidet sommeren 2007 var å utvikle et prototypsystem som kunne ta et bilde med en Nokia N95 telefon, sende dette over et nettverk til en Oracle interMedia database, gjennomføre en innholdsbasert bildespørring (CBIR) i databasen, hente/motta et resultatsett og presentere resultatsettet på telefonen. Den overordnede arkitekturen til systemet illustreres i figur 1 som viser hvilke komponenter systemet er bygget på.



Figur 1 - Oversikt over systemet utviklet sommeren 2007

Systemet som er utviklet gjør det mulig for en Nokia N95 telefon å kommunisere med en Apache Tomcat instans, som videre igjen kommuniserer med Oracle. Applikasjonsserveren, Oracle og databasen er installert på CAIM DB server - bulmeurt.uib.no.

1.2 Gjennomføring

For å utvikle det systemet som er illustrert i figur 1, ble det satt opp en kravspesifikasjon som blir vist i tabell 1. Kravene retter seg mot henholdsvis Nokia N95 telefonen og serveren den kommuniserer med. I tillegg, er systemet avhengig av et databasesystemet som inneholder en bildesamling tilgjengelig for søk, samt muligheter for kommunikasjon med en ekstern server.

MMIR-systemet utviklet sommeren 2007, benytter det samme databasesystemet og tok utgangspunkt i de samme løsningene for innholdsbaserte bildespørringer (CBIR) som ble brukt av VORTEX Image Search Interface – VISI systemet[2], dokumentert i (Næss, 2007).

[1] CAIM-prosjektet er ledet av Randi Karlsen ved UiTø, og dette sommerprosjektet er ledet av Joan C. Nordbotten ved UiB.
[2] 1st utviklet i 2005 av Øyvind Osdal for VED prosjektet.

Tabell 1 - Kravspesifikasjon for systemet utviklet sommeren 2007

Krav no.	Systemkrav	Påkrevd funksjonalitet
1. Nokia N95	Operere som systemklient som kommuniserer med serversiden til systemet.	Mulighet for installasjon av program skrevet i et formålstjenlig programmeringsspråk
2. Nokia N95 til Server	Opprette en kobling mellom Nokia N95 og serversiden.	Mulighet for klient/server kontakt over nettverk ved oppstart av programmet på telefonen
3. Nokia N95 til Server	Overføre bilder fra telefon til server til bruk i CBIR.	Importere bilder fra lokal bildemappe eller ta bilder med N95 telefonen og overføre disse til serveren.
4. Server	Motta bilder fra Nokia N95 til bruk for CBIR.	Kommunisere med databasesystem vha. et passende grensesnitt for databasetilkobling og overføre eksempelbilde som parameter i prosedyrekall.
5. Server	Motta/hente resultat fra databasesystemet	Bruke et passende grensesnitt for databasetilkobling og motta/hente resultat fra CBIR.
6. Server	Prosessere resultat fra databasesystemet.	Konvertere bilde fra formatet brukt i databasesystemet til brukbart format for Nokia N95.
7. Server til Nokia N95	Overføre bilder fra server til Nokia N95.	Implementert funksjonalitet til å tilpasse og overføre bilder til Nokia N95 klienten.
8. Nokia N95	Presentere resultat på N95 skjerm.	Implementert funksjonalitet til å vise de mest relevante treffene 4 og 4.
9. Nokia N95	Fastslå GPS posisjon når et bilde tas.	Hente GPS posisjon med funksjonalitet i Nokia N95 når bildet tas, og tagge bildet med denne posisjonen.
10. Nokia N95	Bruke både bilde og GPS posisjon ved søk.	Spesifisere spørring bestående av bade eksempelbilde og GPS lokasjon.
11. N95 tlf. + Nokia Maps	Lagre lokasjonen der et blide har blitt tatt i telefonens lokale "interessante sted" database for bruk med Nokia Maps.	Tilgang til stedsdatabasen og EXIF-data fra bildet.

Kjerneoppdraget var å møte de 8 første kravene. De resterende kravene anser vi som vesentlige for å øke "brukbarheten" til systemet, men å oppfylle alle kravene i kravspesifikasjonen innen den tidsrammen dette sommerprosjektet hadde var ikke mulig.

Utviklingsarbeidet ble fordelt slik at Roe konsentrerte seg om programmering på telefonen, servletprogrammering, samt oppsett av server. Christian arbeidet med funksjonalitet til bruk på serversiden og konsentrerte seg om kommunikasjon med databasen, prosedyrekall, samt evaluering og eventuell videreutvikling av prosedyrene som allerede var implementert der.

Arbeidet medførte en del utfordringer, spesielt knyttet til at det ikke finns så mye tilgjengelig dokumentasjon på systemer av denne typen, samt at det ved bruk av en mobil enhet blir en reduksjon av prosessorkraft og minne. I tillegg medførte systemutviklingsprosessen og systemarkitekturen til en del uvanlige og uventede problemer som ikke umiddelbart kunne identifiseres og løses, spesielt siden rettighetene vi hadde på både server og databasesystem var relativt begrensede.

1.3 Vurdering av resultat fra sommerprosjektet

Dersom vi skal dra en kort konklusjon kan vi si at hovedmålet for sommerprosjektet stort sett ble oppnådd. Et system som møter de 8 første kravene spesifisert i tabell 1 har blitt laget, og dette systemet gjør bruk av bilder tatt med Nokia N95 overført over nettverk til databasesystemet og brukt som eksempelbilder i CBIR. Resultatene fra bildespørringen hentes ut av databasesystemet og benyttes til å hente ut de bildene med best score i bildespørringen. Disse bildene konverteres til et redusert format, overføres fra server til klient og presenteres for brukeren.

Visningen på telefonskjermen fungerer slik at 4 og 4 bilder blir vist for brukeren i den rekkefølgen databasesystemet bestemmer ut fra utregnet avstand mellom bildesignaturen til eksempelbildet versus bildesignaturene lagret i databasen. Funksjonaliteten er satt opp på en slik måte at brukeren kan navigere seg frem og tilbake gjennom resultatsettet ved hjelp av knappene på telefonen.

Kapittel 2 inneholder en grundigere gjennomgang av de forskjellige delene av systemet.

2 Dokumentasjon av systemet

Dokumentasjonen av MMIR-systemet gitt i dette kapittelet kan fungere som støtte for videre arbeid med systemet, eller som forklaring av systemet og illustrasjon på selve systemutviklingsprosessen. Avsnitt 2.1 presenterer databasedelen av systemet, mens 2.2 tar for seg hovedtrekkene i programvareutviklingen for mobiltelefonen og serversiden som henter bilder fra databasen.

Systemet utviklet sommeren 2007 ble laget ved hjelp av programmeringsspråket Java for klient/tjenerapplikasjonen som kommuniserer med Oracle databasen, mens programmering i databasesystemet ble gjort ved hjelp av Oracles eget programmeringsspråk, PL/SQL.

2.1 Databasesystemet

Databasen, Bergen By, som vi benyttet til test av MMIR-systemet, er implementert v.h.a. en Oracle Database 10g Enterprise Edition Release 10.2.0.3.0. Databasen er under uvikling for CAIM prosjektet og innholder pt. ca. 75 bilder av scener fra Bergen By. Den har tidligere vært aksesserbar gjennom IR prototypen VISI (Næss, 2007), som forutsetter en PC klient. I MMIR-systemet vil klienten være en mobiltelefon (pt. Nokia N95).

Strukturen til, og innholdet i, Bergen By databasen var av en slik karakter at vi kunne benytte de delene vi hadde bruk for så å si uforandret for test av denne versjonen av MMIR systemet. Vi fant det derfor ikke hensiktsmessig å sette opp en egen database, men tilpasset heller strukturen i den eksisterende databasen til vårt formål.

PL/SQL, en ekstensjon av SQL, ble benyttet til å endre litt på tabellstrukturen til Bergen By databasen, samt til å endre den brukerdefinerte prosedyrepakken som definerer funksjonene og prosedyrene brukt internt i databasesystemet for bildeimport og innholdsbaserte spørringer

(CBIR). Endringer knyttet til tabellstrukturen, var oppretting av en resultattype og en resultattabell, mens endringene i prosedyrepakken gikk på å redusere bildestørrelsen til bildene i resultatsettet, samt skrive disse inn i resultattabellen.

2.1.1 Databasestruktur

Databasen vi benyttet oss av i dette sommerprosjektet består av både objekttabeller og nøstede tabeller til bruk for innsetting og oppbevaring av bilder med tilhørende informasjon. Disse tabellene og de ulike funksjonene de har, er beskrevet av Bjørge Næss (2007) og vil ikke bli videre beskrevet her. Av tabellene som allerede var opprettet i databasen benytter MMIR systemet foreløpig kun bildetabellen (IMAGE). Denne består av bildetypeobjekter (IMAGE_TYPE) bestående av bilder og bildeattributter som vist i figur 2.

```
CREATE OR REPLACE TYPE IMAGE_TYPE AS OBJECT
(
  Id              number,
  Caption         CLOB,
  Photographer    REF AUTHOR_TYPE,
  Texts           TEXT_REFTABTYPE,
  Image           ORDSYS.ORDImage,
  ImageSignature  ORDSYS.ORDImageSignature,
  Coordinates     MDSYS.SDO_GEOMETRY,
);
/
```

Figur 2 - Bildetypen brukt i bildetabellen

Figur 2 viser kun kjerneinnholdet i IMAGE_TYPE objektet. Objektet har funksjonalitet ut over det som er illustrert i figur 2. Dette er imidlertid funksjonalitet som ikke er vesentlig for testing av MMIR systemet. For en full oversikt og beskrivelse av typer og tabeller i databasen, se Næss, (2007).

Vi utvidet i tillegg databasen med en tabell for midlertidig oppbevaring av resultatene fra den innholdsbaserte bildespørringen. Årsaken til dette er nærmere omtalt i avsnitt 2.1.3. Resultattabellen består av brukerdefinerte resultatobjekter.

Resultatobjektet og resultattabellen brukt i denne versjonen av MMIR-systemet, består av bilde-ID, bildescore, bildetittel og selve bildet. Det er ingen forventede problemer knyttet til å utvide objektet og tabellen til å omfatte andre verdier som f. eks. GPS koordinater.

2.1.2 Databaseprosedyrer

Som nevnt i seksjon 1.2, tar prosedyrene brukt i MMIR-systemet beskrevet her utgangspunkt i prosedyrepakken VISI_BERGENBY, som er prosedyrepakken brukt av VISI-systemet. Denne pakken inneholder funksjoner og prosedyrer for å gjennomføre import av eksempelbilder, samt utføre bildespørringene i Oracle databasesystemet. Resultatet som returneres er en streng bestående av bilde-ID til resultatbilder sortert etter bildescore.

For å tilpasse funksjonaliteten til MMIR-systemet, ble pakken kopiert og gitt navnet MMIR_BERGENBY. Dette ble gjort for å unngå at eventuelle endringer skulle påvirke negativt inn på funksjonaliteten til VISI-systemet.

Prosedyrepakken MMIR_BERGENBY består i all hovedsak av de samme funksjonene og prosedyrene som VISI_BERGENBY pakken, men er utvidet med en funksjon for å slette innholdet i resultattabellen etter bruk. Pakken består da av følgende funksjoner og prosedyrer (funksjon = f, prosedyre = p):

- SEARCH (p)
- IMPORT_SEED (f)
- IMAGE_SEARCH (f)
- CLEAR_TABLE(p)

SEARCH prosedyren tar inn følgende parametere: Url til bildemappe, filnavnet til eksempelbildet, verdier for vekting av syntaktiske trekk (Shape, Color, Texture, Spatial), terskelverdi, og en resultatvariabel som skal fylles med bildenumrene til bildene som databasesystemet returnerer fra CBIRsøket. Prosedyren bruker deretter funksjonen IMPORT_SEED for å hente eksempelbildet som skal benyttes til bildespørringen fra en ekstern mappe.

IMPORT_SEED funksjonen tar Url til bildemappen, filenavnet til bildet, og bildenummeret til eksempelbildet (en verdi generert i SEARCH prosedyren og satt til til ”1”) som parametere, og setter eksempelbildet inn som bilde nummer ”1” i bildetabellen (IMAGE).

Bildetabellen setter alltid inn eksempelbildet som nr ”1”, så det første funksjonen gjør er å slette eksempelbildet brukt i forrige spørring. Deretter genereres bildesignaturen og bildet settes inn i bildetabellen (IMAGE). Funksjonen returnerer en sann eller falsk boolsk verdi alt ettersom om bildeinnsettingen er vellykket eller ikke. Hvis den returnerte verdien er sann (true), kalles IMAGE_SEARCH funksjonen med id til eksempelbildet, vektene og terskelverdien som parametere.

IMAGE_SEARCH funksjonen gjennomfører en innholdsbasert spørring (CBIR) basert på bruk av eksempelbildet, vektene og terskelverdien, og skriver ut resultatbilde-ID separert med komma som en tekststreng. Denne fylles inn i en resultatvariabel i SEARCH prosedyren som igjen returnerer denne tekststrengen dit prosedyrekallet kom fra.

Prosedyren **CLEAR_TABLE**, som kan kalles både eksternt fra Java applikasjonen eller internt i Oracle, sørger for tømming av innholdet i tabellen som blir gitt som parameter. I MMIR-systemet brukes prosedyren til å tømme den midlertidige resultattabellen slik at denne er klar til å ta i mot resultatene fra neste bildespørring.

2.1.3 Reduksjon av bildestørrelsen i databasen

Et merkbart trekk ved testing av denne versjonen av systemet er at det tok relativt lang tid å laste bilder fra databasen. I og med at klienten er en mobiltelefon med de begrensninger det medfører i minne og prosessorkraft, fant vi det derfor nødvendig å redusere bildestørrelsen før bildene presenteres for brukeren. Til en viss grad er dette også fordelaktig med tanke på mengden data som overføres til mobiltelefonen.

Ett alternativ er å redusere bildestørrelsen på bildene i resultatsettet før disse returneres fra databasesystemet.

Siden bildematerialet som benyttes i databasen kan være underlagt opphavsrettigheter og lignende, er det ikke nødvendigvis sikkert at det er en mulig løsning å redusere bildestørrelsen

på bildene når de legges inn i databasen. Med dette i tankene arbeidet vi for å finne en løsning som sørget for at størrelsen på bildene i resultatsettet ble redusert mens de originale bildene som var lagret i databasen forble uforandret.

Flere ulike tilnærminger til reduksjon av størrelsen på bildene i databasen ble satt opp og evaluert. De fleste av disse er basert på ulike former for direkte manipulasjon av et cursor-objekt opprettet i IMAGE_SEARCH prosedyren.

Selv om noen av disse variantene virket lovende mht. hastighet, var de nokså ustabile mht. om prosedyren faktisk kjørte. Siden tidsrammen for prosjektet var relativt stram, var det ikke tid til å gå grundig inn å analysere hvorfor disse problemene oppsto. Vi besluttet derfor å implementere en noe enklere løsning basert på tre separate operasjoner som behandlet resultatsettet i søkeprosessen.

Tilnærmingen som ble implementert bygger også i all hovedsak på funksjonaliteten implementert i prosedyrepakken VISI_BERGENBY, men i tillegg er IMAGE_SEARCH funksjonen utvidet med funksjonalitet for å skrive resultatene fra bildespørringen til en midlertidig tabell. IMAGE_SEARCH funksjonen er også utvidet med funksjonalitet for å redusere størrelsen på bildene i resultatsettet til ikke å overskride en gitt maksimumsverdi i antall piksler. Samtidig sørges det for at det opprinnelige forholdet mellom bredde og høyde i bildet opprettholdes. Den utvidede delen av IMAGE_SEARCH funksjonen sørger for fylling av resultattabellen med at bilde-ID, bildescore. Bildetittel fra bildespørringen settes inn i den brukerdefinerte resultattabellen samtidig som en instans av ORDImage objekttypen initialiseres. Deretter reduseres bildestørrelsen før resultattabellen oppdateres med det nye bildet.

2.1.4 Konklusjon

Prosedyrespråket PL/SQL og den implementerte funksjonaliteten for innholdsbasert bildefremhenting i Oracle fungerer greit som utviklings og testverktøy for MMIR-systemet hvis man ser bort fra problemer knyttet til den eksisterende funksjonaliteten for innholdsbasert bildefremhenting i Oracle. Disse problemene ligger imidlertid utenfor det denne rapporten og dette sommerprosjektet skal dekke, men hvis det skulle komme forbedringer i Oracle's løsniger for CBIR, vil også MMIR-systemet nyte godt av disse.

Dyptgående endringer av databaseprosedyrene har ikke vært sentralt i dette sommerprosjektet da det overordnede målet her har vært å slå fast om det er mulig å benytte Nokia N95 som klient i innholdsbasert bildefremhenting. Når det gjelder prosedyrene implementert i VISI_BERGENBY pakken, fungerer disse greit også som et utgangspunkt for MMIR-systemet. Når det gjelder VISI_BERGENBY løsningen som returnerer bilde-ID til resultatbildene samlet i en tekststreng for så å splitte denne vha. en løkke som igjen kjørte en SELECT spørring over nettverket for hver bilde-ID, virket denne i utgangspunktet litt tungvint siden Java innehar funksjonalitet for å hente ut hele resultatcursorer og dermed kan hente resultatsettet direkte fra bildespørringen. Siden vi i tillegg hadde behov for å redusere bildestørrelsen, modifiserte vi måten å hente ut resultater fra Oracle databasen. Den opprinnelige funksjonaliteten med retur av tekststrengen er likevel beholdt som en måte å kontrollere at databasesystemet kjører de ulike prosedyrene.

2.2 Mobilapplikasjonen og serverfunksjonalitet

Java ble benyttet til utvikling av applikasjonen fordi dette programmeringsspråket både er velegnet til å skrive applikasjoner til bruk på Nokia N95 og i tillegg har funksjonalitet som enkelt legger til rette for tett kommunikasjon med applikasjonsserver og database.

2.2.1 Teknisk arkitektur

Som tidligere illustrert bygger systemet på hovedkomponenter i form av en Nokia N95 mobiltelefon, en Tomcat Apache web container[3] og Oracle interMedia.

Programvareutvikling for N95 med Java ME

Plattformen for Nokias N95 er s60-plattformen, som baserer seg på telefoner som benytter operativsystemet Symbian OS. De fleste telefoner som kommer med denne plattformen er telefoner som befinner seg innenfor smarttelefonkategorien. I all korthet, vil det si telefoner som er beregnet til annet bruk enn bare telefonsamtaler og tekstmeldinger. Per andre kvartal 2007, er N95 en av de mest avanserte mobiltelefonene på markedet. Av de egenskaper som er mest interessante for sommerprosjektet er innebygd GPS, WLAN og et 5 megapixel kamera[4].

Mulighetene for programvareutvikling beregnet til mobiltelefoner bygd på s60-plattformen, defineres gjennom hvilken versjon av s60-plattformen mobiltelefonen støtter. N95 har støtte for den til nå siste versjonen, "S60 3rd Edition, Feature Pack 1". Dette betyr dermed at det med jevne mellomrom kommer oppdateringer, om mobiltelefonen har støtte for siste oppdatering er avhengig av hvorvidt mobiltelefonens firmware har blitt oppgradert for å støtte den nye versjonen av plattformen. For å kunne utvikle programvare må man dermed benytte en såkalt SDK (Software Development Kit), som gjør tilgjengelig de forskjellige "byggesteinene" man kan bruke i utviklingen av programvaren. Nokia gjør SDKer for Symbian OS tilgjengelig for nedlasting, både for utvikling i programmering i Java og Symbian / C++. De forskjellige telefonprodusentene utgir egne, spesialiserte SDKer, men disse SDKene inneholder visse komponenter som er felles når det gjelder Java.

Nærmere bestemt dreier dette seg om Mobile Information Device Profile (MIDP) og Connected Limited Device Configuration (CLDC), som til sammen er standarden for det som er Java Platform, Micro Edition (Java ME) (tidligere også betegnet som Java 2 Platform, Micro Edition (J2ME)). Disse standardene spesifiseres gjennom en ekspertgruppe i Sun[5].

MIDP 2.0 og CLDC 1.1 er siste versjon som definerer Java-rammeverket for utvikling av programvare for mobiltelefoner. Disse oppdateres også med jevne mellomrom. Dette skjer gjennom en ekspertgruppe i en Java Community Process (JCP). Slike ekspertgrupper jobber også med å definere fremtidige spesifikasjoner som skal gjøres tilgjengelige i rammeverket. Disse Java Specification Request-ene (JSR) kan dermed være en større JSR (for eksempel MIDP 2.0) eller være en del av MIDP 2.0 spesifikasjonen. For eksempel JSR-135, som sier noe om hvilke muligheter som er tilgjengelige i form av lyd, bilde og video[6].

[3] Apache Tomcat betegnes som en Web Container, og skiller seg dermed fra en full J2EE implementasjon.
[4] For flere detaljer rundt hvilke egenskaper N95 har, se: http://forum.nokia.com/devices/N95
[5] Ekspertgruppen består av medlemmer fra mer enn 50 organisasjoner, som inkluderer telefonprodusenter, nettverksoperatører og programvareleverandører: http://java.sun.com/products/midp/
[6] JSR 135, Mobile Media API - http://java.sun.com/products/mmapi/

Mobilapplikasjonen utviklet for dette prosjektet benytter seg av følgende JSR-er, som er bygd opp som pakker tilgjengelig for bruk i Java ME:

- JSR-135 Mobile Media API
- JSR-75[7] FileConnection API
- JSR-197[8] Connection API

Nokia har i tillegg sine egne spesialiserte pakker. Disse har dessverre ikke blitt tatt i bruk, pga. manglende dokumentasjon fra Nokias side på hvordan man kan benytte dem. Når mobilapplikasjonen er rettet mot en bestemt telefon er det større muligheter for å ta i bruk slike spesialiserte pakker, enn hva det hadde vært om mobilapplikasjonen skulle ha vært mer generell for bruk på flere forskjellige typer mobiltelefoner.

2.2.2 Apache Tomcat

For å kunne koble sammen mobiltelefonen med databasesystemet er man avhengig av en form for mellomvare som sørger for kommunikasjonen mellom mobiltelefonen og databasesystemet. Apache Tomcat ble vurdert til å kunne oppfylle kravene som mellomvare basert på at Roe hadde noe tidligere erfaring med bruk av Tomcat og at den er relativt lite kompleks i forhold til andre, mer fullstendige J2EE varianter. I tillegg var omfanget av systemet som skulle lages ikke av en slik størrelse at det krevde noe mer enn servlet-funksjonalitet.

2.2.3 Verktøy

Her følger en liste over hvilke verktøy (programvare) som har blitt brukt i prosjektet så langt. Når det gjelder valg av verktøy for utviklingen av mobilapplikasjonen, er det ikke foretatt noen større vurdering enn at Nokia så ut til å ha tilgjengelig de verktøyene det var behov for. Da utviklingen av mobilapplikasjoner var et ubeskrevet blad, falt valget på disse verktøyene fordi de var tilgjengelige og man kom kjapt i gang med arbeidet:

Eclipse 3.2.2[9] – tidligere erfaring med verktøyet og muligheter for å benytte plugins fra Nokia, gjorde at Eclipse var et naturlig valg.

Nokia Carbide.j[10] – Nokias plugin for utvikling av mobilapplikasjoner i Eclipse. Når man har installert Carbide.j får man en ny meny i Eclipse. I denne menyen har man valg for å kjøre javakoden i emulator og valg for å lage .jad og .jar-filer for installering på mobiltelefonen. I løpet av sommeren har der i mot Nokia lagt ned Carbide.j for videre utvikling og anbefaler heller nå bruk av åpenkildekode alternativer som EclipseME[11] og NetBeans Mobility Pack[12]. For videre arbeid vil det være naturlig å undersøke disse to andre alternativene..

S60 SDK[13] – selve rammeverket som programmeringen baserer seg på. Her følger også med emulator og en debug agent for on-device debugging.

---

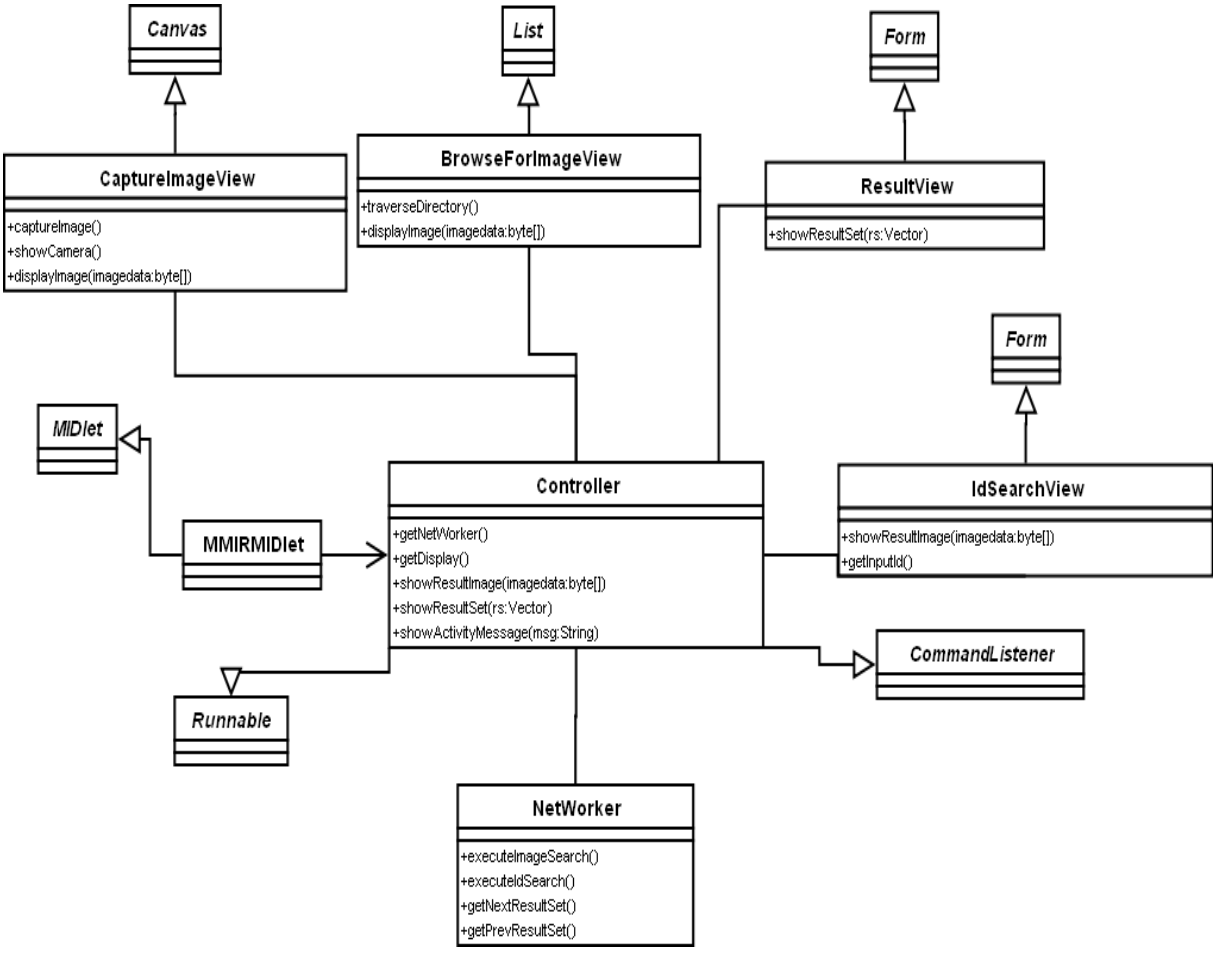
[7] JSR-75 - http://jcp.org/en/jsr/detail?id=75
[8] JSR-197 - http://jcp.org/en/jsr/detail?id=197
[9] Eclipse – http://eclipse.org
[10] Carbide.j er ikke lenger tilgjengelig for nedlasting:
http://forum.nokia.com/main/resources/tools_and_sdks/carbide/what_has_happened.html
[11] EclipseME - http://www.eclipseme.org/
[12] NetBeans Mobility Pack - http://www.netbeans.org/products/mobility/
[13] S60 SDK nedlasting - http://www.forum.nokia.com/info/sw.nokia.com/id/6e772b17-604b-4081-999c-31f1f0dc2dbb/S60_Platform_SDKs_for_Symbian_OS_for_Java.html

Debug Agent – program som må overføres til mobiltelefonen for at man kan kjøre programmet fra utviklingsmiljøet direkte på mobiltelefonen. Installasjonsfilen finnes der man har valgt å installere S60 SDK `<din mappestruktur>\S60\devices\S60_3rd_MIDP_SDK_FP1\s60tools\Ecmt\`

Dia [14] – Åpenkildekode programvare for å tegne forskjellige typer diagrammer i. Dette har blitt benyttet til å tegne ulike UML-diagrammer.

2.2.4 Logisk arkitektur

Figure 3 viser en model over modulene i ***Mobilapplikasjonen***



Figur 3 - klassediagram, som også viser de viktigste metodene og attributtene for hver klasse

Model-View-Controller (MVC)[15] er et designmønster som sørger for at det er en klasse som kontrollerer flyten i programmet og kode for brukergrensesnittet er separert for koden som tar seg av data som vises i brukergrensesnittet. Goyal (2006) bruker også dette designmønsteret i et større eksempel i boken. I dette tilfellet har denne kontrollerklassen fått navnet Controller. Og som man kan se ut i fra klassediagrammet er denne klassen koblet til alle andre klasser. Dette medfører at alle de andre klassene går gjennom Controller-klassen for å kommunisere

[14] Dia - http://www.gnome.org/projects/dia/
[15] MVC - http://en.wikipedia.org/wiki/Model-view-controller

med andre klasser. Dermed styrer Controller-klassen også de fleste av hendelsene som brukeren foretar seg, ved at såkalte lyttere henvises til denne klassen. Det vil i praksis si at om brukeren foretar et valg i en klasse, så utføres selve koden i Controller-klassen. Dermed har man muligheter for å benytte felles metoder som er tilgjengelige i Controller-klassen. I dette tilfellet er dette praksisen som benyttes når applikasjonen ber om å få koble til serversiden.

MVC-patternet blir ikke fulgt i like stor grad gjennom hele applikasjonen, noe av grunnen til det er at Model-delen av MVC ikke er like eksplisitt som de andre klassene. Dvs. at det ikke finnes en egen klasse som håndterer og tar vare på data som flyter i programmet. Applikasjonen blir dermed ikke like ”objekt-orientert” i alle tilfeller, men det er gjort slik for at applikasjonen skal være minst mulig ressurskrevende og fordi det ikke er mulig å sende objekter som holder på data direkte over nettverket i Java ME.

2.2.5 Funksjonalitet

Her følger en oversikt over hvilke ansvarsområder og funksjonalitet de ulike klassene har, samt skjermbilder tatt fra kjøring av applikasjonen i emulator:

MMIRIDlet – Dette er klassen som er selve ”skallet” som starter opp applikasjonen. Videre har den også ansvar for å avslutte applikasjonen når det er aktuelt. Klassen er av typen MIDlet, som er det Java ME oppfatter som et kjørbart program. Denne klassen starter to tråder, en tråd for Controller-klassen og en for nettverkskommunikasjon.

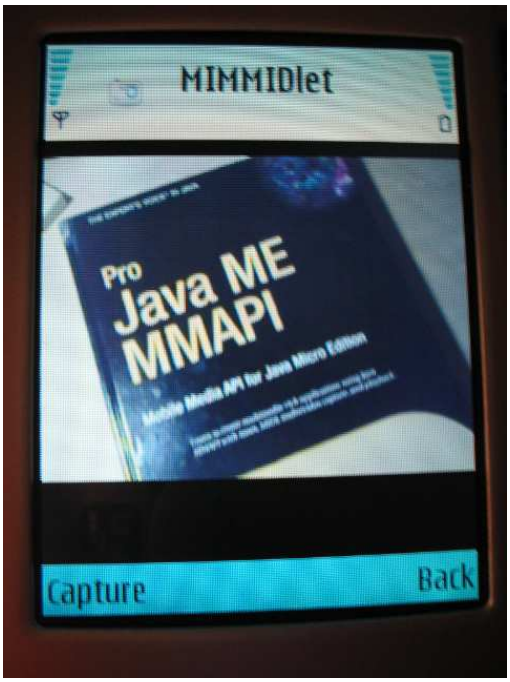
Figur 4 - menyen over mulige valg som vises ved oppstart av applikasjonen.

Controller – Knytter sammen alle klassene og gir tilgang til nettverkskommunikasjon. Lytter på hvilke valg som foretas i de ulike klassene og foretar eksekvering av kode der etter. Ved oppstart er det Controller-klassen som genererer oppstartsmenyen, hvor man velger hva man ønsker å gjøre. Her blir det brukt en meny som er en liste, noe som også benyttes i de innebygde menyene på mobiltelefonen. Ikonene som vises her er for anledningen lånt fra IM-

klienten Pidgin[16]. Applikasjonsikonet er selvkonstruert og med litt godvilje kan man kanskje se at det skal forestille et kjent motiv fra Bergen. Ikonene i menyen er ikke helt tilpasset telefonens skjermoppløsning og fremstår derfor litt ute av posisjon[17].

NetWorker – Kobler til serveren og har metoder som kan kalles for å sende og motta data fra serveren. Klassen tar seg av de rå data som kommer inn og formidler dem videre til Controller-klassen som igjen formidler og viser frem dataene slik de skal være vist frem i det passende view-et. Praksisen med å ha en egen tråd som styrer nettverkstilkoblingen gjør at applikasjonen ikke står og henger, og at applikasjonen ikke krasjer selv om nettverksforbindelsen skulle svikte.

CaptureImageView – Gjør tilgjengelig selve kameraet. Display viser en rute av det man måtte ønske å ta bilde av, i vanlig digitalkamerastil. Menyvalgene viser ”Capture” og ”Back”. For å ta et bilde trykker man ”Capture”. Dette kan også gjøres ved å trykke på den store knappen i midten, som føles mer naturlig å bruke. Når et motiv er valgt og bildet er tatt vises bildet i fullskjerm.



Figur 5 - N95 som kjører applikasjonen og viser funksjonalitet for å ta et bilde.

Brukeren får nå to nye valg ”Search” eller ”Discard”. ”Discard” gir brukeren mulighet til å ta et nytt bilde. Velges ”Search” gjennom føres et bildebasertsøk. CaptureImageView kaller da på en metode i Controller-klassen, som igjen kaller på en metode i NetWorker-klassen. Denne klassen tar i mot bilde som datatypen byte-tabell (byte[]). Og gjør oppkoblingen mot serveren og overføringen av bildedataene.

[16] Instant Messaging-klienten Pidgin – http://pidgin.im
[17] For mer informasjon om bruk av ikoner - http://www.forum.nokia.com/info/sw.nokia.com/id/a309ec6b-5d4c-4f5d-aeaa-421d2f727428/Using_Icons_in_MIDlets_v1_0_en.zip.html

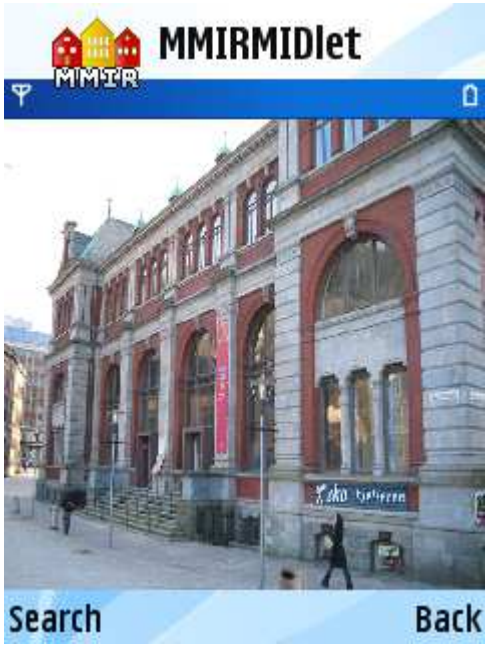
Som man kan se av skjermbildet (figur 5), dekker ikke ”kameralinsen” hele skjermen. Grunnen til dette er man kun har tilgjengelig MIDP-implementasjonen av kameraet og dette er begrenset i forhold til hva man kan oppnå med den innebygde ”kameraapplikasjonen”. Oppløsningen på bildene som blir tatt med applikasjonen er 800 x 600 pixel. Det vil også være mulig å ta bilder med større oppløsning, men dette vil være en avveiing opp mot minneforbruk av applikasjonen. I de tilfeller hvor det ble testet med større oppløsning (1024 x 768) greide ikke applikasjonen å håndtere bildedataene og applikasjonen krasjet. Mer finjustering av bruken av ressurser i applikasjonen kan sannsynligvis gjøre det mulig å ta bilder med større oppløsning.

Javas sikkerhetsprofil skal gjøre det sikrere å kjøre tredjepartsapplikasjoner som denne, noe som går ut på at brukeren må aktivt godkjenne eller avvise om applikasjonen får bruke telefonens ressurser. I dette tilfellet kommer dette frem når man har tatt et bilde og når man skal foreta et søk. Brukeren vil da bli spurt om det tillates at applikasjonen kan benytte seg av henholdsvis kameraet og nettverksforbindelsen.

BrowseForImageView – Tillater navigering i filstrukturen på mobiltelefonens minne og minnekort. Som vist i figur 6 vises kun filer som er av typen *.jpg. Når en jpg-fil er valgt får brukeren opp bildet i fullskjerm. Brukeren har da muligheten til å gjøre et søk ved å velge ”Search”. Søket foregår på samme måte som beskrevet over angående for CaptureImageView og ResultImageView.

Figur 6 - viser navigering i filsystemet for å finne et bilde for opplasting.

figur 7 viser bildet som ble valgt i fullskjerm og valgmulighetene brukeren har. Svakheten ved denne klassen er at Javas sikkerhetsprofil sørger for at for hver gang man åpner en mappe eller en fil, så blir man spurt om du vil tillate applikasjonen å lese innholdet i filsystemet.



Figur 7 - forhåndsvisning av bildet som brukes i søket

ResultImageView – Denne klassen sørger for å vise frem bildene som er resultatet av bildespørringen. Når et bildesøk er gjennomført gjøres et metodekall fra NetWorker, gjennom Controller, til ResultImageView som tar i mot en tabell av fire bilder. Disse bildene vises som et rutenett på skjermen og er nummerert i stigende rekkefølge, re. Figur 8.

Figur 8 - viser hvordan resultatbildene vises på skjermen

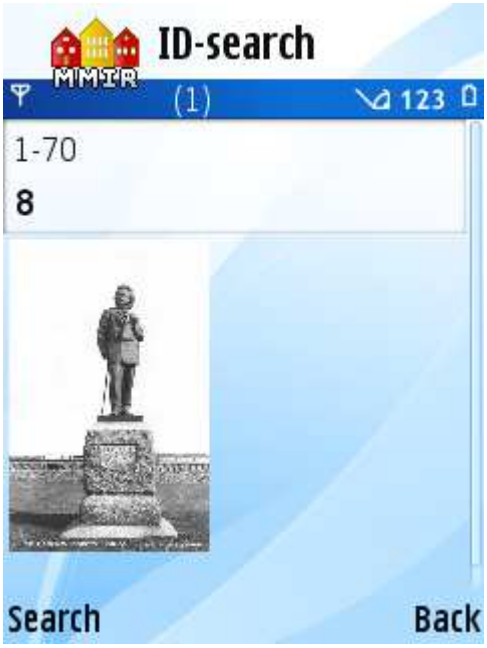
Brukeren får nå opp ulike valg avhengig av hvor man er i navigeringen av bildesettet. Om det er de fire første bildene som vises kan brukeren velge ”Exit” eller ”Next”. Velges ”Next” går igjen programflyten gjennom Controller-klassen og til NetWorker, som kommuniserer med serversiden og ber om fire nye bilder.

Måten man navigerer mellom fire og fire bilder på kunne ha blitt gjort på andre måter. For eksempel ved at man heller hadde latt brukeren bla nedover med navigasjonsknappen, slik man gjør i vanlige menyer. For å oppnå mer flyt i selve navigeringen av bildene kan det også være en idé å se på mulighetene for å laste ned mer enn fire bilder om gangen. Når bildene blir forminsket allerede i det de blir hentet ut fra databasen, vil de ha en vesentlig mindre størrelse (100x100 pixler), noe som gjør at det vil være mulig å håndtere mer enn fire bilder om gangen.

Hvis man ser på vilken oppløsning slike minatyrbilder ofte har i telefoners innebygde funksjonalitet for forhåndsvisning av bilder, ser man at det ofte brukes enda mindre størrelser, slik at det gjerne vil være plass til 4 x 3 bilder. Der i mot vil det i slike tilfeller være muligheter for flere visningsmoduser, som f.eks. at man kan se bildet i fullskjerm og navigere mellom bilder i fullskjerm. Dermed vil dette være et forhold mellom størrelsen på skjermen, antall miniatyrbilder man ønsker å vise frem og hvor store miniatyrbildene må være for at de skal kunne gi et visst inntrykk av hva bildet er av. For videre utvikling av denne applikasjonen bør det prioriteres å lage funksjonalitet som gjør det mulig å vise ett og ett bilde fra resultatsettet i fullskjerm. Slik funksjonalitet vil igjen gjøre det mulig å bruke mindre miniatyrbilder, da man har mulighet til å se nærmere på bildet i fullskjerm.

Navigasjonsfunksjonaliteten er også noe mangelfull og det kan fort oppstå situasjoner hvor nummereringen av bildene ikke stemmer overens med det faktiske nummeret bildet har i rekkefølgen i resultatsettet. Noe av grunnen til dette er at holdes to separate tellere, en på serversiden og en på klientsiden, som holder styr på hvor man er i navigeringen. Dette er ikke en optimal løsning og bør forandres.

IdSearchView - Denne funksjonaliteten ble laget helt i starten som en enkel test for å få til kobling mot databasen og å kunne vise et bilde, hentet fra databasen, på skjermen. Brukeren oppgir en ID, som returnerer det bildet som har den IDen i databasen.



Figur 9 - viser resultatet av et søk etter bilde med ID 8

AboutView – Gir kort informasjon om prosjektet / applikasjonen.

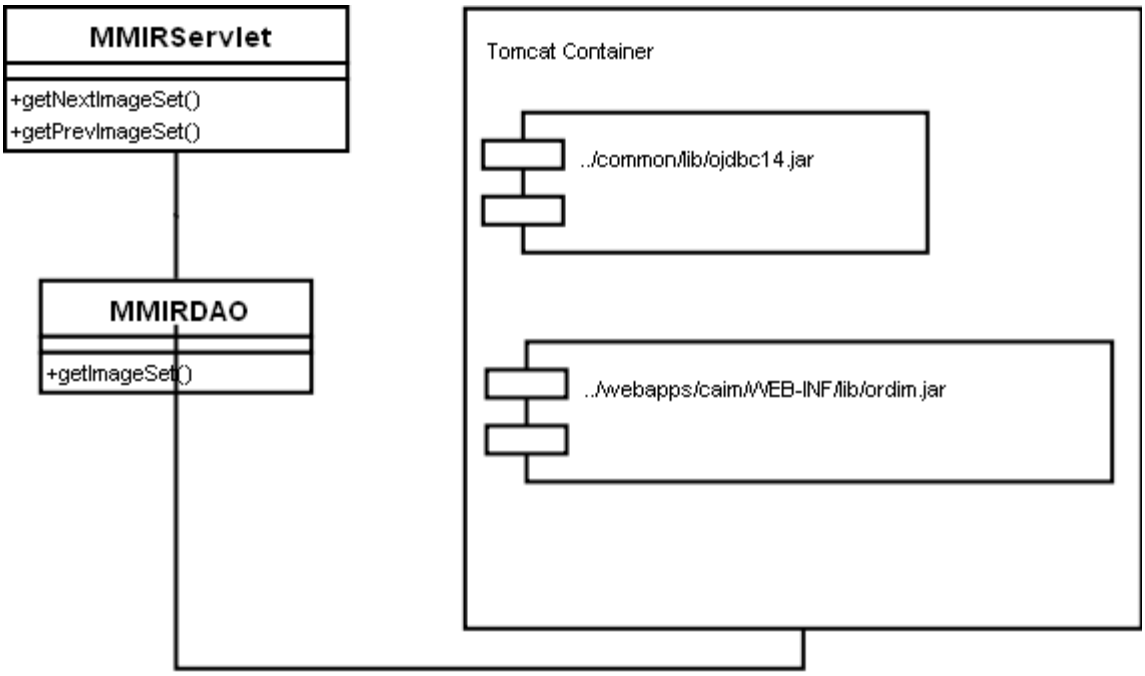
2.2.6 Oppsummering av implementasjon av mobilapplikasjon

Noen av klassene følger ikke MVC-patternet i like stor grad. Grunnen til dette er at koden ikke skal bli for omstendig og for å ha bedre kontroll med ressursbruken i applikasjonen.

Den viktigste funksjonaliteten er i så måte; å kunne ta bilde og få et navigerbart resultatsett tilbake. Appendiks A er et sekvensdiagram, som viser flyten gjennom systemet fra et bilde blir tatt til et resultatsett vises på skjermen igjen.

2.2.7 Serversiden

figur 10 forklarer hvilke to klasser serversiden består av og hvilke klassebiblioteker som benyttes av disse to klassene. Serversiden er mindre kompleks i forhold til antall klasser, da den kun består av to klasser – MMIRServlet og MMIRDAO. Data Access Object (DAO) forteller at dette objektet har ansvar for all rå data som hentes fra databasen.



Figur 10 - diagram som viser klasser og klassebibliotek i bruk av servleten.

Klassebiblioteket `ojdbc14.jar` brukes i dette tilfellet indirekte gjennom Tomcat Containeren. Dette gjør Tomcat for å kunne kontrollere tilkoblinger som gjøres mot databasen og håndtere disse på en fornuftig måte. Med ansvaret for tilkoblinger fra servlet til databasen, tildelt Tomcat slipper man å programmere funksjonalitet for å koble opp mot databasen. Hvilken databasedriver, brukernavn, passord og url konfigureres i Tomcats innstillinger i server.xml-filen.

Det andre klassebiblioteket i diagrammet er `ordim.jar` som inneholder klasser som gjør det mulig å hente ut bilderesultater fra databasen i form av `OrdImage`-objekter. For å få dette til må man ta i bruk databasetilkoblingen og kunne ta i mot resultatsettet man får tilbake i retur fra databasen. Siden det nå er Tomcat som er ansvarlig for denne tilkoblingen, samt håndteringen av resultatsettet som kommer tilbake, må man benytte seg av innebygd funksjonalitet i Tomcat for å få resultatsettet tilbake i mer håndterbar form.

For å aktivere serversiden av applikasjonen i Tomcat, kan man pakke de kompilerte filene sammen i en .war-fil. War-filer må i tillegg ha en spesiell filstruktur for at Tomcat skal kjenne dem igjen som en webapplikasjon.

2.2.8 Problemer under utviklingen

Et av de største problemene under utviklingen var å finne ut av hvordan vi skulle få til koblingen mellom Tomcat og Oracle, slik at Tomcat kontrollerte koblingen. Det er kanskje lett å si det nå, men løsningen som viste seg å fungere, burde ha vært innen rekkevidde å komme frem til på et tidligere tidspunkt.

Et annet problem som tok opp mye tid var testingen av applikasjonen i emulatoren. Emulatoren så til tider ut til å leve sitt eget liv, hvor man fikk startet opp emulatoren, men

ikke fikk kjørt applikasjonen i emulatoren. Her burde man kanskje ha tatt seg tid til å ta en titt på andre utviklingsmiljø / emulatorer, for å vurdere om det ville ha svart seg å bytte til disse for å få en mer stabil emulator.

2.2.9 Veien videre

Naturlig nok vil en fortsettelse fra hvor langt man har komt etter kravspesifikasjonen være et naturlig steg videre, foruten å forbedre den funksjonalitet som allerede er implementert. I følge kravspesifikasjonen vil det dermed være å inkorporere GPS data inn i bilder som metadata. Per i dag støtter ikke Java ME tilgang til bilders metadata. Det vil si at det ikke er mulig å få tak i eller angi meta data på selve mobiltelefonen. Men slik det ser ut nå, er det en ny JSR under arbeid som vil gjøre det mulig å behandle metadata også på mobiltelefonen[18]. Når denne vil være implementert i noen av leverandørenes SDKer er imidlertidig usikkert. N95 legger heller ikke til metadata om GPS-koordinater inn i bilder som blir tatt med det innebygde kameraet. Alternativet nå er å overføre bildedata og metadata separat til serversiden, or så å legge metadata inn i bildet der.

Da må serversiden av systemet utvides og her kunne det nok ha vært en fordel med et mer fleksibelt system, enn det som er status i dagens system. Dette gjelder selvsagt koden slik den er, men også andre applikasjonsservere enn Tomcat kan tenkes å være aktuelle å bytte til. Når det gjelder koden er det spesielt mangel på ”objekt-orienterthet” som kan forbedres.

Roe kommer til å bygge videre på dette arbeidet i form av en masteroppgave og vil i første omgang se på hvordan brukere kan aktivt skille ut objekter i bildet som er av interesse, for så å gjøre en spørring på objektet som brukeren har valgt ut.

[18] “The Java ME GUI APIs at a Glance” - http://developers.sun.com/mobility/midp/articles/guiapis/#5

3 Kilder

Goyal, V. (2006). *Pro Java ME MMAPI.* Apress.

Knudsen, J. (2006). *Beginning J2ME: From Novice to Professional*, Third Edition. Apress.

Næss, B. (2007) *The VISI Prototype*. CAIM-TR-1. Dept. of information and media sciences, Univ. of Bergen.

Internettkilder:

Forum Nokia: http://forum.nokia.com

Oracle interMedia Audio, Image, and Video User's Guide and Reference Release 8.1.6: http://download.oracle.com/otn_hosted_doc/intermedia/inter.816/a67299/index.htm

Appendiks

A - Sekvensdiagram