Utvikling og portering av Java ME applikasjoner

Thomas Vervik
Master Informatikk
Universitetet i Bergen

June 2, 2008

Innholdsfortegnelse

1 Introduksjon til oppgaven **4**
- 1.1 Beskrivelse av eksisterende system . . . 5
  - 1.1.1 Global One Solutions . . . 5
  - 1.1.2 Global One Solutions arkitektur . . . 10
- 1.2 Mobiltelefoner . . . 13
  - 1.2.1 Nokia . . . 14
  - 1.2.2 SonyEricsson . . . 16
  - 1.2.3 SuMobil på forskjellige telefoner . . . 16

2 Java ME standarden og portering **19**
- 2.1 Java ME mål og arkitektur . . . 20
  - 2.1.1 Konfigurasjoner . . . 21
  - 2.1.2 Profiler . . . 26
- 2.2 Java ME Portering . . . 28
  - 2.2.1 Hvorfor eget rammeverk for Java ME utvikling? . . . 30
- 2.3 Portering ved bruk av Aspekt Orientert Programming . . . 31
  - 2.3.1 Introduksjon Aspekt Orientert Programmering . . . 32
  - 2.3.2 Portering ved hjelp av AOP . . . 33
  - 2.3.3 Evaluering av portering av Java ME applikasjoner ved bruk av AOP . . . 35
- 2.4 J2ME Polish . . . 35
  - 2.4.1 Introduksjon J2ME Polish . . . 36
  - 2.4.2 Enhetsdatabasen . . . 37
  - 2.4.3 Introduksjon til byggingen . . . 42
  - 2.4.4 Polish GUI . . . 45
  - 2.4.5 Evaluering av J2ME Polish . . . 47
  - 2.4.6 Evaluering av J2ME Polish kontra bruk av AOP . . . 48

3 Alternative teknologier **49**
3.1 Mobil nettleser - Opera Mini 49
3.1.1 Historie 50
3.1.2 Hvordan virker Opera Mini 50
3.1.3 SuMobil løsningen i Opera Mini 52
3.1.4 Portering ved bruk av Opera Mini 55
3.2 WAP 55
3.2.1 SuMobilløsningen utviklet i WAP 57

4 SuMobil **58**
4.1 SuMobil arkitektur 58
4.2 SuMobil mot forskjellige databaser 59
4.2.1 Distribusjon av SuMobil 60
4.2.2 Belasting av kunder 62
4.3 Sikkerhet i SuMobil 62
4.3.1 Sikkerhetsbehov i SuMobil 63
4.3.2 Sikkerhetsmodellen implementert i SuMobil 64
4.4 SuMobil i bruk 66

5 Oppsummering **67**
5.1 Oppdragsgivers målsetning 67
5.2 Undertegnedes målsetning 68

Figurliste

1.1 Skjermbilde av Global One portalen 7
1.2 Skjermbilde av webshopen til en av Susoft sine kunder, sportsbutikken Platou Sport 8
1.3 Skjermbilde av datatransport administrasjonsiden 9
1.4 To brukerscenario for å beskrive samspillet mellom de forskjellige komponentene i systemet til Susoft 17
1.5 To brukerscenario for å beskrive samspillet mellom de forskjellige komponentene i systemet til Susoft 18

2.1 Illustrasjon av Java ME arkitekturen 22
2.2 Konfigurasjoner er nøstede. 22
2.3 Modell av J2ME Polish arkitekturen 38
2.4 Model av enhetsdatabasehierakiet 42
2.5 Klassediagram av Java ME gui'en 46
2.6 Viser hvordan Polish under preprosseseringen vil erstatte stil preprosesserings elementer i kildekoden og ved hjelp av stilfilen polish.css lage et nytt design ved hjelp av Polish sine egne java klasser 47

3.1 Viser datamengden Opera Mini laster ned forhold til konkurrenter, i tillegg til nedlastingshastighet. 51
3.2 Når en bruker surfer på nettet med Opera Mini, blir forespørselen sent via Genera Packet Radio Service (GPRS) til en av Opera Software sine proxytjenere. Tjeneren vil så hente siden, prosessere, komprimere og sende den tilbake til brukerens mobiltelefon. 51

4.1 En modell over SuMobilløsning arkitekturen. 59
4.2 Skjermbilde av SuMobil 63

Kapittel 1

Introduksjon til oppgaven

Denne masteroppgaven består av to deloppgaver.

Den første oppgaven er å utvide Susoft sitt rapporteringssystem. Susoft leverer i dag kassesystemer til detaljhandelen, og i den anledning tilbyr dem et omfattende sett med rapporter som er tilgjengelig via Internett. Oppgaven en blir å ta gjøre deler av denne rapporteringen tilgjengelig på mobiltelefonen.

Den andre oppgaven omhandler portering av Java ME applikasjoner til mobiltelefoner. Sun's programeringsspråk Java skal være plattformuavhengig. Sun's Java Virtual Machine (JVM) tolker bytekoden under kjøring, og JVM vil oversette disse kommandoene til maskinavhengige instruksjoner. Men slagordet til Sun's Java "Write Once, run everywhere" har av onde tunger ofte blitt omdøpt "Write Once, debug everywhere", eller "Write once, get disappointed everywhere", som er uttrykket som brukes for Java ME[2]. Om dette kanskje har et snev av sannhet for Java Standard Edition (Java SE), er det definitivt tilfellet med Java Micro Edition (Java ME). Hver mobilprodusent implementere selv de virtuelle maskinene i henhold til Sun sine spesifikasjoner. Mobiltelefoner har også veldig forskjellige spesifikasjoner og ressurser tilgjengelig. På skrivebordsmaskiner og tjenere kjører alle maskiner av nyere dato den siste Java SE virtuelle maskinen versjon 1.6 til Sun, Hotspot. Men i mobilverden finnes det i dag to forskjellige virtuelle maskiner, alt etter hvilken konfigurasjon man har valgt. I fremtiden kan dette antallet øke. I tillegg finnes disse konfigurasjonene i forskjellige versjoner. Den virtuelle maskinen til Java SE finnes forsåvidt også i forskjellige versjoner, men denne kan enkelt oppdateres når man trenger den nyeste versjo-

nen på en skrivebordsmaskin eller tjener. Denne muligheten har man ikke på mobiltelefoner, da denne programvaren er preinstallert av produsenten. De alle fleste Java programmer skrevet i Java SE eller Java EE kan uten problemer kjøre på alle fleste skrivebordsmaskiner og tjenere - utvikleren trenger normalt ikke tenke på om brukeren har kraftig nok maskin eller ikke. I mobilverden har telefonene veldig forskjellige spesifikasjoner og ressurser tilgjengelig, som skjermstørrelse, heap størrelse, lagringsplass, biblioteker tilgjengelig, osv.

Løsningen på denne utfordringen er å bygge applikasjonen i forskjellige versjoner mot spesifikke eller grupper av mobiltelefoner. Dette betyr i praksis forskjellige "builds" for enkelte eller sett av mobiltelefoner, hvor man tar hensyn til de forskjellige mobilenes egenart. Til dette formålet behøver man et byggerammeverk, og det vil i denne oppgaven bli gjennomgått hva er byggrammeverk er, hvilke rammeverk som finnes på markedet i dag og valgt et av disse for bruk under utviklingen av applikasjonen beskrevet i oppgave en.

1.1 Beskrivelse av eksisterende system

Susoft er et Bergensbasert itfirma etablert i 1982. Den opprinnelige forretningsideen var å levere konsulttjenester innen it. I 1985 startet dem å utvikle et ERP system. Systemet blir kalt SU Applications og ble installert i 700 firma i Norge, inkludert profilerte firma som Posten. Rundt år 1999 valgte dem å satset på en mindre organisasjon og med en ny type programvare. Internett hadde gjort sitt inntog og Susoft ønsket å utnytte denne muligheten til å lage en ny type butikkdataløsning. I mange systemer kommuniserer ikke kassepunktet i butikken med noen sentral tjener. Vil man ha tilgang til salgsdata, rapporter, lagerstatus, osv må man hente disse ut direkte fra programvaren på kassepunktet. Har man bare en butikk er dette overkommelig. Har man en hel kjede begynner dette å bli et logistisk problem. Kjerneproduktet til Susoft er kassepunkter som kommuniserer med en sentral tjener, og hvor man har tilgang til disse dataene fra hvor enn man er via Internett.

1.1.1 Global One Solutions

Global One Solutions er navnet på butikkløsning rettet mot detaljhandelen. Systemet var i utgangspunktet laget for større kjeder med mange butikker, men har med tiden også blitt solgt inn til mindre kjeder og butikker.

Global One Solutions består av følgende deler:

- SuButikk
- Global One Portal
- Netthandel

SuButikk er programvaren som er installert på kassepunktene ute i butikkene. Denne er utviklet på Susoft og skrevet i programeringsspråket PowerBuilder.

Global One Portal (G1) er adminstrasjonverktøyet på tjener siden. Her har kunden tilgang til blant annet å opprette produktstrukturer, legge inn produckter, bestille varer fra hovedkontoret, tilgang til et utall rapporter, osv. Global One Portal er delt inn i åtte deler:

- G1 Back Office
  - Her styrer man opprettelse av butikker, regioner, valuta, kort, osv. I tillegg styrer man brukertilganger til G1 her.
- G1 Product Management
  - Her opprettes produkter, produktstrukter, priser, osv. Tre typer produkter finnes:
    - Regulær vare - en normal vare
    - Variant vare - en sko kan finnes i mange størrelser og varianter
    - Diverse vare - diverse vare for produkter med forskjellig pris
- G1 Vendor Management
  - Her opprettes og administreres produsenter av varer
- G1 Custom Management
  - Verktøy for kundeoppfølging. Viktigste funksjon her er Broadcast funksjonen som gjør at vi kan sende ut kundebrev til kundene
- G1 eProcurement

  - Er innkjøpsfunksjonen på G1. Ideen når man har store kjeder er at butikkene logisk sett kjøper inn varer fra hovedkontoret. Hovedkontoret tjener sine penger ved at man legger på et lite påslag på prisen man tilbyr videre til butikkene. En butikksjef logger seg inn på G1 med en egen bruker. Fra eProcurement kjøper han varene han trenger i butikken sin. G1 vil så generere en elektronisk pakkseddel som blir sendt ut til Shopkeeper i den aktuelle butikken.

- G1 Business Intelligence
  - Tilbyr et bredt spekter av rapporter, fra totalsalg for alle butikkene i kjeden til salgstall for hver enkel selger i en butikk.

På figur 1.1 kan man se et skjermbilde av Global One portalen.

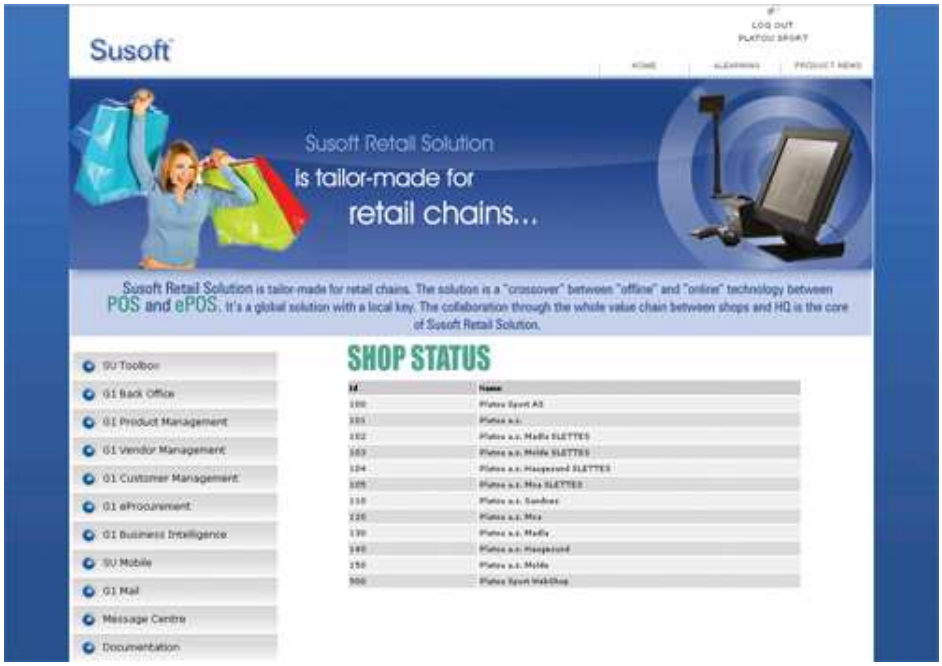

Figure 1.1: Skjermbilde av Global One portalen

Netthandel

Dette er den nyeste delen av Global One Solutions. Ideen er at kundene allerede har lagt inn betydelig med data i form av strukturvarianter, pro-

dukter, produktinformasjon og bilder i systemet. Med en nettbutikk fra Susoft kan kundene enkelt sette opp en nettbutikk med produktene dem har allerede har lagt inn via Global One Portal. Nettbutikken bruker samme database som Global One Portalen. Figur 1.2 viser et skjermbilde av nettbutikken til en av Susoft sine kunder.

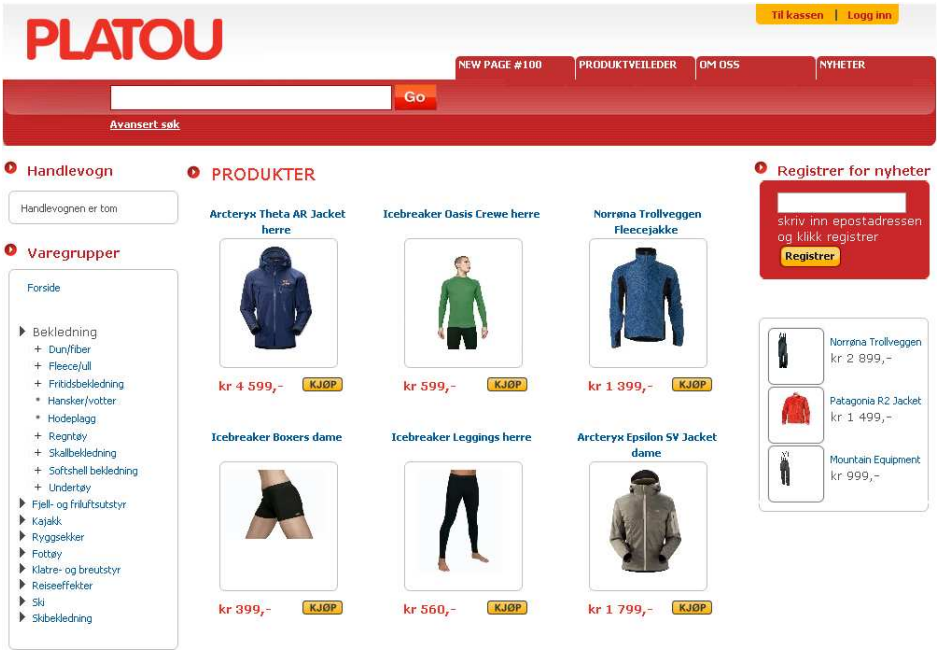

Figure 1.2: Skjermbilde av webshopen til en av Susoft sine kunder, sportsbutikken Platou Sport

Datatransporten

Den fjerde komponenten er "usynlig" for Susoft sine kunder, men allikevel den største delen, og kalles Datatransporten. Denne delen er ansvarlig for å flytte informasjon til og fra SuButikk i butikkene og legge dem inn i databasen på tjenersiden, i tillegg til at den er ansvarlig for import og eksport av data til og fra eksterne it systemer. Datatransporten er skrevet i Java. På figur 1.3 kan man se et skjermbilde av datatransport administrasjonssiden.

Figure 1.3: Skjermbilde av datatransport administrasjonsiden

Mobil løsning

På verdensbasis finnes det mange flere mobiltelefoner enn skrivebordsmaskiner. Lenge hadde mobilene begrenset kapasitet på maskinvaren, som minne, prossesor ytelse, lagringsplass, og ikke minst skjermstørrelse. Disse delene begynner å få en akseptabel kapasitet, og sammen med programmeringsplattformer som .NET Compact og Java ME fra henholdsvis Microsoft og Sun, begynner mobil teknologien å modnes sett fra et utviklingsmessig syn. Disse to løsningene gjør at man utvikle en applikasjon som kjører på alle mobiltelefoner som støtte den gitte platformen.

Et naturlig steg videre var å gi tilgang til Global One Solutions systemet via mobiltelefonen. Mange scenario var mulig, og de fleste oppgaver som i dag gjøres på Global One Portal kunne også blitt gjort gjennom mobilen. Begrensningen på mobilsiden i denne sammenhengen er for det meste skjermstørrelsen og mangel på skikkelig tastatur. I tillegg har vi en konkurrerende faktor i at de fleste i dag har relativt enkel tilgang til skrivebordsmaskiner med tilhørende internettilkobling, så derfor måtte det gjøres en gjennomgang av hvilke rapporter som var ønskelige uansett når og hvor man er. I prosessen med å velge hvilke deler som skulle gjøres tilgjengelig på mobiltelefonen er følgende hensyn tatt:

- mobilen er alltid tilgjengelig, men skrivebordsmaskiner med internett er aldri langt borte i dagens Norge
- hvilke funksjoner på Global One kan tenkes å være interessante uansett hvor man er
- hvilke funksjoner på Global One krever ikke fullt tastatur og stor skjerm

Følgende rapporter ble valgt:

- rapport med salgstall for hele kjeden sortert på siste år, siste måned, siste uke og i går. Salgstallene er også sammenlignet med budsjett og avviket mellom disse er kalkulert
- salgrapport over hvor mye de ansatte har solgt fordelt på hver butikk
- når ansatte logger seg inn på SuButikk skrives denne verdien til Susoft sin tjeneren ved hjelp av en webservice. Ved hjelp av disse dataene man generere en som viser hvem som har logget inn for dagen. Derved kan man se hvem som har kommet på jobb og om de kom presis på jobb via mobilen.
- en rapport over salget de siste 12 månedene. Dataene blir presentert i form av en graf

Mobilløsningen som skal implementeres kalles SuMobilløsningen. Navnet spinner på navnet på det eksisterende systemet på Susoft, Global One Solutions, men undertegnede har valgt å gi løsningen et norsk navn siden oppgaven er skrevet på norsk. SuMobilløsningen vil bestå av to deler - SuMobil Portal og SuMobil klient. SuMobil Portalen vil være en Java EE applikasjon som kjører på en tjener. SuMobil klient vil være en Java ME applikasjon som kjører på mobiltelefoner.

1.1.2 Global One Solutions arkitektur

På figur 1.4 er en model av hvordan Global One Solution arkitekturen er bygget opp.

- runde, blå sirklene representerer databaser
- grønne kvadratene er webapplikasjoner
- gule kvadrater er SuButikk-installasjoner

- den røde firkanten er en samlebetegnelse på Datatransporten, som består av 16 enkeltstående Java SE applikasjoner som kjører som batchjobber eller som webservices
- heldekkede piler betyr at web applikasjonene bruker de tilkoblede databasene
- striplede striper betyr overføring av data over internett

Ute i butikkene har kundene en installasjon av SuButikk på hvert kassepunkt. Denne programvaren kommuniserer med tjenerene hos Susoft ved forskjellige anledninger. En slik anledning er ved kjøp og salg av gavekort. Et gavekortet kan være utstedt i en annen butikk i den aktuelle kjedenm, og brukes i en annen. Ved salg av gavekort i en butikk, vil dette bli sendt til Susoft sin tjener sentralt. Hvis kunden bruker det det i en annen butikk, vil SuButikk programmet her hente frem gavekortet på den samme tjeneren. Denne kommunikasjonen foregår ved hjelp av webservices. En annen, mer fast, anledning er overførsel av kasseoppgjør og lagerstatus hver kveld. Etter butikkpersonalet har tatt kasseoppgjøret om kvelden, vil disse SuButikk generere xmlfiler som så sendes til Susoft sin tjener. På tjenersiden vil en av Datatransportprogrammene parse xmlfilen og legge dataene inn i den aktuelle butikkjedens database.

Hver kunde har sin egen database som innholder all dens data. Global One Portal applikasjonen er litt spesiell på den måten at det bare er èn webapplikasjon som håndtere portalene til alle kundene. Databasetilkoblingen blir bestemt av kjedenavnet man gir som innparameter i slutten av url'en. Gitt for eksempel følgende url: http://g1.susoft.com/platousport. Denne url'en vil så bli omskrevet i webtjeneren Apache til:

```
http://g1.susoft.com/suservletroot/no.susoft.sucom.servlets.
SUServlet?JavaProgram=no.susoft.sudatatransport.servlets.
CheckChainURL&checkurl=platousport
```

Denne url'en vil starte en servlet som tar platousport som innparameter og vil opprette en tilkobling til Platou Sport sin database. Dette forklarer hvorfor det går en pil fra Global One Portal til hver eneste database.

En av databasene har en spesiell rolle fordi den innholder databaseparametrene til alle kundedatabasene. Susoft kaller denne databasen "Masterdatabasen". Det er dette som gjør at Global One Portalen på en dynamisk måte kan skifte database. I tillegg er Masterdatabasen viktig fordi den brukes til logging.

Datatransporten er en samling av 16 Java SE programmer som har som hovedansvar å flytte data mellom kassepunktene i butikkene til og fra databasene sentral hos Susoft.

SuMobil Portalen er bygget opp etter samme prinsipp som Global One Portalen, men henter databaseinnstillingene fra statiske propertiesfiler og ikke fra Masterdatabasen. Hver webshop er koblet til nøyaktig en database.

Brukerscenario

Det vil bli brukt to brukerscenario for å illustrere hvordan komponentene i Global One Solutions henger sammen.

Scenario en- Gitt en kjede med ti butikker blir kunde hos Susoft. Man vil da få tildelt en egen database, tilgang til Global One portalen, og ti SuButikk installasjoner til hvert kassepunkt ute i butikkene. For å kunne selge noe i butikken må vi først ha opprettet varer. Istendenfor å gjøre dette i hver butikk, oppretter man varene på Global One portalen. Varene eksporteres så fra Global One til alle SuButikk installasjonene på de ti kassepunktene.

Scenario to - Ved dagens slutt vil de ansatte ta kasseoppgjøret. Etter kasseoppgjøret er ferdig vil SuButikk vente en time før den tar kontakt med Susoft datatransport tjeneren. SuButikk vil generer xml filer som representere forskjellige data fra SuButikk (som kasseoppgjør, gavekort, lagerstatus, osv) og overføre disse til datatransport tjeneren. Et program på Datatransporten tjeneren vil kjøre som en batch jobb og importere disse dataene i databasen til kunden. Når det er gjort er dataene tilgjengelig på vanlig måte gjennom Global One. Netthandlen bruker samme database som beskrevet ovenfor. Man bruker Global One portalen til å opprette varer og til å legge til varer som skal vises netthandlen. Siden samme database brukes har netthandelen tilgang til alle nødvendige og unødvendige data som lagerstatus, pris, kunderegister, osv. Dette betyr i praksis illustrert med et eksempel at når man gjør et salg av en vare i SuButikk, vil denne transaksjonen overføres til kundedatabasen på tjenersiden etter kasseoppgjør. I det øyeblikket lagertransaksjonene er oppdatert og lagt til i databasen, vil også lageret i netthandlen være oppdatert.

SuMobil løsningen vil også bruke denne databasen, og dermed tilgang til salgstall, ansatte pålogging, osv.

1.2 Mobiltelefoner

Det er i dag mange forskjellige aktører som produserer mobiltelefoner. Finske Nokia er størst, amerikanske Motorola er nest størst, mens andre store produsenter er koreanske Samsung, koreanske LG Electronics og svensk-japanske SonyEricsson. De forskjellige produsentene har forskjellige operativesystem og er organisert i foreskjellige produktserier. Nokia for eksempel har delt sin produktserie i Series40, Series60, Series80 og Series90. Så selv om dem har et utall forskjellige mobilmodeller, er telefonene kategorisert i disse fire seriene. Det er viktig å vite om, for alle mobilene i en serie har mange fellestrekk. I mange tilfeller er det hensiktsmessig å bygge applikasjonen for hver av disse seriene. Det vil her bli en gjennomgang av hvordan programvaren og operativsystemet er bygget opp i Nokia og SonyEricsson sine mobiltelefoner. På mobiltelefoner er programvaren preinnstallert, og man har ingen automatisk programvareoppdatering ala "Windows Update"[15] på mobiltelefoner. Derved må man normalt forholde seg til de programmene og de versjone av programvaren som er preinstallert.

SonyEricsson og Nokia tilbyr to typer operativsystemer (OS). På billigtelefonenen har dem egne properitære OS. Dette er enkle OS som ikke har multitasking og ikke støtte andre tredjepartsprogrammer enn Java ME applikasjoner. Dette gjør samtidig at disse OS'ene er mer stabile og mer responsive enn mobiltelefoner med mer avanserte OS. Både SonyEricsson og Nokia har utviklet hvert sitt properitære OS til sine lavpristelefoner. Siden dem ikke skal støtte andre tredjepartsprogrammer enn Java, er ingen OS API'er gjort tilgjengelig. Det finnes faktisk ikke noen informasjon om disse operativsystemene på nettsidene til de to respektive produsentene utover operativsystemenes eksistens.

På de mer avanserte mobiltelefonene til SonyEricsson og Nokia kjøres Symbian OS. Dette operativsystemet utvikles av Symbian Ldt, et selskap eid av Nokia (47,9 %), Sony Ericsson (13,1%), Panasonic (10,5%), Siemens AG (8,4%) og Samsung (4,5%). Dette er et operativsystem med pre-emptive multitasking og minnebeskyttelse. Preemption betyr at operativsystemet avbryter en oppgave som kjører i prosessoren uten oppgavens tillatelse. Oppgaven som kjøres vil bli byttet ut med en annen ventende oppgavene. Denne byttingen kalles kontekst bytting. Minnebeskyttelse betyr at operativsys-

temet sørger for at en prosess ikke har tilgang til minneområder utover de minneområdene den er tildelt.

1.2.1 Nokia

Nokia er for tiden den største mobilprodusenten med rundt 40 prosent av mobilmarkedet[16]. De har delt inn mobilene sine i fire serier:

Series40

Series40 er lavprisserien til Nokia. Alle Series40 modellene er basert på Nokia sitt eget properitære operativsystem, Nokia OS. Skjermstørrelsen er normal 128*128 piksler. Noen få modeller i denne serien har 240*320 piksler, mens noen få eldre har 96*65 piksler. Series40 har tre forskjellige utviklingsplattformer (Development Platform), altså versjoner av Java ME miljøet.

- Developer Platform 1.0 er basert på CLDC 1.0 og bruker MIDP 1.0 profilen. Skjermstørrelsen på disse mobilene er 128*128.piksler. Den har støtte for Nokia UI api'en. Bare applikasjoner under 64Kb er akseptert og jvm heap størrelsen er 200 KB. Man kan bruke record storen (RMS), altså datalagring, opp til 20 KB. Man bruker Nokia UI api'em for å spille av lyder, bruker full-skjerm modus, kontrollere vibrasjonen og bakgrunnslyset i tillegg til mer avanserte bildemanipulering som rotering av bilder og manipulering av piksel data.

- Developer Platform 2.0 er basert på CLDC 1.1 og støtter profilene MIDP 2.0 og JTWI 1.0. Skjemstørrelsen på disse mobilene er 128*128 piksler. Siden JTWI er inkludert, kan applikasjoner her bruke Wireless Messaging API og sende SMS. Mobil Media API'er er også støttet og gjør at man spille av True Tone luder og midi filer. Noen mobiler i denne kategorien støtter også Bluetooth API'en eller Mobile 3D Graphics API. Selv om Nokia UI api'en er tilgjengelig, bør den ikke brukes. Det meste av funksjonaliteten i denne er integrert i MIDP 2.0. Developer Platform 2.0 aksepterer jar filer opp til 128 KB.

- Developer Platform 3.0 er også basert på CLDC 1.1 og profilene MIDP 2.0 JTWI 1.0. Skjermstørrelsen på disse mobilene er 240*320 piksler

Series60

Series60 platformen er basert på Symbian OS 6 og høyere. Skjermstørrelsen er 176*208 piksler eller høyere og med en fargedybde på 16 bits pr piksel (altså 65.536 farger). Serien er delt inn i tre forskjellige utgaver (editions) som reflekterer forbedringene i det underliggende operativsystemet, Symbian OS. De forskjellige utgavene er:

- First edition bruker Symbian OS 6.1, profilen MIDP 1.0 og konfigurasjonen CLDC 1.0. I tillegg er api'ene Nokia UI API, Mobile Media API og Wireless Messaging API inkludert.
- Second Edition er delt inn i tre feature packs, nummerert en til tre.
  - Feature pack 1 bruker MIDP 2.0 profilen sammen med CLDC 1.0 kongfigurasjonen. Også Mobile Media API 1.1 og Wireless Messaging API er støttet. Operativsystemet er Symbian OS 7.0s
  - Feature pack 2 bruker CLDC 1.1 konfigurasjonen som støtter flyttall or er JTWI kompatible. Andre biblioteker støttet er PDAAPI (FileConnection og PIM API), Bluetooth tilkobling for push registry og Mobile 3D Graphics API for OpenGL-kompatible enheter. Operativsystemet er Symbian OS 8.0a
  - Feature pack 3 tilbyr, i tillegg til alle funksjonene fra Feature pack 2, Web Service API. Operativsystemet her er Symbian 8.1a
- Third edition har i tillegg til det de andre utgavene har, støtte for Security og Trust API, Session Initialization API, The scalable 2D Vector Graphics API, Location API and Wireless Messaging API 2.0. Denne utgaven bruker MIDP 2.0 profilen og CLDC konfigurasjonen.

Series 80

Nokia Series80 platform består av enhenter som kombinerer pda'er og mobiltelefoner. De er rettet mot forretningssegmentet. Nokia Communicator er en eksempel på denne serien. En typisk Series80 enhet ser ut som en normal mobiltelefon, men når man åpner den får man en skjerm på 640*200 piksler og et qwerty tastatur. Serien støtter MIDP 2.0 profilen og CLDC 1.0/CLDC 1.1 konfigurasjonen (avhengig av modell), i tillegg til Personal Java Profilen og CDC konfigurasjonen.

Series 90

Series90 enheter er optimalisert for å presentere media med sin 320*200 piksler skjerm og sin penn baserte input metode. Serien støtter MIDP 2.0 profilen sammen med CLDC 1.1 konfigurasjonen.

1.2.2 SonyEricsson

SonyEricsson telefoneme kommer med Java ME i syv utgaver - 1, 2, 3, 4, 5, 1-Symbian og 2-Symbian. De fem første utgavene kjører på et eget properitært SonyEricsson OS, mens smarttelefonene bruker et operativsystem basert på Symbian UIQ (et penn basert brukergrensesnitt). Tabell 1.1 viser en oversikt over hvilke Java ME konfigurasjoner og profiler de forskjellige Javaplatformene til SonyEricsson utstyres med.

Table 1.1: SonyEricsson Javaplatformer

Java Plattform	OS	Profile	Konfigurasjon	Skjermstørrelse
1	SE OS	MIDP 1.0	CLDC 1.0	128*160
2	SE OS	MIDP 2.0, JTWI 1.0	CLDC 1.1	176*220
3	SE OS	MIDP 2.0, JTWI 1.0	CLDC 1.1	128*128, 128*160, 176*220, 240*320
4	SE OS	MIDP 2.0, JTWI 1.0	CLDC 1.1	176*220
5	SE OS	MIDP 2.0, JTWI 1.0	CLDC 1.1	176*220
1-Symbian	Symbian UIQ OS	MIDP 1.0	CLDC 1.0	208*320
2-Symbian	Symbian UIQ OS	MIDP 2.0	CLDC 1.0	208*320

1.2.3 SuMobil på forskjellige telefoner

SuMobil applikasjonen er laget for alle javabaserte Nokia og Sony-Ericsson mobiltelefonene på markedet i dag.

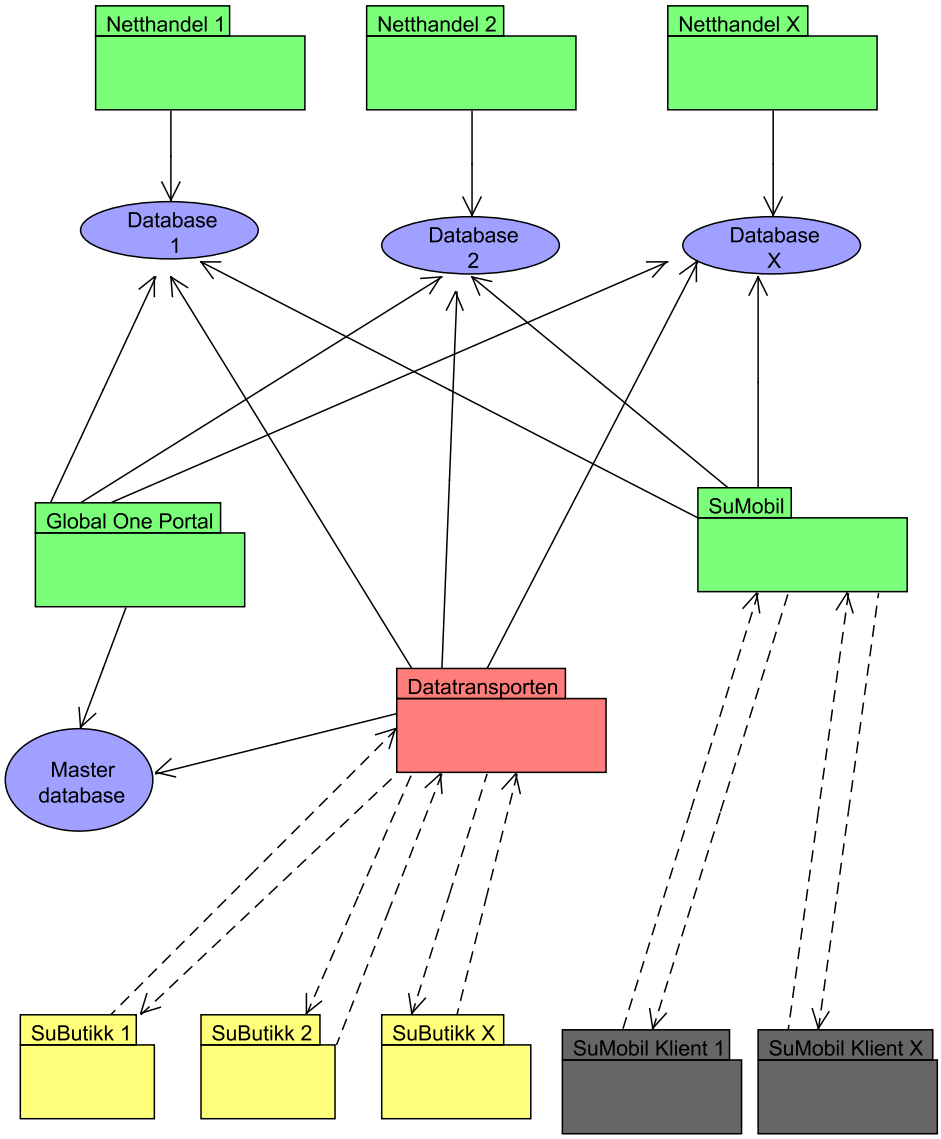

Figure 1.4: To brukerscenario for å beskrive samspillet mellom de forskjellige komponentene i systemet til Susoft

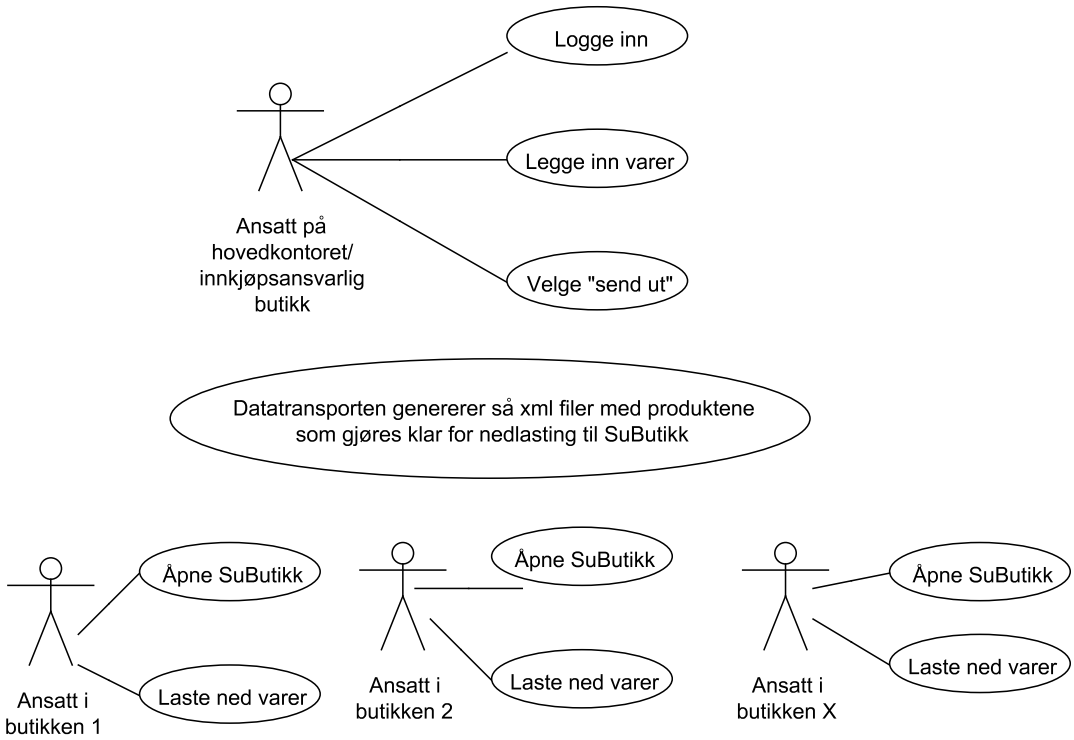

Figure 1.5: To brukerscenario for å beskrive samspillet mellom de forskjellige komponentene i systemet til Susoft

Kapittel 2

Java ME standarden og portering

Da denne masteroppgaven omhandler mobilløsningen generelt og klientsiden spesielt, vil det bli brukt noen sider på å beskrive hva Java ME er, hvorfor det trengtes en egen spesifikasjon for trådløse enheter og hva som skiller Java ME fra de to andre Java utgavene. Det vil også bli beskrevet hvordan disse arkitektoriske forskjellene har påvirket utviklingen av mobilapplikasjoner. Det er viktig med en beskrivelse av Java ME arkitekturen for å få en forståelse av hvilke utfordringer man står ovenfor når man programmerer for mobiltelefoner, og hvordan disse håndteres i Java ME. Det vil også gi en forståelse av hvorfor Java ME ble valgt som teknologi for Sumobilløsningen i denne oppgaven.

Java familien er delt opp i tre grupper: Java Micro Edition (Java ME), Java Standard Edition (Java SE) og Java Enterprise Edition (Java EE). Java SE var den første Java utgaven og er beregnet for skrivebordsmaskiner. Det spesielle med denne utgaven kontra de to andre er et stort utvalgt av GUI biblioteker. Java SE inkluderer en Java Virtual Machine. Denne forkortes JVM.

Java EE bygger på Java SE og er en hel pakke med verktøy og biblioteker for å implementere Service Orienterte Arkitekturer og neste generasjons webapplikasjoner[11]. Siden Java EE bygger på Java SE bruker dem samme JVM.

Java ME er en nedstrippet versjon av Java SE og har en litt annen oppbygging av arkitekturen for å støtte mangfoldet av mobiltelefoner og forbrukselektronikk generelt.

Global One Portalen og netthandlen er begge Java EE applikasjoner.

Tjenersiden av mobilløsningen er også en Java EE applikasjon. Datatransporten er et sett (ca 16 applikasjoner, Susoft kaller dem RunCentrals) med Java SE applikasjoner. Klientsiden av SuMobilløsningen er laget i Java ME.

2.1 Java ME mål og arkitektur

Følgende mål var satt for Java ME[30]:

- Tilby støtte for et bredt utvalg av enheter med forskjellig spesifikasjoner og muligheter. Disse enhetene er forskjellige på områder som brukergrensesnitt, datalagring, nettverksmuligheter, båndbredde, strømforbruk og sikkerhet.
- Tilby en arkitektur som er optimalisert for enheter med lite ressurser og tilby en arkitektur med et lite footprint.
- Tilby nettverkstilkobling over et vidt område av nettverksmuligheter og tjenester.
- Tilby en måte å levere applikasjoner og data over nettverkstilkoblinger. Ofte er nettverk den foretrukkede måten å levere Java ME applikasjoner på. Applikasjoner må ha muligheten til å bli installert på enheten, lastet direkte inn til minne og forkastet etter kjøring.
- Maksimere cross-platform mulighetene mens man fremdeles tar i bruk fordelene hver enkelt unike enhet kan tilby.
- Maksimere fleksibilitet og tilby metoder for å støtte et kjapt forandrende marked og å støtte eksisterende og fremtidige Java ME applikasjoner.
- Tilby en måte for tredjeparts utviklere å skrive og utvikle applikasjoner til Java ME støttede enheter uten å måtte ta hensyn til spesifikke mobilprodusenter eller mobiltelefoner.
- Tilby en måte å skalere applikasjoner over enheter med forskjellige teknologier, funksjoner og prosesseringsmuligheter

Kort fortalt ønsker man å støtte et bredt omfang av enheter, men samtidig ikke legge begrensninger på funksjonalitet som en gitt enhet tilbyr. Dette er to motstridende hensyn, og er løst ved at Java ME introduserer to nye arkitektuelle konsepter - konfigurasjoner og profiler. Konfigurasjoner skal løse den første utfordringen ved å støtte et bredt omfang av enheter,

mens profilene skal sørge for at man kan bruke enhetenes funksjonalitet fullt ut.

Konfigurasjoner er bygget opp av et sett med lavnivå API'er som definerer egenskaper ved det spesifikke Java ME miljøet. En viktig egenskap med konfigurasjoner er at de er nøstet. Det betyr at alle konfigurasjoner må kunne kjøre på den største Java ME konfigurasjonen. Gitt at det er definert tre konfigurasjoner, k1, k2 og k3, der k1 er minst mens k3 er størst. Hvis applikasjonen først er laget for k1, men så finner man ut man har lyst å kjøre den på en enhet med bedre spesifikasjoner, skal denne kjøre uten videre på både k2 og k3.

Profiler angriper mer enhetsspesifikk og brukerspesifikk API'er som funksjonalitet i brukergrensesnittet, datalagringsmekaniser og enhetsspesifikke funksjoner som bruk av IR (infarød) porter eller for å aksessere telefonspesifikke funksjoner i mobiltelefon.

For å konkretisere - konfigurasjoner inkluderer et sett med kjerne Java funksjonalitet som String, System, Thread, og Object, i tillegg til at den har funksjonalitet som å håndtere I/O strømmer og nettverkstilkobling. Profiler tilbyr funksjonalitet som brukergrensesnitt, handlingshåndtering og datalagring, i tillegg til at man kan legge til valgfrie profiler som utvider den tilgjengelige funksjonaliteten.

Et komplett Java ME miljø er bygget opp av en konfigurasjon og en eller flere profiler. Siden disse to arkitektuelle konseptene kan blir mikset rundt og rearrangert gitt et spesifikt behov, blir Java ME arkitekturen smibar nok til å støtte de diverserte behovene i Java ME verden.

Figur 2.1 er en illustrajon på arkitekturen. Man kjenner igjen at den virtuelle maskinen er et lag over operativsystemet fra Java SE, men i tillegg er to nye lag introdusert ved konfigurasjonen og en eller flere profiler. Konfigurasjonen vil teknisk sett være den virtuelle maskinen, mens profilene vil være biblioteker (jar filer) [12].

2.1.1 Konfigurasjoner

Konfigurasjoner er spesifikasjoner innen Java ME arkitekturen som er definert av en ekspertgruppe innen Java Community Process (JCP)[19]. For øyeblikket har JCP definert to konfigurasjoner [10]:

- Connected Limited Device Configuration (CLDC)
- Connected Device Configuration (CDC)

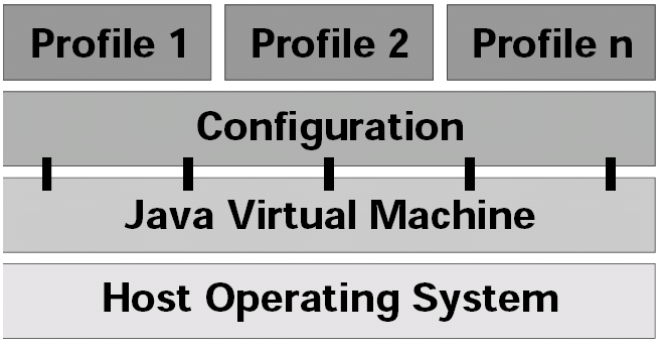

Figure 2.1: Illustrasjon av Java ME arkitekturen

CLDC er tiltenkt enheter med strikte begrensninger på minne, cpu, strømforbruk og nettverkstilkobling. CDC er tiltenkt kraftigere enheter som kraftige pda'er, hjemmesentre, osv. Figur 2.2 nedenfor viser hvordan CLDC og CDC er nøstet.

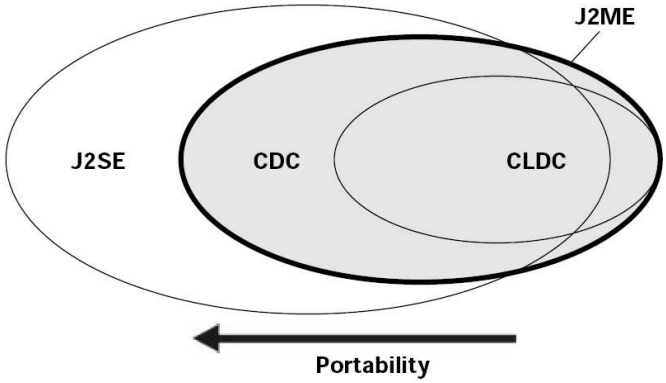

Figure 2.2: Konfigurasjoner er nøstede.

Konfigurasjoner definerer kontrakten mellom profilene og den virtuelle maskinen. Hver konfigurasjon sin egen virtuelle maskin. CDC bruker C-Virtual Machine (CVM) og CLDC bruker The Kilobyte Virtual Machine (KVM). Implementasjonene av de virtuelle maskinene må følge spesifikasjonen til hver av konfigurasjonene.

Connected Limited Device Configuration

Målet med denne konfigurasjonen er å tilby et mininal footprint med et minste felles multiplum av funksjonalitet tilgjengelig til ressursesvake enheter. En ressurssvak enhet har følgende karaktistikker:

- 160 kB til 512kB med total minne tilgjengelig for Java miljøet
- 16-bit eller 32-bit prosessor
- Lavt strømforbruk, som regel batteridrevet
- Støtter en eller annen tilkobling til et netverk. Normalt er denne tilkobling midlertidig, og som regel over linjer med lav båndbredde eller trådløse mobilnett.

CLDC er en strippet utgave av Java SE, men utelater noe funksjonalitet. Man laget CLDC ved å starte med blanke ark og legge til det som var nødvendig fra Java SE basert på de følgende kriteriene:

- Er denne funksjonen passende for denne typen enheter?
- Behøver denne funksjonaliteten mye lagringplass, minne og/eller mange cpu sykler?
- Kan utviklere gjenskape funksjonaliteten hvis de allikevel trenger den?
- Støtter disse enhetene generelt den gitte funksjonaliteten?
- Finnes det sikkerhetsrisikoer knyttet til den gitte funksjonaliteten på en ressursbegrenset enhet?

For å møte kravet om lite footprint har det blitt fjernet mange funksjoner som er tilgjengelig i et Java SE miljø. Nedenfor er en liste av funksjonene som er fjernet i CLDC konfigurasjonen[6].

- Java Native Interface, JNI. Dette er biblioteker som gjør at man kan kalle programmer skrevet i andre programmeringsspråk, for eksempel programmer skrevet i C eller C++.
- Bruker definerte klasselastere. I Java SE kan man arve de eksisterende klasselasterne og selv implementere hvor og hvordan den virtuelle maskinen skal laste inn klasser. Dette er av ressursmessige og sikkerhetsmessige grunner ikke lagt til Java ME.

- Refleksjon gjør det mulig å inspisere og forandre en applikasjon under kjøring i den virtuelle maskinen. Refleksjon har fordeler og ulemper. Ulemper angår sikkerhetsmessige og ressursmessige grunner. Disse ulempene har mye større konsekvenser i en CLDC enhet, og refleksjon er derfor utelatt fra denne konfigurasjonen.

- Trådgrupper og deamontråder. Trådgrupper gjør det mulig å manipulere flere tråder på en gang. Denne funksjonenen kan gjenskapes ved å legge tråder i en vektor og iterere denne om man ønsker å forandre alle trådene i steg. Deamonstråder er tråder som kjører helt til alle andre tråder er døde. Er det bare deamonstråder igjen i minne til den virtuelle maskinen, avsluttes maskinen. Deamonstråder er fjernet fra CLDC konfigurasjonen

- Finalization er en metode som kjøres rett før objektet hentes og skal vrakes av garbage colletoren i den virtuelle maskinen. Ideen we at man kan bruke denne funksjonen til å frigjøre ressurser i objektet. Problemet med denne metoden er at man aldri har noen garanti for når garbage colletoren kommer for å samle inn objektet. Garbage collectoren i Java er ikke-deterministisk - man har ikke noen garanti for at objektet blir samlet inn og finalization metoden blir kjørt innen en viss tidsperiode. Dermed kan objektet holde på ressurser lengre enn man hadde forventet. Siden finalization funksjonen er så upålitelig, er den fjernet fra CLDC konfigurasjonen[13].

- Svake referanser er referanser som ikke beskytter objektet fra å bli samle inn av garbage collectoren. Slike referanser blir brukt for å unngå sirkulære referanser og for å angi hvilke objekter som ikke er kritiske. På den måten kan man minimalisere antall unødvendige objekter i minnet ved å bruke svake referanser til dem. Svake referanser brukes også ofte til mellomlagring av store elementer som bilder, filmer, osv.[34]

- Flyttall datatyper, som float og double var fjernet fra første versjon av CLDC konfigurasjonen fordi dem krevde for mye ressurser i form av minne og cpu. Det ble senere lagt til i versjon 1.1 av CLDC.

- Noen sikkerhetsfunksjoner og API'er.

- Multidimensjonale tabeller er fjernet fordi dem krever for mye ressurser.

- I Java SE og Java EE blir klasser verifisert før dem lastes inn i minnet i den virtuelle maskinen. Dette er derimot en ressurskrevende operasjon, og er derfor utelatt fra Java ME. Man har isteden et eget steg etter kompilering, preverifisering. I Java Virtual Machine spesifikasjonen er det verfiseringen av en klassefil definert i fire steg[7]. Dette er steg som at man sjekker at klasser er definert som final ikke blir arvet, at klassefilen ikke er korrupt, at alle klasser med unntak av Object har en superklasse, osv. De første tre stegene her gjøre ved hjelp av et preverifiseringsverktøy som Sun leverer med sitt Wireless Toolkit for Java ME. Det siste steget er selve lastingen inn til minnet i den virtuelle maskinen, og det gjøres selvfølgelig av den virtuelle maskinen selv.

- Noen begrenset feilhåndtering (ikke alle unntak er inkludert)

The Kilobyte Virtual Machine

CLDC konfigurasjonen må implementeres som en egen virtuel maskin som oppfyller spesfikasjonen. Sun har laget en virtuel maskin spesifikasjon (KVM) som støtter CLDC konfigurasjonen. KVM spesifikasjonen inneholder både obligatoriske og valgfrie elementer. Alt etter hvilke valgfrie elementer som implementeres, kan KVM støtte nye versjon av eksisterende konfigurasjoner eller helt nye konfigurasjoner. I dag er det bare definert to konfigurasjoner, CLDC og CLC, men det kommer sannsynligvis flere med tiden. KVM spesifikasjonen har tatt hensyn til dette.

En konfigurasjon er teknisk sett et sett med biblioteker og virtuel maskin funksjoner. Den virtuelle maskinen må tilby de funksjonene konfigurasjonen krever, og det er denne jobben KVM gjør. Sun har laget både KVM spesifikasjonen og en referanse implementasjon av denne. Da KVM ble laget hadde man to tilnærmingsmuligheter - enten starte med Java SE virtual machinen og fjerne funksjoner og biblioteker om man trengte dem, eller å starte med "blanke ark" og bare legge til funksjoner man trengte. Sun valgte den siste tilnærmingen slik at man eksplisitt fikk se hvilke avhengigheter som var mellom funksjonene og bibliotekene.

Sun sin implementasjon av KVM har et footprint på mellom 40 kB og 80 kB avhengig av kompileringvalg og enheten den skal kjøres på. KVM spesifikasjonen angir både obligatoriske elementer som må være med i implementering og valgfrie elementer. Ved å bare implementere den obligatoriske delen av KVM spesifikasjonen kan størrelsen halveres samtidig som implementasjonen oppfyller spesifikasjonen [8]

Connected Device Configuration

CDC er den andre av de to konfigurasjonene som for øyeblikket er definert innen Java ME verden. Den brukes ikke i SuMobilløsningen, men er kort beskrevet for å vise hvordan en annen konfigurasjon er bygget opp og for å dekke Java ME arkitekturen på en "komplett" måte. CDC er beregnet for enheter og apparater med mer ressurser enn CLDC enheter. CDC kjøres på en C-Virtual Machine (CVM) som er fullt ut kompatible med Java Virtuel Maskin Spesifikasjonen [7], altså den virtuelle maskinen for Java SE og Java EE. Konfigurasjonen er rettet mot enheter med så lite som 512kB med minne, men er designet spesielt for enheter med rundt 2 MB tilgjengelig minne. Normalt har enhetene som bruker denne konfigurasjonen betraktelig med prosesseringskraft, de kan ofte plugges rett inn i veggen og støtter bredbånd internett tilkobling[5].

C-Virtual Machine

Selv om CVM oppfyller hele Java Virtuel Maskin spesifikasjonen, er implementasjonen anderledes enn den den virtuelle maskinen som er implementert for Java SE og Java EE på den måten at den er optimalisert for nettverksenheter. For eksempel er garbage colletor (gc) algoritmen fullstendig separert fra den virtuelle maskinen, noe som gjør forskjellige gc algoritmer kan bli brukt avhengig av enhet. For å øke portabilitet har referanse implementasjonen implementert tråder inni den virtuelle maskinen. Tråder som er implementert innenfor virtuelle maskiner er kalt grønne tråder. Disse trådene gjør det enklere å portere den virtuelle maskinen til andre maskiner siden de ikke gjøres kall til det underliggende operativsystemet slik som det gjøres på den virtuelle maskinen for Java SE. Det er dog mulighet for produsenter å implementere OS spesifikke tråder hvis de velger å gjøre det på sin enhet og plattform. Klasseverifiseringen gjøres under lasting av klasser til den virtuelle maskinen, akkurat som i Java SE miljøet. Det er ingen preverifiseringssteg i CDC[4].

2.1.2 Profiler

Profiler tilbyr API'er som fokuserer på enkelte enheter eller på en gruppe av relaterte enheter som mobiltelefoner og personsøkere. Enhetene som støtter en profil har ofte mye til felles i hvordan disse blir brukt, hvilket brukergrensesnitt dem har, inputmetoder (tastatur, talltastatur, osv), hvordan en kobles til nettverk (mobil bruker telefonnettet, pda ofte trådløse nett), hvordan enheten lagrer data, osv. Profiler adresserer de mest spesifikke funksjonene i

Java ME arkitekturen.

Det er viktig å skille to ting. Er profiler konseptuelle definisjoner definert i en spesifikasjon eller er det programvare? Dette er et viktig skille. Profiler er laget av mange deltakere gjennom Sun's Java Community Process (JCP). Det dem primært lager der er spesifikasjoner, ikke programvare. Det blir ofte i denne sammenhengen også laget referanse implementasjoner, men prinsippielt sett er det opp til utstyrsleverandørene å lage en implementasjon av spesifikasjonen. Utstyrsleverandøren kan dog velge å bruke referanseimplementasjonen, som ofte blir laget av Sun, eller dem kan implementere spesifikasjonen selv. Akkurat som konfigurasjonene definerer kontrakten mellom profilen og den virtuelle maskinen, definerer profilen kontrakten mellom enheten og applikasjonen. Hvis utstyrsleverandøren velger å implementere profiler selv må dem implementere absolutt hele spesifikasjonene. Det er deler av spesifikasjonene som er valgfri å implementere, men i praksis er det få deler[30]. Spesifikasjonen angir hvilke deler av den som er valgfri og obligatorisk.

To typer profiler

Det gjøres et skille mellom to typer profiler[9]:

- enhetsprofiler
- funksjonsorienterte profiler (Sun kaller disse valgfrie pakker/profiler)

Normalt vil profilene tilby brukergrensesnittet, inputmetodene og lagringsmekanismene for en gitt gruppe enheter. Disse profilene vil definere et komplett utviklingsmiljø for et spesifikt sett av enheter og kan derfor sees på som enhet profiler. Men profiler kan også defineres til å oppfylle mer spesifikke tjenester. Eksempler her er en Remote Method Invocation (RMI), webservices eller en multimediaprofil.

Man kan også samle tjenester for et gitt markedssegment, for eksempel lage en profil som støtter banktransaksjoner. Disse profilene kan sees på som funksjonsorienterte profiler. Fordelen ved å samle et sett med funksjoner i profiler er at man da enkelt kan gjenbruke dem i mange forskjellige enheter.

Dette skillet gir også modularitet og fleksibilitet ved at utstyrsleverandørene kan velge hvilke funksjoner som er nødvendige eller mest viktige for en gitt enhet. La oss eksemplifisere. Gitt vi har en mobiltelefon, PDA og et hjemmevideosenter. Anta at disse tre enhetene tilbyr kortkjøp over internett. Høyst sannsynlig vil hver av disse enhetene bruke tre forskjellige enhetsprofiler pga forskjellige skjermløsninger, inputmetoder, lagringsmetoder, tilgjenglige ressurser, osv. Men alle disse tre enhetene kan støtte

kortkjøpprofilen. Dette fordi dem kjører en enhetsprofil i bunn, og i tillegg kan legge til så mange funksjonsorienterte profiler som ønskelig ved siden av enhetsprofilen, som for eksempel en banktransaksjonsprofil. Fordi profiler er så spesifikke og modulære kan vi vente at det blir utviklet mange forskjellige profiler i fremtiden. Så oppsummert, hvordan kan profiler være modulære og støtte alle enhetspesifikke API'er uten å bli monolistiske eller introdusere dublikate klassebiblioteker i de forskjellige profilene? Ganske enkelt fordi en enkelt profil eller et sett av profiler ikke utgjør et komplett Java ME miljø. Profiler adresserer bare spesifikke funksjoner. Under profilene er konfigurasjonene som den virtuelle maskinen som tilbyr kjerne java støtten.

I profilspesifikasjonen er det angitt hvilken konfigurasjon den krever. Hvis profilen ønsker å bruke funksjonalitet som finnes i for eksempel CDC konfigurasjonen, kan den bare kjøres der og ikke under for eksempel CLDC konfigurasjonen. Men det motsatte er mulig pga av den nøstede arkitekturen til Java ME, som vist på figure 2.2

2.2 Java ME Portering

Java ME utvikling er i sin natur anderledes enn med Java SE eller Java EE. Hvis man tar utgangspunkt i den klassiske iterative utviklingsmodellen har man et ekstra steg under Java ME utvikling man kaller "Portering". I dette steget tilpasses og bygges applikasjonen mot forskjellige mobiltelefoner. For å støtte dette steget under utvikling er det laget byggerammeverk som skal støtte denne porteringen. Java ME applikasjoner er forskjellige fra andre Java applikasjoner på den måten at dem er skrevet for mobiltelefoner som varerierer veldig i hvilken konfigurasjon og profil(er) dem støtter, skjermstørrelse, tilgjengelig diskplass/minnestørrelse, maks størrelse på applikasjonen, osv. For å løse disse utfordringene bør man bygge en egen jar fil for hver mobiltelefon eller for grupper av mobiltelefoner. Tabell 2.1 viser noen eksempler på valgfri funksjonalitet som skal være med under bygging av en gitt Java ME applikasjonen avhengig av hvilken telefon man bygger mot. Det er byggerammeverket sin oppgave å styre hvilken del av koden og ressurser som skal være med i jar filen som bygges mot den aktuelle telefonen eller gruppen av mobiltelefoner.

Når man utvikler mobilapplikasjoner kan man løse porteringsproblematikken på en av fire måter:

- Bruke minste felles multiplum

Table 2.1: Byggescenario

hvis mobilen....	....så skal applikasjonen
støtter lyder	spille av lyder
støtter alfa-blending	presentere menyer med varierende gjennomsiktighet
har begrensninger på .jar filstørrelsen	fjerne alle ikke-nødvendige bilder
støtter full skjermstørrelse	bruke full skjermstørrelse
har nok heap minne og støtter store .jar filer	bruke valgfri funksjoner som bitmap fonter
har kamera	bruke kamera, for eksempel til å ta bilder til bruk i applikasjonen
støtter bluetooth	bruke sms eller http tilkobling for å kommunisere med andre brukere
støtter kjøring av bakgrunnsprogrammer	kjøre applikasjoner i bakgrunnen

  - Hvis man holder seg til MIDP 1.0 og CLDC 1.0 konfigurasjonen og bare bruke de enkle, høynivå gui elementene i MIDP 1.0, er sjansen stor for at den samme applikasjonen vil kjøre relativt smertefritt på alle enheter.

- Bruke forskjellige kildekode avhengig av enhet koden skal kjøres på
  - Man kan også ha forskjellig kildekode for hver enhet man ønsker å støtte, men dette vil kjapt utvikle seg til en vedlikeholdsmareritt.

- Bruke dynamisk kode som tilpasser seg under kjøring
  - JavME støtter ikke refleksjon slik vi kjenner det fra Java SE, men man kan fremdeles instansiere klasser dynamisk, og på den måten skrive dynamisk kode. Man kan også bruke et annet programmeringsparadigme, Aspekt Orientert Programmering (AOP). Ved bruk av dette paradigme kan man abstrahere forskjellene mellom forskjellige mobiltelefoner på en helt annen måte enn hva som er mulig med Objekt Orientert Programmering (OOP).

- Bruke preprosessering. Preprosessering betyr at et program går gjennom koden og manipulere den før den blir kompilert av Java kompilatoren.

De to første løsningene er trivielle løsninger. De to siste løsningene vil bli gjennomgått i detalj hver for seg.

2.2.1 Hvorfor eget rammeverk for Java ME utvikling?

I 1997 lanserte Sun sin servlet spesifikasjonen, noe som gjorde det mulig å lage web applikasjoner på Java plattformen. Senere ble den utvidet med Java Server Pages (jsp) for å støtte Model View Control modellen (MVC)[21], som går ut på å skille logikk og presentasjon. Senere kom også rammeverk som Apache Struts, Spring, etc, som forenklet utvikling ytterligere. Dette er MVC rammeverk som automatiserer mange oppgaver, for eksempel mer avanserte måter å styre presentasjonen på, enklere måter å motta innparemetre fra html skjema, osv. Disse rammeverkene har gjort web utvikling mye enklere og raskere, og har betydd mye for web applikasjonsutvikling.

Java ME ble lansert av Sun i 1999. På samme måte måte som utvikling av web applikasjoner har blitt forenklet ved MVC rammeverk som Apache Struts og Spring, skjer det for tiden ligende ting ved utvikling av mobile applikasjoner. Frem til nå har utviklere brukt Sun sine Java ME api'er under utvikling, men disse api'ene har svakheter. På samme måte som det er kløne te å hente ut hvert enkelt input parameter fra en html form manuelt i en servlet, har også Java ME en del funksjoner som kan forenkles. I Apache Struts for eksemple definerer man servlets som er JavaBeans. Når brukeren sender et skjema, hentes den aktuelle servleten som representerer skjemaet på tjenersiden og bruker set metodene i servleten til automatisk å sette innparametrene. I Java ME er det på samme måte også mange ting som kan forenkles, men siden Java ME utvikling byr på andre utfordringer, må disse rammeverkene løse andre problemer. Et eksempel her er utfordringen ved å skrive applikasjoner som automatisk kan kjøres på forskjellige mobiltelefoner. Hver mobilprodusent kan implementere sin egen virtuelle maskin (men selvfølgelig basert på den samme spesifikasjonen), men som utviklere vet vi også at da får vi forskjellige bugs i hver implementasjon. Også begrensede ressurser på mobiltelefoner er en utfordring. En pda kan ha mange megabyte med minne og lagringsplass i tillegg til stor skjerm, mens en billigere telefon kan ha mindre resursser og liten skjerm. Dette er ikke et problem under webapplikasjon utvikling hvor de aller fleste tjenere har mer enn nok med minne, cpu og lagringsplass, men under Java ME utvikling er dette forskjeller man må ta alvorlig.

Hvordan Java ME utvikling vil fortone seg om tre år kan ingen sikkert si. Men mange mener, inkludert undertegnede, at de mobil rammeverkene på markedet i dag vil bety like mye for mobilapplikasjonsutvikling som MVC rammeverkene har gjort for Java EE utvikling. Undertegende har identifis-

ert tre rammeverk som er på markedet i dag, to tyske og et amerikansk. Det første er et åpent kildekodeprosjekt, Ant Antenna. Dette er et rammeverk som bygger på Ant, og er et rent byggerammeverk. Det andre Java ME rammeverket er J2ME Polish. Det er som Antenna også et byggerammeverk, men har noe mer avansert funksjonalitet i forhold til selve byggingen (som enhetsdatabase, osv). I tillegg har J2ME Polish noen ekstra moduler som retter opp noen svakheter i Java ME standarden. Dette er for eksempel støtte for en mer avansert GUI, i tillegg noen støtteverktøy og java klasser. Begge disse rammeverkene baserer seg på bruk av preprossesering for å gjøre portering. Det tredje rammeverket er rammeverk utviklet av Tira Wireless, et amerikansk selskap med hovedkvarter i California. Dette rammeverket gjør portering ved bruk av Aspekt Orientert Programmering og egenutviklede Jumplets.

For å implementere SuMobilløsningen har undertegnede valgt å bruke J2ME Polish. Dett er utviklet av et tysk firma, Enough Software. Rammeverket er gratis for studenter, men man må kjøpe linsens hvis man bruker det til kommersiell programvare.

2.3 Portering ved bruk av Aspekt Orientert Programming

I OOP representer ofte objekter fysiske gjenstander som en bil, hjul, ratt, osv. I objektene kan man også definere oppførsel til de fysiske gjenstandene ved hjelp av metoder. For eksempel kan en bil ha metoder som "start motor", "kjør", "stopp" og "stopp motor". Man har i OOP definert objekter som abstraktere oppførsel og data i en klasse. En klasse har normalt et ansvarsområde. Bruker vi bilen som eksempel, så har et bilobjekt ansvaret for å styre bilen. Denne koblingen mellom objekter og gjenstander gjør at språk som bruker dette programmeringsparadigme er enkle og forstå. Spesielt merker man fordelen ved at objektene representerer ting man kan forholde seg til når programmer og systemer blir store.

Ideen med OOP er altså at et objekt har et ansvarsområde. Men det finnes tilfeller hvor man har funksjonalitet som er spredt utover mange klasser og objekter. Et eksempel på dette er logging. Logging er sjelden hovedansvaret til en enkelt klasse, men allikevel bruker man dette i mange klasser. Man sier at logging sprer seg (eng:cross cuts) utover mange klasser, og at dette er et "cross cutting concern". Undertegnede kjenner ikke til noen god norsk oversettelse av dette begrepet, og vil derfor videre i teksten bruke det engelske uttrykket eller forkortelsen CCC. Slike CCC bryter med

prinsippet i OOP at hver klasse har et enkelt ansvarsområde, og slike CCC gjør koden mindre modulær og fleksibel.

2.3.1 Introduksjon Aspekt Orientert Programmering

Aspekt Orientert Programmering er et paradigme som separerer slik "cross cutting concerns" i egne komponenter som kalles aspekter. Et aspekt er en modulær komponent som innholder implementeringer av funksjonalitet som sprer seg over flere klasser. La oss gå rett på sak og illustrere med et eksempel. Nedenfor ser man hvordan man ville gjort logging ved bruke av OOP paradigme:

```
void paint(Graphics g{
   logging("entering paint");
   // your paint code
   logging("leaving paint");
}
void keyPressed(int keyCode){
   logging("entering keyPressed");
   // your key processing code
  logging("leaving keyPressed");
}
```

Det optimale hadde vært om vi kunne spesifisert "legg til logging i begynnelsen og/eller slutten av de følgende to metodene". Det er akkuart det man gjør ved bruk av AOP paradigme. La oss illustrere med følgende pseudokode:

```
loggingAspect{
   loggableCalls = paint, keyPressed;

   insertBefore: loggableCalls{
      logging("entering " + $methodName);
   }
   insertAfter: loggableCalls{
      logging("leaving " + $methodName);
   }
}
```

For å forstå AOP er det tre viktig begreper man må kunne:

Et Pointcut er et punkt eller punkter under kjøring av applikasjonen hvor et "cross-cutting concern" må utføres. I eksempelet ovenfor er følgende et "pointcut":

```
loggableCalls =paint, keyPressed;
```

"Pointcuts" kan være en kombinasjon av metodekall, initialisering av et objekt, initialisering av en en feltvariable, mottak av unntak, eller andre veldefinerte punkt langs kjøringen av programmet.

Et Advice er koden du ønsker skal kjøre når man kommer til en visst "pointcut". I koden ovenfor vil følgende kodesnutt være et "advice":

```
insertBefore: loggableCalls{
   logging("entering " + $methodName);
}
insertAfter: loggableCalls{
   logging("leaving " + $methodName);
}
```

Et Aspect er kombinasjonen av et "pointcut" og et "advice". Derav navnet Aspekt Orientert Programmering.

2.3.2 Portering ved hjelp av AOP

I denne oppgaven er det tatt utgangspunkt i løsningen Tira Wireless har utviklet til portering av Java ME applikasjoner ved hjelp av Aspekt Orientert Programmering. En ulempe ved OOP er at det ikke er mulig å definere "cross-cutting concerns" (CCC). Det er mange CCC i Java ME kode, og mangelen på støtte av dette kompliserer porteringen. Et slikt CCC kan være en spesifikk feil i implementasjonen av Java ME på mobiltelefonen, gjøre tekst mindre/større alt etter hvilken mobil vi bygger mot eller redusere antall oppdateringen av skjermen for å få et spill til å kjøre kjappere på en treg mobiltelefon. Uten å bruke AOP er det nesten umulig å utvikle løsninger disse til gjenbrukbare objekter. Ved bruk av preprosessering må man legge disse preprosesseringselementene hver plass i koden hvor det er et CCC. Ved bruk av AOP kan man abstraktere dette ut av koden og definere det en gang i et aspekt.

Tira Wireless har definert noe dem kaller "Jumplets" [36, 35]. Dette er en konfigurerbar komponent som tillater innpakking av felles kode, aspekter eller hele subsystemer til en gjenbrukbar komponent. Med en Jumplet har man abstraktert ut porteringsproblematikken ut fra selve Java ME koden. Dette har flere fordeler. For det første vil koden bli enklere fordi man slipper å manuelt programmere inn løsninger på porteringproblematikken slik man gjør ved preprosessering. Siden porteringsløsningene er abstraktert i aspekter som er pakket sammen i en Jumplet, kan en slik komponent gjenbrukes til å portere også andre applikasjoner. Den porterte applikasjonen vil ikke inneholde ekstra metadata eller noen ekstra størrelse. En Jumplet inneholder også et sett med metadata hvor man kan kategorisere og beskrive hva den gjør.

I AOP har man en del av implementasjonen som tar seg av å flette aspektene inn i resten av Javakoden. Denne modulen kalles på engelsk for en "weawer", eller fritt oversatt til norsk - en fletter. Denne vil for hvert "pointcut" i koden sette inn koden i fra det tilhørende "advice". Denne flettingen kan bli gjort på to forskjellinge måter[3]. Den første måten er å gjøre flettingen i kildekoden. Da vil "advice" bli lagt til i kildenkoden ved de respektive "pointcuts". Denne metoden kalles kildekodefletting. Den andre måten kalles bytekodefletting. Da vil flettingen skje under kjøring når den virtuelle maskinen laster klassen inn i minne.

En meget utbredt Java implementasjon av AOP i dag, AspectJ, bruker bytekodefletting. Selv om bytekodefletting har visse fordeler [3] for Java EE baserte systemer, er det merarbeid å gjøre dette under kjøring. På så ressursesvake enheter som mobiltelefoner må merarbeid unngås. I tillegg mangler de fleste AOP implementasjonene mulighet til å gi mer detaljert kontroll av kodeinnsettingen. For eksempel, hvis man bruker AspektJ, kan man bare modifisere tilstanden til alle "drawString" metodekallene inni en "paint" metode. Det er derimot ikke mulig å modifisere hver femte forekomst av "drawString" mens de andre "drawString" kallen forblir urørt. Uten detaljert kontroll på kodeinnsettingen er det veldig vanskelig å portere applikasjonen på en skikkelig måte. Tira Wireless har utviklet en AOP implementasjon som er en modifisert utgave av Javassist, (*Java* Programming *Assist*ant). Denne komponenten er kjernen i rammeverket dem har utviklet for portering av Java ME applikasjoner, Tira Jump 2008. Tira Jump 2008 inneholder også en meget omfattende enhetsdatabase som angir spesifikasjoner, støttede konfigurasjoner og profiler, telefonoperatører, osv. I dette rammeverket angir man hvilke mobiltelefoner man vil bygge mot, og

hvilke Jumplet's, altså aspekter, man ønsker skal brukes til å portere applikasjonen til en gitt telefon. Alt dette uten å ta eksplisitt hensyn til det i kildekoden.

2.3.3 Evaluering av portering av Java ME applikasjoner ved bruk av AOP

Ved å bruke AOP kan man abstraktere hele porteringenproblematikken ut av selve applikasjonskoden til gjenbrukbare komponenter som Jumplets. Dette er en ryddigere måte å løse porteringen på enn ved å bruke preprosessering. Løsningen til Tira Wireless er en properitær løsning, og dem tilbyr egne kurs og undervisningsopplegg i bruk av deres løsning. Undertegnede hadde ønsket å laget en demo ved bruk av denne løsningen da den i teorien virker som den smarteste og mest riktig måte å løse porteringproblemer på. Denne løsningen er derimot sansynligvis dyr (man kan ikke kjøpe løsningen via nettet, men må sende en forespørsel om kjøp av produktet). I tillegg fant undertegnede denne løsningen og teorien rundt sent i oppgaven og tiden ville ikke strukket til for å implementere en versjon av SuMobilløsningen ved bruk at dette paradigme og Tira Jump 2008. Men teorien rundt Tira Jump 2008 og deres bruk av AOP, enhetsdatabase og Jumplets er godt beskrevet og ikke så veldig vanskelig å forstå. Undertegnede ble imponert av løsningen.,

2.4 J2ME Polish

Polish J2ME spinner på det engelske ordet polish som oversatt betyr "å polere". På mange måter forenkler Polish Java ME utvikling på samme måte som Struts, Spring og and MVC verktøy forenklet Java EE utvikling. Det vil bli beskrivet noen utfordringer ved Java ME utvikling og nevne noen svakheter ved Java ME standarden som Polish har løst.

I Java ME verden finnes alt fra enheter uten skjerm, til enheter med små skjermer til store skjermer. Noen enheter har kamera, andre ikke. Andre igjen har støtte for waw lyd, mens andre bare har støtte for midi. Noen har kanskje ikke støtte for lyd i det hele tatt. Noen enheter aksepterer ikke programmer større enn noen få kilobyte, andre takler programmer på mange megabyte. I koden må man ta hensyn til utfordringene beskrevet ovenfor, men i Java ME slik den fremstår i dag må dette gjøres manuelt ved å bruke et av de tre første alternativene beskrevet i punktlisten om porteringsalternativer. Polish har løst porteringsproblematikken ved å bruke det fjerde alternativet - preprosesseringen.

En annen svakhet ved Java ME er støtten for gui. I dag tilbyr Java ME to typer brukergrensesnitt - høynivå og lavnivå. Ved bruk av høynivå gui'en har man liten eller ingen kontroll over hvordan gui elementet presenteres på skjermen. Med lavnivå gui'en kan man tegne opp pikseler manuelt. Denne er beregnet for spill og grafer. Polish tilbyr biblioteker som utvider høynivå gui'en og gjør at man har mer kontroll på designet av brukergrensesnittet uten å måtte bruke lavnivå gui'en og ulempene som følger med dette.

I Java ME finnes ingen innebygget støtte for å lokalisere applikasjonen, altså enkel støtte for mange forskjellige språk. Polish tilby lokaliseringsverktøy som gjør at man kan bygge applikasjonen for mange forskjellige språk på en gang.

Java ME tilbyr også dårlig støtte for logging. Det finnes ikke noe loggrammeverk ala slik man har i Java EE verden med for eksempel Apaches log4j[1]. Polish har et logging rammeverk som er meget inspirert av Apaches log4j. Under bygging er det mulig å fjerne alle spor etter debugging i koden ved hjelp av preprosesseringstags, noe som er viktig i Java ME verden for å holde applikasjonene små.

I Java EE og Java SE verden utvikler man en applikasjon som så normalt kjører uten problemer på alle maskiner som har Java installert. Maskinenvaren er som regel alltid kraftig nok til å kjøre applikasjonen, og selv om man kanskje av og til må øke minne og tilgjengelig heap minne for den virtuelle maskinen, kan vi si at maskinvare sjelden er flaskehalsen for å få applikasjonen til å kjøre (selv om hastighet og hvor høy belastning applikasjonen tåler selvfølgelig påvirkes kraftig av forskjellig maskinvare). I Java ME verden er det ikke fullt like enkelt. Mobiler og trådløse enheter varierer voldsomt i ressurser tilgjengelig. En billig mobiltelefon kan ha 128 kB minne, mens en pda kan ha mange megabyte minne tilgjengelig. Noen mobiler akseptere ikke applikasjoner større enn 64 kB, andre igjen har ingen øvre grense. Noen har stor skjem, andre har liten skjerm. Dette må tas hensyn til for eksempel når man presenterer data. I SuMobil applikasjonen er det for eksempel en test på om skjermen er mindre enn en gitt størrelse. Hvis ja, skal salgsdataene presenteres vertikalt i to kolonner. Hvis skjermen er større enn en gitt størrelse, skal dataene presenteres horisontalt i fire kolonner.

2.4.1 Introduksjon J2ME Polish

Polish er delt inn i fire deler, se også figur 2.3:

- Byggerammeverket. Dette er et ant baserte rammeverket som lar deg preprosessere kildekoden, kompilerer, preverifiserer og pakker applikasjonen for forskjellige mobiler og språk.

- Klientrammeverket. Klient rammeverket er et sett med api'er som man kan bruke under mobilutviklingen. Den største biten er gui api'en som utvider MIDP 1.0 profil brukergrensesnittet. Polish gui'en tilbyr en rekke metoder og klasser man kan bruke til å gjøre mobilgrensenittet penere. Hele gui'en på sumobil klienten er designet ved hjelp av denne api'en. I tillegg er det lagt til en del hjelpeklasser, som:
  - ArrayList. Java ME har ikke inkludert klassen ArrayList som vi kjenner fra Java SE. Polish har implementert denne klassen, og fjernet synkroniseringen for å gjøre klassen kjappere. Brukes dette objektet av forskjellige tråder må man selv passe på synkronisering.
  - BitMapFont. Gjør at man kan bruke egendefinerte fonter i applikasjonen
  - TextUtil, hjelpeklasse som gjør at teksten man gir som input tilpasser seg skjermen på den gitte mobiltelefonen.
- IDE plug-ins. Polish tilbyr plugins til Eclipse. Navnet på dette tillegget er Mepose, og det er fremdeles under utvikling. Men en beta versjon er lagt til Polish 2.0. Den tilbyr funksjonalitet som:
  - Velge enheten du bygger mot
  - Kjøre og debugge applikasjonen. Med kjøre her menes at Mepose starter build scriptet
  - Klassisk syntaks utheving av polish spesifik kode for lettere lesing av sistnevnte
  - Funksjonalitet for å automatisk indentere polish spesifik kode
  - Autokomplettere polish prepressesing direktiver
- Hjelpeverktøy. Polish tilbyr verktøy for forskjellige formål. The binary data editor er spesialisert for å lage og modifisere strukturerte binære filer, som level data, mens font editoren lager bitmap fonter ut av hvilken som helst true type font.

Disse lagene henger tett sammen. Logging rammeverket har en klientside api, men er kontrollert av byggerammeverket, bare for å nevne et eksempel.

2.4.2 Enhetsdatabasen

Byggerammeverket til Polish angriper mange av utfordringene en står ovenfor når man bygger applikasjoner for Java ME verden. Den første utfordringen den løser er hvordan man bygger en applikasjon mot mange forskjellige

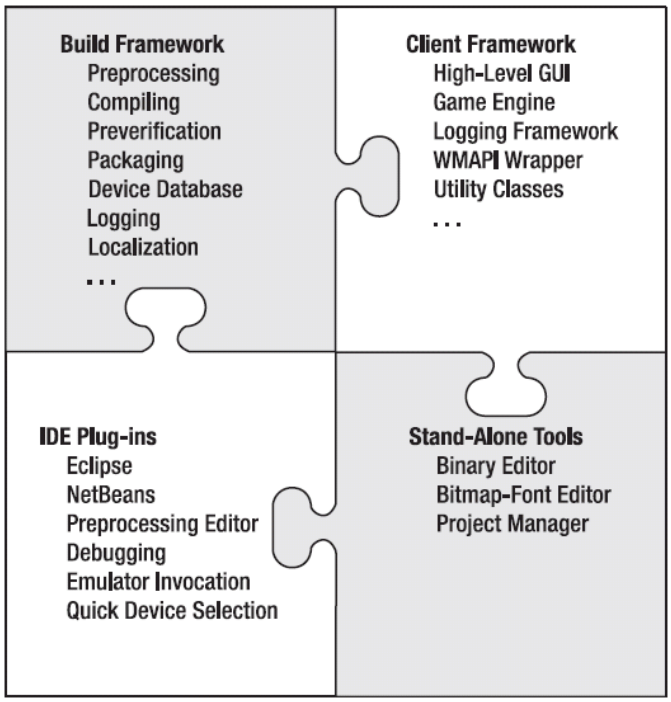

Figure 2.3: Modell av J2ME Polish arkitekturen

enheter. Selv om to mobiler begge har implementert midp 2.0 standarden, kan den ene mobilen ha kamera og støtte for mp3, mens den andre ikke har kamera og bare støtter midi lyd filer. Andre forskjeller kan være størrelsen på skjermen, antall farger, osv. For å løse dette har Polish opprettet en enhetsdatabase med teknisk informasjon om alle mobiltelefoner på markedet. Enhetsdatabasen består av fire file: device.xml, vendors.xml, groups.xml, apis.xml og bugs.xml. Under bygging bruker Polish informasjonen i disse filene for å fastslå hvilken deler av kildekoden og resursser den skal legge til den endelig jar filen rettet mot den aktuelle mobiltelefonen. La oss illustrere med et eksempel: Gitt vi bygger for Nokia E61 og Nokia 6310i. Førstnevnte støtter Midp 2.0 profilen har har stor skjerm. Nokia 6310 støtter bare Midp 1.0 profilen og har liten skjerm. Når vi bygger Sumobil klienten vil ikke Nokia 6310 telefonen støtte rapporten som presenteres som en graf, fordi tredjepartsbibliotekter vi bruker til å tegne opp denne grafen bruker metoder som bare finnes i Midp 2.0 profilen. Et annet eksempel er at Nokia E61 har stor skjem som gjør at vi kan presentere en rapport over fire kolonner, Nokia 6310 har så liten skjerm at vi må presentere den samme

rapporten over to kolonner og heller liste dataene vertikalt.

Device.xml

Device.xml inneholder en liste over de aller fleste mobiltelefonen på dagens marked. Nedenfor har vi et eksempel med Nokia E61.

```
<identifier>Nokia/E61</identifier>
```

Identifier elementet identifiserer mobiltelefonen ved å angi navnet på produsenter, etterfølt av enkelt slash og modelnavnet.

I features tagen definerer funksjoner mobilen har. Eksempler er at mobilen har kamera, kan motta peke events, osv.

```
<features>hasCamera, hasPointerEvents</features>
```

I groups tagen forteller man hvilken gruppe mobilen tilhører. En mobil tilhører alltid en gruppe. E61 tilhører for eksempel gruppen Seriees60E3 siden den er en del av Series 60 serien til Nokia og E3 fordi den har extension 3, altså version tre i denne serien. Som en digresjon kan det nevnes at Series60E3 er et barn av Series60 serien, men dette blir tatt opp i avsnittet med groups.xml.

```
<groups>Series60E3</groups>
```

For å definere mer komplekse egenskaper ved mobilen bruker man capability tagen. Nednefor er et eksempel som viser skjermstørrelsen

```
<capability name="ScreenSize" value="176x208"/>
```

```
<devices>
    <device>
        <identifier>Nokia/E61</identifier>
        <features>hasCamera</features>
        <groups>Series60E3</groups>
        <capability name="OS" value="Symbian OS 9.1"/>
        <capability name="JavaPlatform" value="MIDP/2.0, JTWI/1.0"/>
        <capability name="JavaConfiguration" value="CLDC/1.1"/>
        <capability name="JavaPackage" value="locationapi, mmapi1.1, wmapi1.0,
            wmapi2.0, m3g, pdaapi, btapi, sipapi, webservice, securityapi"/>
```

```
        <capability name="BitsPerPixel" value="24"/>
        <capability name="ScreenSize" value="320x240"/>
        <capability name="FullCanvasSize" value="320x240"/>
        <capability name="VideoFormat" value="3gpp, mpeg-4, realvideo"/>
        <capability name="SoundFormat" value="midi, midi64, amr, nb-amr, aac,
            mp3, mp4, aac+, eaac, eaac+, realaudio, truetone, wb-amr"/>
        <capability name="HeapSize" value="dynamic"/>
        <capability name="MaxJarSize" value="dynamic"/>
        <capability name="StorageSize" value="75 MB"/>
        <capability name="DRM" value="OMA DRM Forward Lock"/>
        <capability name="Region" value="Asia-Pacific, Europe, Africa"/>
        <capability name="AnnouncementDate" value="2005-10-12"/>
        <capability name="RootCertificates" value="UTI Root, Symbian A,
            Symbian B, Symbian C, Symbian D"/>
    </device>
</devices>
```

Vendors.xml

Alle produsenter man referer til i devices.xml må være definert i vendors.xml. Her kan man også liste opp felles funksjonalitet eller spesielle egenskaper alle enhetene til en gitt produsent tilbyr. Et eksempel nedenfor med mobilprodusentene Nokia og Simens

```
<vendors>
   <vendor>
      <name>Nokia</name>
      <features></features>
      <capability name="key.ClearKey" value="-8" />
      <capability name="key.ChangeInputModeKey" value="35" />
   </vendor>
   <vendor>
      <name>Siemens</name>
      <features></features>
      <capability name="Emulator.Class" value="SiemensEmulator" />
   </vendor>
<vendors>
```

Groups.xml

Det finnes mange enheter som deler feller funksjonalitet. Et eksempel er Nokia Series60 og Series40. Mobilene i hver av disse seriene har mange fellestrekk og kjører som regel også samme operativsystem. Istedenfor å bygge en applikasjon til både E60 og N73, kan man bygge applikasjonen til gruppen Series60. Eksempel fra groups.xml:

```
<groups>
   <group>
        <name>Series60</name>
        <parent>Nokia-UI</parent>
        <capability name="OS" value="Symbian OS 6.1" />
        <capability name="key.ClearKey" value="-8" />
        <capability name="JavaPlatform" value="MIDP/1.0" />
        <capability name="JavaConfiguration" value="CLDC/1.0" />
        <capability name="JavaPackage" value="wmapi, mmapi, chart" />
        <capability name="ScreenSize" value="176x208" />
        <capability name="FullCanvasSize" value="176x208" />
        <capability name="CanvasSize" value="176x144" />
        <capability name="Font.small" value="14" />
        <capability name="Font.medium" value="15" />
        <capability name="Font.large" value="19" />
        <capability name="Bugs" value="ExitCommandAlwaysPresent,
            drawRgbOrigin, sizeChangedReportsWrongHeight,
            TransparencyNotWorkingInNokiaUiApi,
            displaySetCurrentFlickers,
            requiresHardcodedCanvasDimensionsInFullScreenMode" />
        <capability name="dtmf.separator" value="/"/>
    </group>
</groups>
```

Bugs.xml

I denne filen er det listet opp feil i implementasjonen av Java ME. Ikke bare feil kan legges til, men også hvis en implementasjon av en funksjon i en viss enhet ikke oppfører seg som i henhold til spesifikasjonen.

Xml filene ovenfor (med unntak av bugs.xml) er organisert i et hieraki. Funksjonalitet definert i groups.xml overkjører funksjonalitet definert i vendors.xml. Funksjonalitet definert i device.xml overkjører de to andre xml filene. Figur 2.4 viser hierarkiet.

Siden alle disse konfigurasjonsfilene er skrevet i xml formatet, kan dem redigeres. Hvis en ny telefon kommer på markedet kan man legge den til disse filene i. Polish har i tillegg til enhver tid oppdaterte filer tilgjengelig på nettsiden sin, så den mest hensiktsmessige er å laste ned nyeste versjon fra Polish sine nettsider.

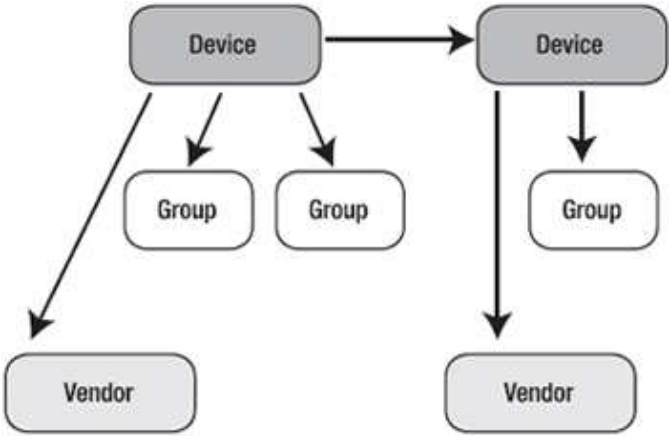

Figure 2.4: Model av enhetsdatabasehierakiet

2.4.3 Introduksjon til byggingen

Byggefasen består av syv steg:

- preprosessering
- kompilering
- obfuskering
- preverifikasjon
- pakking
- starte emulatoren

Velge hvilken mobil man bygger for

Hver byggefase starter med å velge mobilen man bygger for. Polish vil kjøre byggefasen en gang for hver telefon eller gruppe man har valgt. Man lister telefonene man bygger for i deviceRequirements seksjonen. Her kan man angi både konkrekte mobiler, grupper av mobiler eller produsenter.

```
<j2mepolish>
..........
    <deviceRequirements>
        <!-- her bygger vi for tre konkrete mobiltelefoner -->
        <requirement name="Identifier" value="Nokia/E61" />
        <requirement name="Identifier" value="Nokia/5300" />
        <requirement name="Identifier" value="Motorola/V8" />

        <!-- her bygger for en en gruppe mobiltelefoner, Nokia Series 60-->
        <requirement name="Identifier" value="Nokia/Series60" />

        <!-- her bygger vi en jar fil for alle Nokia telefoner -->
        <requirement name="Identifier" value="Nokia" />

    </deviceRequirements>
..........
</j2mepolish>
```

Preprosessering

I preprosesseringsfasen blir kildekoden manipulert før den blir kompilert. Sammen med enhetsdatabasen i Polish gjør dette at man kan tilpasse koden til forskjellige telefoner uten å måtte ha forskjellige sett med kode. Det finnes et utall forskjellige preprosesseringsdirektiver man kan bruke for å tilpasse applikasjonen til forskjellige telefoner. I Sumobil klienten brukes disee preprosesseringsdirektivene forskjellige plasser i koden, blant annet en sjekk om telefonen støtter midp2. Hvis ja, kan det legges til en graf funksjon som bruker et tredjeparts bibliotek som krever midp2. Hvis telefonen bare støtter midp1, vil Polish under preprosesseringen fjerne all kode som forutsetter midp2. Nedenfor er kode fra eksempelet ovenfor. En tilsvarende test gjøres mange plasser i koden:

```
//#if polish.midp2
    choice = new String[]{chain, shop, salepersons, montlysales};
//#else
    choice = new String[]{chain, shop, salepersons};
//#endif
```

Kompilering

Polish legger automatisk til alle valgfrie api'er som støttes av den gitte mobilen i classpathen til kompilatoren.

Obfuskere

Å obfuskere Java applikasjoner er bra av to grunner:

- gjør som regel applikasjonen betraktelig mindre ved at den fjerner alle ubrukte biblioteker, klasser, metoder og variabler, i tillegg til at klassenavn, metoder og variabler gis nye, kortere navn som gjør at de opptar mindre plass.
- Obfuscating steget gjør at det er nesten umulig å dekompilere applikasjonen.

Innbygget i Polish er en obfuskator som heter ProGuard[20]. Men man kan også bruke andre obfuskatorer. SuMobil klienten gikk fra 1400 kb til 163 kb etter å ha blitt kjørt gjennom obfuskatoren.

Preverifisering

Polish bruker preverifisereren som følger med Wireless Toolkitet til Sun.

Pakking

I pakkesteget blir den endelige applikasjonen pakket sammen med alle filene den skal inneholder. Polish bruker følgende konsept: Man har en mappe 'resources'. Her legges alle ressurser som bilder, css, lyd, tekstfiler, språkfiler, osv.

Produsent og modell spesifikke ressurser. Man kan også plassere ressurser for spesifikke mobilprodusenter på formen resources/[vendor-name]. For spesifikke modeltyper hos denne mobilprodusentene, brukes filstien resources/[vendor-name]/[device-name]. Skal man for eksempel legge til ressurser bare til til Nokia E61, legger dem i følgende filsti resources/Nokia/E61/

Inkludere gruppe spesifikke ressurser. Man kan organisere ressurser i grupper for å oppnå bedre og mer logisk abstraksjon. Gitt at man oppretter en mappe under ressurser man kaller "BitsPerColor+16", og legg

høyoppløselige bilder der. Da vil de bli lagt til applikasjonen til den mobilentypen man bygger mot har en skjerm som støtter 16 eller flere farger. Alle mobiler er medlemmer i en eller flere grupper, enten implisitt ved egenskapen til telefonen, eller eksplisitt definert i groups.xml.

I Polish.css kan man definere de forskjellige stilene man ønsker å bruke i applikasjonen. Polish.css er en spesiell fil i forhold til de andre filene i ressursmappen. Dette fordi man kan ha en stilfil i ressursmappen. Da vil denne stilfilen ligger i roten. Videre kan man legge inn stilfiler spesifikt for spesifikke mobiltelefoner eller grupper av mobiltelefoner på samme måte som beskrevet ovenfor i de to foregående paragrafene. Stilene i disse stilfilene vil bli lagt til stilene som allerede ligger i roten, istedenfor å bli erstattet fullt ut som det ville blitt gjort med vanlige filer.

Dette brukes når sumobil klienten bygges. Der har man en polish.css i bunn, og to polish.css i de to undermappene hvor man overskriver ".turnoverform" stylen, altså den som styrer hvor mange kolonner salgsdataene skal vises over.

Bruke språk spesifikke ressurser. Man kan legge språkfiler i egne mapper, så vil Polish plukke ut den riktige språkfilen for det aktuelle språket under bygging. I Sumobil klienten har vi to mapper, "no", og "en", som inneholder hhv norsk og engelsk språkfil

Man kan også velge å bruke en annen resource mappe under bygging. Derved kan hele designet drastisk endres ved å bare forandre en linje i Polish build filen.

2.4.4 Polish GUI

Brukergrensesnitt i standard Java ME

I standard Java ME er det profilene som er ansvarlig for å tilby biblioteker for brukergrensesnittet. Den desidert mest brukte enhetsprofilen i dag er midp, det er også denne som er brukt i Sumobil klient løsningen. Det vil derfor bli en gjennomgang av hvordan gui håndteres i midp profilen, og hvordan Polish har utvidet denne funksjonaliteten.

Midp profilen tilbyr to forskjellige api'er for brukergrensesnitt til utvikleren:

- høynivå gui
- lavnivå gui

I høynivå gui'en er det predefinert et sett med gui elementer man kan bruke under utvikling. Gui elementene er implementert i profilen, og det er mobiltelefonen som bestemmer hvordan disse elementene skal presenteres på skjermen. Presentasjon er scrolling, navigasjon, visuelle karikastikker som farger, form, font, osv av gui elementene. Det gjør det enkelt å bruke dette brukergrensesnittet, men gjør samtidig at man har liten kontroll hvordan brukergrensesnittet blir presentert på mobiltelefonen.

Med lavnivå gui'en har applikasjonen bedre kontroll med hvordan brukergrensesnittet blir presentert. Denne gui'en ble laget spesielt med tanke på spill hvor man må ha nøyaktig kontroll på plassering av gui elementer. Men det er tidkrevende og relativt komplisert å lage brukergrensesnitt med denne gui'en.

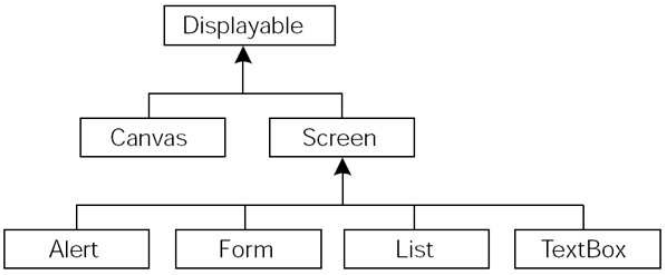

Figure 2.5: Klassediagram av Java ME gui'en

Gui i Polish

Polish utvider (arver) høynivå gui'en til MIDP 1.0 profilen. Polish GUI er et kraftig verktøy som gjør det mulig å definere hvordan hele applikasjonen skal presenteres på skjermen, uten å måtte bruker lavnivå gui'en i MIDP. Konseptet bak Polish GUI er beskrevet i diagrammet nedenfor. Ideen er at man man skiller presentasjon og logikk, på samme måte som man gjør på nettsider med html/css. Man har en ekstern fil som inneholder alle stilene, så refererer man til disse stilene i kildekoden. Man skriver stilene på css formatet og lagrer dem i en fil med navn Polish.css. Det blir allikevel feil å kalle dette stilsettet for css slik man gjør i J2ME Polish. Css (Cascade Style Sheet) er et stilark for html sider, ikke for Java ME applikasjoner. Man kan derfor ikke bruke alle css stilene slik vi kjenner dem fra den vanlige standarden. Men at Polish har brukt css syntax og den samme måten å tenke på, gjør at det er enkelt å forstå konseptet og enkelt å sette seg inn i.

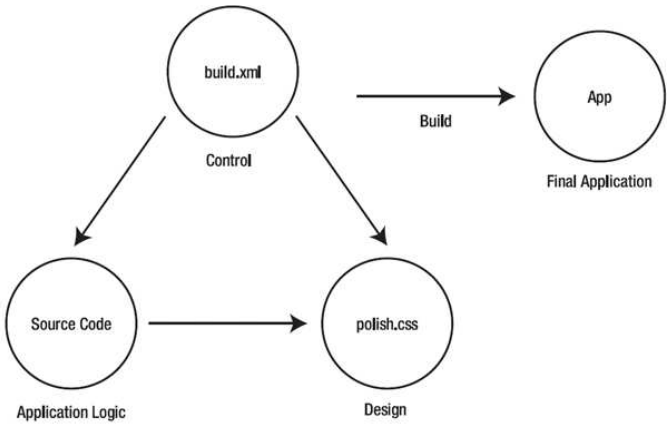

Figure 2.6: Viser hvordan Polish under preprosseseringen vil erstatte stil preprosesserings elementer i kildekoden og ved hjelp av stilfilen polish.css lage et nytt design ved hjelp av Polish sine egne java klasser

Gui'en til Polish blir brukt i Sumobil klienten. På den måten har det vært mulig å lage et pent brukergrensesnitt. Presentasjonen av nesten alle elementer i brukergrensesnittet er direkte eller indirekte styrt av Polish. Det er også mulig å definere presentasjon av tabeller og kolonner. Kolonnefunksjonen er blant annet brukt under presentasjon av rapportene.

2.4.5 Evaluering av J2ME Polish

J2ME Polish forenkler portering og utvikling av Java ME applikasjoner. Byggedelen av dette rammeverket gjør utfordringene med portering til mangfoldet av mobiltelefoner overkommelig. Uten et slikt rammeverk måtte man gjort porteringen manuelt. I praksis ville dette medført at man sannsynlig hadde utviklet applikasjoner til bare en liten del av markedet, for eksmpel bare Nokia telefoner, for telefoner med spesifikasjoner over en viss størrelse eller/og telefoner som støtter en gitt profil, osv. Med bruken av preprosessering har man lagt til et ekstra nivå som gjør at man kan angi i kildekoden hvilken mobiltelefon man utvikler mot uten at man denne ekstra informasjonen øker størrelsen eller kompleksiteten til koden som blir sendt videre til kompilatoren for kompilering. Dette er viktig da hver kilobyte er viktig å spare fordi man utvikler mot så relativt ressurssvake maskiner som mobiltelefoner.

2.4.6 Evaluering av J2ME Polish kontra bruk av AOP

Hvis man tar utgangspunkt i de to ikke-trivielle måtene man kan løse porteringsproblematikken har J2ME Polish og Tira Jump 2008 brukt hver tilnærming. Den enkleste av løsningene er J2ME Polish, og denne løsningen gjør at man på en relativt intuitiv og nøyaktig måte kan ta hensyn til porteringsproblematikk direkte i koden. Undertegnede har ikke fått prøvd Tira Jump 2008, men sitter igjen med et inntrykk at at denne løsningen er mer avansert og automatiserer porteringproblematikken ved bruk av gjenbrukbare komponenter. Bakdelen er nok at denne tilnærmingen ikke er like treffsikker som bruk av preprosseseringelementer og J2ME Polish. På Tira Wireless sine nettsider ser man at dem har et samarbeid med Enought Software som utvikler J2ME Polish, og i mange sammenhenger kan en kombinasjon av disse løsningene være den beste tilnærmingen.

Kapittel 3

Alternative teknologier

SuMobilløsningen kunne vært utviklet på andre måter enn å skrive en Java ME applikasjon. Mange av de alternative teknologiene ville i tillegg vært mer passende og mindre tidkrevende å bruke, men læringsutbytte ville blitt betydelig lavere. Med kunnskapen om Java ME utvikling, utfordring og muligheter i denne teknologien, samt bruk av rammeverk som Polish i bunn gjør at man har bedre forutsetning for å evaluere og forstå andre løsninger. Det vil bli en beskrivelse av de alternative teknologier, etterfulgt av en beskrivelse av hvordan SuMobilløsningen kunne være løst ved bruk av dem.

3.1 Mobil nettleser - Opera Mini

Norske Opera Mini er en nettleser laget for mobiltelefoner. Den er utviklet i Java ME og krever at mobiltelefonen har innbygget støtte for Java. Med denne nettleseren kan man surfe på nettet på alle vanlige html sider, men alle sider tar turen innom Opera sin tjener for komprimering og tilpassing før de blir sendt ut til mobilen. Nettsider for skrivebordsmaskiner kan være på mange megabyte. Størrelse er ikke et stort problem for en skrivebordspc med bredbåndstilkobling, men på mobiltelefoner med tregt trådløs oppkobling og begrenset oppløsning på skjermen og langt mindre minne og cpu, er denne komprimeringen gull verdt. Et annet poenget, kanskje den viktigste fordelen, er at teleselskapene normalt tar betalt for antall megabyte man laster ned til mobilen, i motsetning til på skrivebordspcer hvor man normalt har en fast pris på en bredbåndslinje.

3.1.1 Historie

Opera Mini er basert på Operas nettleser for skrivebordsmaskiner. Den var i utgangspunktet utviklet for mobiltelefoner som vanligvis ikke ville være i stand til å kjøre en nettleser. Opera Mini ble introdusert 10 August, 2005 som et pilotprosjekt i samarbeid med den norske fjernsynskanalen TV2, og var da bare tilgjengelig gjennom TV2. Den første betaversjonen ble lansert for vanlige brukere oktober 2005.

Opera Mini 2.0 ble lansert mai 2006. Den inkluderte nye funksjoner som muligheten for å laste ned filer, nye brukervalgte skin, søkemotor valg i den nylig innebyggede søkelinjen, en speed dial funksjon og forbedret navigasjon.

Opera Mini 3 beta ble lansert november 2006 og introdusert sikker nettsurfing (krytert på socketnivå), RSS feeds, foto opplasting og skjuling av lister (eng:content folding). "Content folding" betyr at lange lister med innhold skjules for bedre leslighet av hele siden. Et pluss tegn (+) vil vises og man kan klikke på dette for å vise hele innholdet.

Opera Mini 4 ble lansert november 2007. Denne versjonen inkluderer muligheten for å vise nettsider relativt likt som på en skrivebordsmaskin ved hjelp av to funksjoner - oversikt/zoom, i tillegg til landskapsvisning. I oversiktsmodusen kan brukeren få en oversikt over hele nettsiden ved at opera zoomer ut og viser hele siden. Ved hjelp av peker funksjonen på telefonen kan man bruke zoom funksjonen til å velge et område på siden man ønsker å lese, ergo zoome inn på. I tillegg har dem bygget inn Opera Link, som er et verktøy som gjør man kan synkronisere bokmerkene man har både på mobiltelefonen og på Opera på skrivebordsmaskinen.

3.1.2 Hvordan virker Opera Mini

Ulikt vanlige nettlesere vil Opera Mini gjøre en forespørsel til en av Opera's proxy tjenere når den vil hente en nettside. Denne proxy tjeneren vil på vegne av Opera Mini gå ut på nettet og hente siden. Siden blir så komprimert og levert til telefonen i et markup format som kalles OBML (Opera Binary Markup Language). Dette formatet sammen med komprimeringen av innhold på siden (som bilder, mm) reduserer den aktuelle nettsidens størrelse med opptil 90 prosent ifølge Opera's egne beregninger [24].

Ved tilkobling til proxy tjenerene til Opera har man to valg - socket eller http tilkobling. Opera Mini bruker som standard socket tilkobling, men man

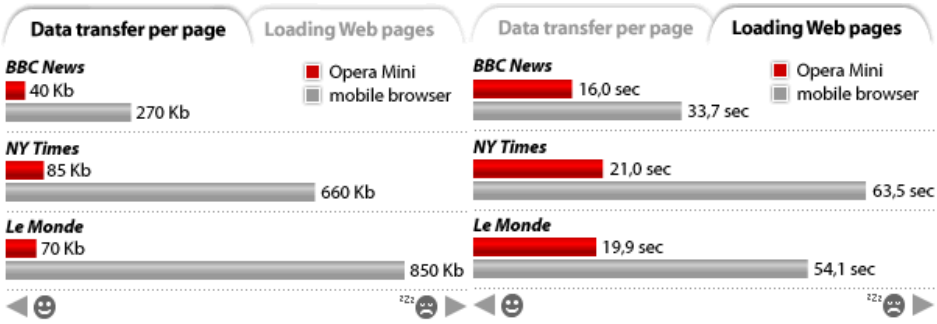

Figure 3.1: Viser datamengden Opera Mini laster ned forhold til konkurrenter, i tillegg til nedlastingshastighet.

kan velge å bruke http tilkobling. Fordelen med socket tilkobling er at denne bare åpnes en gang og kan brukes til mange overføringer. I tillegg åpner det for at Opera tjenerene kan dytte data til din mobil, som for eksempel Opera Link [22]. Http tilkoblinger vil åpne en tilkobling for hver side man forespør, og vil derfor være tregere. Men ikke alle mobiltelefoner støtter socket tilkoblinger, og da må man bruke http tilkobling.

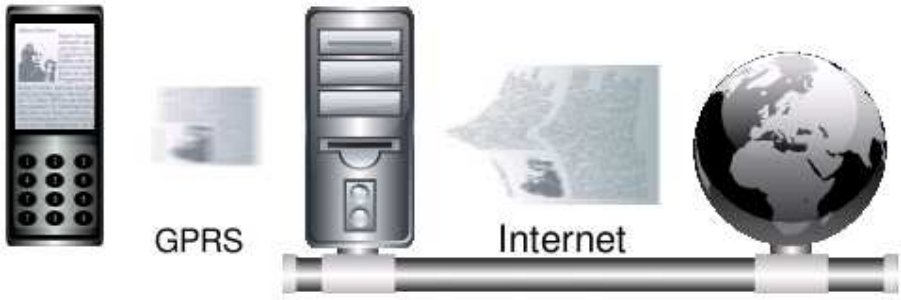

Figure 3.2: Når en bruker surfer på nettet med Opera Mini, blir forespørselen sent via Genera Packet Radio Service (GPRS) til en av Opera Software sine proxytjenere. Tjeneren vil så hente siden, prosessere, komprimere og sende den tilbake til brukerens mobiltelefon.

Opera Software har over 100 proxytjenere som håndterer Opera Mini trafikk, og kjører på Linux [18].

Presentasjonmoduser i Opera Mini

Full skjerm presentasjon. Standard presentasjonsmåte for mobiler med skjerm større enn 128 pixel er full skjerm visning, også kjent under "desktop redering mode". I denne modusen, når Opera laster en webside, vil den først zoome ut og vise siden på nesten samme måte som den ville bli vist på en skrivebordsmaskin. Hvis mobiltelefonen har en berørbar skjerm, kan brukeren velge en seksjon av siden og zoome inn. Alternativt kan brukeren bruke mobilens piltaster til å kontrollere en virtuell peker og bruke denne til å zoome inn. Opera Mini automatisk gjetter hvor sidens innhold begynner, så istedenfor å finne og scrolle til starten av innholdet, kan brukeren klikke en gang for å gå til det anbefalte startpunktet.

Liten skjerm presentasjon. For mobiler med skjermer som er 128 piksler eller mindre er standard presentasjonsmodus Small-Screen Rendering (SSR). I denne modusen er siden formatert til en enkelt vertikal kolonne hvor man bare scroller opp og ned. Lange lister og navigasjons bars blir automatisk kollapset av en funksjon som kalles "content folding", altså skjuling av innhold. Et pluss (+) symbol blir vist ved siden av det kollapsede innholdet. Når det klikkes, vil det skjulte innholdet vises. I denne modusen vil alle bilder bli skalert ned til maks 70 prosent av skjermstørrelsen i begge retninger. Denne modusen kan testes i en vanlig Opera nettleser ved å klikke shift+F11.

3.1.3 SuMobil løsningen i Opera Mini

Sumobil løsningen kunne blitt laget gjennom Opera Mini istedenfor å skrive en Java ME applikasjon fra bunnen. På mange måter var dette den orginale måten mobilløsningen til Susoft var tiltenkt å bli laget på, men forskjellige faktorer gjorde at dette ikke ble den endelige løsningen.

Det vil i denne delen ble beskrevet hvordan Sumobil løsningen kunne blitt utformet for å bli løst i Opera Mini. Selv om Opera Mini er ment for å surfe på nettet på vanlige nettsider, og Opera proxytjenerene tilpasse disse sidene til mobilen, er ikke denne løsningen idiotsikker. Det finnes retningslinjer[25] man bør følge og teknologi man kan bruke når man utvikler sider til mobiltelefoner. Noen av disse teknikkene kan brukes slik at siden ser bra ut både på en skrivebordsmaskin og en mobiltelefon.

Det spesielle med web på mobiltelefoner

Mobiltelefoner kan oppsummeres ved at dem er trege og små. Å designe nettsider for mobiltelefon er ikke så forskjellig fra å lage nettsider for skrivebordsmaskiner, men man må ta større hensyn til hva som er relevant å ha med og hvordan det kan presenteres effektivt i form av størrelse og kompleksitet. Overflødig innhold bør fjernes da det ikke er like enkelt å skumlese eller ignorere dette som på en stor skjerm.

Design av nettsiden

Den foretrukkede layouten for mobiltelefoner er en enkel kolonne, med den viktige informasjonen nær toppen av siden. Overflødig informasjon bør være fjernet eller skjult. Horistontal scrolling under lesing er veldig distraktherende og bør unngås. Følgende strategier kan brukes for et godt design for mobiltelefoner[23]:

- **Bruk flytende design**. Design som er basert på tables, frames eller css absolute position bør ikke brukes som design for sider for mobiltelefoner.
- **La de mobile nettleserene gjøre reformateringen**. Lag et design som er optimalt for en normalt skrivebordsmiljø, og la nettlesere på mobiltelefonen ta seg av presentasjonen. Denne strategien virker bra hvis siden er basert på et flytende design og er laget i henhold til web standarder.
- **Lag mange versjoner av nettsiden**. Lag et design for skrivebordsmaskiner og et for mobiltelefoner, og la tjeneren gjennom sniffing avgjøre hvilken versjon den skal sende.
- **La innholdet tilpasses utlike måter for presentasjon**, ved å bruke CSS handheld media, Media Queries eller Javascript.

Andre tips det bør tas hensyn til under designfasen er:

- Unngå visuelle avhengigheter mellom elementer på nettsiden. Ikke la elementer være avhengig av å være ved siden av hverandre. To elementer som er ved siden av hverandre på en stor skjerm vil normalt bli plassert under hverandre på eller til og med være langt fra hverandre på en mobilskjerm, avhengig av enhet og hvordan siden er kodet.

- La siden nedgraderes på en smidig måte. Hvis et element ikke kan vises på mobiltelefonen, la siden fremdeles presenteres på en akseptable måte.
- ikke inkludere unødvendige elementer
- Frames and css posisjonering bør ikke brukes på sider for mobiltelefoner. Vær også forsiktig med tables og floats.

Nettverkshastighet

Mediafiler, hovedsaklig bilder, utgjør mesteparten av den brukte båndbredden når man laster ned nettsider. Siden bilder vil bli skalert ned på mobilskjermen, er det sløsing av båndbredde å ha inkludert overdimensjonerte bilder. Andre media filer som lyd, video eller binære filer kan ofte ikke spilles av på mobiltelefoner, og bør derfor ikke legges til siden.

En strategi er å bruke bilder med forskjellig størrelse for skrivebordsmaskiner og mobiltelefoner. Dette kan gjøres på tjenersiden (ved sniffing) eller på klientsiden. Man kan bruke CSS3 'content' egenskapen til å forandre innhold, som:

```
@media screen {#image {content: url(big-image.png)}}
@media handheld {#image {content: url(small-image.png)}}
```

Det kan også gjøres ved hjelp av JavaScript

Prosessorhastighet

Prosessorene i mobiltelefoner vil være betydelig tregere enn i skrivebordsmaskiner. I tillegg er batteri alltid en begresning. Mens forskjellen mellom et gjennomsnittlig og et dårlig design på et skrivebordsmaskin er ubetydelig, kan det utgjøre en stor forskjell på mobiltelefoner. All nettleserprosessene tar tid - parse html/css/javascript dokumentene, presentere bilder, kjøre script, presentere html dokumentet, aksessere DOM treet, bruker interaksjon, osv. Spesielt noen prosesser er dyre i form av ressurser, som å reloade en hel side eller DOM tre manipulering [23]. Til mer kompleks en side er, til mer er det å hente ved å effektivisere disse prosessene.

Minne

Mobiltelefoner har alltid lite minne sammenlignet med skrivebordsmaskiner, så en bør identifisere hva på en nettside som konsumerer mest. Når man

laster ned bilder vil disse lagres i minne. Man bør derfor gjøre bilder så små som mulig. En annen ting er mellomlagring (eng:caching). Hvis bilder er store, vil mellomlageret tømmes oftere, og elementer må derfor ofte lastes ned på ny hvis man går tilbake til siden. Markup tagger tar mer minne enn tekst når nettleseren bygger opp DOM treet [23]. Til færre markup elementer som blir brukt, til kjappere vil siden respondere

Brukergrensesnitt

Mobiltelefoner har små skjermer og som regel ikke et fullt tastatur. Selv i de tilfellene hvor dem har fullt tastatur, er disse små og mer tidkrevende å taste inn på enn vanlige tastatur. Å navigere på en nettside og spesielt å skrive inn tekst er mer kløne enn på en skrivebordsmaskin, så dette må tas hensyn til når man designer brukergrensesnittet. Tommelfingerregelene må være at det er enklere å velge noe enn å skrive inn noe. Bruk derfor heller valg bokser i html koden enn textbokser. Det er også lettere å velge fra en flervalgsliste (eng:multiple choice) enn en nedtrekksliste (eng:drop down list). Mobiltelefoner har som regel ikke mus, så ikke lag et grensesnitt som forventer dette. Ikke bruk onmouse og lignende handlinger (eng:events). De fleste mobile nettlesere viser en side om gangen, og selv om det finnes noen mobile nettlesere som støtter det, ikke bruk popup vinduer. Ikke bruk dynamiske menyer da de fleste mobile nettlesere vil utvide (eng:expand) dem automatisk.

3.1.4 Portering ved bruk av Opera Mini

Med en Opera Mini løsning hadde man ikke gjort portering mot spesifikke mobiltelefoner eller grupper av mobiler som med Java ME applikasjoner, men mot Opera Mini. Mini er en Java ME applikasjon, og Opera har allerede gjort porteringen i sin applikasjon. SuMobil løsningen ville vært et lag over Java ME. I praksis vil porteringen i denne løsningen være at man tar hensyn til de designrådene til Opera som er beskrevet i avsnittene ovenfor. Opera Mini vil så sørge for at SuMobilløsningen vil presenteres som den skal på selve mobiltelefonen.

3.2 WAP

Wap står for "Wireless Application Protocol" og er en åpen standard for applikasjoner som bruker trådløst kommunikasjon. Motivet med standarden var å åpne for tilgang til internet fra mobiltelefoner og pda'er. Wap

spesifiserer en hel protokollstabel. Nederst har man Datagram Protocol (WDP) som gjør at nettverkslaget ligner på udp, altså upålitelig datagram overføring. Det er dette laget som på er ansvarlig for peer-to-peer kommunikasjonen, altså det ansvaret udp/tcp har i den velkjente tcp/ip protokoll stabelen. Wap protokoll stabelen har også definert en protokoll, wtp, som mer effektivt enn tcp håndterer utfordringene i trådløse nett. Det beste eksempelet her er tcp som vil bremse utsendelsen av pakker hvis det er et høyt pakketap. Dette er gjort under en forutsetning at pakketap i et trådbasert nettverk skyldes kø i nettet fordi trådbaserte nett har relativt få pakketak, mens pakketap i trådløse er en del av trådløse netts natur, og bør løses ved at senderen sender flere pakker, ikke færre, som tcp gjør.

Wap har et eget format som er veldig likt (x)html - Wireless Markup Language (WML). Teknisk sett er wml sider som validerer mot wml dokument typen (eng:doctype). Et wml dokument er kjent under termonologien "kortstokk" (eng:deck). Data i kortstokken er strukturert under et eller flere "kort" (eng:cards). Hvert kort er en side som representerer et enkelt samspill med brukeren.

Det er forskjellige måte å tilby wap til kunder på. Man kan enten har en egen wap-tjener som kundene kobler seg direkte til, eller man kan ha en vanlig web-tjener og en gateway mellom denne og brukeren som oversetter html siden til en mobilvennlig wml side som sendes til mobilen.

Wap var kjent som teknologien som skulle bli for mobilen det http og html var for skrivebordsmaskinene. Men suksessen uteble. I ettertidklokskapens lys kan vi finne grunner til dette:

- Wml formatet gjorde at man stengte mobilbrukere ute fra internettsamfunnet. Man kunne bare lese side som var på wml formatet, og tilbudet av disse sidene ble aldri særlig stort.

- Under-spesifisering av terminal krav. Wap sider oppførte seg veldig forskjellig fra mobil til mobil siden det ikke var strenge nok krav til hva mobilprodusentene skulle støtte. Eksempler her er at på noen mobiler kunne man taste 4 for å komme til link fire i wml siden, på andre mobiler ble det lagt egne nummer på linkene til venstre for dem, mens på atter andre mobiler var ikke denne funksjonen støttet i det hele tatt

- Problemer med begrensede brukergrensesnitt muligheter. Mange mobiler har så små skjermer og i svart/hvitt som gjør lesing av informasjon vanskelig. Dette har dog bedret seg i dag, men da wap kom hadde de fleste mobiler veldig små skjermer.

- Mangel på åpenhet. Mange mobilselskaper tilbydde bare sider fra innholdsleverandører som var "godkjent" på forhånd eller til selskaper som betalte for seg. Muligheten til å fritt surfe på nettet ble var små. Det fantes teleselskapet som gjorde det mulig å surfe fritt på nettet, men ofte laget dem en startportal som man startet i, og fordi det i brukergrensesnittet var det vanskelig å skrive inn url'er manuelt, tok ikke brukerne seg bryet med å surfe fritt.

- Mangel på å dra med seg innholdstilbydere. Teleselskapene trodde at så lenge man laget infrastrukturen, så ville innholdet fylles av seg selv. Slik gikk det ikke. Et nærmere samarbeid med innholdsleverandører på forhånd kunne kanskje økt tilbudet av wap sider.

[33]

3.2.1 SuMobilløsningen utviklet i WAP

SuMobilløsningen kunne vært utviklet med bruk av wap teknologien. En slik løsning ville måtte ta mange av de samme hensynene som den måtte hvis den ble laget for Opera Mini.

Kapittel 4

SuMobil

SuMobil er navnet på mobilløsningen i denne oppgaven. SuMobil gjør at man får tilgang til et utvalg rapporter på mobiltelefonen. Løsningen består av en mobilklient og en applikasjon som kjører på en av Susoft sine tjener. Mobildelen av løsningen kalles SuMobil Klient, mens applikasjonen på tjeneren kalles SuMobil Portal. Hele løsningen kalles SuMobil Løsning, eller kort for SuMobil. Brukes bare benevnelsen SuMobil i teksten vil dette referere til hele løsningen, både klient og tjenersiden. Det vil i følgende kapittel bli en gjennomgang av hvordan SuMobil er bygget opp og implementert.

4.1 SuMobil arkitektur

Figur 4.1 viser en modell av SuMobil arkitekturen. Mobilklienten er en Java ME applikasjonen som kjører på mobiltelefonen og som tar seg av rapportvalg, forespørsler og presentasjon av rapporter, mens tjenerapplikasjonen er en Java EE applikasjon som er ansvarlig for å levere rapportene og utseendelse av nedlastingslinker på sms. I tillegg tilbyr SuMobil Portalen statistikker over utsendte sms'er og hvem som har installert SuMobil Klienten.

SuMobilklientene distribueres til kundens mobiltelefoner, så det er mange klienter som kobler seg mot tjeneren. På tjeneren er det èn applikasjon som håndterer alle forespørsler. Denne vil dynamisk koble seg til den korrekte butikkjedes database. Bakerst i systemet har vi databasene hvor hver butikkjede har sin egen database.

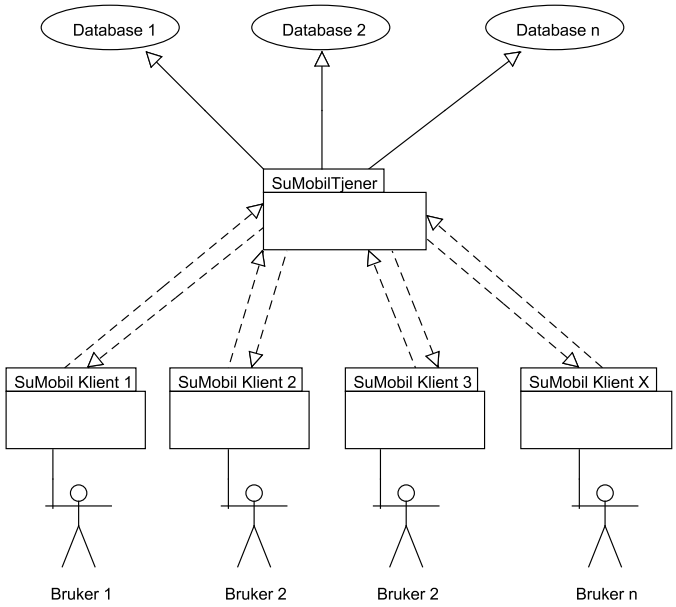

Figure 4.1: En modell over SuMobilløsning arkitekturen.

4.2 SuMobil mot forskjellige databaser

Siden hver kunde hos Susoft har en egen database, vil SuMobil Portal måtte koble seg til mange forskjellige databaser. SuMobil klienten må også på en eller annen måte angi hvilken database den vil hente rapporter fra.

Det vil først bli beskrevet hvordan en bruker laster ned og installere applikasjonen på mobiltelefonen, så en gjennomgang over alternative måter oppgaven med valg av database kunne blitt løst på, og til slutt hvilken løsning som ble valgt. Det kan kort nevnes at tre av de fire løsningsalternativene faktisk ble implementert, og at prøving og feiling gjorde hver ny implementasjon litt smartere.

Vi har følgende brukerscenario:

- Kunden logger seg inn på sin Global One portal
- Trykker knappen "Få tilsendt SuMobil"
- Kunden får tilsendt en sms med nedlastingsurl
- Applikasjonen lastes ned på mobilen, installeres og kjøres

SuMobil Tjener Som databaserammeverk er det valgt Apache iBatis[31]. Dette er et persistence rammeverk som gjør at man kan koble SQL spørringer til Plain Old Java Objects (pojo) [32]. Selve spørringene skrives i en xml fil og gis et navn. I Java kildekoden angir man navnet på spørringen man ønsker å kjøre, og i retur får man et pojo objekt. Dette gjør at man får skilt ut spørringene fra kildekoden, noen som gjør utvikling og ikke minst vedlikehold enklere. På tjenersiden er databasetilkoblingene manuelt angitt i en iBatis konfigurasjonsfil. SuMobil Portal tar tre innparametre - brukernavn, passord og databasenavn. Det siste parameteret mappes så mot iBatis sin konfigurasjonsfil og et tilkoblingsobjekt opprettes. Tilkoblingen lagres i brukerens sesjon. For hver forespørsel denne brukeren gjør vil databasetilkoblingen hentes fra sesjonen og brukes.

SuMobil klient. På klientsiden var det tre måter å løse oppgaven på:

- Det første alternativet var å få brukeren til å skrive inn navnet på databasen han/hun ville koble seg mot i Java ME applikasjonen på mobiltelefonen. Ulempen ved denne løsningen var at det var tungvint å skrive inn et ekstra felt, i tillegg til at databasenavnet måtte vært stavet helt korrekt. Fordelen var at vi her bare trengte å generere en jad fil for hver mobiltelefon.

- Det andre alternativet var å legge databasenavnet i jadfilen som brukeren lastet ned. Dette ville gjort at brukeren hadde sluppet å skrive inn databasenavnet ved innlogging. Ulempen er at det må genereres en jadfil for hver kjede og for hver mobiltelefon. Denne løsningen ble faktisk implementert, og da var det et eget program som autogenererte jad filer med riktig innhold

- Det tredje alternativet og det som ble den endelige løsningen var å lage en servlet som genererte jad filer som inneholder databasenavnet. Servleten legger til egenskapen MIDlet-chain=¡dbnavn¿ i jadfilen som sendes ut til mobiltelefonen.

4.2.1 Distribusjon av SuMobil

"Enkelt" er et nøkkelord når det kommer til suksess for mobilapplikasjoner. Få er komfortable med programmer på mobiltelefoner og det må gjøres ultimat enkelt å:

- laste ned

- installere
- kjøre applikasjonen.

Det første steget gjøres ved å sende ut en sms med nedlastingslink til brukeren. Denne sms'en vil inneholde en link på følgende format

```
http://<server>/<appname>/jad?user=<user>&password=<password>&
lang=no_NO=<språk>&mype=<mobiltype>&r_type=<rapporttype>&
r_value=<rapportverdi>
```

De fleste moderne mobiltelefoner vil kjenne igjen at dette er en http link, og derved gjøre det mulig å enkelt klikke på den. Denne linken vil gjøre et kall til en servlet på tjenersiden som vil generere og returnere en jad fil med alle obligatoriske parametre, i tillegg til brukerdefinerte egenskaper. De brukerdefinerte egenskapene blir hentet i to steg. Den vanlige inngangsporten til SuMobil Portalen vil være gjennom Global One Portalen. Man har allerede logget seg inn på Global One Portalen med brukernavn og passord i tillegg til at man har angitt database, og de samme innstillingene brukes også i SuMobil Portalen. Av den grunn er det laget slik at disse parametrene blir videresendt til SuMobil portalen når man aksessere denne gjennom Global One Portalen. På den måten oppnår man Single Sign On [26].

Når man bestiller en nedlastingslink på SuMobil Portalen angir man om man ønsker salgsdata for kjede, en region eller en butikk. Denne informasjonen legges til i jadfilen i form av tre parametrene - report type, report value og report name. De to siste parameter angir hvilken tjener og hvilken port den aktuelle SuMobil Portalen lytter på.

```
MIDlet-chain: db
MIDlet-user: username
MIDlet-password: password
MIDlet-report_type: rapporttype
MIDlet-report_value: rapportverdi
MIDlet-report_name: rapportnavn

MIDlet-server: server
MIDlet-port: port
```

Ved mottak av jadfilen vil mobiltelefonen starte nedlasting og installering av applikasjonen. Her har vi et potensielt problem ved at ikke alle mobiltelefoner akseptere usignerte Java ME applikasjoner. På de fleste mobiler

er det mulig under innstillinger å velge at mobilen skal akseptere usignerte applikasjoner, men undertegnede har allikevel opplevd serfikatproblemer på noen mobiltelefoner.

Men om alt har gått fint vil applikasjonen installeres og være klar for kjøring. Man slipper å taste inn brukernavn/passord/database fordi disse allerede er lagt inn i jadfilen man fikk tilsendt.

4.2.2 Belasting av kunder

Hver SMS som sendes ut koster X antall ører hos Susoft sin SMS leverandør, Netcom. Dette skal kunden belastes for. Derfor er det satt opp et system der kunden blir belastet for hver sms som sendes ut. Løsningen er å logge hver SMS som sendes ut. Det har blitt laget tre rutiner:

- en loggmetode som logger navn, telefonummer, mobiltype, kjede, bruker, nedlastingsurl og nedlastingstidspunkt i en database
- en funksjon som leser fra databasen og kalkulerer hvor mange SMS en gitt kjede har sendt
- to måter å presentere loggen på - enten ved å skrive ut innholdet i datadasen direkte på websiden, og/eller få dataene representert i form av en graf.

For å få tilgang til de to siste funksjonene, må man logge inn som superbruker med et eget brukernavn/passord. Disse passordene er lagret i en ekstern tekstfil.

4.3 Sikkerhet i SuMobil

Det er mange forskjellige grupper brukere som har tilgang til Global One. Ledelsen ved hovedkontoret til en gitt kjede har normalt tilgang til alle salgsdata, rapporter, osv på Global One Portal. Dette gjør at dem kan sammenligne salgstall i de forskjellige butikkene, kjøpe inn varer til alle butikkene, osv. En annen gruppe er regionsledere. Ofte er kejder delt opp i regioner, for eksempel en vestlandsregion og en østlandsregion. Hver region har sin regionsledelse, og disse ønsker bare rapporter for sin region. En tredje gruppe er butikksjefer i de enkelte butikkene. Disse skal ikke ha tilgang til salgstall relatert til andre butikker i kjeden enn sin egen butikk.

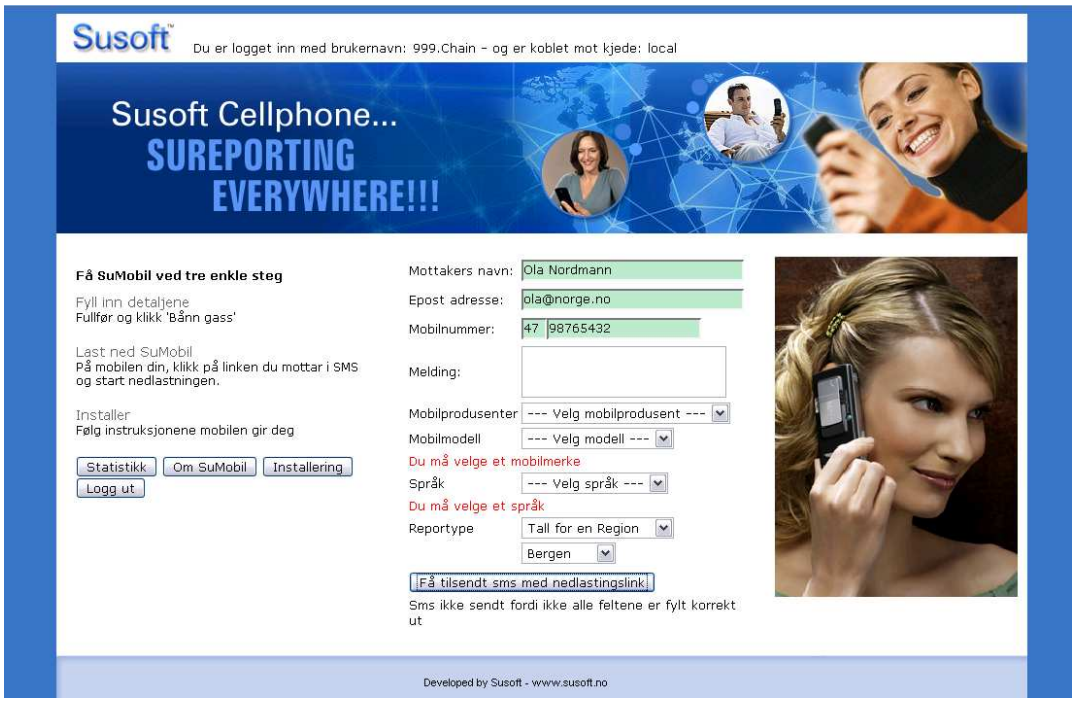

Figure 4.2: Skjermbilde av SuMobil

4.3.1 Sikkerhetsbehov i SuMobil

SuMobil tilbyr et utvalg av rapportene som i dag finnes på Global One Portal. Tjenesten skal tilbys til de samme brukerene som i dag bruker Global One Portal, og har derved samme behovet for sikkerhet. På Global One Portal er det definert ni forskjellige tilgangsnivåer som styrer hvor mye en gitt bruker har tilgang til. Øverst i tilgangspyramiden er en bruker med tilgangsnivå ni. Det er en bruker som bare ansatte på Susoft har tilgang til. Den gir tilgang til absolutt alt i portalen. Nest øverst er tilgangsnivå åtte, som er for kjedeledelsen ved hovedkvarteret. Disse har full tilgang til data, men mangler noe tilgang til noen administrasjonsverktøy som tilgangsnivå ni har. Tilgangsnivå syv er tildelt regionsledelsen. Disse har all tilgang til data relatert til sin region. Tilgangsnivå seks er for butikksjefer og gjør at dem har tilgang til alle data relatert til sin butikk. Tilgangsnivåene under seks kan defineres av kjedeledelsen. For eksempel kan kjedeledelsen opprette en brukergruppe og manuelt tildele denne gruppen tilgang til gitte funksjoner. Denne gruppen kan så for eksempel tildeles tilgangsnivå fem. Bare brukere

med tilgangsnivå seks eller høyere har tilgang til SuMobilløsningen.

4.3.2 Sikkerhetsmodellen implementert i SuMobil

Det har vært ønskelig med et Single Sign On system gjennom hele SuMobilløsningen. Det er tre steder i Susoft system man må authentisere seg for å bruke SuMobilløsningen.

- Logge seg på Global One Portalen for å aksessere SuMobil Portal
- Logge seg på SuMobil Portalen
- Logge seg på SuMobil Portalen fra SuMobil klienten

Målet har vært at brukeren bare skal logge seg på Global One Portalen og deretter få den nødvendige tilgangen på SuMobilløsningen. Det vil bli en kort gjennomgang av hvert av de tre påloggingsstegene og hvordan der er implementert.

Brukere logger seg på Global One Portalen. Brukernavnet og databasenavnet lagres i brukerens Global One Portal sessjon.

Det er to måter å aksessere SuMobil Portalen på. Den første er gjennom Global One Portalen. I menyen vil det da være et menypunkt, SuMobil. Klikker man her vil man bli videresendt til SuMobil Portalen. Den andre måten er å skrive inn URL'en til applikasjonen i en nettleser. Gitt det første scenario har brukeren allerede logget seg inn med brukernavn/passord/database på Global One Portalen. I scenario to må brukeren logge inn direkte på SuMobil Portalen.

Gitt scenario en vil Global One Portal hente brukernavn/passord/database informasjonen fra brukerens sessjon. Denne informasjonen vil brukes til å bygge en URL som man så videresendes til. Ved at Global One Portal sender med påloggingsparameterne til SuMobil Portalen gjør at vi oppnår Single Sign On. Dette gjør først og fremst at brukerne slipper å huske mange forskjellige passord. Det muliggjør også å bruke den allerede eksisterende og avanserte sikkerhetskontrollen i Global One Portalen.

Den største ulempen sett fra et sikkerhetsståsted med både Global One Portalen og SuMobil Portalen er at brukernavn og passord sendes ukryptert mellom klient og tjener. Hvis en tredjepart har tilgang til nettet imellom klient og tjener, kan han med relativt enkle verktøy kunne fange opp brukernavn og passord. En annet moment er hvordan brukernavnet og passordet utveksles mellom Global One Portal og SuMobil Portal. Når man logger på Global One Portalen, gjøres dette gjennom et html skjema (eng:form), og skjemaet sendes til tjeneren ved overføringsmetoden 'post'. Ved bruk av

'post' legges datasettet[29] i kroppen, mens ved bruk av den andre tilgjengelige overingsmetoden, 'get', legges datasettet til i slutten av selve URL'en. Bruk av 'post' anbefales når man sender informasjon som har konsekvenser på tjenersiden, som for eksempel brukerpålogging [28]. Når man videresendes fra Global One Portalen til SuMobil Portalen gjøres dette ved at man legger til brukernavnet og passordet som parametere i URL'en, på samme måte som en 'get' når man sender inn et htmlskjema. En svakhet med 'get' kontra 'post' er at påloggінglinken lagres i nettleserhistorien, og da vil brukernavn og passord være synlig der. Ved bruk av 'post' blir ikke slik informasjon lagret i nettleserhistorien.

Men distinksjonen ovenfor mellom 'post' og 'get' er uvesentlig i forhold til det egentlige problemet - at passord sendes ukryptert til tjeneren. Fordi dagens løsning på Global One Portal er så usikker vil systemet totalt sett ikke blir særlig tryggere om vi kryterer påloggingen til SuMobil Portalen - en eventuell hacker vil jo uansett kunne se passordet når brukeren logger på Global One Portalen.

Men de to neste påloggingsstegene - overføringen av kredentialene fra Global One Portalen til SuMobil Portalen og påloggingen fra SuMobil Klienten til SuMobil portalen - kunne vært hashed. Målet er å oppnå pålogging på en tjener uten at passordet sendes som klartekst over nettet. Dette kan oppnås ved å bruke internettstandarden RFC2617, Digest Access Authentication [27]. Denne bruker som standard MD5 hashing. Kjernen i påloggingssystemet er at tjeneren sender en "utfordring" til klienten. Dette er en unik tekststreng og vil være forskjellig fra hver gang. Klienten vil så konkatinere denne "utfordringen" med passordet som ble tastet inn i nettleseren, og gi denne strengen som input til en MD5 funksjon. MD5 algoritmen vil generere en tilnærmet unik 128 bit tekststreng basert på innstrengen. Denne hashede strengen sendes så til tjeneren. Tjeneren har både "utfordringen" og passordet til brukeren, så den kalkulerer også den hashede verdien av disse konkatinert. Til slutt sammenlignes verdien klienten sendte og verdien tjeneren kalkulerte. Hvis de er like er passordet riktig.

Ved pålogging til SuMobil Portalen fra en nettleser kan krypteringen på klientsiden implementeres ved hjelp av et Javascript bibliotek [14]. Ved pålogging til SuMobil Portalen fra SuMobil klienten kan MD5 algoritmen kjøres i Java på begge sider. Et MD5 bibliotek for JME finnes i URL'en listet opp i referanse [17].

4.4 SuMobil i bruk

SuMobil løsningen er i dag i bruk hos mange av Susoft sine kunder. Vi har brukt Platou Sandnes som pilotkunde. Tilbords sine mellomledere begynne aktivt å bruke SuMobilløsningen i disse dager for å se hvordan salget går ute i butikkene og hvem som selger bra. I tillegg er det planlagt en en mer avansert mobilapplikasjon for mer kraftige pda'er som har større skjermer og tilbyr mer avanserte gui biblioteker.

Kapittel 5

Oppsummering

Når man gjør masteroppgaven ekstern i et firma ligger det innbygget en interessekamp mellom student og arbeidsgiver. Arbeidsgiver ønsker et fullverdig system som umiddelbart kan settes i drift, mens man som student må ta hensyn til at oppgaven må ha elementer av nyskaping og en viss akademisk ballast.

5.1 Oppdragsgivers målsetning

Susoft ønsket å tilby deler av dagens eksisterende rapporteringssystem på web til mobiltelefoner. I det eksisterende systemet har man muligheter for å få generert rapporter på forskjellige formater som Excel, HTML og Word. I begynnelsen av prosjektet brukte undertegnede mye tid på sette seg inn i koden og det eksisterende systemet til Susoft. Dette viste seg å være et blindspor av forskjellige grunner. Den viktigste grunnen var at i halvåret mellom beslutningen om å gjøre masteroppgaven på Susoft og høsten da undertegnede begynte å jobbe med selve oppgaven, sluttet alle de tre utviklerne på Susoft, inkludert utviklingssjefen. Systemet på Susoft er bygget opp av mange egenutviklede systemer. Dette inkluderer eget databaserammeverk, xmlparsere, Java EE rammeverk, sesjonshåndtering, mm. Global One Portalen var for eksempel utviklet ved bruk av mange ikke-standard metoder. I det ligger at Global One Portalen ikke var bygget i noe MVC rammeverk. Dette kompliserte arbeidet med å følge programflyten. I tillegg var systemet fullt av egenutviklede løsninger, for eksempel en egen brukersesjonhåndtering istedenfor å bruke applikasjonsserveres egne sesjonhåndteringsmekaniser. Et annet eksempel er egenutviklede xmlparsere. Alle disse egenutviklede løsningene har sin naturlige forklaring

i at store deler av disse systemene ble utviklet før standarder for dette var utbredt i markedet. Men man forsatte også å bruke disse properitære løsningene etter at dem burde være fjernet til fordel for standardløsninger. Dette byttet ble ikke gjort fordi utviklerne som jobbet på Susoft hadde dyptgående kunnskaper til de eksisterende løsningene, og for dem ville ikke standardmåter å gjøre disse tingene på gjøre hverdagen enklere. Men dette problemet kom frem i lyset da alle sluttet på en gang uten at denne kunnskapen var viderført på en tilfredstillende måte. Ved oppstart av masteroppgaven var undertegnede den personen som satt med mest kunnskap om systemet. Det sa mer om Susoft sin tilstand enn undertegnedes kompetanse på den tiden. Denne situasjonen var hemmende i begynnelsen av SuMobilprosjektet, og gjorde at undertegnede brukte en god stund før han kom skikkelig i gang med oppgaven. I begynnelsen av desember bestemte undertegnede seg for å ikke viderutvikle det eksisterende systemet, men heller å utvikle SuMobilløsningen fra bunnen. Det eneste som ble brukt videre var den eksisterende databasen. Etter den tid ble det mer fart på prosjektet. I dag er SuMobilløsningen i drift hos Susoft og mange kunder har begynt å bruke systemet. Når man starter å rulle ut en så ny løsningen som rapportering på mobil er det vanlig at nye behov vil melde seg etterhvert som brukerne blir vant med løsningen og begynner å se nye behov dem vil ha dekket gjennom mobiltelefonen. Susoft og undertegnde regner derfor med at SuMobilløsningen vil bli viderutviklet i tiden fremover.

5.2 Undertegnedes målsetning

Undertegnedes målsetning var å lære Java ME utvikling. Utvikling av applikasjoner for mobiltelefoner er i stor vekst, og hvor det ikke eksisterer veletablerte standarder for utvikling slik man kjenner fra Java SE og Java EE utvikling. Målsetningen var derfor å lære særegenhetene til Java ME og hvordan man utvikler Java ME applikasjoner av ikke-triviel kompleksitet og størrelse. Gjennom SuMobilprosjektet har undertegnede ikke bare fått innblikk i Java ME utvikling og utfordringer rundt dette, men også innblikk i hvordan bygge videre på eksisterende system og utfordringer rundt dette.

Bibliography

[1] Apache. Log4j logging. Tilgjengelig fra: http://logging.apache.org/log4j.

[2] Jamie Flournoy. write once, be disappointed everywhere.

[3] Jim Hugunin.

[4] Sun Inc. The c virtual machine (cvm).

[5] Sun Inc. Connected device configuration spesification. Tilgjengelig fra: http://java.sun.com/products/cdc/overview.html.

[6] Sun Inc. Connected limited device configuration spesification. Tilgjengelig fra: http://java.sun.com/products/cldc/overview.html.

[7] Sun Inc. The javatm virtual machine specification. Tilgjengelig fra: http://java.sun.com/docs/books/jvms/secondedition/html/VMSpecTOC.doc.html.

[8] Sun Inc. The k virtual machine (kvm). Tilgjengelig fra: http://java.sun.com/products/cldc/wp.

[9] Sun Inc. Mobile information device profile. Tilgjengelig fra: http://java.sun.com/products/midp/overview.html.

[10] Sun Inc. Sun community source licensing. Tilgjengelig fra: http://www.sun.com/software/communitysource/j2me/cldc.

[11] Sun Inc. Web developer resources for java ee. Tilgjengelig fra: http://java.sun.com/javaee/overview/webdev.jsp.

[12] Sun Inc. Which apis come from the j2se platform. Tilgjengelig fra: http://developers.sun.com/mobility/midp/articles/api.

[13] Javaworld. Object finalization and cleanup. Tilgjengelig fra: http://www.javaworld.com/jw-06-1998/jw-06-techniques.html.

[14] Paul Johnston. Digest access authentication javascript. Tilgjengelig fra: http://pajhome.org.uk/crypt/md5/.

[15] Microsoft. Windows update. Tilgjengelig fra: http://windowsupdate.microsoft.com.

[16] Nokia. Nokia regnskap. Tilgjengelig fra: http://www.nokia.com/2007/Q4/index.html.

[17] The Legion of the Bouncy Castle. *Legion of the Bouncy Castle Java cryptography APIs*.

[18] Christen Krogh PCWorld. Opera officially launches mini browser. Tilgjengelig fra: http://www.pcworld.com/article/id,124473-page,1/article.html.

[19] Java Community Process. Java community process. Tilgjengelig fra: http://jcp.org/en/home/index.

[20] Proguard. Obfuskator. Tilgjengelig fra: http://proguard.sourceforge.net.

[21] Govind Seshadri. Understanding javaserver pages model 2 architecture. Tilgjengelig fra: http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html.

[22] Opera Software. Opera link. Tilgjengelig fra: http://link.opera.com/.

[23] Opera Software. The phone factor. Tilgjengelig fra: http://dev.opera.com/articles/view/the-phone-factor-2.

[24] Opera Software. Size and compression. Tilgjengelig fra: http://www.operamini.com/features.

[25] Opera Software. Designing for opera mini, 2008. Tilgjengelig fra: http://dev.opera.com/articles/view/designing-with-opera-mini-in-mind.

[26] Virksomhetsportalen. Hva er single sign-on ? Tilgjengelig fra: http://www.virksomhetsportalen.no/hva-er-single-sign-on-.365859-32039.html.

[27] W3C. Digest access authentication. Tilgjengelig fra: http://www.w3.org/Protocols/rfc2069/rfc2069.

[28] W3C. Form submission. Tilgjengelig fra: http://www.w3.org/TR/html401/interact/forms.html.

[29] W3C. Processing form data. Tilgjengelig fra: http://www.w3.org/TR/html401/interact/forms.html.

[30] James White and David Hemphill. *Java in small things*. Manning Publications, 2002.

[31] Wikipedia. ibatis. Tilgjengelig fra: http://en.wikipedia.org/wiki/IBATIS.

[32] Wikipedia. Plain old java object. Tilgjengelig fra: http://en.wikipedia.org/wiki/PlainOldJavaObject.

[33] Wikipedia. Wap. Tilgjengelig fra: http://www.pcworld.com/article/id,124473-page,1/article.html.

[34] Wikipedia. Weak reference. Tilgjengelig fra: http://en.wikipedia.org/wiki/Weakreference.

[35] Tira Wireless.

[36] Tira Wireless. New jumplets. Tilgjengelig fra: https://gomobile.tirawireless.com/xwiki/bin/view/Jumplets.