NTNU
Det skapende universitet

Intelligent Helper

en agentbasert kontekstsensitiv hjelper for mobilen

Pål M. Bakkejord

Master i informatikk
Oppgaven levert: August 2007
Hovedveileder: Asbjørn Thomassen, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Sammendrag

Lokasjon som kontekstinformasjon blir benyttet stadig mer. Den har vist sin nytteverdi gjennom den mengde av teknologier og applikasjoner, som baserer seg på denne informasjonen. Denne typen applikasjoner er ofte brukersentrert, og gir økt funksjonalitet til en bruker.

Presentasjon av en brukers lokasjon, varierer ut fra forskjellige behov og formål. Fordelene og nytteverdien denne informasjonen gir, er i de fleste tilfeller åpenbar.

Arbeidet i denne oppgaven ser på hvordan kontekst kan benyttes for å gi assistanse til brukere, med fokus på besøkende og ansatte ved Norges Teknisk-Naturvitenskapelige Universitet (NTNU), i *Den Intelligente Korridor*.

Hvordan brukermodeller kan benyttes for å gi økt funksjonalitet for brukere av systemet blir utforsket, og et løsningsforslag for hvordan dette kan gjøres blir presentert. Brukermodellene gir et forslag til hvordan informasjon kan innhentes og presenteres for brukere, ut fra hvilke interesser de har.

Arbeidet har basert seg på agentrammeverket Java Agent DEvelopment Framework (JADE), og design av systemet blir gjort ved å følge en ny metode for utvikling av multi-agentsystem. Design av et system blir presentert, hvor alle funksjoner i systemet er basert på agenter, og interaksjonen mellom disse.

Arbeidet avsluttet med en prototyp av det designede systemet, og en demonstrasjon av systemet viser hvordan det kan benyttes.

Forord

Denne rapporten dokumenterer arbeidet på min masteroppgave ved Institutt for Datateknikk og Informasjonsvitenskap (IDI), NTNU. Arbeidet ble utført i perioden høst 2006 til vår 2007.

Jeg ønsker å takke min veileder, Amanuensis Asbjørn Thomassen, for viktige innspill under arbeidet med oppgaven. Videre vil jeg takke Christian Laverton, min romkamerat, kamerat og fly-buddy. Hans innspill var uvurderlige ettersom oppgavens frist krøp nærmere. Mest av alt vil jeg rette en stor takk til min Ida. Hun har gått gjennom siste del av svangerskapet som alenemor, og vårt andre barn er ventet den 6. august.

Trondheim, 1. august, 2007

_____-

Pål Moum Bakkejord

Innhold

1 Introduksjon **1**

1.1 Bakgrunn og motivasjon 1

1.2 Mål 2

1.2.1 Delmål 3

1.3 Forskningsmetode 3

1.4 Struktur 3

2 Problemutdypelse **5**

2.1 Assistanse 5

2.2 Funksjonalitet for ansatte 6

2.2.1 Informasjonsflyt 6

2.3 Scenario 7

2.3.1 Scenario 1 7

2.3.2 Scenario 2 8

2.3.3 Scenario 3 8

3 Bakgrunn **9**

3.1 Kontekst 9

3.2 Kontekstparametere 10

3.2.1 Tid 10

3.2.2 Lokasjon 11

3.3 Allestedsnærværende og gjennomtrengde 21

3.3.1 Allestedsnærværende 21

3.3.2 Gjennomtrengende 21

3.4 Kontekstsensitive applikasjoner 23

3.4.1 Context Toolkit 24
3.4.2 SmartReminder 25
3.4.3 ContextPhone 26
3.4.4 The Nidaros Framework 28
3.4.5 Klassifisering av kontekstsensitive system 30
3.4.6 Utviklingsteknologi 30
3.5 Brukermodellering 35
3.5.1 Brukerdimensjoner 35
3.5.2 Stereotyper 37

4 Agent **38**
4.1 Definisjon 38
4.2 Agenttyper 39
4.3 Multiagentsystem 40
4.4 The Foundation for Intelligent Physical Agents 42
4.5 Agentkommunikasjon 43
4.5.1 FIPA-ACL meldinger 44
4.5.2 Interaksjonsprotokoller 45
4.5.3 Communicative Acts 47
4.5.4 Content Language 48
4.6 Ontologi 48
4.6.1 Representasjon 48
4.7 Java Agent DEvelopment Framework 49
4.7.1 Java Agent DEvelopment Framework kjernen 49
4.7.2 Agent Management Specification 50
4.7.3 Agent Management Reference 50
4.7.4 Kommunikasjonsmodell 52
4.7.5 Tilstander 52
4.7.6 JADE GUI 53
4.7.7 Light Extensible Agent Platform 54

5 Intelligent Helper **59**

5.1 Metode 59
5.1.1 Unified Modelling Language 60
5.2 Designvalg 60
5.2.1 Funksjonalitet 62
5.3 Analyse & Kravspesifikasjon 62
5.3.1 Steg 1 - Bruksmønsterdiagram 63
5.3.2 Kravspesifikasjon 71
5.3.3 Steg 2 - Agent identifikasjon 72
5.3.4 Steg 3 - Ansvarsidentifikasjon 74
5.3.5 Steg 4 - Identifisering av relasjoner 76
5.3.6 Steg 5 - Agentraffinering 76
5.3.7 Steg 6 - Agent utplassering 78
5.4 Design 78
5.4.1 Steg 1 - Agent splitting/fusjon/omdøping 79
5.4.2 Steg 2 - Interaksjons-spesifikasjon 80
5.4.3 Steg 3 - Definisjon av Ad-Hoc interaksjonprotokoller . . . 82
5.4.4 Steg 4 - Message Templates 82
5.4.5 Steg 5 - Beskrivelse for registrering/søk 82
5.4.6 Steg 6 - Interaksjon med agentressurser 82
5.4.7 Steg 7 - Agent-Bruker interaksjon 83
5.4.8 Steg 8 - Indre agent oppførsel 84
5.4.9 Steg 9 - Definere en ontologi 87
5.4.10 Steg 10 - Valg av innholdsspråk 88

6 Implementasjon 90

6.1 Utviklingsverktøy og programmeringsspråk 90
6.1.1 Programmeringsspråk 90
6.1.2 Protégé 91
6.2 Ontologi 91
6.3 Server 91
6.3.1 Administrasjon av tjenester 91
6.3.2 LokaliseringsAgent 93

6.3.3 BrukerManagerAgent 96
6.3.4 KalenderAgent & BrukerModellAgent 97
6.4 Mobil 98

7 Demonstrasjon 104
7.1 Scenario 104
7.1.1 Besøkende 104
7.1.2 Registrerte brukere 108
7.2 Evaluering av design og implementasjon 109
7.2.1 Oppnåelse av funksjonelle krav 110
7.2.2 Oppnåelse av ikke-funksjonelle krav 112
7.2.3 Styrker & svakheter 112
7.2.4 Mulige forbedringer 113

8 Konklusjon 114
8.1 Måloppnåelse 114
8.1.1 Delmål 1 114
8.1.2 Delmål 2 115
8.1.3 Delmål 3 115
8.1.4 Delmål 4 115
8.1.5 Delmål 5 115
8.2 Evaluering 116
8.3 Videre arbeid 116

Bibliografi 118

Appendiks 120

Figurer

1.1 Forskningsmetode. 4

3.1 Norge inndelt i UTM soner. 12
3.2 GPS enhet, og en illustrasjon av GPS satelitter i bane rundt jorden. 13
3.3 Time Of Arrival posisjonering i et GSM-nettverk. 14
3.4 En Ekahau T301 Wi-Fi posisjoneringsbrikke. 16
3.5 En Cordis RadioEye™. 17
3.6 Dekningsområdet til Cordis RadioEye™. 18
3.7 Eksempel på XML sendt til Cordis RadioEye™ 19
3.8 Definering av soner i Cordis RadioEye™. 20
3.9 Eksempel på et Extensible Markup Language (XML) dokument. . 20
3.10 Autentisering av XML til Cordis RadioEye™ 21
3.11 In/Out Board applikasjonen laget med Context Toolkit. 25
3.12 ContextContacts: en utvidet kontaktliste. 28
3.13 Arktitekturen til Nidaros Framework lokasjonsserver 29
3.14 J2ME teknologien og relasjon til andre Java ™ (Java) teknologier 32
3.15 Midlet livssyklus 34

4.1 Agent og dens forhold til et miljø 39
4.2 Reduksjon av nettverksbelastning 41
4.3 En FIPA spesifikasjons livssyklus 42
4.4 Agent Communication Language melding etter FIPA-ACL standard. 45
4.5 FIPA Request interaksjonsprotokoll. 46
4.6 Eksempel på en *query-if*, kommunikativ handling. 47
4.7 Respons på *query-if* kommunikativ handling. 47

4.8 JADE intern arkitektur. 52
4.9 FIPA standard: komponenter i kommunikasjonsmodellen. 53
4.10 JADE RMA GUI. 54
4.11 LEAP Internal Transport Protocol. 55
4.12 JADE powered by LEAP kjøremiljø. 56
4.13 JADE powered by LEAP *Stand-alone*. 57
4.14 JADE powered by LEAP *Split*. 57

5.1 Metode for utvikling av JADE multiagentsystem - Analysefasen. . 63
5.2 Brukesmønsterdiagram - Systemets hovedfunksjonalitet. 64
5.3 Brukesmønsterdiagram - Systemets hovedfunksjonalitet. 65
5.4 Bruksmønsterdiagram - Bruker søker Online brukere. 65
5.5 Bruksmønsterdiagram - Bruker viser Online brukere på kart. . . . 66
5.6 Bruksmønsterdiagram - Bruker registrerer en brukerprofil. 67
5.7 Bruksmønsterdiagram - Bruker registrer en ny avtale/hendelse. . . 70
5.8 Bruksmønsterdiagram - Opprette en ny brukermodell. 70
5.9 Agentidentifikasjon 73
5.10 Transformat teknikk for eksterne ressurser i agentsystem. 74
5.11 En modell av Prolog-motor. 74
5.12 Identifisering av relasjoner. 77
5.13 Raffinering av agentdiagram. 78
5.14 Plassering av agentene i systemet. 79
5.15 Steg 5 - Tjeneste registrering og søk hos Directory Facilitator (DF). 83
5.16 Systemets hovedkomponenter. 84

6.1 Kommunikasjon mellom senderagenter og en mottaker. 93
6.2 Klassediagram til LokaliseringsAgent. 94
6.3 Interne klasser i LokaliseringsAgent. 94
6.4 Klassediagram til BrukerManagerAgent. 96
6.5 Interne klasser i BrukerManagerAgent. 96
6.6 Klassediagram til BrukerAgent. 98
6.7 Klassediagram til BrukerAgent. 99

6.8 Interne klasser i BrukerAgent. 100
6.9 Aktivitetsdiagram for kartvisning. 103

7.1 *Intelligent Helper* velkomstsone. 105
7.2 *Intelligent Helper* login. 105
7.3 *Intelligent Helper* - meny for uregistrerte brukere. 106
7.4 *Intelligent Helper* (IH) - søkefunksjon 106
7.5 IH - søkefunksjon 107
7.6 IH - søkefunksjon 107
7.7 IH guider besøkende til destinasjon. 107
7.8 IH guider besøkende - Steg 2. 108
7.9 IH guider besøkende - Steg 3. 108
7.10 IH - registrering av profil. 109
7.11 IH - varsler mangler i registrering. 109
7.12 IH - valg i kartrepresentasjon. 109
7.13 Søkefunksjon 109
7.14 Scenario 2. 110

1 Registrering av, og søk etter, tjenester hos DF. 134
2 Konsepter i IH. 135
3 Konsepter i IH. 136
4 Agenthandlinger i ontologien for posisjonering i IH. 137
5 Agenthandlinger i ontologien for brukerhåndtering i IH 137
6 Agenthandlinger i ontologien for avtalehåndtering i IH. 138

Tabeller

3.1 Mobilutbredelse i Norge. 22

3.2 Kategorisering av kontekstsensitive system. 30

4.1 FIPA-ACL meldingsstruktur. 45

5.1 Design spørsmål og valg. 61

5.2 Bruksmønster 1: Søk Online brukere. 66

5.3 Bruksmønster 2 - Se Online brukere på kart 67

5.4 Bruksmønster - Bruker registrerer en brukerprofil. 68

5.5 Bruksmønster - Bruker registrerer en ny avtale/hendelse. 69

5.6 Bruksmønster - Opprette en ny brukermodell. 71

5.7 Oversikt over funksjonelle krav. 71

5.8 Oversikt over ikke-funksjonelle krav. 72

5.9 Ansvarstabell. 76

5.10 Interaksjonstabell for BrukerAgent. 81

5.11 Innledende oppførsler. 87

5.12 Konsepter i ontologien. 88

5.13 Predikat i ontologien. 88

5.14 Agenthandling i ontologien. 88

7.1 Oversikt over funksjonelle krav. 111

7.2 Oversikt over ikke-funksjonelle krav. 112

1 Ansvarstabell 122
2 Interaksjonstabell for BrukerAgent 124
3 Interaksjonstabell for LokaliseringsAgent 126
4 Interaksjonstabell for BrukerManagerAgent 128
5 Interaksjonstabell for KalenderAgent 130
6 Interaksjonstabell for BrukerModellAgent 132

Terminologi

AC Agent Container

ACC Agent Communication Channel

ACL Agent Communication Language

AI Agent Identifier

AMR Agent Management Reference

AMS Agent Management System

AP Agent Platform

API Application Programming Interface

Bit *BI*nary Digi*t*

CDC Connected Device Configuration

CIF Content Interchange Format

CL Content Language

CLDC Connected Limited Device Configuration

CORBA Common Object Request Broker Architecture

CRE Cordis RadioEye™

DF Directory Facilitator

DTD Document Type Definition

EPE Ekahau Positioning Engine

FIPA The Foundation for Intelligent Physical Agents

FIPA-ACL FIPA-Agent Communication Language

GHz Gigahertz

GPRS General Packet Radio Service

GPS Global Positioning System

GSM Global System for Mobile Communications

GUI Graphical User Interface

HTML Hyper Text Markup Language

HTTP Hyper Text Transfer Protocol

HTTPS Secure Hyper Text Transfer Protocol

IDI Institutt for Datateknikk og Informasjonsvitenskap

IEEE Institute of Electrical and Electronics Engineers

IIOP Internet Inter-ORB Protocol

IP Interaction Protocol

ITP Internal Transport Protocol

JAD Java Application Descriptor

JADE Java Agent DEvelopment Framework

JADEpL JADE powered by LEAP

JAR *J*ava *Ar*chive

JDK Java Development Kit

JRE Java Runtime Environment

JSR Java Specification Request

JVM Java Virtual Machine

J2ME Java Platform, Micro Edition

J2SE Java Platform, Standard Edition

J2EE Java Platform, Enterprice Edition

LAN Lokal Area Network

LEAP Light Extensible Agent Platform

MAC Media Access Control

MAS Multi-Agent System

Mbit Megabit

MIDP Mobile Information Device Profile

MMS Multimedia Messaging Service

MTP Message Transport Protocol

MTS Message Transport Service

NTNU Norges Teknisk-Naturvitenskapelige Universitet

OTA Over-the-air

OWL Web Ontology Language

PC Personal Computer

PDA Personal Data Assistent

PKI Public Key Infrastructure

RFID Radio Frequency Identification

RMA Remote Management Agent

RMI Remote Method Invocation

RMS Record Management System

RPC Remote Procedure Call

SDK Software Development Kit

SGML Standard Generalized Markup Language

SMS Short Message Service

SSL Secure Sockets Layer

SUN Sun Developer Network

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol/Internet Protocol

UML Unified Modelling Language

USB Universal Serial Bus

UTM Universal Transverse Mercator

VM Virtual Machine

WAP Wireless Application Protocol

WLAN Wireless Local Area Network

WTK Wireless Toolkit

XML Extensible Markup Language

Kapittel 1

Introduksjon

Kontekst er et ord som er vanskelig å definere. Allikevel står det for en informasjonskilde av ytterst nødvendighet og en stor ressurs.

Bruk av kontekstinformasjon blir mer og mer vanlig, og det åpner for mange nye muligheter og arenaer for systemutvikling. Det er ennå mye som ikke er gjort og som kan forbedres. Vi har nå tilgjengelig en mengde teknologi som sammen kan assistere brukere i mange av de rutinemessige oppgavene som utføres i løpet av en dag.

Ved å kombinere den teknologien vi har nå i dag, er det mulig å posisjonere objekter og oss selv. Dette gir muligheter som til nå har vært lite benyttet, men som er et "hett" tema innen kunstig intelligens(AI) teknikker.

Hva om Personal Computer (PC)'en akkurat var ferdig med å slå seg på når en ankommer kontoret om morgenen? Eller at kaffemaskinen var ferdig med nytraktet kaffe, selv om en var én time senere til lunsj.

"Intelligent Homes & Buildings" og "Smart Homes" er begrep som omhandler integrasjon av slik teknologi, i private hjem og andre bygg. Der er fokus på tilpassning av hjemmet eller bygget, til brukerene og miljøet rundt. Fokuset er ofte på kontrollering av miljøet som systemene er i, og å forbedre bruken av ressursene. Eksempel på dette er regulering av lys etter lysforhold, regulere temperatur ved å kontrollere den varmen som skal genereres.

1.1 Bakgrunn og motivasjon

Kontekst er et "diffust" begrep, men som er en nødvendighet for forståelsen av en setning eller en hendelse. Denne viktige kilden til informasjon er ofte avgjørende for beslutninger som skal tas og for forståelsen for de ulike situasjoner man befinner seg i.

Ved riktig bruk kan kontekstinformasjon automatisere ulike steg i en prosess. Mye av den informasjonen en bruker må legge manuelt inn i et system, kan

automatiseres ved å trekke informasjonen ut av brukerens kontekst.

Informasjonen som blir viktig, vil være forskjellig i ulike situasjoner. Informasjonen vil være avhengig av hvilket foremål den skal brukes til, og for hvem den skal omhandle.

Den Intelligente Korridor, i 3. etasje i IDI bygget er et miljø hvor det utforskes på forskjellig bruk av AI teknikker. Miljøet der består av mange ansatte, en rekke kontor, møterom, og forskjellige studielokaler. Miljøet her er lite oversiktlig, og inneholder flere bortgjemte rom, hvor blant annet ansatte og studenter befinner seg. Informasjon om hva og hvem som befinner seg her, er plassert i den ene enden av korridoren. Den andre enden inneholder lite informasjon om verken ansatte eller rom, annet enn at det er angitt navn på dørene.

Dette kunne med fordel vært mer brukervennlig, for både besøkende og ansatte ved Den Intelligente Korridor. Hvordan kan en da benytte seg av "kontekstinformasjon" for å gi denne assistansen?

Disse spørsmålene er ikke begrenset til Den Intelligente Korridor, men omhandler alle miljø hvor forholdene er uoversiktlige. Dette er ofte et problem ved institusjoner slik som NTNU og andre store virksomheter.

For å bedre denne situasjonen kan informasjon som lokasjon benyttes i større grad. Den Intelligente Korridor er et område for testing og utvikling av State-of-the-Art (SOTA) teknologi, og der befinner også Cordis RadioEye™ (CRE) seg. Dette verktøyet angir posisjoner til Wireless Local Area Network (WLAN) klienter, og gir da kontekstinformasjonen lokasjon. Utnyttelsen av CRE i dette miljøet kan være stort, og det vil være forskjellige måter å benytte seg av den.

Ordet *agent* har forskjellige betydninger, men i det domenet som AI utgjør handler det som oftest om en programvarekomponent. Hvordan designer man et system bygget opp av agenter? Og hvordan kan agenter benyttes for å løse det uoversiktelige miljøet som Den Intelligente Korridor utgjør.

1.2 Mål

Hovedmålet for oppgaven er å designe et system som skal være kontekstsensitivt og tilby funksjonalitet til besøkende og ansatte. Systemet skal jobbe for å være allestedsnærværende(ubiquitous) og sømløst integrert med omgivende teknologi. En prototyp av systemet skal utvikles og evalueres. Dette innebærer også oppsett og konfigurasjon av nødvendig maskinvare.

Et fremtredenede element i oppgaven er bruk av lokasjon. Posisjoneringsteknologien skal komme i form av Radionor Communications sitt CRE, som gir posisjoner i et lokalt koordinatsystem. Det skal derfor defineres et lokalt koordinatsystem for Den Intelligente Korridor, i 3. etasje i IDI-bygget, hvor det er hengt opp 2 radioøyer, og de skal begge konfigureres inn i det lokale koordinatsystemet.

For at systemet skal være State-of-the-Art skal det benyttes en agentarkitektur

som utfører de forskjellige oppgavene systemet skal utføre, sammen med Radionor sitt CRE. Systemet skal fungere opp mot mobile enheter med støtte for Java teknologi. Java Agent DEvelopment Framework skal være utviklingsrammeverket for oppgaven.

En ny metode for uvikling av multi-agentsystem skal benyttes under analyse og design av systemet. Dette er for å få en hjelp til utviklingen, samt å prøve ut den nye metoden.

Bruk av brukermodeller skal undersøkes og det skal foreslås en utnyttelse av brukermodeller for å gi ytterligere forbedringer i systemet.

1.2.1 Delmål

Hovedmålet for oppgaven er å designe et system basert på agenter, som skal være kontekstsensitivt og tilby funksjonalitet til besøkende og ansatte.

Målet er videre delt opp i delmål:

- **Delmål 1**: Definere krav til systemet som skal designes
- **Delmål 2**: Design av et agentsystem, basert på agentrammeverket JADE.
- **Delmål 3**: Benytte kontekstinformasjon i systemet.
- **Delmål 4**: Benytte brukermodeller i systemet.
- **Delmål 5**: Implementere, demonstrere, og evaluere en prototyp av det modellerte systemet.

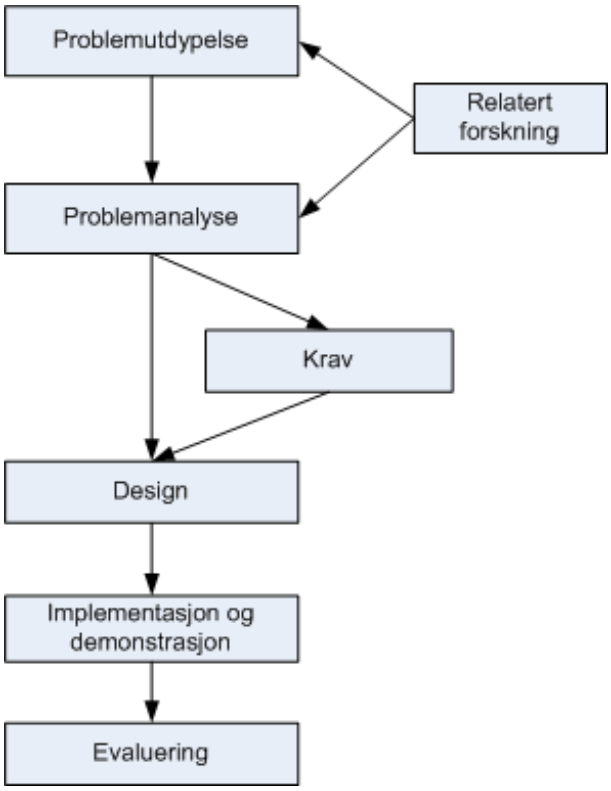
1.3 Forskningsmetode

Forskningsmetoden benyttet for oppgaven er som illustrert i Figur 1.1. I første steg utdypes problemet, og problemstillingen , støttet opp av relatert forskning. Neste steg gjør en grundigere analysere av problemet, også støttet opp av relatert forskning, for å øke forståelsen for problemet. Denne bakgrunnen benyttes i det neste steget, hvor det gjøres analyse og design av systemet. Fjerde steg gjør en implementasjon av en prototyp av det designede systemet. Femte og avsluttende steg er evaluering av det modellerte systemet, og ser på hvordan dette systemet kan være en løsning på problemstillingen for oppgaven.

1.4 Struktur

Oppgavens struktur består av 6 kapittel som er satt opp på følgende vis:

- 2. **Problemutdypelse** - ser på ulike elementer i problemstillingen. Scenario belyser noen aspekter ved problemstillingen.



Figur 1.1: Forskningsmetode.

- 3. **Bakgrunn** - ser på de viktigste konseptene for oppgaven. Kapittelet støtter dette opp med presentasjon av State-of-the-Art system.
- **4. Agent** - ser på definisjonen av en agent. Her presenteres også The Foundation for Intelligent Physical Agents og agentsystemetJava Agent DEvelopment Framework.
- **5. Intelligent Helper** - presenterer kravspesifikasjon, analyse og design av *Intelligent Helper*.
- **6. Implementasjon** - viser implementering av systemet og den funksjonaliteten som er i prototypen.
- **7. Demonstrasjon** - demonstrerer den implementerte prototypen.
- **8. Konklusjon** - evaluerer design og implementasjon.

Kapittel 2

Problemutdypelse

Dette kapittelet ser på hvilke problem systemet skal løse. Det tar opp ulike aspekter med den funksjonalitet som skal tilbys brukere og besøkende og hva som må undersøkes av relatert forskning.

Seksjon 2.1 ser på hva en besøkende trenger for å få best mulig assistanse.

Seksjon 2.2 ser hvilken funksjonalitet som kan være nyttig for ansatte.

Seksjon 2.3 belyser problemstillingen gjennoom scenarioer.

2.1 Assistanse

Det er i problembeskrivelsen fastlagt at posisjonering av brukere skal komme i fra CRE. Men hvordan skal en benytte denne teknologien for å gi assistanse til noen som kommer inn i Den Intelligente Korridor?

Besøkende som kommer inn i Den Intelligente Korridor og søker personer eller et lokale, vil trenge hjelp i form av informasjon om hvor disse befinner seg. Dette bringer opp et spørsmål om hvordan en skal *presentere* den hjelpen systemet tilbyr i Den Intelligente Korridor, slik at best mulig *assistanse* blir gitt.

Direktiver til den ansatte eller det lokale som søkes, kan presenteres i tekstet form på en skjerm, eller ved tale over en høyttaler. Hvis den besøkende søker en ansatt vil denne informasjon sannsynligvis ikke noen ny informasjon, da brukeren trolig kjenner navnet, og alle dører er merket med navn.

En annen mulighet er at det vises et kart over Den Intelligente Korridor på en skjerm ved inngangen til korridoren, slik at brukeren kan se på kartet og deretter bevege seg i riktig retning. Dette kartet kan vise hvor kontoret til brukeren er, eller angi posisjoner til brukeren.

Mennesker er ikke så flinke på å huske informasjon som de har fått kun en kort presentasjon av. Korttidshukommelsen, eller arbeidshukommelsen holder bare på informasjonen i 20-30 sekunder før den begynner å gå tapt [31]. Hvis brukeren må

huske informasjon som blir presentert ved inngangen til Den Intelligente Korridor og frem til destinasjon, kan dette bli et irritasjonsmoment.

Begge mulighetene gir brukeren informasjon om hvordan han kan gå frem for å finne den ansatte eller riktig lokale. Men er hvordan skal en presentere informasjonen, slik at den besøkende hele tiden er trygg på at han kommer frem til sitt mål? Vil en mer mer dynamisk, grafisk representasjon gjøre det lettere for en besøkende ved Den Intelligente Korridor?

2.2 Funksjonalitet for ansatte

Hvilke funksjoner kan øke produktivitet og gi assistanse for ansatte i Den Intelligente Korridor i løpet av en arbeidsdag? Hvilke behov er det og hvordan kan de løses?

Dagens hverdag er fyllt opp av informasjon, i mange former, og i et stort antall. Informasjonen kan bli så påtrengende at den ofte fremstår mer som en disktraksjon, enn det som var intensjonen.

2.2.1 Informasjonsflyt

Informasjon er ikke bare viktig for å skal utføre oppgaver i det daglige, men også en kilde til inspirasjon og kunnskap. På en institusjon som NTNU er dette noe det er en overflod av, både inspirasjon, kunnskap og ikke minst, **informasjon**.

Slike institusjoner vokser og forandrer seg over tid og noen ganger veldig raskt. Under slike perioder vil noen systemer bli utrangert, eller i verste fall bare forbigått av andre. Det man ender opp med at den voksende mengden med informasjon må presenteres i en rekke ulike system, av en rekke ulike mennesker. Da er det ikke alltid like enkelt for brukere å få med seg alt, eller om ikke annet, det som faktisk er viktig og interessant for den enkelte.

Kan informasjonen *finne veien selv*, til de riktige brukerene. En mulighet er å angi for all informasjon hvilke brukere som skal motta den, isteden for å sende all informasjon til samtlige.

Hvis brukererns preferanser kan bestemme hvilken informasjon som skal mottas, vil avsenderen slippe å registrere alle mulige mottakere, det være seg grupper eller enkeltpersoner.

2.2.1.1 Informasjonsfiltrering

Informasjon til de ansatte kan med en fordel *filtreres*, slik at distraksjonene avtar og produktiviteten for den enkelte øker. Filtrering av informasjon kan utføres på forskjellige måter, og med forskjellig nøyaktighet. Med nøyaktighet menes det

her hvordan informasjonen filtreres i forhold til den enkelte bruker, da dette vil oppleves ulikt fra person til person.

For å tilrettelegge for brukere og for at den tilgjengelige informasjon skal presenteres riktig for den enkelte, må brukerens preferanser representeres i systemet.

Hvordan brukeren *representeres* i systemet må undersøkes og hvordan denne informasjonen kan benyttes for å forbedre funksjonalitet. En vanlig måte er å benytte en database hvor brukeren selv, gjennom en applikasjon, lagrer informasjon om seg selv. Dette er en veldig statisk måte å løse problemet på, men det vil gi brukeren en forutsigbar funksjon i et system. En av oppgavens formål er å se på brukermodeller, som en "alternativ" representasjon av brukeren.

2.3 Scenario

I denne seksjonen presenteres og diskuteres tre scenarioer for å belyse problemet, og det gis en kort diskusjon av hendelsen i scenarioet.

2.3.1 Scenario 1

Del 1

Roger ankommer 3. etasje i IDI-bygget, til Den Intelligente Korridor. Roger skal finne frem til Asbjørn, som har et kontor i denne etasjen. Roger er ikke kjent her og han ser ingen andre personer. Roger vet ikke hvor han skal henvende seg for å finne ut av problemet.

Hvordan kan besøkende som ankommer Den Intelligente Korridor få assistanse til å finne frem til den, eller de personer som søkes. Et system som tilbyr tjenester for besøkende kunne gjøre det enklere og den besøkende kan selv finne ut av, og løse problemet der hvor han er.

Del 2

Et system merker Roger, og annonserer sine tjenester på en skjerm ved inngangen til Den Intelligente Korridor. En vennlig stemme forteller Roger om at han har ankommet "Den Intelligente Korridor" og at han kan få hjelp. Roger blir presenter en rekke valg, hvor en oversikt over Den Intelligente Korridor er ett av dem. Roger velger å se på oversikten, som også viser piler fra en liste over navnene til ansatte, og bort til deres kontor. Det er en rotete oversikt og det tar tid for Roger å se hvor han skal finne kontoret.

Hvordan skal assistansen som tilbys besøkende representeres for å gi best mulig assistanse. Det finnes flere muligheter for å presentere hjelp til en besøkende, hvor informasjon kan presenteres ved hjelp av tekst, lyd og bilde eller en annen grafisk representasjon.

Del 3 *Roger finner frem til kontoret, men det viser seg at Asbjørn ikke er der,*

og Roger vet ikke helt om han bør lete, eller komme tilbake senere.

Informasjon som indikerer hvor ansatte er, eller hva de er opptatt med, kan gi den informasjonen besøkende trenger for å ta denne avgjørelsen.

Hvilken informasjon om brukeren som skal benyttes, vil være avhengig av hvilken muligheter det er for innhenting av informasjonen. Hvilken nytteverdi vil forskjellige typer informasjon tilbringe. En status, slik som "ute-til-lunsj" eller "møte", kan legges inn av den ansatte selv og indikere hvorvidt han/hun er tilgjengelig.

2.3.2 Scenario 2

Rita skal opp til Morten som skriver sin hovedoppgave og sitter ved en studieplass i Den Intelligente Korridor.

Morten har vært på plassen sin hver dag den siste uken, og nå skal Rita ta lunsj med Morten. Rita kommer frem til studieplassen til Morten, men han er ikke der. Det er heller ingen andre i området, så det er ingen å spørre om hvor Morten befinner seg.

Rita går bort til skjermen som er plassert ute i korridoren, og søker på Morten. Når Rita finner Morten i systemet, har han verken lagt igjen beskjed om hvor han er, eller endret sin status til "ute-til-lunsj". Rita bestemmer seg for å legge igjen en beskjed til Morten, på brukersiden hans i systemet. Mens hun avslutter beskjeden skifter statusen til Morten tekst, og det står nå " Opptatt (forelesning(ITV200))".

Rita får hentet opp en oversikt over området hvor forelesningen foregår, ser en stund på oversikten og forlater Den Intelligente Korridor, i en foreløpig riktig retning til ITV200.

En grafisk oversikt, vil i de fleste tilfeller være tilfredsstillende. Hvis avstanden til destinasjonen derimot er stor, eller at det er momenter som gjør at det vil ta tid å nå dit, vil den besøkende etter hvert, sannsynligvis, føle føle seg usikker.

2.3.3 Scenario 3

Harald går gjennom eposten sin for å planlegge hvilke møter og forelesninger han skal delta på denne uken. Han får hver uke en stor mengde poster, og det kan være vanskelig og tidkrevende å skille ut de som er av interesse. En stor del av epostene omhandler møter og forelesninger som ligger utenfor hans fagfelt. Hadde han bare hatt et verktøy for å filtrere ut interessante eposter.

Som tidligere nevnt får vi mye informasjon, fra mange hold. Denne informasjonen kan være forstyrrende, og er ikke alltid nødvendig i alle situasjoner.

Kapittel 3

Bakgrunn

Dette kapittelet ser på begreper som er viktige for denne oppgaven.

Seksjon 3.1 ser på hva kontekst er og hvordan det har blitt definert i forskjellige sammenhenger.

Seksjon 3.3 ser på begrepene *allestedsnærværende* og *gjennomtrengende*.

Seksjon 3.4 ser på hvordan applikasjoner kan benytte seg av kontekst og viser SOTA system som har gjort dette tidligere.

Seksjon 3.5 tar for seg brukermodellering.

3.1 Kontekst

Hva er kontekst og hvordan benyttes det? Kontekst blir ofte nevnt i sammenheng med språk, hvor kontekst kan være den omgivende tekst eller tale rundt et ord. Et ord kan ha forskjellig betydning etter hvilken sammenheng det benyttes i.

Mennesker benytter kontekst i den fysiske sammenhengen når en hendelse forklares. Her vil det ofte være snakk om tid, plassering og andre aspekter som vil gi viktig forklarende informasjon om hvordan omgivelsene og omstendighetene rundt hendelsen var. Denne informasjonen er noe mennesker bruker uten å tenke for mye over det, men som er essensiell for forståelsen av situasjonen.

Mange forsøk har blitt gjort for å gi en definisjon på hva kontekst er, men det ser ut til å være vanskelig å gi en klar definisjon som holder i alle situasjoner. Anind K. Dey og Gregory D. Abowd kom i [9] med en definisjon.

> Kontekst er enhver informasjon som kan bli brukt til å karakterisere situasjonen til en entitet. En entitet er en person, et sted, eller objekt som regnes relevant til samhandlingen mellom en bruker og en applikasjon, inkludert brukeren og applikasjonen.

Dey og Abowd drøfter tidligere definisjoner for å kategorisere ulike typer kon-

tekstinformasjon. De kommer frem til fire primærtyper, som er:

- **lokasjon**
- **identitet**
- **aktivitet**
- **tid**.

De deler videre konteksten opp i to nivå:

- **primær**; Første nivå inneholder de fire primærtypene.
- **sekundær**; Informasjon som for eksempel en e-post adresse eller telefonnummeret til en person. Den sekundære informasjonen innhentes ved å benytte den primære som en indeks. De mener kategoriseringen vil hjelpe utviklere av kontekstsensitive applikasjoner til å finne den mest nyttige kontekstinformasjonen for sine applikasjoner.

3.2 Kontekstparametere

Kontekst en informasjonskilde, med beskrivende info for alle situasjoner. Kontekst består av mange forskjellige segmenter, eller forskjellige *parametere*.

3.2.1 Tid

Riktig informasjon til riktig tid er en velkjent strofe etter hvert som det har blitt en overflod av informasjon. Som Edward O. Wilson sa i [42]:

> We are drowning in information, while starving for wisdom. The world henceforth will be run by synthesizers, people able to put together the right information at the right time, think critically about it, and make important choices wisely.

Tid er et begrep som har vist seg vanskelig å plassere, og det har vært forsøkt definert mange ganger opp gjennom tiden. Betydningen av tid har vært diskutert og svært forskjellige meninger om tiden har vært lagt fram.

Teorier om tid er blant annet *Lineær*, hvor tid blir sett på som en bevegelse, med en begynnelse og en konstant bevegelse fremover. En annen teori er tiden som en *sirkulær* enhet, hvor fremtiden blir til fortid. En tredje måte å er *kronologisk*, hvor tiden blir sett på i forhold til *fortid*, og hvor hendelser i fortiden kan plasseres i en sekvens.

Tid blir gitt som en standard av *International System of Units(SI)* [1] og måler tiden som en måleenhet, oppdelt i enheter som sekunder, minutter, timer og dager.

[1] http://physics.nist.gov/cuu/Units/index.html

3.2.2 Lokasjon

Lokasjon som kontekstparameter har lik tid, i de fleste situasjoner et meget sterkt utnyttelsespotensiale og er ofte en sentral informasjonskilde. Det kan ofte være avgjørende og veiledende nok til å ta beslutninger eller utføre proaktive handlinger på vegne av en bruker.

Når en benytter seg av lokasjon kan det benyttes på ulike måter å representere det på [22]:

- **Geografisk posisjon**
  - **Relativ**; Angir posisjon ut fra distansen fra et referansepunkt og til punktet som skal posisjoneres.
  - **Absolutt**; Posisjonen angis i grader i et koordinatsystem.
- **Beskrivende**; Angi lokasjonen ved å beskrive den med informasjon, for eksempel en adresse.
- **Topologisk**; Lokasjonen blir angitt etter hvordan den er koblet sammen i en topologi. For eksempel i et datanettverk kan en maskin angis med hva den er knyttet til i nettverket.

Geografiske absolutte posisjoner kan angis på ulike måter, og i forskjellige koordinatsystem. To velkjente globale koordinatsystem er lengde- og breddegrader, og Universal Transverse Mercator (UTM).

3.2.2.1 Lengde- og breddegrader

Lengde- og breddegrader deler jorden opp i et tenkt rutenett, med meridianer(lengdegrader) som går fra nord til sør og parallellsirkler(breddegrader) som går fra øst til vest. Meridianene er parallelle med en nullmeridian, og måler distansen øst eller vest for denne. Nullmeridianen er satt til å gå gjennom Greenwich i England. På den motsatte siden av jorden er det 180° øst eller vest. Breddegradene er parallelle med ekvator og gir distansen nord eller sør for ekvatorlinjen. Breddegradene oppgis som 0° ved ekvator til 90° nord(nordpolen), og 90°sør(sørpolen). For å gi en mer nøyaktig posisjonering deler man gradene inn i 60 minutter og et minutt deles inn i 60 sekunder.

Når man oppgir en posisjon i lengde- og breddegrader oppgis breddegraden først og spesifiserer posisjonen nord eller sør for ekvator. Deretter oppgis lengdegraden, øst eller vest for nullmeridianen.

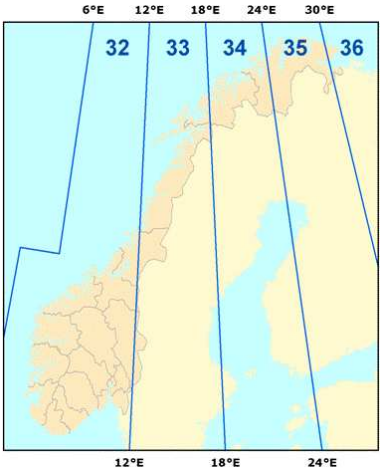
3.2.2.2 Universal Transverse Mercator

UTM er en måte å prosjisere et gradnett på jorden i en sylinderform, hvor sylinderen er liggende på tvers(transversal) [33]. Koordinatsystemet UTM deler jorden

inn i soner og belter. Det defineres 60 soner, hvor sonene deles inn med tangeringsmeridianer på 6° mellom seg. Hver sone har en sentralmeridian slik som for eksempel sone 32, som har 9° som sin sentralmeridian [33].

Beltene er parallellsirkler som har fått tildelt bokstaver fra **C** til **X**, men hvor **I** og **O** er utelatt. Hvert belte har en bredde på 8° og dekker området fra 80° sørlig bredde til 84° nordlig bredde. En sone i UTM vil bli angitt med sone nummer og bokstav, og få betegnelsen sonebelte.

Et eksempel på en bruk UTM vises i Figur 3.1 [2]. Her vises hvordan Norge dekkes av syv sonebelter, som er 32V, 33V, 32W, 33W, 34W, 35W og 36W. Trondheim ligger i sone 32V.



Figur 3.1: Norge inndelt i UTM soner.

Det finnes mange ulike teknologier for posisjonering av trådløse enheter. I seksjonen under presenteres et lite utdrag av de mange typene som finnes.

Direkte posisjonering

Direkte posisjonering er en metode som gir en direkte posisjon i et koordinatsystem over jorden. Noen teknologier som benytter seg av en slik posisjonering er Global Positioning System (GPS), Global System for Mobile Communications (GSM) og WLAN.

Global Positioning System

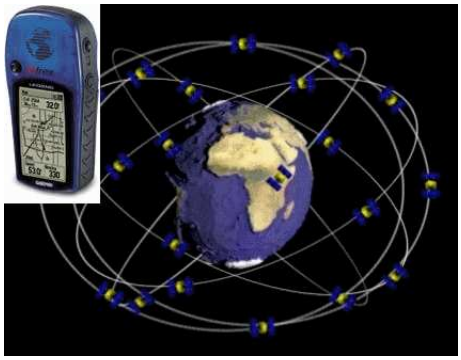
GPS er den mest kjente posisjoneringsteknologien og blir benyttet av ulike grupper til forskjellige formål. GPS ble utviklet av *U.S. Department of Defence* som en militærteknologi, men ble åpnet for allmennheten og kommersielt bruk i 1980 årene. Den er nå godt etablert i markedet og blir nå benyttet mye til veivisnings- og kartapplikasjoner.

[2] Figur er hentet fra http://www.mesterkart.no/images/UTM-soner.gif

GPS fungerer ved at 24 satellitter rundt jorden, samarbeider med kontrollstasjoner for å gi mest mulig eksakt data til en GPS-mottaker. Dataene bestemmer lokasjonen til GPS-mottakeren i fire dimensjoner, som er lengdegrad, breddegrad, høyde og tid. For å bestemme lokasjonen må GPS-mottakeren ha kontakt med minst 4 satelitter. Det er mulig å benytte kún tre satelitter, men da blir posisjonen kun angitt i to dimensjoner. Den fjerde satelitten utvider med en dimensjon og angir posisjonen også i høyde.

Det finnes ulike typer GPS som gir litt forskjellig nøyaktighet, men det spenner fra rundt 15 meter og helt ned til 3-5 meter. Dette er svært nøyaktig for et system som fungerer over hele verden og under alle forhold, såfremt man har kontakt med minimum 3 satelitter.

GPS begrenser seg ofte til utendørsbruk og GPS-enheter mister ofte kontakten hvis man har en obstruksjon over GPS-mottakeren. GPS-mottakere har tidligere vært begrenset til dedikerte enheter som oppgav posisjonen til brukeren og kanskje med et grafisk kart. Figur 3.2 viser satelittene i bane rundt jorden og en GPS mottaker [3].



Figur 3.2: GPS enhet, og en illustrasjon av GPS satelitter i bane rundt jorden.

GPS har etter hvert blitt mer og mer integrert i andre enheter som mobiltelefoner, bærbare PC'er og kjøretøy. Det er også et stort antall forskjellige GPS-mottakere med forskjellige tilkoblingsmuligheter, som Bluetooth eller Universal Serial Bus (USB), laget for å integreres med nåværende mobiler og bærbare PC'er.

3.2.2.3 Global System for Mobile Communications

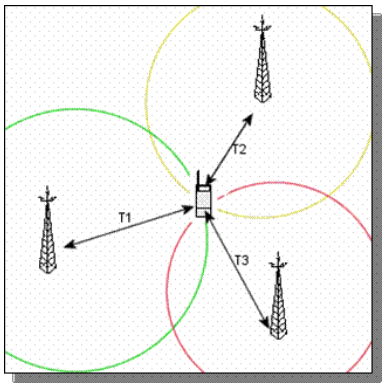
GSM posisjonering fungerer ved at mobiltelefoner kobler seg opp til en av de mange basestasjonene som er plassert rundt omkring for å gi mobilbrukere god dekning. Basestasjons dekning er kalt en *celle*, og posisjonen blir den cellen mobilen befinner seg i. Ved å benytte én eller flere basestasjoner kan man bestemme posisjonen med nøyaktighet som varierer fra rundt 1000 meter og ned til 60 meter.

Det finnes ulike typer posisjoneringsteknikker som benytter seg av det eksisterende GSM nettverket.

[3] GPS mottakeren er en typen *Garmin eTrex Legend GPS*

CELL-ID er en teknologi som benytter benytter én basestasjon for posisjonering av GSM-enheten. Med hvilken nøyaktighet man kan anslå GSM-enheten bestemmes utifra hvor stor cellen er. En forbedring i nøyaktigheten kan oppnås ved å benytte flere alternative celler, såkalt *Enhanced CELL-ID*. Fordelene til denne metoden er at det er billig da det ikke trengs ny hardware for å utnytte den.

Time Of Arrival benytter seg av minimum tre basestasjoner for å gjøre en triangulær posisjonering av en GSM-enhet. Den kalkulerer posisjoner utifra hvor lang tid det tar for signalene å gå fra en mobil til tre ulike stasjoner. Denne teknologien gir bedre nøyaktighet enn bruken av én enkelt basestasjon, men krever en oppgradering av hardware i mobilnettverket. Figur 3.3 viser denne metoden.



Figur 3.3: Time Of Arrival posisjonering i et GSM-nettverk.

Det finnes også flere andre metoder ved bruk av GSM-nettverk, hvor noen er hybride nettverk med en kombinasjon av et GSM-nettverk og GPS.

Indirekte posisjonering

Andre teknologier vil benytte seg av en *relativ* posisjon, hvor man tar utgangspunkt i den enheten som lokaliserer andre enheter i et eget koordinatsystem. Videre kan da disse plasseres i et av de verdensomspennende koordinatsystemene hvis det er nødvendig. I de følgende seksjonene presenteres noen av disse teknologiene.

3.2.2.4 Bluetooth™

Bluetooth er en kortdistanse radioteknologi som blir mye brukt for å koble sammen bærbare enheter og er en erstatning til kabler [4]. Det er mye brukt på blant annet mobiler, bærbare maskiner, tastatur og mus. Bluetooth teknologien benytter seg av det traffikerte 2.4Gigahertz (GHz) frekvensbåndet, og er derfor utsatt for mye støy. Dette håndteres ved at Bluetooth implementerer en frekvenshopping, som vil si at den skifter mellom 79 forskjellige kanaler. Dette gjøres 1600 ganger i løpet av ét sekund. Bluetooth spesifiserer 3 forskjellige klasser etter

hvilket strømforbruk det er, hvor strømforbruket måles i milliWatt(mW). Rekkevidden for de ulike klassene er som følger:

- **100mW**; gir en teoretisk rekkevidde på 100 meter.
- **25mW**; gir en teoretisk rekkevidde på 20 meter.
- **1mW**; gir en teoretisk rekkevidde på 10 meter.

Posisjonering av Bluetooth enheter vil normal gjøres ved en triangulær posisjonering. Den mest utbredte Bluetooth klassen har en rekkevidde på kun 10 meter, og vil da være den rekkevidden en posisjonering med Bluetooth opererer med. Posisjonering vil da kreve en rekke enheter plassert rundt omkring i det området som skal dekkes, slik at man hele tiden har minst tre forskjellige referansepunkter.

3.2.2.5 Wireless Local Area Network

WLAN teknologien posisjonerer trådløse enheter som benytter seg av 802.11 standarden. Dette er standarden trådløse nettverkskort benytter i dag i bærbare PC'er, PDA'er og nå også et voksende antall mobiler.

802.11 er en familie av Institute of Electrical and Electronics Engineers [4] standarder for trådløse lokalnettverk(WLAN) som ble designet for å utvide 802.3 [5] inn i det trådløse domenet. 802.11 ble først sluppet i 1997 og spesifiserte da to rå datahastigheter på 1 og 2 Megabit/s [6] som ble overført via infrarøde(IR) signaler. 802.11 opererer på 2.4GHz frekvensen som er et ulisensiert frekvensområde, og som derfor er en mye brukt blant trådløsteknologiene. signaler.

WLAN teknologien har vært preget av ustabilitet og har også en mye tregere overføringshastighet enn Lokal Area Network (LAN). **11n** er et tillegg til 802.11 familien, som enda ikke er bekreftet av IEEE som et tillegg til 802.11, men er allerede tatt i bruk og er ute på markedet. 11n skal oppnå en maksimal hastighet på 540 Mbit/s optimalt og ha en typisk dataoverføringshastighet på 200 Mbit/s. Overføringshastigheten skal være opp til 50 ganger hasigheten til 11b og 10 ganger hastigheten til 11a og 11g som er de mest brukte standardene. Teknologien som blir brukt for å oppnå dette er Multiple-Input Multiple-Output(Multiple-Input Multiple-Output (MIMO)). MIMO er en abstrakt matematisk modell for multi-antenne kommunikasjonssystemer hvor en sender har flere antenner som kan sende uavhengige signaler. De nye standardene har økt hastigheten for trådløs overføring betraktelig, men er fortsatt noe preget av ustabile forbindelser.

802.11 standarden og enheter som benytter seg av denne, blir ofte kalt *Wi-Fi* kompatible enheter, men har også blitt brukt som et generelt utrykk for WLAN. Wi-Fi er egentlig et varemerke fra *Wi-Fi Alliance* [7].

[4] Institute of Electrical and Electronics Engineers (IEEE) er en internasjonal non-profit, profesjonell organisasjon for utvikling av teknologi.

[5] Det fysiske laget i trådbasert Ethernet.

[6] Megabit per sekund. Megabit (Mbit) er en mengde informasjonslagring. 1 Mbit = 10^6 = 1,000,000 bits = 125,000 bytes.

[7] http://wi-fi.org/

For å benytte de nye teknologiene og standardene som utvikles, må både basestasjonene [8] og klientene støtte disse standardene. Ny basestasjoner og nettverkskort er som oftest bakoverkompatible slik at nettverk kan gradvis videreutvikles for å støtte de nye og bedre standardene.

Det finnes en del system som benytter seg av WLAN for posisjonering og de benytter ulike teknikker for posisjoneringen. De aller fleste har teknikker som benytter seg av en eksisterende WLAN topologi, og legger sine teknikker oppå disse. Under presenteres forskjellige teknikker.

Ekahau Positioning Engine er et programvarebasert system som man installerer på en servermaskin som støtter systemkravet, og er knyttet til det lokale nettverket [19]. Ekahau Positioning Engine (EPE) benytter seg av eksisterende trådløse basestasjoner som er Wi-Fi kompatible, og klienter som bærbare PC'er og Personal Data Assistent (PDA)'er. EPE beregner posisjonen ved at en klient som kjører EPE Client-programvare, registrerer signalstyrken til alle de aksesspunktene den har kontakt med. Deretter sender den dette til en EPE posisjonsserver som beregner posisjonen utifra dette.

Administrasjon og overvåkning av sporede enheter foregår i et web-basert grensesnitt, samt at det finnes interface for applikasjonsutvikling mot EPE. Arkitekturen er tilrettelagt for lett integrasjon av dette systemet med den eksisterende middleware-teknologiene som måtte benyttes. Dette gjør de ved blant annet å muligjøre for kommunisering ved hjelp av XML 3.2.2.6 dokumenter over Hyper Text Transfer Protocol (HTTP).

Ekahau har også utviklet små Wi-Fi [9] brikker som kan posisjoneres lik andre trådløse enheter. Disse brikkene er ment å festes til personer eller utstyr for å få status og oversikt over et foretaks aktiva. Posisjoneringen har en nøyaktighet på inntil 1 meter. En stor styrke til EPE er at det ikke krever ekstra hardware utenom det trådløse nettverket en allerede har. Figur 3.4 viser en Ekahau T301 Wi-Fi brikke. EPE's begrensninger til hvilke enheter som kan posisjoneres ligger i

Figur 3.4: En Ekahau T301 Wi-Fi posisjoneringsbrikke.

at de må ha en klient installert. Denne klienten er for tiden laget for Windows ™ enheter, hvor listen inneholder PC'er, bærbare og en del PDA'er.

[8] Basestasjon, eller aksesspunkt, er i sammenheng med lokalnettverk en fellesbetegnelse for enheten trådløse klienter kobler seg opp mot, og som fordeler ut IP adresser.

[9] http://www.wi-fi.org/

CRE utviklet av Radionor Communications er en teknologi som posisjonerer trådløse enheter. Den består blant annet av intelligente antenner, nye prosesseringsstrategier og avanserte matematiske beregningsalgoritmer. Cordis RadioEye™ kan har en del egenskaper som gjør at den har en sterk posisjon blant posisjoneringsteknologiene og er den mest presise teknologien. I neste seksjon gis en mer detaljert presentasjon av CRE.

Cordis RadioEye™

Radionor Communications er et selskap som ble dannet i 2000 og er lokalisert i Trondheim. Et av deres kjernesatsninger er *Cordis RadioEye™*. Cordis RadioEye™ er en hardware basert lokasjonssensor som lokaliserer trådløse enheter i et trådløst lokalnettverk. CRE består av et panel av intelligente antenner, et hovedkort med en 300MHz prosessor, 2 ethernet nettverkskort, og kjører et Linux operativsystem. CRE kjører en egen server for å kommunisere mellom CRE og en klient. Siden CRE holder sitt eget system for dataprosessering, trenger det ingen andre system for å kjøre. Figur 3.5 viser en CRE.

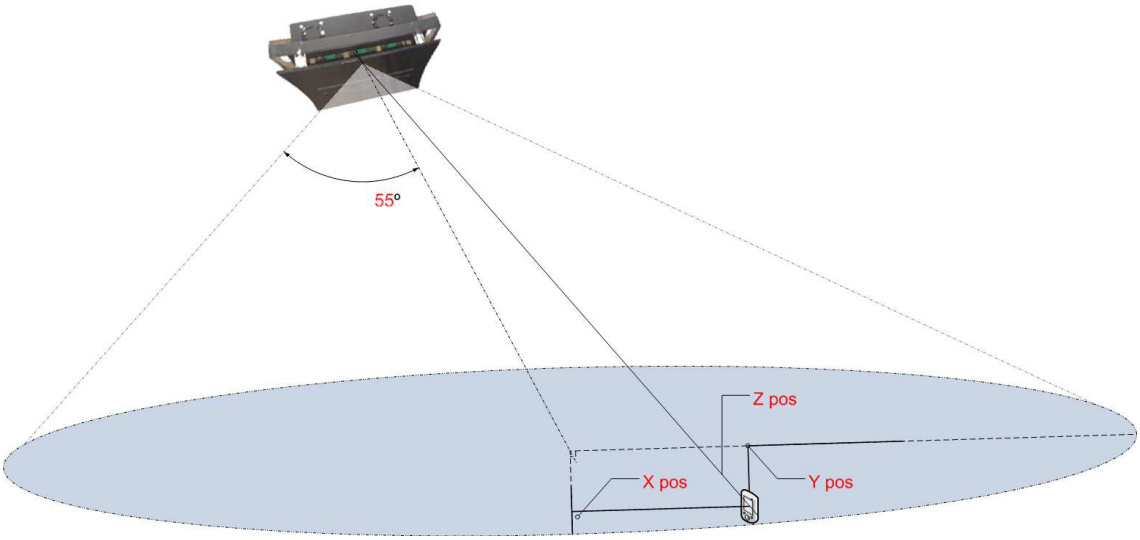
Figur 3.5: En Cordis RadioEye™.

CRE posisjonerer enheter ved å gjøre en spektrumsanalyse av de mikrobølger som trådløse enheter avgir. En geografisk posisjon gis til de trådløse enhetene, og identifiserer dem ved å dekode MAC adressen. CRE må konfigureres i et lokalt koordinatsystem, og posisjonen til enhetene blir beregnet utifra deres relative posisjon i forhold til CRE.

En CRE vil ofte plasseres i taket for å gi best mulig dekningsområdet. Dekningsområdet vil da være et kjegleformet område ut fra CRE. Dekningsgraden er oppgitt i [1] som 110°. Figur 3.6 viser dekningsområdet for CRE.

Det trengs bare én CRE for å gi fullverdige posisjoner, men det er også mulig å benytte flere som arbeider sammen for å gi utvidet og mer nøyaktige posisjoner under vanskelige forhold. Man vil da også kunne få et nytt koordinat, som angir posisjoner i høyden. Èn CRE blir da satt som hovedenhet og den vil innhente data fra de andre.

Nøyaktigheten til CRE er oppgitt til å være helt ned til 50cm. Denne nøyaktigheten kan påvirkes av de forholdene den er plassert i og nøyaktigheten kan derfor bli noe større. Bygningsstrukturen kan gjøre det vanskelig for CRE å fungere optimalt. Vegger kan gjøre at signalstyrken blir svakere og signalene kan også reflekteres



Figur 3.6: Dekningsområdet til Cordis RadioEye™.

av veggene som gir inferens for CRE.

CRE må konfigureres når det blir satt opp og defineres inn i det lokale koordinatsystemet. De endringer i det området som er satt til å overvåke, vil bli automatisk håndtert av CRE.

For å administrere CRE, er det laget et webgrensesnitt hvor man kan stille inn de nødvendige konfigurasjoner for at CRE skal gi nøyaktige posisjoner. Dette grensesnittet aksesseres ved å benytte en nettleser og går til CRE's egen internettadresse. Videre er det laget et XML interface for å forespørre etter posisjonsdata fra CRE. Interfacet er kalt *Wireless Positioning Protocol*(WPP) og har en en formalisert meldingsstruktur gjennom en XML Document Type Definition (DTD). Denne DTD'en spesifiserer hvordan kommunikasjon til og forventet svar er for de som skal kommunisere med CRE. Det er definert tre hoved XML-dokument for kommmunikasjon som er:

- Wireless Positioning Protocol Init(wpp_init)
- Wireless Positioning Protocol Triggered Location Report(push_event)
- Wireless Positioning Protocol Response(wpp_response)

wpp_init ber CRE om posisjoner, definere soner eller utnytte brukerspesifikk konfigurasjon. Man kan be om posisjoner på to ulike måter. Den ene er av typen *Location Immediate Request*(LIR) som ber om en posisjon utifra visse kriterier som man sender med. Svaret man får fra CRE er i form av en *Location Immediate Answer*(LIA), og det er en LIA for hver LIR man sendte med. Inni en LIR forespørsel vil man spesifisere utifra kriterier spesifisert i DTD'en, hva og hvordan svar fra CRE skal rapporteres.

Den andre typen posisjonsanmodning er en *Triggered Location Reporting Request*(TLRR) hvor man spesifiserer visse kriterier som må være oppfyllt for at

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE wpp_init SYSTEM "wpp_init.dtd">
<wpp_init version="1.4.1">
  <authentication>
    <userid>BRUKERNAVN</userid>
    <password>PASSORD</password>
  </authentication>
  <lir>
    <report_format>
      <report_coordinates/>
    </report_format>
    <report_multiple>
      <positioned/>
      <mac_type/>
    </report_multiple>
  </lir>
</wpp_init>
```

Figur 3.7: Eksempel på XML sendt til Cordis RadioEye™

man skal motta en rapport. Et umiddelbart svar sendes i retur i form av en *Triggered Location Reporting Answer(TLRA)* og forteller om TLRR anmodningen ble godtatt.

Under i Figur 3.7 vises et eksempel på en XML LIR forespørsel sendt til CRE. En mer detaljert forklaring på XML gis i 3.2.2.6.

Eksempelet viser hvordan man i en LIR spesifiserer at svaret skal rapporteres(<report_format>) med koordinater(<report_coordinates>), og hva som skal rapporteres(<report_multiple>); samtlige posisjonerte(<positioned/>) og identifiser med mac adresser(<mac_type/>).

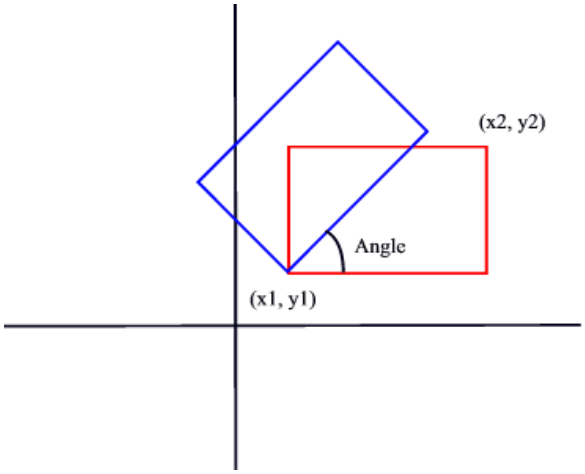
Triggered Location Report(TLR) sendes som en *push_event message* fra CRE, og er rapporter som har blitt anmodet via en TLRR.

Utnyttelse av soner skjer ved å sende en *Zone Definition Request*(ZDR), hvor man kan opprette nye soner, slette en sone og få status på de nåværende soner. Soner defineres i det lokale Kartesiske koordinatsystemet som rektangulære bokser. Sonene oppgis ved å gi X, og Y koordinat og en eventuell verdi for vinkelen på rektangelet. Figur 3.8 [10] viser hvordan sonene defineres.

3.2.2.6 Extensible Markup Language

Extensible Markup Language [11] er en undergruppe, eller nærmere en forkortning av Standard Generalized Markup Language (SGML) [12]. XML er, lik SGML, et metaspråk hvor man definerer sitt eget markeringsspråk. XML er laget for å dele data mellom ulike informasjonssystemer og er enklere å analysere og prosessere

[10] Bildet er hentet fra manualen til Cordis RadioEye™.
[11] http://xml.silmaril.ie/
[12] SGML er et ISO standardisert metaspråk, hvor man kan definere markeringsspråk.



Figur 3.8: Definering av soner i Cordis RadioEye™.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE XML_TEST SYSTEM "XML_TEST.DTD">
<adresse>
  <gatenavn>Kongens gate 1</gatenavn>
  <postnummer>7011</postnummer>
  <by>Trondheim</by>
</adresse>
```

Figur 3.9: Eksempel på et XML dokument.

enn SGML.

XML er en trestruktur av informasjon, hvor treet baserer seg på *elementer* og *attributter*. XML dokumentet trenger en spesifikasjon over de elementene og attributtene som skal være lov å benytte, for å beskrive dataene i XML dokumentet. Dette gjøres ved å benytte DTD eller *skjemaer*(schemas), som blir definert i et skjemaspråk. Noen av de mest kjente skjemaspråkene er *DTD*, *XML Schema* eller *RELAX NG*. DTD var det første skjemaspråket, og har noe begrensninger i forhold til de nyere språkene.

Skjemaene tillegger syntaksen i XML restriksjoner, hvor det typisk er tillatte attributt- og elementnavn, samt hvilket innhold som kan være i elementene. For eksempel kan et element "adresse" defineres til å kun inneholde 3 element, "gatenavn", "postnummer" og "by". Figur 3.9 viser et lite eksempel på et XML dokument, hvor det defineres en adresse.

Cordis RadioEye™ håndterer sikkerhet i kommunikasjon ved å benytte seg av den sikre tilkobling Secure Hyper Text Transfer Protocol (HTTPS). HTTP over Secure Sockets Layer (SSL), eller HTTPS, er en sikker versjon av den vanlige HTTP protokollen. Videre er det et krav om brukernavn og passord for å gjøre anmodninger til CRE og for å utføre konfigurasjoner. Alle XML dokumenter som sendes til CRE må ha med elementer for autentisering, slik som i Figur 3.10:

Følgende Seksjon ser på to aktive felt innen forskning, som ser på hvordan en

```
...
 <authentication>
   <userid>BRUKERNAVN</userid>
   <password>PASSORD</password>
 </authentication>
 ...
```

Figur 3.10: Autentisering av XML til Cordis RadioEye™

skal integrere teknologien, på best mulig måte, inn i hverdagen.

3.3 Allestedsnærværende og gjennomtrengde

Ubiquitous computing(allestedsnærværende) og *pervasive computing*(gjennomtrengende), er aktive felt for utvikling både innen forskning og applikasjonsutvikling.

3.3.1 Allestedsnærværende

Allestedsnærværende komputasjon handler om å integrere teknologien slik at benyttelsen av den glir sømløst inn i våre daglige rutiner. Som Weiser sa i [41], teknologien skal bli *literally visible, effectively invisible*. Det er de løsningene som på mest mulig effektiv måte lar deg utføre dine oppgaver uten å tenke på teknologien som utfører dem for deg. Vi har for lengst sluttet å bry oss med hva det er som gjør at vi kan ringe tvers over landet og planeten vår. Vi benytter oss av telefonen uten å vie tankene til all den bakenforliggende teknologien som gjør dette mulig. Slik var det Mark Weiser så for seg at datateknologien ville være i fremtiden [40].

> Aller viktigst, allestedsnærværende datamaskiner vil være med på å løse problemet med informasjonsoverflod. Det er mer informasjon tilgjengelig ved våre fingertupper gjennom en tur i skogen enn det er i hvilket som helst datasystem, allikevel føles en spasertur blant trærne avslappende og datamaskiner frustrerende. Maskiner som passer menneskelig miljø, istedefor å tvinge mennesker å entre deres, vil gjøre bruken av en datamaskin like oppfriskende som en spasertur i skogen.

3.3.2 Gjennomtrengende

Gjennomtrengende komputasjon handler om det å ha datamaskiner og sensorer "overalt", og i forskjellige former. For eksempel kan dette være små sensorer integrert i PC'er, mobiler, apparater, hjelpemiddel og selv i klærne. Videre på alle de forskjellige områdene vi oppholder eller beveger oss i, slik som i hjemmet og på

jobben, på sykehus og fabrikker. Det som gjør at teknologien heter *gjennomtrengende komputasjon* er også at alle enhetene og sensorene kommuniserer sammen, og samhandler slik at de blir integrert i våre daglige rutiner.

De fleste av oss benytter mange teknologiske "duppeditter" i løpet av en dag og det er mange av dem som har blitt usynlige for oss. Datamaskinen er ennå ikke kommet til dette stadiet, selv om de aller fleste har nå en eller flere datamaskiner i hjemmet og på kontoret. Datamaskinen skal integreres sammen med et voksende antall tekniske enheter som forsøker å samhandle for å gjøre tilværelsen din lettere. For tiden er disse "duppedittene" et avansert mas for å få til denne samhandlingen, og langt fra usynlig.

For at den teknologien vi benytter oss av og har gjort oss så avhengig av, skal bli så allestedsnærværende som vi ønsker, er man nødt til å gjøre noe nytt. Da er mobiliteten som preger vår hverdag nødt til å inkorporeres og tas høyde for i den videre utviklingen.

Vi forflytter oss hjemmenfra og til kontoret, og rundt om i kontorbygget. Vi "krever" nå at teknologien skal tilby de samme tjenestene mens vi forflytter oss og er utenfor de normale bruksområdene til vår stasjonære PC.

Kommunikasjon og informasjonsflyt er en pådriver for utviklingen av mobile enheter. De enheter som har blitt tatt opp er enheter som er like mobile som brukerene selv, og som tilbyr de funksjoner forbrukeren behov for. Den enheten som har tatt den klare førsteplassen er mobilen.

I Norge er mobilen blitt allemannseie og vi er på verdenstoppen i bruk av denne redskapen. Det er nå flere mobiler i Norge enn det er innbyggere og det øker enda. Tabell 3.1 viser utbredelsen av mobiltelefoner pér innbygger i Norge.

Mobilutbredelse i Norge						
	2002	2003	2004	2005	1. halvår 2006	3. kvartal 2006
Abb. per 100 inbygger	84	89	98	102	105	106

Tabell 3.1: Mobilutbredelse i Norge.

Den store utbredelse av mobilen gjør at utviklingen av denne teknologien går veldig raskt og den er også en fremdriver for andre teknologier. Batteri og radioteknologier videreutvikles for å holde takten med mobilene. Forbrukerne i Norge er ikke bare storforbrukere av mobiler, men de er også avanserte mobilbrukere og ønsker mer avanserte mobiler [38]. Mobilene øker i nå hurtig i minne, prosessorkraft og lagerplass og dette gjør at de takler større og mer avanserte applikasjoner.

Batterikapasiteten henger dessverre litt etter slik at noen teknologier og bruksområder får svi. Spesielt ved bruk av ekstrafunksjoner som noen mobiler tilbyr, slik som WLAN, Bluetooth og GPS. Den mest utbredte teknologien som bruker en del strøm er WLAN og det er begrenset hvor mye det er ønskelig å ha en slik funksjon på.

GPS er i sin spede begynnelse som en innebygd teknologi i mobiltelefonen, men det har eksistert eksterne enheter en stund. Da vil man koble seg opp til GPS'en med Bluetooth.

Personal Data Assistent, eller PDA som den er bedre kjent som, er en håndholdt enhet som har en del særegne egenskaper. Disse egenskapene var først enkle ting som kalender, adressebok, oppgaveliste og kanskje mulighet for å ta notater. Etter hvert har disse enhetene fått mange flere funksjoner og utvidet funksjonalitet, som e-post håndtering, dokumenthåndtering, web-lesing, og mange flere. Selv om PDA'en over en lang periode har hatt en rekke funksjoner som ikke mobilen kunne stille opp med, har PDA'en aldri fått like stor oppslutning som mobiltelefonen. Gjennom tiden til PDA har den vært en relatvit stor, i forhold til mobilen.

Smartphone er betegnelsen på mobiltelefoner som har fått store utvidelser i funksjonalitet. Dette er mobiler som gir samme funksjonalitet som en PDA, og som var ment for å fylle det spriket som var mellom PDA'er og mobiltelefoner.

3.4 Kontekstsensitive applikasjoner

Context-Aware, eller kontekstsensitive applikasjoner, er applikasjoner som tar hensyn til konteksten brukeren er i og tilbyr utifra dette riktig informasjon eller tjenester. Dey og Abowd gav i [9] denne definisjonen på hva det vil si å være kontekstbevisst, også kjent som kontekstsensitivt.

> Et system er kontekstbevisst hvis det bruker kontekst for å gi relevant informasjon og/eller tjenester til brukeren, hvor relevans er avhengig av brukerens oppgaver.

Kontekstsensitive applikasjoner benytter kontekst til å påvirke og endre applikasjonen slik at den tilpasser seg miljøet.

Hvordan applikasjonene benytter kontekstinformasjon kan variere og også hvordan informasjonen blir innhentet. Når og hvordan skal informasjonen innhentes? Skal det gjøres automatisk på gitte tidsintervaller, eller skal det hentes på brukers kommando?

3.4.0.1 Kontekst innhenting og tilbydere

For at system skal få tilgang til forskjellig kontekstinformasjon må det være tjenester som forsørger disse dataene. Et system kan selv innhente data om brukeren ved å spørre brukeren som da gir denne informasjonen gjennom de direkte input som systemet støtter(ofte mus og tastatur).

Alternativt kan systemet benytte seg av andre distribuerte kilder for informasjon. Dette kan være sensorer som tilbyr informasjon utover det som ligger internt i systemet. Ulike typer sensorer kan innhenter data om fysiske forhold som

værdata(barometerverdier og luftfuktighet), lysforhold, andre personer eller egen posisjon.

Dey og Abowd presenterer i [9] tre kategorier for kontekstsensitive applikasjoner, hvor kategoriseringen ser på ulike egenskaper.

1. **Presentasjon** av informasjon og tjenester for en bruker. Benytter seg av konteksten for å presentere den informasjon eller de tjenester som er nyttige i den gitte konteksten. Slik vil det bli færre valg, og enklere og hurtigere for brukeren å utføre oppgaver.
2. **Automatisk *eksekvering*** av en tjeneste, på vegne av en bruker ut i fra kontekstinformasjon.
3. **Merking** av kontekst til informasjon, for senere gjenfinning og for å kunne aktivere informasjon ved en gitt kontekst.

Videre blir fire State-of-the-Art system som benytter seg av kontekst presentert.

De ulike løsningene diskuteres etter hver presentasjon og til slutt blir de de satt opp mot kategoriseringen gitt av Dey og Abowds.

3.4.1 Context Toolkit

Dey og Abowd er velkjente navn innen forskning på bruk av kontekst. Deres forskning på kontekst har resultert, blant annet, i et rammeverk for utvikling av kontekstsensitive applikasjoner, kalt *Context Toolkit* [32].

Rammeverket baserte seg på *widgets*, separate programvarekomponenter, og da de velkjente Graphical User Interface (GUI) komponentene som blir brukt for å programmere brukergrensesnitt. Det som var utviklernes mål med Context Toolkit, var å skjule kompleksiteten, gi en hensiktsmessig tolkning av kontekst og lette den helhetlige konstruksjon ved gjenbruk av widgets. Widgetene fungerte mellom applikasjonen og det miljøet den opererer i, slik som de GUI-baserte komponentene.

Context Toolkit ble etter hvert et bibliotek med forskjellige widgets, blant annet for å detektere nærvær, identitet og aktiviteter til folk og ting. Widgetene var ment for å benyttes i en distribuert arkitektur, noe som gjenspeilet seg i de tre distribuerte komponentene som widgetene var avhengig av. Disse tre komponentene var:

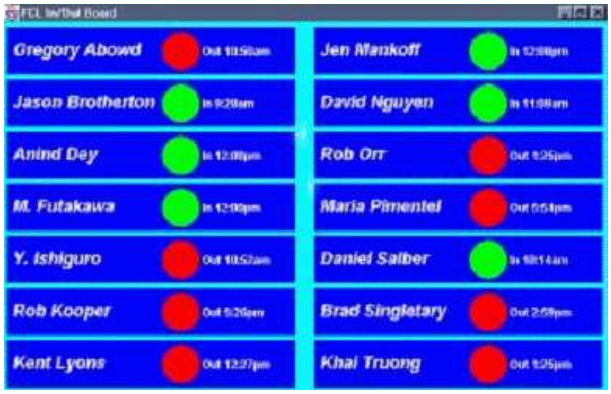
- **Generatorer**; for å samle inn kontekstinformasjon. Dette var komponenter som innkapslet èn, eller flere like sensorer, og den software som samlet inn rå data fra dem.
- **Tolker(interpreter)**; for å trekke ut informasjon fra innsamlet kontekstinformasjon.
- **Servere**; for å samle innsamlet data.

Context Toolkit's eneste forutsettelse var at underliggende system støttet Transmission Control Protocol/Internet Protocol (TCP/IP). Videre benyttet de seg av XML over HTTP for å holde kravene for utnyttelsen av Context Toolkit nede.

I [32] nevner de to utviklede kontekst-widgeter for å demonstrere Context Toolkit, henholdsvis *IdentityPresence* og *Activity*. IdentityPresence widgeten ble utplassert og opplyste om hvem som kom og gikk, samt identiteten deres. Activity widgeten ble også utplassert og opplyste om endringer i aktivitetsnivået på dette området. Widgetene skulle kunne benytte seg av forskjellige sensorer for sine oppgaver, og eksempelvis kunne Activity widgeten benytte med infrarød sensor eller videoanalyse. For oppgaven den ble testet på av utviklerne, ble det benyttet en mikrofon.

3.4.1.1 In/Out Board

En applikasjon utviklet, kalt *In/Out Board* benyttet seg av IdentityPresence widgeten som ble lokalisert ved inngangen til bygningen deres. Applikasjonen viste en oversikt over de ansatte, hvem som var tilstede og tiden da de ankom eller forlot bygningen. Ansatte som var tilstedet fikk en grønn prikk, og en rød hvis de var borte. Figur 3.11 viser In/Out Board applikasjonen.



Figur 3.11: In/Out Board applikasjonen laget med Context Toolkit.

Tilnærmingen til bruk av kontekst i ContextToolkit senker terskelen betraktelig for å utforskning av de forskjellige aspekter ved bruk av kontekst. Abstraksjonen av sensorene gjør også at utvikling av applikasjoner, i teorien, kan benytte seg av alle tilgjengelige teknologier, bare den blir skjult bak en Widget. Dette typen abstraksjon kan sammenlignes med teknikker for *tjenesteoppdagelse*(Service Discovery), hvor beskrivelser og identifikasjon av tjenester og dens tilbydere er eneste grensesnitt for applikasjonene.

3.4.2 SmartReminder

I 2001 vant SmartReminder [23], en applikasjon utviklet for å forske på kontekstsensitive applikasjoner, *John G. Kemeny Computing Price*, hvor kategorien var

beste innvative programvare [13].

SmartReminder i seg selv er en påminnelse-applikasjon som påminner brukeren utifra en gitt kontekst. Applikasjonen gikk ut ifra brukerens timeplan, som ble lastet inn når applikasjonen startet. I stedet for å gi hver påminnelse en viss tid før hver hendelse, får brukeren påmindelsen en tid før som er satt utifra kontekst. Kontekst som ble tatt i betraktning var lokasjonen til brukeren, destinasjon, tidspunkt, vær- og trafikkdata, slik at en estimert reisetid for å komme seg til avtalestedet kunne beregnes. SmartReminder fikk ikke benyttet lokasjonsdata under utprøvning da dette ikke ble satt opp fysisk, men systemet var klargjort for dette.

Systemet skulle også komme opp med varslinger utifra *umiddelbar nærhet* til andre enheter, og da påminne brukeren om noe som eventuellt skulle diskuteres med personen.

SmartReminder ble også designet slik at det skulle ikke stoppe opp ved manglende kontekstinformasjon, men nedgraderes gradvis, så hvis lokasjonsdata manglet ble siste posisjon benyttet ved beregning.

Systemet ble designet på Solar [8], som er en åpen plattform for utvikling av mobile kontekstsensitive applikasjoner og benyttet samme objektorienterte design i SmartReminder. Denne arkitekturen tillater en modulær oppbyning som gjør tilføyelser til systemet enklere, og uten å endre for mye på kjerneapplikasjonen.

SmartReminder viser sitt design med en del av sine kontekstsensorer i mobilenheten selv, slik som sensorene for lokalisering og umiddelbar nærhet. Disse ble ikke implementert i systemet, men forskjellige lokasjonsmetoder ble diskutert. De kom frem til at det var behov for to forskjellige sensorer utifra hvorvidt brukeren var ute- eller innendørs.

SmartReminder's gradvise degradering av funksjonalitet øker applikasjonens anvendelighet betraktelig. Hvis applikasjonen opprettholder en viss kjernefunksjonalitet selv når deler av applikasjonen er uanvendelig, vil den opplevde brukervennligheten øke. Dette gjør at applikasjonen øker muligheten for å bli annerkjent av brukere og forbrukere.

3.4.3 ContextPhone

ContextPhone er en open-source plattform for utvikling av kontekstsensitive applikasjoner, utviklet av forskere ved University of Helsinki [14] and Helsinki Institute for Information Technology [15] [27].

Context Phone Project er et pågående prosjekt hvor det forskes på karakterisering og analyse av informasjon vedrørende en brukers kontekst, og bruken av denne til proaktiv adapsjon. Det er fra dette prosjektet at ContextPhone har blitt en

[13] http://www.cs.dartmouth.edu/newsinfo/kemeny/2001/final.html
[14] http://www.helsinki.fi/university/
[15] http://www.hiit.fi/

realitet.

ContextPhone ble laget med menneskerettet(human-centered) utviklingsstrategi for å tilrettelegge konstruksjonen av allestedsnærværende og sømløst integrerte applikasjoner. ContextPhone plattformen består av fire integrerte moduler som stiller bibliotek for tilgang til og styring av mobiltelefonen. De fire modulene er:

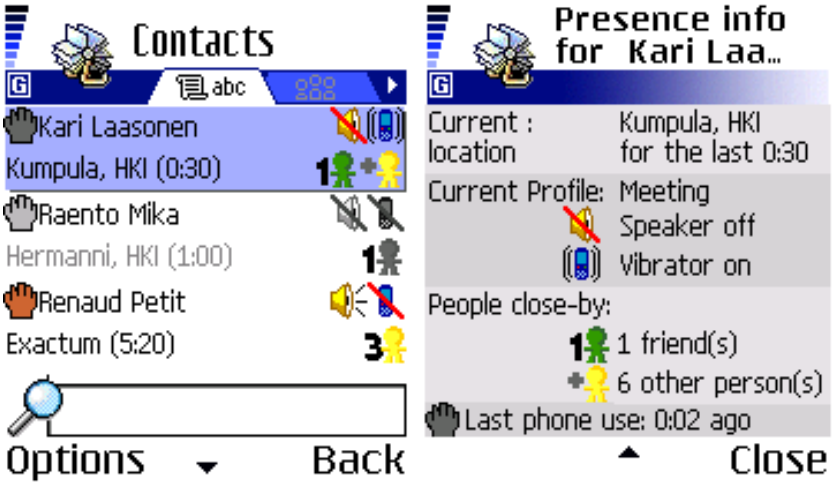
- **Sensorer**; Skaffer kontekstdata fra de forskjellige dataene mobilen er satt opp med. Mulige sensorer er basestasjon(Cell id), GPS eller mobilbruk.
- **Kommunikasjon**; Styrer tilkobling til eksterne tjenester, for eksempel til å sende tilstedeværelses data. Benytter Internett protokoller via General Packet Radio Service (GPRS), Bluetooth, Short Message Service (SMS) eller Multimedia Messaging Service (MMS).
- **Tilpasningsdyktige applikasjoner**; Erstatter innebygde mobilapplikasjoner slik som kontaktbok eller siste samtaler oversikt. Applikasjonene som følger med heter *ContextLogger*, *ContextContacts* og *ContextMedia*.
- **Systemtjenester**; Styrer bakgrunnstjenester, logging, gjennoppretting og Status display.

Komponentene og den logiske strukturen er lik *Context Toolkit* sine *kontekst-widgets*, men avgrenset til et enkelt system. Komponentene bruker en *publish-subscribe* modell. Komponenter abonnerer på *events* fra andre komponenter som kaster events etter at de har registrert en endring.

ContextPhone benytter seg for det meste av kontekstinformasjon som hentes fra mobilen i seg selv og utnytter den for å forbedre og utvide funkasjonaliteten til mobilen. *ContextContacts* for eksempel, vil gi utvidet informasjon om dine venner eller kollegaer i kontaktboken. Den kan vise om kontakter er tilgjengelige, utifra om mobilen er satt til "møte" eller "privat" osv.. Slik informasjon gjør at man slipper å forstyrre en person som sitter i et møte eller sitter på en kino. Den kan også vise hvem vedkommende er i nærheten av ved at den registrere andre mobiler nærheten ved hjelp av Bluetooth. Figur 3.12 viser kontaktlisten slik den ser ut med informasjon om andre brukere til venstre, og hvordan man kan velge sette informasjon som skal vises i de andres kontaktlister til høyre.

ContextPhone benytter seg av et allerede akseptert og integrert hjelpemiddel, mobilen, ved å forbedre allerede innebygde funksjoner. Overgangen fra mobilens innebygde funksjoner til ContextPhone blir lettere for forbrukere og gjør da at applikasjonen blir akseptert raskere.

Integrasjonen i mobilens eksisterende applikasjoner er en begrensning, da det finnes en mengde ulike operativsystem for mobiler. En frittstående applikasjon vil være tilgjengelig for et større antall mobiler.



Figur 3.12: ContextContacts: en utvidet kontaktliste.

3.4.4 The Nidaros Framework

Et lokalt og spennende prosjekt kalt *The Nidaros Framework* [39], ble gjennomført for å forske på lokasjonsbevisste applikasjoner. Dette rammeverket benytter seg av Cordis RadioEye™.

The Nidaros Framework ble laget for å utvikle lokasjonsbevisste(location-aware) applikasjoner. Målet for rammeverket var å benytte flere ulike teknologier for posisjonering som GPS, GSM og WLAN, samt at dette skulle håndteres av èn lokasjonsserver. Ved å benytte en enkelt lokasjonsserver for samtlige lokasjons-sensorer, ble det lettere å takle overlappende posisjonsdata. Figur 3.13 viser arkitekturen til lokasjonsserveren.

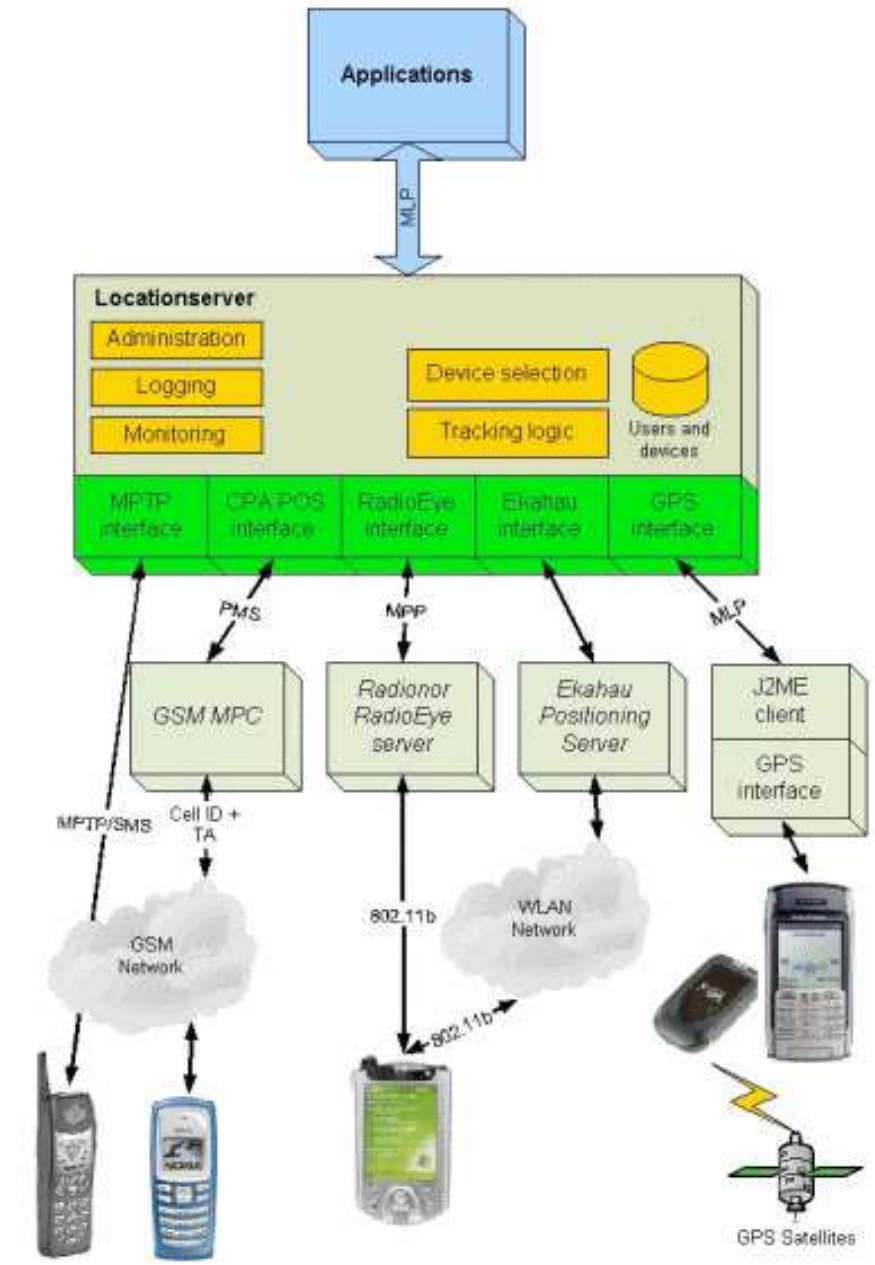
For å teste rammeverket ble det satt opp en turguide i Nidarosdomen [16] for både mobiltelefoner og PDA'er. Denne implementasjonen erstattet domkirkens eksisterende PDA-baserte guide, *Nidaros Pocket Guide*(NPG), en ren PDA basert guide uten lokasjons-sensor.

Den en nye guiden var en klient for PDA og den ble laget i Flash MX for å støtte tekst, lyd og video. Mobiltelefonklienten ble implementert ved bruk av Hyper Text Markup Language (HTML) og web-browseren i mobilen. Sidene ble sendt til mobilen med *WAP Push* og viste også sider med video og lyd.

Oppkobling til serveren fra mobilen ble gjort med GPRS siden mobilen ikke hadde innebygd WLAN kort. Manglende WLAN kort gjorde at de ikke fikk posisjonert mobilen og de utstyrte da brukerene med WLAN brikker som ble festet på brukeren. Mobilen og WLAN brikken ble paret ved at man oppgav Media Access Control (MAC)-adressen til WLAN-brikken i mobilklienten, samt telefonnummeret til mobilen.

Demonstrasjonen som ble gjort i Nidarosdomen fikk veldig god mottakelse og spesielt mobiltelefonens applikasjon. Dette viser at interessen for en slik benyttelse

[16] http://www.nidarosdomen.no



Figur 3.13: Arktitekturen til Nidaros Framework lokasjonsserver

av mobilen blir lett tatt opp av forbrukerne.

Mobiltelefonen er en av våre aller mest aksepterte og integrerte teknologier og gjør dette Nidaros prosjektet veldig spennende. Begrensningen prosjektet erfarte under testing, hvor mobilen manglet WLAN, er på vei til å endres.

Prosjektet ser også på hvordan man må endre på presentasjon av innhold etter hvilke begrensninger og muligheter enheten som kjører applikasjonen har. Fordelen prosjektet viste til med å sende XML dokumenter mellom klient og server, var at det gav en høy fleksibilitet i styring av den grafiske presentasjonen.

Sending av XML dokumenter innebar også en del problemer med denne løsningen.

Applikasjonens stilte da høye krav til både prosessor og minne på enhetene, og batteriene ble tappet for strøm på kort tid.

XML dokumentene som ble sendt frem og tilbake mellom klient og server, ble et mulig hinder hvis det ble mange brukere på samme tid. Responsiviteten til systemet kunne minke ettersom antallet brukere økte, på grunn av den store belastning XML dokumentene la på nettverket.

3.4.5 Klassifisering av kontekstsensitive system

Applikasjonene presentert over benytter seg alle av kontekstinformasjon med noe ulike hovedfokus, men samtlige benytter seg av lokasjon. Tabell 3.2 viser systemene utifra kategoriene **P**resentasjon, *Autotmatisk **E**ksekvering* og ***M**erking*, Dey og Abowd [9] presenterte.

System	**P**	**A**	**M**
Context Toolkit - In/Out Board	X		
SmartReminder	X		X
ContextPhone	X	X	
The Nidaros Framwork	X		

Tabell 3.2: Kategorisering av kontekstsensitive system.

Applikasjonene skiller seg også fra hverandre i hvilke teknologier de baserer seg på. Under presenteres noen av teknologiene brukt i applikasjonene og en utvidet presentasjon Java.

3.4.6 Utviklingsteknologi

Fire av de fem presenterte systemene benyttet seg av mobilen som hovedgrensesnitt for brukerene. Utvikling av applikasjoner for små enheter som mobilen er krevende på grunn av de store begrensningene på tilgjengelige ressurser.

ContextPhone 3.4.3 utviklet applikasjoner siktet for enkelte versjoner under operativsystemene Nokia Series [17] og Symbian OS [18].

The Nidaros Framework utviklet et grensesnitt i HTML og sendte det til mobilen via Wireless Application Protocol (WAP) Push.

Wireless Application Protocol er en protokoll for kommunikasjon og den ble utviklet for å kunne tilby Internett i de trådløse nettverkene. WAP er ment for å tilby Internett for mobiltelefoner og PDA'er og skal tilpasse innholdet til de restriksjoner som disse små enhetene har. Internett-sider laget for WAP blir omdannet til *Wireless Markup Language*(WML), som er tilpassede sider.

[17] http://www.s60.com/
[18] http://www.symbian.com/

WAP Push er et tillegg til WAP som tillater å "dytte" en melding til mobiltelefonen, og brukeren må godtar oppkoblingen til WAP siden. Det WAP Push gjør er å sende en kodet melding til mobiltelefonen med en adresse til WAP siden.

SmartReminder benyttet seg som nevnt av SOLAR [8] plattformen, og denne var implementert i Java.

Sun Developer Network (SUN) stiller med forskjellige Java plattformer for utvikling av applikasjoner for forskjellige kjøremiljø.

3.4.6.1 Java Platform, Standard Edition

Java Platform, Standard Edition er den originale distribusjonen i Java familien. Den består i hovedsak av to, hovedkomponenter; *J2SE Java Runtime Environment* og *Java Development Kit (JDK)* [19].

J2SE Java Runtime Environment sørger for kjøremiljøet til Java applikasjoner gjennom Java Virtual Machine og forsørger bibliotek. I tillegg sørger Java Runtime Environment (JRE) for flere komponenter, deriblandt Java Plug-in [20] som muliggjør for kjøring av en applet i en browser.

Java Development Kit er et "superset" av JRE. Det inneholder alt som er i JRE og i tillegg en rekke verktøy, slik som kompilatorer og feilsøkingsprogramvare(*debugger*) som muliggjør utvikling av applikasjoner for Java.

Java SE API definerer hvordan bibliotek kan benyttes, ved å spesifisere hvordan applikasjoner kan benytte seg av den tilgjengelige funksjonaliteten i biblioteket.

Java Virtual Machine er programvare som kjører Java bytecode, produsert av kompilatorer. Java Virtual Machine (JVM) sørger for uavhengighet til maskinvare og operativsystem, for applikasjoner laget på Java plattformen.

Java tilbyr mange muligheter for å lage sikre applikasjoner, med verktøy, implementasjoner av sikkerhetsalgoritmer(kryptografi), mekanismer og protokoller for nettverkskommunikasjon [34]. Java tilbyr et Application Programming Interface (API) for å opprette en sikker tilkobling over TCP/IP, og sikrer med dette all kommunikasjon mellom to applikasjoner som støtter protokollen. Java tilbyr også HTTPS og for å benytte dette er det nødvendig for applikasjonene å autentisere seg. dette kan gjøres på ulike måter, hvor en av dem er Public Key Infrastructure (PKI).

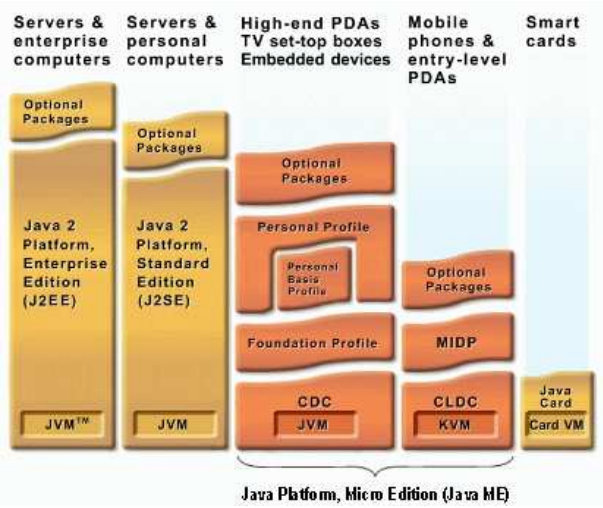
Public Key Infrastructure (PKI), er et rammeverk for utveksling av sikker kommunikasjon, basert på elektronisk legitimasjon og signaturer. Det lages da en *public* og en *private* nøkkel samtidig, ved å benytte den samme algoritmen. Den private nøkkelen plasseres fast på en maskin, og sendes ikke over nettverket, hvor den andre derimot plasseres slik at den er åpen og tilgjengelig for alle, som en del av et sertifikat. Denne nøkkelen kan da benyttes for å kryptere en melding, som

[19] http://java.sun.com/javase/technologies/
[20] Java Plug-in er en kobling mellom nettleser og Java plattformen.

sendes videre til mottakeren hvor den private nøkkelen ligger, og som da benyttes for å dekode meldingen.

3.4.6.2 Java Platform, Micro Edition

Java Platform, Micro Edition er SUN's svar på ressurs-begrensningene til små mobile enheter og er en samling spesifikasjoner for utvikling av små applikasjoner [25]. Spesifikasjonene, eller Java Specification Request, er spesifikasjoner satt av Java Community Process, og Java Specification Request (JSR)'ene i Java Platform, Micro Edition (J2ME) er et subsett av de komponentene man finner i en standard Java distribusjon, slik som Java Platform, Standard Edition (J2SE). J2ME ble originalt laget for å håndtere de restriksjoner som er assosiert med konstruksjonen av applikasjoner for små enheter. Figur viser J2ME teknologien 3.14, og dens relasjon til resten av Java familien.



Figur 3.14: J2ME teknologien og relasjon til andre Java teknologier

J2ME teknologien er basert på tre hovedelementer;

- **Konfigurasjon**
- **Profil**
- **Valgfri java-pakke**

Konfigurasjonene har blitt delt opp i to basis konfigurasjoner hvor man har små mobile enheter og en som er mer rettet mot kraftigere mobile enheter som smartphones og digitalbokser. Konfigurasjonene for små enheter er kalt *Connected Limited Device Configuration* og for de mer kapable enhetene er *Connected Device Configuration*. Hver av de forskjellige konfigurasjonene har profiler tilgjengelig for å legge til flere funksjoner. Profilen som legges til er styrt av hvilken konfigurasjon

man benytter. Oppå konfigurasjonen og profilen kan man legge til valgfrie pakker for ekstra funkasjonalitet.

Connected Limited Device Configuration (CLDC) er en konfigurasjon som sikter seg inn på små ressurs-svake enheter slik som mobiler og er designet for å møte de krav som stilles for at en Java plattform skal kunne kjøre på en mobil. Kravene er satt for å imøtekomme de begrensede ressurser på minne, prosesseringskraft og grafiske kapabiliteter som er på mobilene. CLDC gir et basis sett API og en virtuell maskin tilpasset slik at Java applikasjoner kan kjøres på en mobil. En mye brukt kombinasjon for mobiler er å kombinere CLDC med Mobile Information Device Profile (MIDP) profilen for å stille et komplett Java applikasjonsmiljø.

CLDC 1.1 er en revisjon av den tidligere 1.0 versjonen og inkluderer noen nye funksjoner, hvor de viktigste er flytende komma. I tillegg er det inkludert en Java funksjon som sletter objekter som ikke lengre er i bruk. Denne fjerner objekter med en så kaldt *weak reference*(svak referanse). Dette er en viktig minnebesparende funksjon for små enheter som mobiler.

Connected Device Configuration (CDC) er en konfigurasjon for J2ME som er rettet mot litt større enheter med mer kapasitet og nettverksforbindelse. På CDC konfigurasjonen er det definert tre profiler, og hver av profilene har et sett valgfrie pakker, som selveste applikasjonen kjører på.

- **Foundation Profile**
- **Personal Basis Profile**
- **Personal Profile**

Profilene er en samling Java API'er som støtter en konfigurasjon for å gi visse egenskaper til en spesiell enhet(mobil el.l.), eller en retning innen markedet.

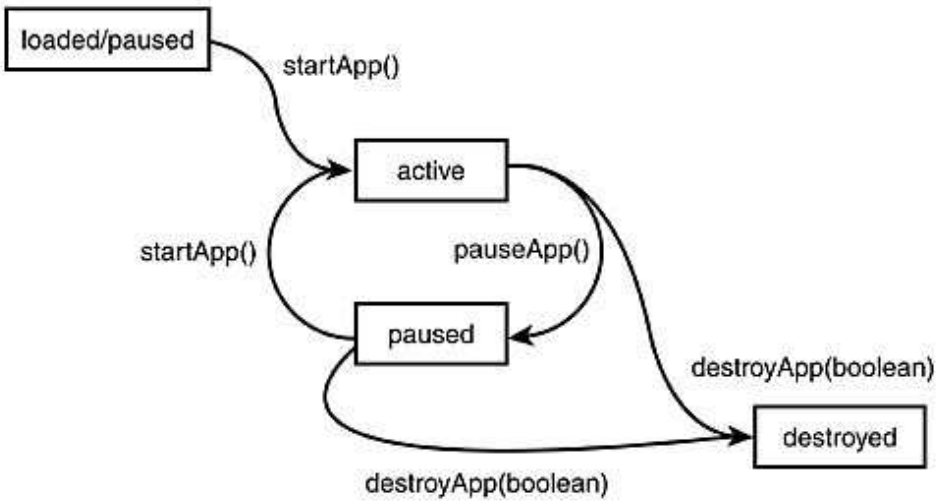
MIDP er en spesifikasjon for bruk av Java innebygd i mobiler og PDA'er og er en del av J2ME rammeverket. MIDP profilen ligger oppå CLDC konfigurasjonen og er et høynivå-API som ytterligere definerer applikasjonen. Applikasjoner laget for CLDC/MIDP miljøet blir kalt MIDlet'er.

MIDP 2.0 er en revidert utgave av 1.0 versjonen og har nytt forsterket brukergrensesnitt, multimedia og spillfunksjonalitet, bedre tilkoblingsmuligheter, Over-the-air (OTA) overføring og ende-til-ende sikkerhet.

Valgfrie java-pakker kan benyttes når applikasjoner har behov for annen funksjonalitet en det som tilbys som standard i noen av konfigurasjonene eller profilene. Problemet med å benytte tilleggspakkene er at man ofte ekskluderer en rekke mobiler som ikke har støtte for andre pakker utenom standard oppsettet.

En applikasjon laget for små enheter som mobiler og J2ME's virtuelle maskin(VM), kalles for en **MIDlet**. Når en MIDlet er laget kan den kjøres på enhver enhet som samsvarer med spesifikasjonen for J2ME teknologien. En MIDlet kan være lagret på et lagringssted og på forespørsel bli lastet ned *OTA* av en kompatibel enhet.

En MIDlet har en livssyklus som den går gjennom i dens levetid. Den starter ved at en *programadministrator*, kalt *Application Management Software* [21] gjør et kall til MIDlet'ens konstruktor og den starter da i statusen *loaded*. Når programadministratoren kaller startApp() metoden startes MIDlet'en og den går da videre til *active* tilstanden. Dette er frem til programadministrator kaller pauseApp(), hvor MIDlet'en pauses og venter på ny instruksjon. Til slutt vil programadministratoren kalle destroyApp(boolean), MIDlet'en har status *destroyed* og er terminert. Figur 3.15 viser livssyklus til en MIDlet.



Figur 3.15: Midlet livssyklus

JAD & JAR er to basiskomponenter man får når man lager en applikasjoner for J2ME. Sammen vil de utgjøre en *MIDlet suite*. Den ene av disse to komponentene virker noen ganger litt overflødig, men den har en viktig funksjon.

Java Application Descriptor er en fil som beskriver MIDlet'en, og peker til plasseringen til *J*ava *Ar*chive (JAR)'en. Videre vil Java Application Descriptor (JAD)'en gi navn på MIDlet suit'en, størrelsen på JAR filen, og hvilken konfigurasjon og profil den er basert på. Andre valgfrie egenskaper kan defineres av utvikler, eller utifra MIDP spesifikasjoner.

***J*ava *Ar*chive** er arkivet som inneholder selveste applikasjonen og de nødvendige tilleggsbibliotekene. I tillegg vil JAR'en inneholde en *manifest* fil, som spesifiserer mye av det samme som ligger i JAD filen. JAR filen vil i tillegg til Java klassefiler og manifestet, kunne inneholde andre ressurser, slik som bilder og andre filer.

J2ME's Wireless Toolkit (WTK) er en samling verktøy. For utvikling av applikasjoner for mobiler har SUN laget em samling verktøy som gir utviklere et emuleringsmiljø, dokumentasjon og eksempler til utvikling av Java applikasjoner for CLDC/MIDP mobiler.

[21] Application Management Software er den programvaren som styrer kjøringen av applikasjoner.

J2ME stiller med er meget sterkt utviklingsmiljø, tilgjengelig på de fleste mobiltelefoner på markedet i dag.

3.5 Brukermodellering

For å hjelpe en bruker eller tilby tjenester som er av nytte for brukeren ut fra konteksten rundt brukeren, er det nødvendig å ha en forståelse av hvem brukeren er. Når en utvikler et system har man alltid en "brukermodell", en idè om hvordan en modell av den generelle brukeren som vil benytte seg av systemet. Som Elaine Rich [30] sa;

> De fleste system som samhandler med mennesker inneholder, om enn bare implisitt, en form for modell av skapningene de kommer til å samhandler med.

Brukermodeller(user models) er en intensjon om å bygge opp standard modeller som reflekterer brukerene av et system. Modellene benyttes videre for å kunne så raskt som mulig finne ut hvilken type person en bruker er, for å da kunne tilrettelegge den funksjonalitet som systemet skal tilby til den enkelte bruker. Dette er også referert til som konstruksjon av *stereotyper* 3.5.2.

Innen feltet maskinlæring er brukermodeller vensentlige for å tilpasse seg den individuelle brukeren. I [24] viser Martin E. Müller til maskinlæringsfeltet, hvor metoder for å konstruere brukermodellene kan vise til noen av de bedre prestasjonsresultatene. Müller presiserer **prestasjon** som må oppfylle visse kvalitetsmål.

For *informasjonsgjenfinning* måles graden av *presisjon*, og *fremkalling* av korrekte verdier til brukermodellene. Videre måles representasjonen av brukermodellen med graden av *presisjon* og *dekning*. Prestasjon og ytelse måles også etter hvor effektivt systemet er.

Müller gjør i [24] en uvanlig benyttelse av en kjent maskinlæringsteknikk, *induktiv logisk programmering*(ILP), når de lager OySTER [22]. ILP er kort sagt, en teknikk for å konstruere nye predikater(logiske programmer) fra positive og negative eksempler. OySTER benytter en brukermodell for å raffinere søket til brukeren, før det sendes videre til andre søkemaskiner. Den samme modellen benyttes så for å filtrere søkeresultatene.

3.5.1 Brukerdimensjoner

For å bygge opp en brukermodell trenger en å ta stilling til ulike karakteristikker ved det å modellere en bruker. Elaine Rich beskrev i [29] tre dimensjoner som kan klassifisere brukermodeller:

[22] http://www.cogsci.uni-osnabrueck.de/ ai/oyster.html

- **generell vs. individuell bruker**
- **eksplisitt vs. implisitt modellering**
- **lang- vs. kortsiktig modell**

3.5.1.1 Generell vs. Individuell

Generell versus individuell brukermodellering baserer seg på hvor mange brukere som skal modelleres og hvor mange ulike brukertyper som vil være nødvendig.

Den generelle består i å finne en brukermodell som best passer "alle".

Individuell brukermodellering gir større fleksibilitet i forhold til å tilpasse system og brukerinteraksjon, og "skreddersy" brukergrensesnittet og samhandlingen mot brukeren. Den individuelle brukermodellen skal i motsetning til den generelle, oppdateres under systemets levetid og gjennom den samhandlingen som foregår mellom system og bruker.

3.5.1.2 Eksplisitt vs. Implisitt

Spørsmålet om bruk av eksplisitt eller implisitt modellering handler om hvordan systemet skal modellere og vedlikeholde brukermodellen. Skal informasjonen som skal gjøre opp brukermodellen hentes inn ved å spørre brukeren direkte. Eller skal systemet opprette og oppdatere brukermodellen ved å se på handlinger og den adferden brukeren utøver i sin interaksjon med systemet over tid.

Et enkelt eksempel på en eksplisitt modellering kan være at brukeren selv angir hvilken printer som skal være standard. En implisitt modellering vil her være at systemet etter en viss tid eller gjentakende valg av en annen printer, angir den printeren som standard.

Bruk av *kunstig intelligens*(AI) teknikker innen brukermodellering, brukes ofte for å implisitt modellere brukeren. Teknikker fra AI feltet kan bruker for å resonnere over en brukers tro og kunnskap, og fra dette anslå sannsynligheter for at brukeren har en gitt kunnskap.

3.5.1.3 Lang- vs. kortsiktig

En lang- eller kortsiktig modell handler om hvorvidt modellen skal representere hver bruker over tid eller med korte hurtige interaksjoner med en rekke forskjellige brukerere.

Brukermodeller med et langsiktig perspektiv vil ofte oppføre seg ulikt fra de med kort, fordi det stilles noe ulike krav til umiddelbar respons til læring og forandring i brukermodellen. Langsiktige brukermodeller vil kunne oppdatere seg ut fra brukerens adferd mer gradvis(selv om dette varierer) for å tilpasse seg mer riktig mot brukerens preferanser, uten å gjøre store forandringer ut fra enkelt handlinger.

3.5.2 Stereotyper

Stereotyper er en måte å simplifisere en kompleks virkelighet ved å trekke ut og standardisere likheter. Det er en helt naturlig funksjon som mennesker benytter for å takle de komplekse inntrykkene en treffer på. Stereotyper er bygd opp av et sett med attributter, eller egenskaper, som beskriver en standard definisjon for et individ eller en brukergruppe.

System som bruker stereotyper må også ha et sett med *utløsere*, som signaliserer at en spesiell hendelse har skjedd slik at de korrekte sluttninger kan tas. Et eksempel fra [29] er når en bruker benytter seg av avanserte funksjoner i et system, så kan dette være en utløser for at en *ekspert-bruker* stereotyp blir aktivert. Stereotypens egenskaper blir da lagt til brukeren.

Et eksempel på bruk av stereotyper under en eksplisitt brukermodellering kan være at brukeren selv angir hvilken "stereotype" han/hun anser seg selv for å være, ut fra noen gitte stereotyper. I en implisitt brukermodellering vil handlingene til brukeren gjøre at brukermodellen oppdaterer seg. Nye egenskaper legges til brukeren fra stereotyper, som etter brukerens handlinger, passer til brukeren [29].

Kapittel 4

Agent

Systemet skal utvikles som et multi-agentsystem og dette kapittelet tar derfor for seg agenter og hvordan en agent blir defineret gjennom dens egenskaper.

4.1 Definisjon

Det er gjort flere forsøk på å definere hva som er en agent og det har ofte vært heftige debatter om hva som er den riktige betegnelsen. Hvilke egenskaper må en agenten ha for å bli definert som en agent.

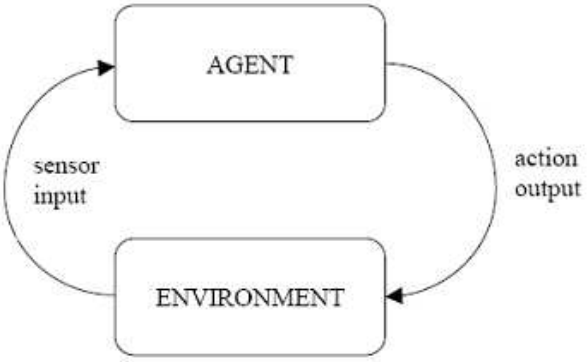
Noen kaller, riktig eller uriktig, en termostat på et varmeapparat for en agent. Den utfører sin oppgave uten menneskelig interaksjon, ved å reagere på en temperatur, og slår av eller på varmen hvis den er over eller under en gitt terskelverdi. Michael Wooldridge [43] gav en tilpasset definisjon, fra den tidligere publikasjonen [44].

> En *agent* er et datasystem som er *situert* i et *miljø*, og som er kapabel til *autonome* handlinger i dette miljøet for å oppnå sine mål.

En agent oppholder seg i et miljø, observerer via en eller flere sensorer og agerer i miljøet. Wooldridge viste i [43] en figur av en agents interaksjon med et miljø 4.1.

Agenter ofte med forsøkt definert med èn kort definisjon. Senere har disse blitt splittet opp utifra hvilke egenskaper de innehar. Michael Wooldridge viser i [44] hvordan han deler agentene opp i to generelle hovedgrupper, henholdsvis *svake* og *sterke* agenter. Egenskapene til en svak agent;

- **autonom**; agent som operer uten direkte innblanding av mennesker eller andre, og har en form for kontroll over egne gjerninger og intern tilstand.
- **sosial** agenter som interagerer med andre agenter(muligens mennesker) via en form for *agentkommunikasjonsspråk*



Figur 4.1: Agent og dens forhold til et miljø

- **reaktiv**: agenter oppfatter miljøet(som kan være den fysiske verden, en bruker via et brukergrensesnitt, en samling av andre agenter, Internett, eller muligens samtlige av disse, og responderer inne rimelig tid til forandringer som skjer i miljøet.
- **proaktiv**: agenter agerer ikke kun som en reaksjon til miljøet, men er også i stand til å *ta iniativ*.

I [44] vises det videre til sterke agenter, agenter som innehar mer mentale og menneskelige egenskaper som kunnskap, tro, intensjon og forpliktelse.

4.2 Agenttyper

Det finnes ulike typer agenter alt etter som hvilke egenskaper de har. Under beskrives det noen ulike typer agenter.

4.2.0.1 Autonome agenter

En velkjent definisjon av en autonom agent er gjort av Pattie Maes i [21].

> Autonome agenter er komputasjonelle systemet som holder til i et komplekst, dynamisk miljø, observerer og agerer autonomt i dette miljøet, og med dette realiserer et sett med mål eller oppgaver som de er designet for.

Autonome agenter er fullstendige agenter, som følger og setter opp sin egen plan, samt at de har sensorer og kan utføre handlinger i miljøet.

4.2.0.2 Intelligente agenter

Intelligente agenter er det neste steget i å benytte agenter til å utføre oppgaver uten for mye interaksjon med sin eier. Med intelligent menes det anlegg til å inferere og utføre aksjoner som behøves, samt å søke og inkorporere relevant informasjon, gitt visse mål. Det er en enhet som har sensorer i sitt miljø og som agerer i miljøet, samt gjør proaktive handlinger for å fremme målet sitt. Agenten må ha visse egenskaper for å komme under definisjonen av en *intelligent agent*. Egenskapene som kjennetegner en intelligent agent er;

- **tilpasningsevne**; agenten må kunne sanse og observere sitt miljø og endre eller tilpasse seg til det etter behov.
- **læringsegenskap**; agenten må lære seg miljøet og hvordan den skal reagere og handle i det. Dette skjer ofte gjennom *prøve-og-feile*(trial-and-error) metoden, hvor man ser på adferd i miljøet og suksessen på dette.

4.2.0.3 Mobile agenter

For klient-server [1] kommunikasjon vil det foregå nettverkskommunikasjon frem og tilbake mellom server og klient, som regel gjennom bruk av Remote Procedure Call (RPC) [2] teknikken. Dette kan være vanskelig hvis det ikke er gode nettverksforbindelser for klienten under hele interaksjonen.

En av de store fordelene til mobile agenter er at de kan forflytte seg til hvor de har en oppgave å utføre og returnere når oppgaven er utført, og med dette redusere nettverksbelastningen betraktelig. I et nettverk hvor forbindelsene kan variere og tilgjengelige aksesspunkt er sparsomme, vil mobile enheter minske nettverkstrafikken som normalt trengs for å utføre samme oppgaven.

Med mobile enheter og WLAN er det alltid en usikkerhet med nettverksforbindelsen og det skjer ofte brudd i nettverksforbindelsen. En mobil agent vil ikke ha behov for konstant nettverkstilgang og kan utføre oppgavene lokalt for så å avvente til det er mulig å opprette en nettverkstilkobling, og returnere med resultatet.

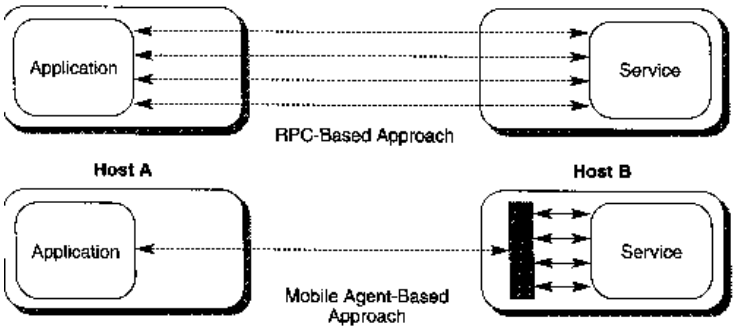
En illustrasjon over reduksjon av nettverkskommunikasjon vises i Figur 4.2.

4.3 Multiagentsystem

Multiagentsystem(Multi-Agent System (MAS)) er et system som har flere agenter som samhandler. Agentene i systemet er autonome og handler på vegne av en eier.

Kommunikasjonen består som oftest i å sende meldinger på et felles agentspråk. Agenter vil som oftest operere på vegne av sin eier og ha forskjellige motivasjoner

[1] Den mest vanlige arktitekturen for maskin til maskin kommunikasjon over et datanettverk.
[2] Teknikk for å opprette kommunikasjon mellom maskiner distribuert over et datanettverk.



Figur 4.2: Reduksjon av nettverksbelastning

og mål. Dette gjør at agentene vil få bruk for egenskaper for å samarbeide, koordinere og forhandle med andre agenter. Karakteristikken til et multiagentsystem er ifølge Katia Sycara [35];

- Hver agent har ufullstendig informasjon eller muligheter til å løse problemet og har et begrenset synspunkt.
- Det finnes ingen global systemkontroll.
- Data er desentralisert.
- Komputasjon er asynkron.

4.3.0.4 Fordeler og ulemeper med MAS

MAS har sine fordeler og ulemper og det er et felt i utforskningsfasen. Det har funnet flere utnyttelsesområder hvor det utfører sine oppgaver bedre og sikrere en før. Under vises noen fordeler og ulemper.

Fordeler med et MAS i forhold til en sentralisert tilnærming[36]:

- Multiagentsystem distribuerer sine ressurser over nettverket med sine sammenkoblede agenter. Et system med en enkelt agent er sentralisert og kan være utsatt for ressursbegrensninger, flaskehalser og kritiske feil. Ved å benytte et multiagentsystem kan man overkomme disse problemene.
- Et multiagentsystem vil klare seg hvis en agent svikter og faller bort.
- MAS tillater sammenkobling av flere eksisterende systemer ved å benytte en innkapslingsteknikk(*agent wrapper*), og en sammenkobling til et agentsystem.
- MAS' modellering av problemer, med autonome samarbeidende komponenter, viser seg å være en mer naturlig måte å håndtere problemene på.
- MAS øker systemytelsen, og spesielt med tanke på komputasjonell effiktivitet, påliteligheg, utvidbarhet, vedlikeholdbarhet, hurtig respons, fleksibilitet og gjenbruk.

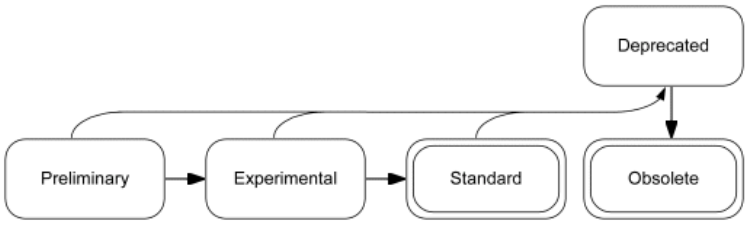
Ulemper med MAS, her hentet fra Viktor R. Lesser i [20]:

- Multiagentsystem har behov for en viss båndbredde og en god del av ressursene må gå til dette.
- Heterogeniteten til agentene, gjør det vanskelig å dele informasjon. Konkurranse mellom agenter og beskyttelse av egeninteresser stopper dem i å dele informasjon.
- Planlegging i MAS blir mye mer komplisert på grunn av flere agenter å ta med i beregningene.

4.4 The Foundation for Intelligent Physical Agents

The Foundation for Intelligent Physical Agents (FIPA) er en en IEEE standardiseringsorganisasjon for agenter og multiagent systemer og ble offisielt anerkjent i 2005. FIPA ble opprinnelig opprettet i 1996 som en sveitsisk organisasjon for å produsere standarder for heterogene og samhandlende agenter, og agentbaserte systemer. Det sveitsiske selskapet ble oppløst i 2005 og FIPA ble da akseptert som en IEEE standardiseringsgruppe for å legge til rette for integrasjon av agentbasert software med andre teknologier.

FIPA spesifikasjonene identifiseres etter hvilken posisjon de har i *spesifikasjonslivssyklusen*. Syklusen er til for å vise utviklingen for en spesifikasjon fra sin opprinnelse til den endelige løsningen [3]. Figur 4.3 viser livssyklusen.



Figur 4.3: En FIPA spesifikasjons livssyklus

Livssyklusens stadier fortoner seg slik;

- **Preliminary**; Statusen spesifikasjonen får i begynnelsen av et teknisk utvalg, som også genererer spesifikasjonen. Spesifikasjonen på dette stadiet blir sett på som et utkast under oppføring og går ofte gjennom mange revisjoner. Den blir ikke implementert på dette stadiet.
- **Experimental**; Status satt etter at en spesifikasjon er godkjent av *FIPA Architecture Board*. Spesifikasjoner som har fått denne statusen blir åpnet for implementasjon og testing på FIPA-støttende plattformer.
- **Standard**; Spesifikasjonen er nå implementert og testet tilstrekkelig av FIPA Archtiecture Board og FIPA medlemmene til å få denne statusen;

[3] http://www.fipa.org/specifications/lifecycle.html

Spesifikasjonen er nå ansett som stabil og er dermed egnet for implemetering på FIPA-støttende plattformer.

- **Deprecated**; Potensielt overflødige eller unødvendige spesifikasjoner blir satt som deprecated for en viss tid. Dette er for å sikre seg at spesifikasjonen virkelig er overflødig. Hvis spesifikasjonen derimot re-evalueres til fortsatt å være nødvendig, vil den bli satt tilbake til statusen den ble tatt ut fra.
- **Obsolete**; Spesifikasjoner blir ansett som "overflødig" på grunn av teknologiendringer eller endringer i andre FIPA-spesifikasjoner. Spesifikasjonen får etter en periode som *deprecated* denne statusen. Denne statusen er den endelige i livssyklusen.

Det finnes 25 spesifikasjoner som har fått statusen *Standard* i livssyklusen, men det finnes mange spesifikasjoner som enda ikke har oppnådd denne statusen. Spesifikasjonene er delt inn i kategorier etter hvilket emne i agentdomenet de er laget for. Kategoriene er *Agent Communication*, *Agent Message Transport*, *Agent Management*, *Abstract Architecture* og *Applications*. Av disse kategoriene er det agentkommunikasjon som skiller seg ut som kjernekategorien til FIPA.

4.5 Agentkommunikasjon

FIPA's arbeid på agentkommunikasjon(*Agent Communication Language (ACL)*), har gjort deres FIPA-ACL 4.5.1 til en velkjent og annerkjent spesifikasjon. Agentarktitekturer trenger både representasjon og kommunikasjonsmodeller for å tilrettelegge for agentkommunikasjon. For at en agent skal kunne kommunisere med andre agenter må den være i stand til å;

- oppdage eksistensen til en agent, nettverksadresse, egenskaper og/eller roller til agenter
- kommunisere med andre agenter gjennom et standardisert agentkommunikasjonsspråk.

For å muliggjøre oppdagelse av andre agenter stiller agentsystemene med navneservere og fasilitatorer. Agentene registrerer seg hos en navneserver og sine tjenester hos en fasilitator. Disse fungerer som *hvite* og *gule* sider; oppslagsverk for agentene. Når agenter har oppdaget hverandre må de kommunisere gjennom et felles agentspråk. Et agentspråk består i hovedsak av;

- **Agent Communication Language**: Et ytre språk som stiller et sett av applikasjonsuavhengige primitiver. Dette gjør at en agent kan ytre sin intensjon i et forsøk på å kommunisere med andre agenter. De fleste ACL baserer seg på en kommunikasjonstori og spesifiserer *performativer* slik som **ask**, **tell** og **achieve**.
- **Content Interchange Format**: Et indre språk, som tillater en agent å uttrykke sitt applikasjonsavhengige innhold med den hensikt å dele.

- **Ontology**: Domenespesifikt vokabular. Agenter vil ikke forstå hverandre, selv om de snakker samme språk, hvis de benytter forskjellige vokabular for å representere konsepter i sitt domene.

FIPA's kommunikasjonskategori for agenter, omhandler fire aspekter ved kommunikasjon;

- **ACL Messages**; Et språk som tillater agenter å kommunisere med hverandre ved hjelp av meldinger.
- **Interaction Protocol**; Modeller for typiske kommunikasjonssamhandlinger.
- **Communicative Acts**; Grunnblokker for kommunikasjon mellom agenter. Består av to typer; *primitive* og *sammensatte* handlinger.
- **Content Languages**; Beskrivelse av hvordan meldingen er kodet under sending mellom to agenter.

En *ontologi* vil i samarbeid med innholdsspråket(Content Language (CL)) støtte tolkningen av innholdet i en ACL melding. Ontologier vil bli beskrevet senere i dokumentet.

4.5.1 FIPA-ACL meldinger

Speech-act teorien er det de fleste ACL er basert på. Speech-act teorien kommer opprinnelig fra språkvitenskapen og er en teori for å forstå hvordan mennesker benytter språk. Speech-act omhandler hvordan språk benyttes for å utføre handlinger. Speech-act kategoriserer "yttringer" i ulike kategorier utifra intensjonen til den som ytrer meldingen, effekten meldingen har på mottakeren og ethvert fysisk utslag meldingen ytrer.

4.5.1.1 FIPA-ACL Message Structure Specification

Denne spesifikasjonen er en foreslått standard for agent-kommunikasjon, og definerer et sett parametere som kan være i en FIPA melding [10]. Det eneste parameteret som er obligatorisk er *performative*. Tabell 4.1 viser parameterene:

Parameter	Kategori
performative	Type kommunikativ handling
sender	Deltaker i kommunikasjon
receiver	Deltaker i kommunikasjon
reply-to	Deltaker i kommunikasjon
content	Innhold i meldingen
language	Beskrivelse av innhold

Fortsetter på neste side.

```
(inform
  :sender x
  :receiver y
  :language FIPA-SL
  :ontology weatherForecast
  :protocol query-if
  :content
    (raining (location Trondheim))
)
```

Figur 4.4: Agent Communication Language melding etter FIPA-ACL standard.

Tabell 4.1 – forts. fra forrige side.	
Parameter	Kategori
encoding	Beskrivelse av innhold
ontology	Beskrivelse av innhold
protocol	Kontroll over konversasjon
conversation-id	Kontroll over konversasjon
reply-with	Kontroll over konversasjon
in-reply-to	Kontroll over konversasjon
reply-by	Kontroll over konversasjon

Tabell 4.1: FIPA-ACL meldingsstruktur.

Hvilke en velger å benytte er avhengig av situasjon, men som oftest vil *sender* og *receiver* være utfyllt. Det er også viktig i mange tilfeller å meddele hvilket språk og koding(*encoding*) som benyttes, for å sikre at innholdet kan tolkes. *Conversation-id* benyttes under konversering og sørger for at meldingene som mottas faktisk hører til samme konversasjon. Figur 4.4 viser et eksempel på en melding som følger FIPA-ACL standard, hvor en agent **x** spør agent **y** om det regner i Trondheim.

4.5.2 Interaksjonsprotokoller

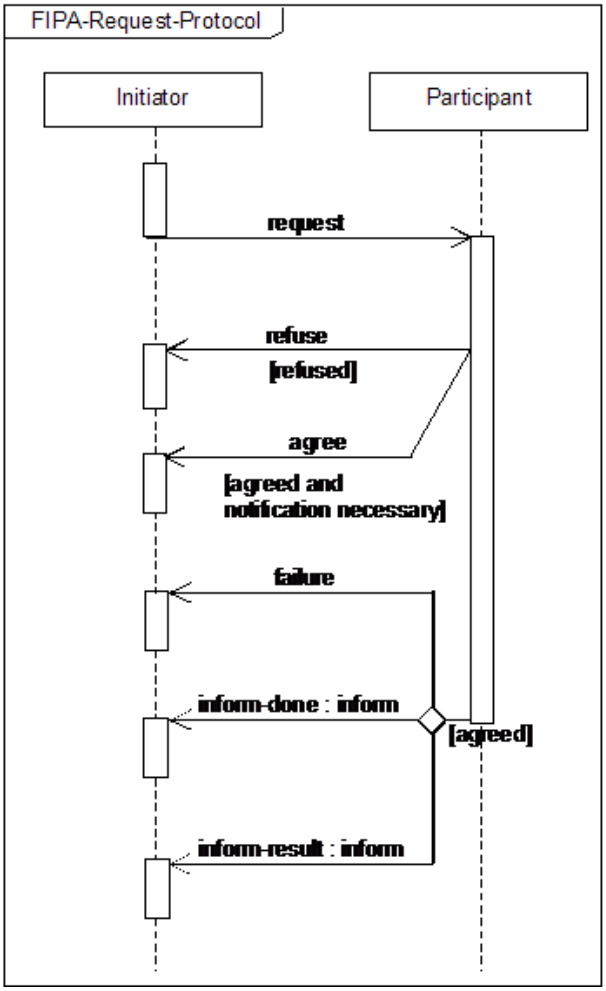
Interaksjonsprotokoller(Interaction Protocol (IP)) er standardiserte måter å kommunisere mellom to eller flere parter. Protokollene spesifiserer hvordan agentene skal utføre ACL meldingsutveksling med andre agenter for å oppnå ulike resultat. FIPA har standardisert et utvalg protokoller for de mest vanlige scenarioer, hvor et utvalg av dem er:

- **FIPA Request**; lar en agent be en annen om å utføre en handling.
- **FIPA Query**; lar en agent be om å få utføre en handling på en annen agent.
- **FIPA Contract-Net**; lar en agent spørre flere agenter om å utføre en handling. Deltakerene vil svare nei, eller gi et "tilbud" på beste "pris" for

å utføre handlingen.

- **FIPA Subscribe**; lar en agent forespørre mottakeren om å utfør en en handling på et senere tidspunkt, når et referert objekt forandres. Objektet refereres til i ACL meldingen.

Figur 4.5 viser en representasjon av interaksjonsprotokollen FIPA Request. For å benytte en slik protokoll vil man sette sette *fipa-request* i protokollfeltet i en ACL-melding [13].



Figur 4.5: FIPA Request interaksjonsprotokoll.

Interaksjonen i figuren 4.5 viser en *Initiator* som initierer samhandlingen. Dette gjøres med en ACL melding til en annen agent, deltakeren(participant), hvor protokollfeltet er fyllt inn med *fipa-request*. Deltakeren kan enten avvise(refuse) eller godta(agree) forespørselen fra initiatoren. Hvis deltakeren godtar og utfører handlingen, avslutter den protokollen ved å sende èn av tre mulige svar til initiatoren. Meldingen kan inneholde;

- **failure**; deltaker feiler i sitt forsøk på å oppfylle initiatorens forespørsel.

```
(query-if
    :sender (agent-identifier :name i)
    :receiver (set (agent-identitfier :name j))
    :content
      "((registered (server d1) (agent j)))"
    :reply-with r09
    ...)
```

Figur 4.6: Eksempel på en *query-if*, kommunikativ handling.

```
(inform
    :sender (agent-identifier :name j)
    :receiver (set (agent-identifier :name i))
    :content
      "((not (registered (server d1) (agent j))))"
    :in-reply-to r09)
```

Figur 4.7: Respons på *query-if* kommunikativ handling.

- **inform-done: inform**; deltaker fullfører forespørselen, og gir respons til initiator om at forespørsel er fullført.
- **inform-result: inform**; deltaker vil indikere fullført forespørsel, og gi initiatoren resultatet.

4.5.3 Communicative Acts

FIPA har standardisert et bibliotek bestående av 22 kommunikative handlinger for å fremstille forskjellige intensjoner med kommunikasjonen. FIPA's kommunikative handlinger forteller intensjonen med meldingen, men er uavhengig av innholdet i meldingen. De kommunikative handlingene er, som nevnt tidligere, delt opp i to klasser:

- **Primitive**; atomiske handlinger, bestående av kun én kommunikativ handling. Angir kommunikative handlinger slik som *inform* som forteller om en agents intensjon til å informere om noe den vet.
- **Sammensatte**; satt sammen av flere kommunikative handlinger. Operatorene for sammensetting er *komposisjonsoperatorer* slik som ";" for sekvenser, eller "—" for disjunkte handlinger [12]. *query-if*, består av *request* og *inform* i en sekvens, og får meningen "Jeg *forespør* at du *informerer* meg om det regner".

Eksempelet i Figur 4.6 viser hvordan i (initiator) forespør j (deltaker), om j er registrert hos domeneserver *d1*. I den kommunikative handlingen *query-if* innebærer dette at initiator i forespør j om å informere(*inform*) hvorvidt proposisjonen er sann.

Videre i Figur 4.7 svarer agent j at det er det ikke(**not**).

4.5.4 Content Language

Content Language, eller *innholdsspråk*, beskriver hvordan ACL meldinger blir omkodet før sending.

4.6 Ontologi

En ontologi er en *eksplisitt spesifikasjon av en konseptualisering* [16]. Ontologier gir domenebaserte vokabularer for representasjon og kommunikasjon av kunnskap om et emne, og i tillegg et sett relasjoner og egenskaper angitt av dette vokabularet. Agenter benytter ontologier for å beskrive konsepter når de skal dele kunnskap. For et gitt domene sikrer dette at en term, et objekt eller en entitet, har en uniform mening blant alle agenter. En velkjent definisjon av en ontologi, fremlagt av James Hendler [17] siteres under:

> En ontologi er en formell definisjon på en samling av kunnskap. Den mest typiske typen av ontologier i bruk under konstruksjon av agenter involverer en strukturell komponent. Hovedsakelig en klassifikasjon av klasse- og subklasse-relasjoner kombinert med definisjoner på relasjonene mellom disse tingene.

Ontologien benyttes i et agentsystem for å inngå en "ontologisk forpliktelse" mellom agenter, hvor en ontologisk forpliktelse er en avtale om å benytte et vokabular på en måte som er konsistent i henhold til teorien spesifisert av en ontologi [16].

Utvikling og benyttelse av ontologier har etter hvert beveget seg fra AI området og ut til andre kunnskapsfelt. Domeneeksperter benytter nå ontologier for å kommentere og dele informasjon i deres felt.

4.6.1 Representasjon

Representasjon av ontologier kan gjøres på to forskjellige måter, hvor en enten benytter seg av Web Ontology Language (OWL) eller Frame Ontology.

- **OWL** er et markup-språk for å definere og instansiere ontologier på Web [4]. Ontologier laget i OWL kan inneholde beskrivelser av klasser, samt egenskapene på klassene og eventuelle instanser.
- **Frames Ontology** definerer basiskonsepter for en rikholdig objekt-orientert representasjon. Beskrivelsen her er hentet fra *Protégé*'s implementasjon av Frames Ontology. Protégé-Frames består i et sett klasser organisert i et hierarki, hvor det representerer et domenes fremtredende konsepter. Videre

[4] World Wide Web(Web)

er det et sett av felt, assosiert til klasser for å beskrivere deres egenskaper og relasjoner [5].

En ontologi vil sammen med instanser av klassene utgjøre en kunnskapsbase.

4.7 Java Agent DEvelopment Framework

Java Agent DEvelopment Framework (JADE) er et rammeverk for agenter distribuert over et nettverk og på mobile terminaler. JADE er open-source, agent-orientert *middleware* [6] som muligjør konstruksjonen av uavhengige og autonome prosesser, som med sine egenskaper utgjør agenter. JADE rammeverket består av to hovedkomponenter; en FIPA-oppfyllende agent-plattform og en pakke for å utvikle Java agenter [2].

4.7.1 Java Agent DEvelopment Framework kjernen

JADE plattformen er bygget på en distribuert kjerne som støtter basis plattform-nivå operasjoner som *meldingslevering*, og *livssyklus-administrasjon*. Den nåværende plattformen er tredje versjon, og utviklingen fra de to første generasjonene har gitt mer modularitet og støtte for hele Java familien. I de to foregående versjonene hadde *første generasjon* kun støtte for J2SE miljøet. *Andre generasjon* utvidet til hele Java familien og gav mer modularitet ved å kunne velge hvilke egenskaper som ble inkludert.

Tredje og nåværende versjon splittet kjernen opp slik at den forutsette modulariteten ble en realitet. Strategien deres var rettet mot den en hybrid løsning av trådløs og fast nettverksdistribusjon.

Den nye kjernen introduserte *Services*, eller tjenester, en ny strukturell komponent som grupperte et sett av egenskaper utifra en kontekstuell sammenheng. Eksempel på slike tjenester er *agent administrasjon* og *event notification*. Det er slike tjenester som distribueres og sammen utgjør plattformen.

Andre steg var å forandre på *Container* 4.7.3.6'en. Den nye versjonen fikk en utvidet semantikk, hvor den i tillegg til å holde agenter nå også inneholdt de nye tjenestene.

For utviklingen av dette nye designet laget de en liste for den nye arkitekturen[7].

- Forskjellige tjenester kan bli satt sammen.
- Ethvert subsett av de tilgjengelige tjenester kan bli tatt i bruk på enhver Container, og kan ved behov fjerne eller legge til en tjeneste under normal kjøring av plattformen.

[5] http://protege.stanford.edu/overview/protege-frames.html

[6] Middleware er en softwarekomponent som tilrettelegger for kommunikasjon mellom to ulike typer software applikasjoner.

- Noen tjenester kan være tilgjengelig på kun et subsett av containerene, eller kan være tilgjengelig med ulike tjeneste-nivå på forskjellige containere.
- Tjenestene kan kjøres på forskjellige Java versjoner støttet av JADE, muligens med en gradivis degrasjon på ressurssvake enheter.

Applikasjoner laget i JADE vil opprette kun én java applikasjon på hver host og derfor kun én JVM. Nye agenter er implementert som java tråder og tilhører en *Container* som gir agenten mulighet til å kjøre. For å utføre sine oppgaver kan prosessen kan være vedholdende og kreve kommunikasjon for å samarbeide eller kanskje konkurrere med andre agenter.

4.7.2 Agent Management Specification

JADE er som sagt en FIPA-oppfyllende plattform og består derfor av komponenter etter FIPA's spesifikasjoner. Under vises *FIPA Agent Management Specifikation*, spesifikasjonen for håndtering av agenter, satt opp av FIPA [15].

4.7.3 Agent Management Reference

Agent Management Reference Model(AMR) er det normgivende rammeverket FIPA agenter eksisterer og opererer på. Det oppretter den logiske referansemodellen for oppretting, registrering, lokasjon, kommunikasjon, forflytning og avslutning av agenter. De forskjellige logiske komponentene i AMR er agenter og fungerer som tjenester(services). To agenter som kan ses på som hovedkomponenter, er *Agent Management System (AMS)* og *DF* agentene.

4.7.3.1 Directory Facilitator

Directory Facilitator er en av de obligatoriske komponentene til en Agent Platform (AP) og er en agent som fungerer som et oppslagsverk for agentene, ofte beskrevet som agentenes gule sider. Agentene som vil tilby sine tjenester kan melde seg til DF slik at andre agenter som trenger disse tjenestene kan forhøre seg hos DF for å finne frem til den riktige agenten. Dens oppgave er å sørge for oppdatert, komplett og tidsriktig liste over agenter på AP. Det må være minst én DF på hver AP til enhver tid. En AP kan holde så mange DF'er som det er behov for, og de kan registrere seg hos hverandre.

Tjenester som DF'en tilbyr er *register*, *deregister*, *modify* og *search*. Disse tillater andre agenter å registrere sine tjenester, samt å søke etter andre agenters tjenester.

4.7.3.2 Agent Management System

AMS er en obligatorisk komponent(agent) på en AP. Det vil kun være én AMS

per AP til enhver tid. AMS'ens oppgaver er å administrere all virksomhet på en AP, slik som å skape nye agenter, slette agenter, samt å avgjøre om en agent kan dynamisk registrere seg hos en DF. Hvis *mobilitet*, flytting av agenter, er støttet av AP, vil AMS'en håndtere dette. Når agenter registrerer seg hos AMS vil de få en gyldig agent-id(AID). AMS tilbyr også *white-pages* tjenester for andre agenter.

4.7.3.3 Message Transport Service

Message Transport Service er tjenesten som kontrollerer agentens private kø med ACL meldinger. Message Transport Service (MTS)'en håndterer transporten slik at agenten ikke trenger bry seg om på hvilken måte meldinger må sendes. MTS'en velger den transportmekanismen som gir minst mulige kostnader ved sending. Den benytter flere forskjellige protokoller for transport av meldinger, slik som *HTTP*, *Remote Method Invocation (RMI)* [7] og *Internet Inter-ORB Protocol (IIOP)* [8].

4.7.3.4 Agent Communication Channel

Agent Communication Channel (ACC) er en agent som forsyner en tjeneste direkte til agenter på en AP. Denne forsyner kommunikasjonskanalen og sørger for kommunikasjon mellom agenter på og utenfor AP. ACC har tilgang til informasjon fra de andre AP tjenestene, slik som AMS og DF for å utføre sine oppgaver med å transportere meldinger[11].

4.7.3.5 Agent Platform

Agent Platform forsyner den fysiske infrastrukturen hvorpå agenter kan plasseres. AP består av maskinene, operativsystemet, tilleggsprogramvare til agenter, FIPA administratorkomponenter(DF, AMS og MTS) og alle andre agenter. FIPA tar kun for seg hvordan kommunikasjon mellom agenter foregår, både for de som hører til den lokale AP og agenter utenfor.

4.7.3.6 Container

Container er betegnelsen for kjøremiljøet for agentene, og er en Virtual Machine (VM). Dette er også kjøremiljøet for JADE plattformen. Det er, og må være, kun én *Main-Container*, eller hovedbeholder. Main-Container holder et sentralt register for informasjon og tjenester for hele JADE plattformen, og den holder AMS og DF tjenestene. Det kan opprettes flere beholdere, som da blir distribuerte noder på flere maskiner i et nettverk. Disse blir *Container*'e underlagt Main-Container, og er distribuerte AP'er i seg selv.

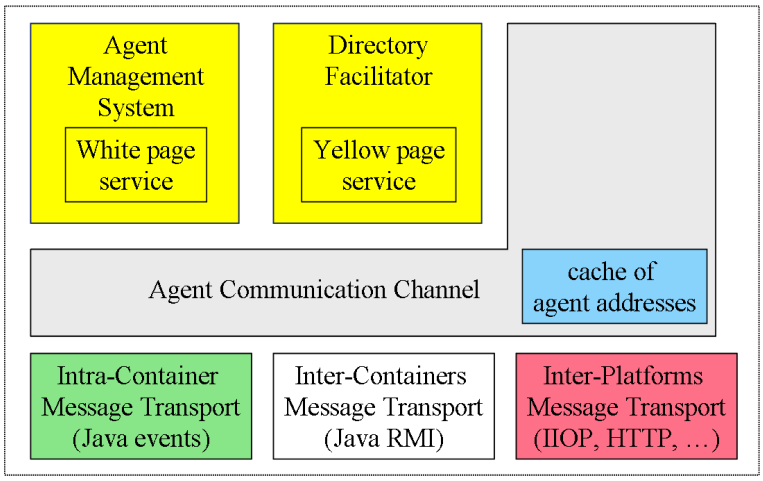
[7] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
[8] http://www.omg.org/news/whitepapers/iiop.htm

4.7.3.7 Message Transport Protocol

Message Transport Protocol er transportlaget som sørger for fysisk kommunikasjon mellom lokale agenter og eksterne agenter eller applikasjoner. Dette transportlaget skjuler logikken ved transport av kommunikasjonsobjekter. MTS blir tilbydd på enhver AP av en ACC og benytter seg av Message Transport Protocol (MTP) for å overføre FIPA ACL meldinger mellom agenter på hvilken som helst AP.

Det finnes flere ulike MTP'er som støtter ulike FIPA-oppfyllende transport protokoller, slik som HTTP MTP som gir støtte for meldinger med HTTP protokollen. JADE benytter seg av IIOP MTP som default protokoll for kommunikasjon mellom agentene.

Som en oppsummering av JADE's interne arkitektur og de komponentene som til sammen gjør opp JADE plattformen, vises Figuren 4.8.



Figur 4.8: JADE intern arkitektur.

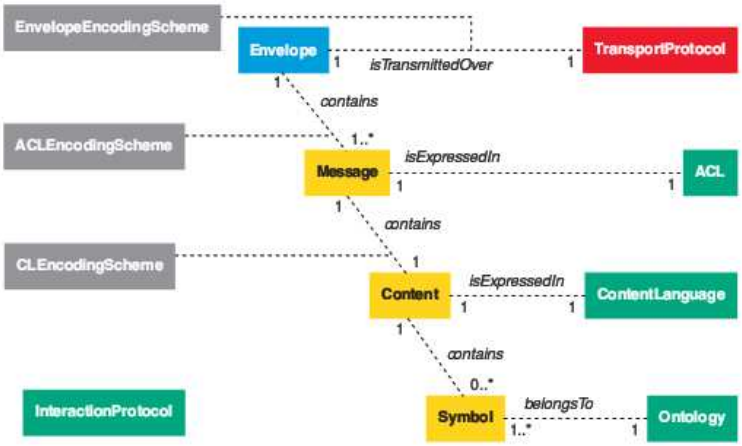
4.7.4 Kommunikasjonsmodell

JADE/FIPA har en kommunikasjonsmodell for interaksjon mellom agenter i deres multiagentsystem [14]. Agenter sender og mottar Java objekter som representerer ACL meldinger og følger protokollene for interaksjon. JADE skjuler koding og dekoding av alle meldinger. Figur 4.9 viser FIPA's kommunikasjonsmodell.

4.7.5 Tilstander

Agenter implementert i JADE kan være en av mange tilstander(states) som følger FIPA's spesifikasjoner. Tilstandene som er mulig å være i er:

Initiert Agenten er laget, men har ikke registrert seg selv hos AMS, den har



Figur 4.9: FIPA standard: komponenter i kommunikasjonsmodellen.

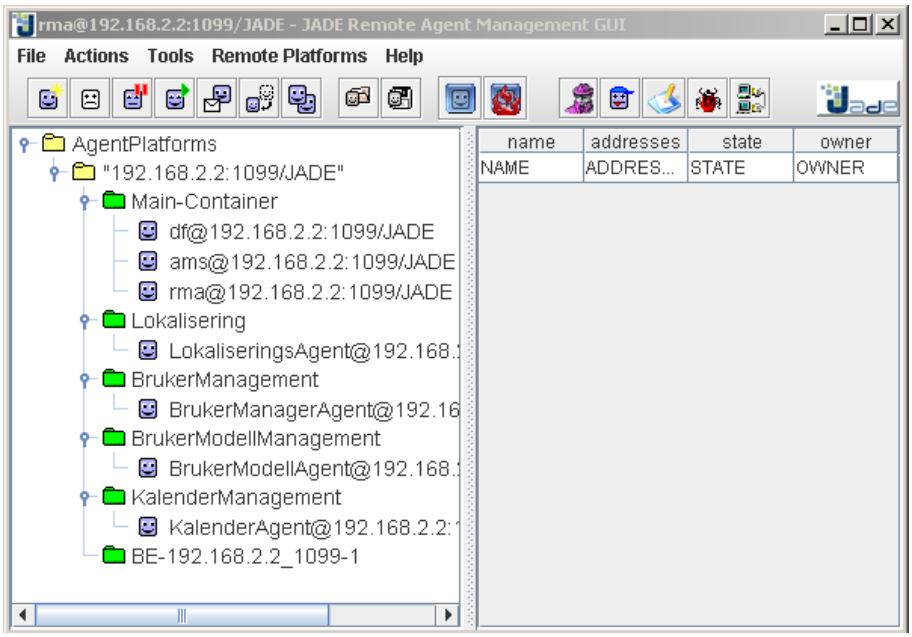
heller ikke et navn eller en adresse og kan da ikke kommunisere med andre agenter.

- **Aktiv** Agenten er registrert hos AMS, har et navn, en adresse og kan aksessere tjenestene i JADE.
- **Avbrutt** Agenten er midlertidig stoppet, den interne tråden er midlertidig stoppet og det eksekveres ingen agent-atferd.
- **Ventende** Agenten er blokkert, venter på noe. Den interne tråden er stoppet, mens den venter på at en forutsetning er møtt(typisk; en melding ankommer).
- **Slettet** Agenten er død. Den interne tråden har terminert sin eksekvering og agenten er ikke lengre registrert hos AMS.
- **Transport** Mobile agenter går inn i denne tilstanden når de forflytter seg til en ny destinasjon. Systemet vil lagre meldinger til den i et buffer, som deretter sendes til agentens nye destinasjon.

Agenten vil gå inn i disse tilstandene ettersom hva agenten skal utføre i løpet av sin livssyklus. Den kan kun utføre oppgaver når den er satt i "aktiv" tilstand.

4.7.6 JADE GUI

For å administrere og overvåke kjøring av agenter stiller JADE med et sett grafiske verktøy, eller et GUI. Dette grafiske verktøyene er implementert som en agent, kalt *Remote Management Agent (RMA)*. Alle funksjonene i grensesnittet er implementer som agenter i seg selv, og utfører spesifikke administrative eller overvåkende oppgaver. JADE stiller med to grafiske verktøy, *IntrospectorAgent* og *DummyAgent*, for monitorering og debugging av agenter. Figur 4.10 viser det grafiske grensesnittet til RMA.



Figur 4.10: JADE RMA GUI.

4.7.7 Light Extensible Agent Platform

Light Extensible Agent Platform (LEAP) er en lettversjon av JADE. JADE-LEAP, også kalt *JADE powered by LEAP*(JADE powered by LEAP (JADEpL)), er laget for å kjøre på en mobil eller mobile enheter hvor det er behov for et sterkt tilpasset miljø. Dette gjøres ved å strippe bort deler av JADE rammeverket som gir mindre kode, et redusert *footprint* [9], og forminske bruk av lagringsplass og et mindre minneforbruk. Mobilens prosesseringskraft og minne under kjøring er sterkt redusert i forhold til stasjonære maskiner, samt at det er begrenset lagringskapasitet, og ikke minst kort batterilevetid. Den korte batterilevetiden gjør at man ikke ønsker å være tilkoblet nettverket med et trådløst nettverkskort over lengre perioder da denne kommunikasjonen tapper batteriet hurtig. En agent kan residere på en mobil enhet og tilkoble seg med gjevne mellomrom for å oppdatere seg mot agentplattformen.

De funksjonelle egenskapene til LEAP er [3]:

1. Den kjører på stasjonære PC'er og håndholdte enheter med begrensede ressurser, som PDA'er og Java-enabled mobiltelefoner.

2. Den benytter trådløst nettverk, som TCP/IP over GSM og IEEE 802.11 WLAN, som garanterer tilkobling for trådløse enheter.

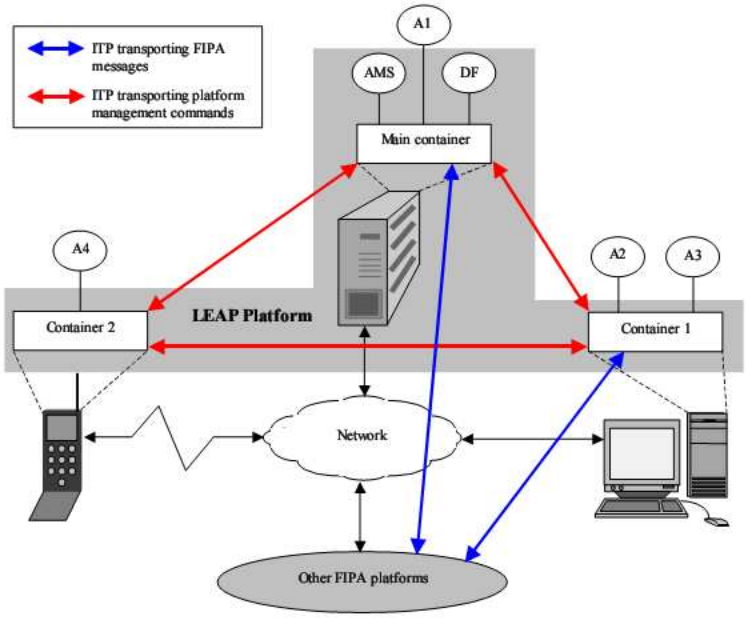
3. Den er uavhengig av operativsystem, ved at den benytter Java som støttes av de fleste operativsystem.

[9] Samlebetegnelse for de ressurser en applikasjon benytter.

4.7.7.1 Kommunikasjon

Kommunikasjon i LEAP blir administrert slik at dynamikken rundt sending av meldinger er fleksibel i forhold til plattformen, men enkel for utvikleren. Protokoller for kommunikasjon, blir kalt *Internal Transport Protocol*(ITP) [3], og LEAP håndterer nå ulike protokoller, som hver for seg håndterer ulike situasjoner. Protokollene som LEAP støtter nå er de samme som ble nevnt tidligere for JADE, og var IIOP, Java RMI og proprietære protokoller over TCP/IP.

For håndholdte enheter derimot er det kun den siste som er praktisk i bruk, selv om implementasjoner av Common Object Request Broker Architecture (CORBA) [10] vil være tilgjengelig. Kommunikasjonsmekanismene i LEAP sørger allikevel for en sømløs håndtering av den kommunikasjon som måtte foregå mellom plattform administrasjonen, AMS, og mellom agentene. Figur 4.11 viser hvordan Internal Transport Protocol (ITP) håndterer kommunikasjon mellom Main-Container på en servermaskin, en stasjonær maskin og 1 mobiltelefon.



Figur 4.11: LEAP Internal Transport Protocol.

Figur 4.11 viser hvordan den ITP som er tilgjengelig på to Containere, benyttes for kommunikasjon mellom dem. På figuren vises dette med en **rød** pil mellom Containere hvor det her transporteres interne administrasjonsmeldinger, uten bruk av IIOP. **Blå** pil viser hvor det er støtte for IIOP protokollen, og derfor direkte kommunikasjon med andre FIPA plattformer. Der hvor det ikke er noen slik ITP tilgjengelig, vil Main-Container fungere som en ruter, og videresende meldinger mellom Containere. I figuren vises dette mellom Container−2 og den eksterne FIPA plattformen.

[10] CORBA er en standard som tillater kommunikasjon mellom programvarekomponenter på forskjellige programvarespråk.

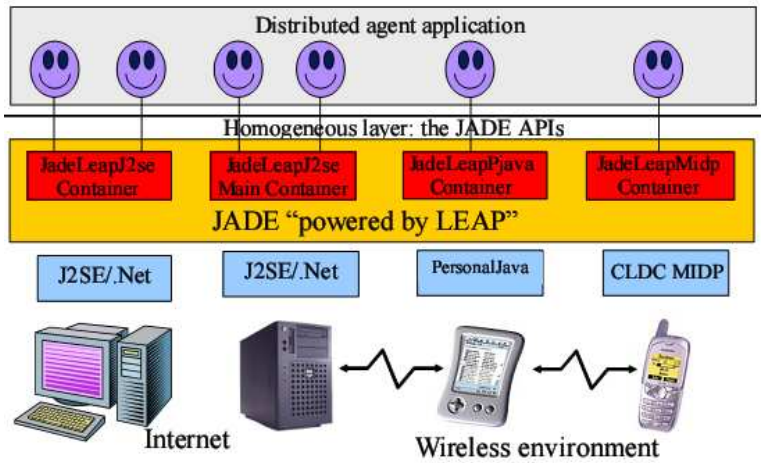
4.7.7.2 JADE powered by LEAP kjøremiljø

LEAP er levert som en add-on til JADE og tilpasser JADE til det kjøringsmiljøet det skal være på [6]. Dette gjør man ved konfigurering og kompilering av JADEpL på serveren slik at kjøringsmiljøet de mobile enhetene skal koble seg til er kompatibelt med deres miljø. LEAP add-on vil kombinert med JADE bytte ut noen komponenter i JADE kjernen og det modifisert runtime miljøet blir da JADEpL. Det er mulig å konfigurere JADEpL til tre typer Javamiljø, for tre ulike typer enheter;

- **j2se**; for å kjøre JADEpL på PC og servere på det faste nettet.
- **pjava**: for å kjøre JADEpL på håndholdte enheter som støtter J2ME med CDC eller *PersonalJava* konfigurasjonene for større enheter som PDA'er.
- **midp** for å kjøre på små håndholdte enheter med MIDP 1.0(eller senere) slik som mobiler i dag.
- **.NET**; for å kjøre JADEpL på PC og servere, som kjører Microsoft .NET Framework 1.1 eller nyere.

4.7.7.3 LEAP API

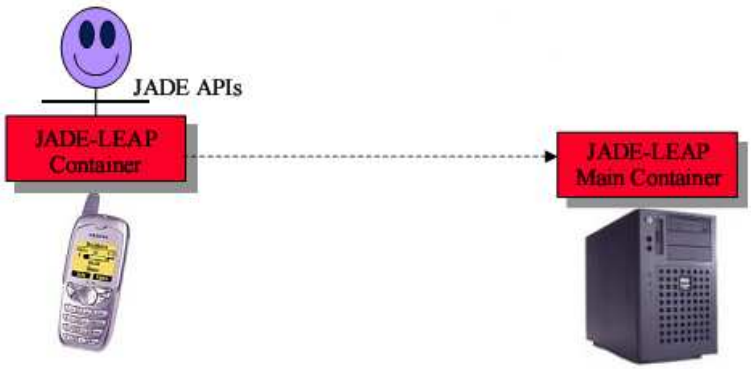
API'et til JADEpL er det samme som JADE og dette gjør at omstillingen til å programmere for JADEpL er liten. Dette skal gjøre at utviklere kan utvikle agenter uten for mye tanke på hvor den skal implementeres, og også benytte applikasjoner laget for JADE uten å måtte omskrive koden. Figur 4.12 viser de ulike versjonene, og hvordan de benytter seg av det samme API'et av JADEpL.



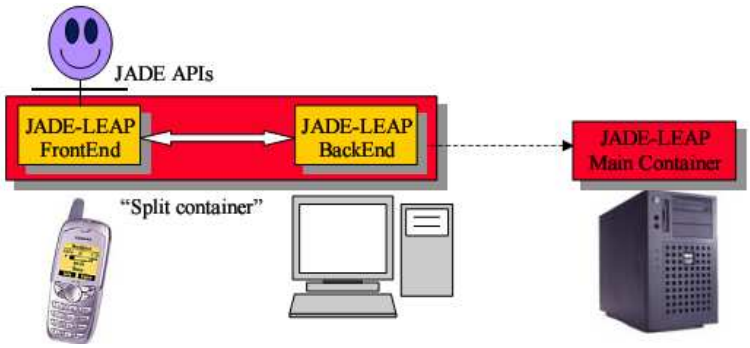
Figur 4.12: JADE powered by LEAP kjøremiljø.

4.7.7.4 Split vs. Stand-alone

For å håndtere ressursbegrensningene til de bærbare enhetene LEAP var ment for, er det utviklet to forskjellige kjøremoduser. Det er den normale *Stand-alone*, hvor en komplett Container kjøres på en maskin hvor JADE kjøremiljøet ble aktivtert. Den andre er *Split*, hvor Container'en blir splittet opp i en *FrontEnd*, og en *BackEnd*. FrontEnd kjører der hvor JADE kjøremiljøet ble aktivert, da ofte en mobiltelefon eller PDA. BackEnd Container'en blir satt opp på en servermaskin, som er koblet til en permanent Internett-tilkobling. Figur 4.13 viser JADEpL's Stand-alone modus, og figuren 4.14 viser Split modus.



Figur 4.13: JADE powered by LEAP *Stand-alone*.



Figur 4.14: JADE powered by LEAP *Split*.

Split modusen er en lettvekt-versjon av en komplett Container, og har en del trekk som gjør at den egner seg for ressursbegrensede enheter [6]. Oppstartsfasen er mye raskere på grunn av at det meste av kommunikasjon under oppstart, foregår mellom BackEnd og Main-Container, og oppstarten spares derfor for trådløs kommunikasjon.

JADEpL har ingen sikkerhet i kommunikasjon mellom FrontEnd og BackEnd, slik at når en benytter MIDP versjonen er dette en svakhet i agentsystemet. Protokollen for kommunikasjon mellom FrontEnd og BackEnd, er en proprietær protokoll kallt **JICP!** (**JICP!**) som går over Transmission Control Protocol (TCP). Denne kommunikasjonen er ikke mulig å gjøre sikker, ved for eksempel å benytte SSL, på

grunn av de begrensningene som er på MIDP mobiltelefoner. Det finnes *add-ons* for JADE, men de støtter ikke Split konfigurasjonen av JADE [11].

Add-on *JADE-S* for JADE er et tillegg som gir muligheter for sikkerhet i multiagentsystem [5]. Sikkerheten innebærer blant annet kryptering og signering av meldinger. Det finnes også autentisering som skal sikre at en bruker som starter en JADE plattform, med Containere og agenter, er en legitimert bruker i forhold til systemet som kjører Main-Container'en. Dette gjennomføres ved at det skal gis brukernavn og passord til Main-Container når den nye plattformen startes.

Den gjennomgåtte teorien har økt forståelsen for problemet og for teknologien som utgjør IH. Dette gjør det mulig å gå videre med spesifikasjon av krav til IH. og til å gjnnomføre en analyse og design av systemet.

[11] http://avalon.tilab.com/pipermail/jade-develop/2006q1/008213.html

Kapittel 5

Intelligent Helper

Dette kapittelet tar for seg analyse og design av *Intelligent Helper*.

Seksjon 5.1 presenterer metoden som er brukt under utviklingen av IH.

Seksjon 5.2 presenterer noen designvalg som er tatt.

Seksjon 5.3 presenterer første del av metoden som er analysefasen og og kravspesifikasjonen til IH.

Seksjon 5.4 presenterer designfasen.

5.1 Metode

Analyse og design av IH følger en metode for utvikling av multiagentsystemer på JADE rammeverket [26]. Metoden er utviklet av Magid Nikraz, og gjennom et samarbeid mellom en rekke forfattere, fra blant annet Telecom Italia Lab [1],grunnleggeren av JADE.

Metoden benytter seg til dels av Unified Modelling Language (UML) notasjon og noe videreutviklet notasjon som er tilpasset agentdomenet. UML's bruksmønster notasjonen brukes for det meste til objektorientert systemutvikling, men dette er ikke en restriksjon for UML. Under agentutviklingen kan man benytte seg av UML's brukesmønster uten modifikasjon. Forklaringen av metodens steg blir gjort under design av systemet.

Denne metoden er ikke helt ferdigutviklet og har ikke med første naturlig steg, planlegging og heller ikke siste, implementasjon og testing. UML notasjonen ble brukt for å støtte systemutviklingen der hvor metoden ikke dekket opp.

[1] http://www.telecomitalialab.com

5.1.1 Unified Modelling Language

UML er et generellt, standardisert modelleringsspråk for *objektorientert systemutvikling*, og et grafisk språk som definerer ulike elementer for å beskrive et system og systemdeler på ulike abstraksjonsnivå.

UML ble utviklet av Jacobson, rumbaugh og Booch i begynnelsen av 1990-årene, men ble overtatt som et standardiseringsobjekt av *Object Management Group (OMG)* [2] i 1997. OMG er en organisasjon for godkjenning av nye standarder innen objektorienterte aktiviteter [18]. UML inneholder en rekke ulike modelleringstyper, hvor noen er: *klassediagram*, *sekvensdiagram*, *bruksmønsterdiagram*, *aktivitetsdiagram*.

Siden den første versjonen, UML 1.0, som kom ut i 1997, har det etter hvert blitt reviderte versjoner, frem til nå hvor UML 2.0 er den siste videreutviklingen.

5.2 Designvalg

Gjennomgangen av problemanalysen, teknologi og State-of-the-Art system reiste noen spørsmål, som sammen avgjør videre design av systemet. Spørsmålene som kom opp vises var:

- **Kontekstinformasjon**; Det er en rekke mulige elementer som kan benyttes for å øke brukervennlighet og funksjonalitet. For kontekstinformasjon ble det indentifisert, **Lokasjon**, **Tid**, **Bruker**, omgivende **Miljø** og **Hardware**.
- **Push vs Poll** Mulighet for å både be om posisjoner fra CRE eller sette opp at posisjoner skal bli tilsent når en bruker entrer en definert sone.
- **XML parsing**; Hvor skal XML dokument fra CRE parses for å trekke ut posisjoner.
- **Mobil enhet**; Hvilken mobil enhet skal IH implementeres for.
- **Lagring**; Hvor skal avtaler, brukerdata og brukermodell lagres.

Disse spørsmålene og hva som er tatt av valg oppsummeres i Tabell 5.1.

Nr.	**Spørsmål**	**Mulig løsning**	*Valgt løsning*
1	**Kontekst-informasjon** Tid Bruker Miljø Hardware	Lokasjon Tid Bruker	Lokasjon

Fortsetter på neste side.

[2] http:www.omg.org

Tabell 5.1 – forts. fra forrige side.			
Nr.	**Spørsmål**	**Mulig løsning**	*Valgt løsning*
2	**Push vs Poll** Poll	Push	Poll
3	**XML parsing** Server	Mobil enhet	Server
4	**Mobil enhet** Mobil	Mobil	PDA
5	**Lagring** Mobil enhet	Server Mobil enhet	Server

Tabell 5.1: Design spørsmål og valg.

Besvarelsen av disse spørsmålene er avgjørende for retningen i den videre analysen og designet av systemet. En forklaring på valgene som er tatt vises under:

- **Kontekst-informasjon**;
  - **Lokasjon**;Valget med å benytte lokasjon er gitt på forhånd og satt opp i delmålene for oppgaven.
  - **Tid**; Elementet tid er lett å benytte og er ofte avgjørende når beslutninger skal tas.
  - **Brukere**; Informasjon om brukere vil bli benyttet. Interesser til en bruker og lokasjonen kan sammen trigge et varsel om en nært forestående hendelse er planlagt. Dette kan for eksempel være et foredrag som er av interesse for brukeren, tatt ut av informasjon registrert om brukeren.
  - **Miljø**; Det skal av avgrensende grunner ikke benyttes andre sensorer en CRE i IH. Muligheten her hadde vært å sjekke andre forhold som vær, temperatur og lignende.
  - **Hardware**; Informasjon om applikasjonens og mobilens status kommer ikke til å å bli benyttet som kontekstinformasjon. Det vil si at det ikke kommer til å bli utført sjekk på belastning av nettverksforbindelse, gjenværende batterikapasitet og lignende.
- **Push vs Poll**; Valget for dette designet falt på å hente posisjoner fra CRE da dette gir mer kontroll over nettverkskommuniksjonen. Behovet for posisjoner fra CRE vil være svært varierende.
- **XML parsing**; Parsing av XML kan skje på både serversiden og på mobilen. Prosjektet *The Nidaros Framework* viste til erfaring med parsing på mobilen, og hvor dette hadde en del negative sider. Applikasjonen ble nærmest overbelastet med all parsingen av XML dokumentene og ved en eventuell økning i oppdateringsfrekvens av posisjoner i en lignende applikasjon, vil dette kunne være fatalt.
- **Mobil enhet**; Valget på å benytte mobile som mobil enhet, er gjort av den grunn at det er i stor grad, flere mobiler på markede, enn noen andre

mobile enheter. De fleste mobiler i dag er også utstyrt med Java teknologi, da med noe forskjellig oppsett.

- **Lagring**; For at mobilen skal ha en gradvis degradering av funksjonalitet blir deler av dataene lagret på mobilen. Dette skal gjøre at mobilapplikasjonen kan benyttes selv om deler av systemet blir midlertidig nede.

5.2.1 Funksjonalitet

Systemet skal benytte en mobil, sammen med CRE for å gi assistanse til besøkende og ansatte ved Den Intelligente Korridor. Funksjonalitet som skal designes, og vurdert som nyttige for målgruppen, er:

- **Søk etter ansatte, og grafisk representasjon av brukere, ved å benytte CRE.**
- **Registrering av hendelser, som distribueres ut fra interesser til bruker.**
- **Benytte brukermodeller for å tilpasse IH til den enkelte bruker.**

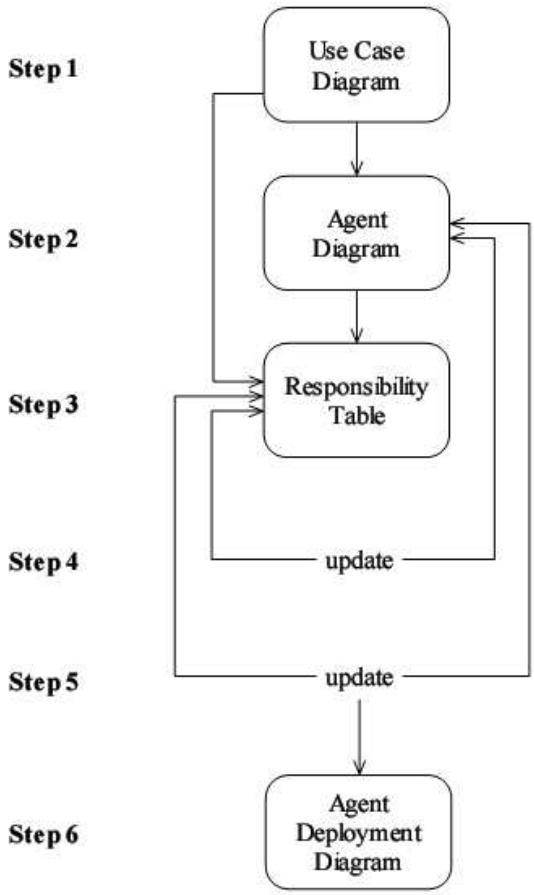
Hendelser i systemet skal også kunne registreres som en **avtale**, mellom to eller flere personer.

5.3 Analyse & Kravspesifikasjon

I analysens første del settes kravene som stilles til systemet. Kravspesifikasjonen utarbeides videre og presenteres gjennom bruk av bruksmønster, og beskriver den funksjonalitet som systemet skal oppnå. Kravspesifikasjonen er rettledende for systemutviklingen og det ferdige produktet.

Metoden tar for seg 6 steg for å kartlegge den funksjonalitet systemet skal ha, de ulike agentene som trengs og deres ansvar i systemet. Fasene i metoden er som følger:

1. **Steg 1 - Bruksmønster**
2. **Steg 2 - Innledende agentidentifikasjon**
3. **Steg 3 - Ansvarsidentifikasjon**
4. **Steg 4 - Identifisering av relasjoner**
5. **Steg 5 - Agentraffinering(Agent Refinement)**
6. **Steg 6 - Agentutplassering**



Figur 5.1: Metode for utvikling av JADE multiagentsystem - Analysefasen.

I analysefasen vil man bevege seg opp og ned i mellom de ulike stegene. Dynamikken i analysefasen vises i Figur 5.1, tatt fra en oppsummering i [26].

Første steg i metoden er å benytte seg av UML bruksmønster for å avdekke funksjonelle krav til agentsystemet. Bruksmønsterene lages med standard UML notasjon.

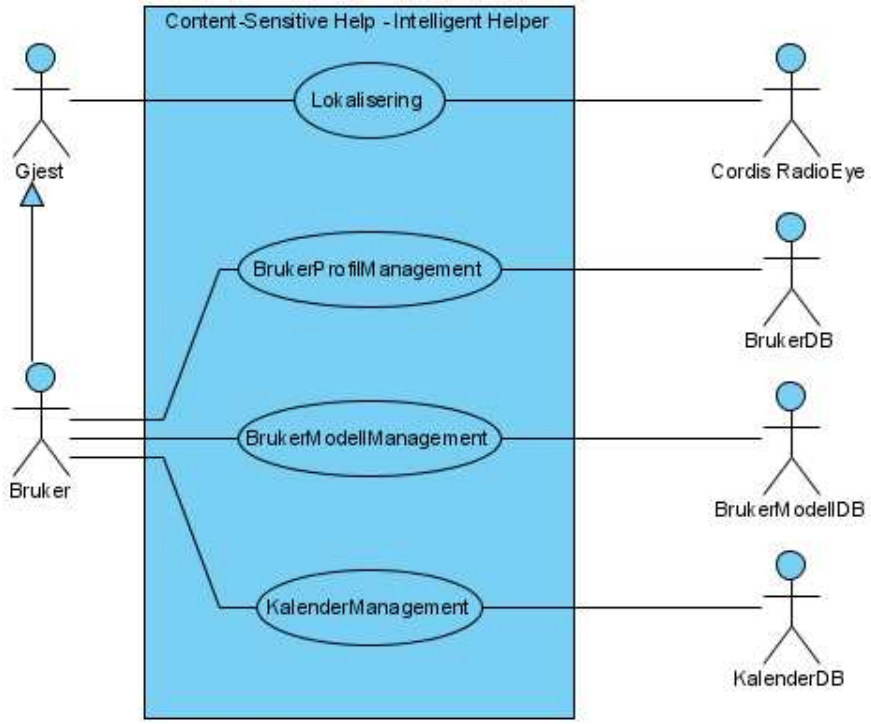
Presentasjon av funksjonalitet vises gjennom bruksmønsterdiagram, med påfølgende detaljerte tekstbaserte bruksmønster. De tekstbaserte bruksmønsterene er basert på [28].

5.3.1 Steg 1 - Bruksmønsterdiagram

Analysen førte frem til et bruksmønskerdiagram som viser hovedfunksjonaliteten i systemet, og vises i Figur 5.2.

Aktørene som ble identifisert under analysen var:

- **Gjest**; er en bruker som ikke har registrert en brukerprofil.



Figur 5.2: Brukesmønsterdiagram - Systemets hovedfunksjonalitet.

- **Bruker**; er en registrert bruker. Profilen inneholder personopplysninger, brukermodell og tilgangsstyring for profilen.

Videre er systemaktørene:

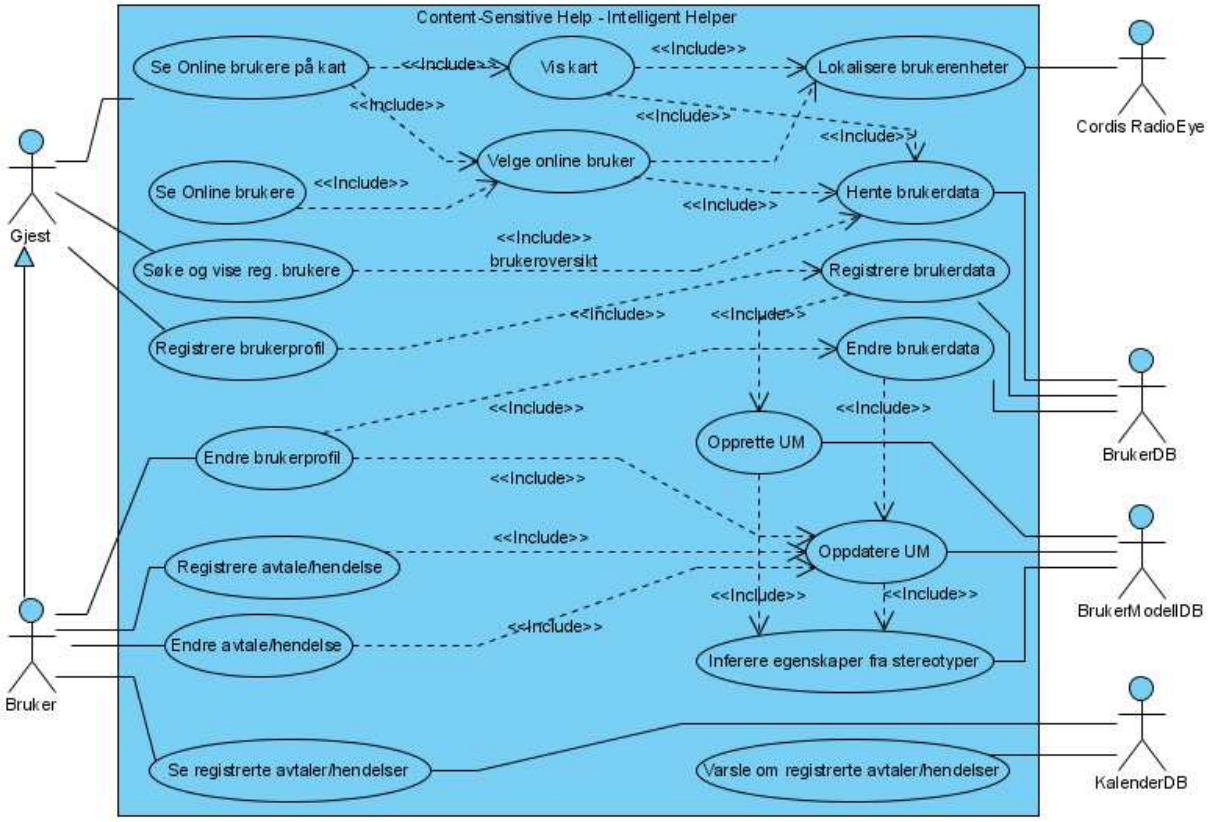
- **Cordis RadioEye™**
- **Brukerdatabase**
- **Kalenderdatabase**
- **Brukermodelldatabase**

Bruksmønsterdiagram 5.3 viser IH i sin helhet, og presenterer de funksjoner som utgjør systemet.

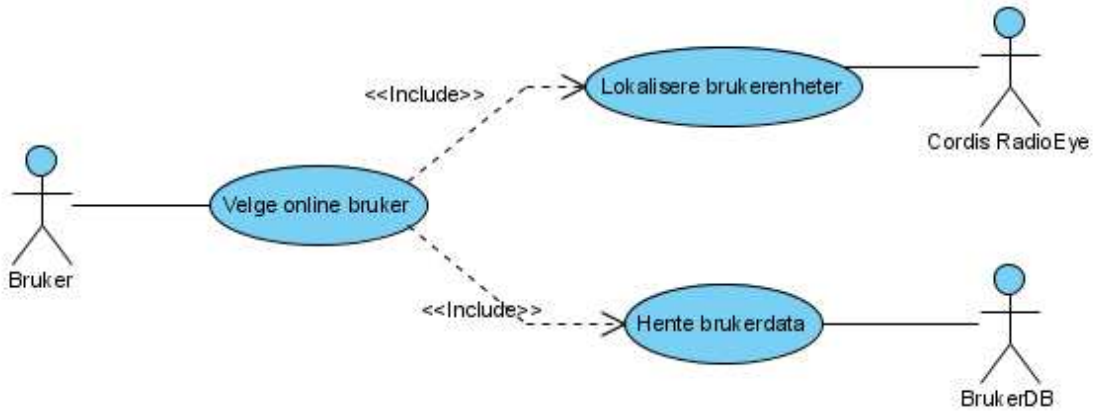
Videre et utvalg av bruksmønsterene vist, hvor hver agent er representert.

5.3.1.1 Søk Online brukere

Bruksmønsteret *Søk Online brukere* viser aktøren "bruker" som ønsker å søke opp ansatte som er registrert i systemet og som er tilstede i 3. etasje på IDI-bygget for øyeblikket. Brukeren skal få opp en liste som tar utgangspunkt i de posisjoner og MAC-adresser som CRE registrerer og sjekker disse opp mot brukerdatabasen over registrerte brukere. Figur 5.4 viser bruksmønsterdiagrammet "Søk Online brukere" og Tabell 5.2 viser bruksmønsteret i tekstet form..



Figur 5.3: Brukesmønsterdiagram - Systemets hovedfunksjonalitet.



Figur 5.4: Bruksmønsterdiagram - Bruker søker Online brukere.

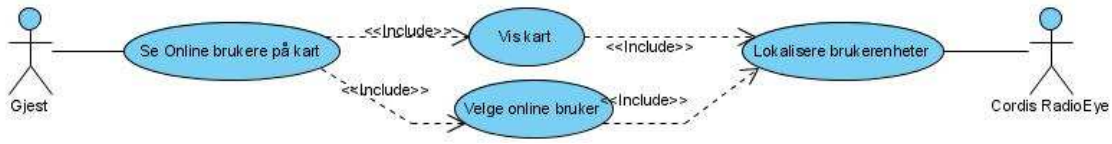
Bruksmønster 1	Søk Online brukere
Beskrivelse	Brukeren velger å søke etter registrerte brukere som registreres av CRE.
Aktører	Bruker
	Fortsetter på neste side.

Tabell 5.2 – forts. fra forrige side.		
Bruksmønster 1	Søk Online brukere	
Bruksmønster forbindelser	Lokaliser brukerenheter	
Miljø	Mobiltelefon, Server	
Pre-betingelser	Brukeren har installert applikasjonen på mobilen.	
Post-betingelser	Bruker får opp en valgbar liste over online-ansatte.	
Normal flyt	1	Brukeren velger å søke etter Online brukere.
	2	Systemet henter posisjoner fra CRE.
	3	Systemet henter brukerdata for registrerte brukere fra BrukerDB.
	4	Systemet viser liste over brukere på mobilapplikasjon.
Alternativ flyt	2.a	Systemet finner ingen Online brukere.
	3.a	Systemet finner ingen informasjon om brukerene hentet fra CRE.

Tabell 5.2: Bruksmønster 1: Søk Online brukere.

5.3.1.2 Se Online brukere på kart

Bruksmønsterdiagrammet *Se Online brukere på kart* vist i Figur 5.5, og den medfølgende Tabell 5.3, viser hvordan en bruker kan se en kartoversikt over utvalgte Online brukere:



Figur 5.5: Bruksmønsterdiagram - Bruker viser Online brukere på kart.

Bruksmønster 2	Se Online brukere på kart
Beskrivelse	Brukeren velger å se Online brukere på kart.
Aktører	Bruker
Bruksmønster forbindelser	Søk Online brukere
Miljø	Mobiltelefon, Server

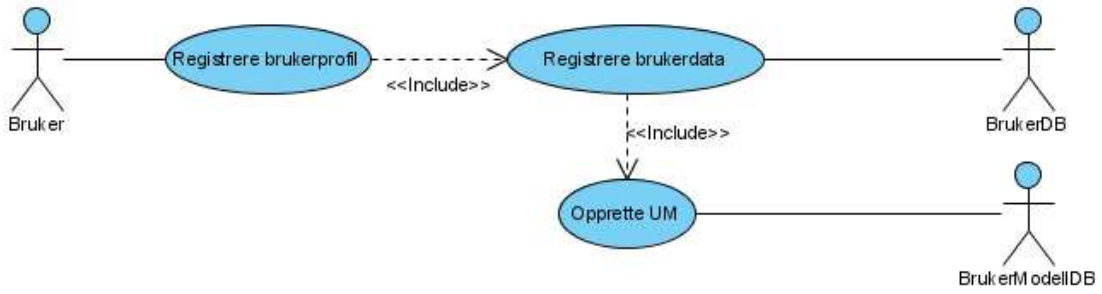
Fortsetter på neste side.

Tabell 5.3 – forts. fra forrige side.		
Bruksmønster 2	Se Online brukere på kart	
Pre-betingelser	Brukeren har installert applikasjonen på mobilen.	
Post-betingelser	Bruker får opp en kartvisning som viser valgte online-ansatte.	
Normal flyt	1	Brukeren velger å søke etter Online brukere.
	2	Systemet henter posisjoner fra CRE.
	3	Systemet henter brukerdata for registrerte brukere fra BrukerDB.
	4	Systemet viser liste over brukere på mobilapplikasjon.
	5	Brukeren velger de*n* bruker*ene* som skal posisjoneres og trykker OK.
	6	Systemet henter oppdaterte posisjoner til de valgte brukerene fra CRE.
	7	Systemet viser kart med de valgte brukerene og oppdaterer visningen til bruker avslutter kartvisningen.
Alternativ flyt	2.a	Systemet finner ingen Online brukere.
	3.a	Systemet finner ingen informasjon om brukerene hentet fra CRE.
	7.a	Systemet finner ikke alle posisjoner og viser oversikt over siste kjente posisjon til brukere med tapte posisjoner.

Tabell 5.3: Bruksmønster 2 - Se Online brukere på kart

5.3.1.3 Registrere brukerprofil

Dette bruksmønsterdiagrammet *Registrer brukerprofil* vist i Figur 5.6, og medfølgende Tabell 5.4 viser registreringen av en brukerprofil.



Figur 5.6: Bruksmønsterdiagram - Bruker registrerer en brukerprofil.

Bruksmønster 3	Bruker registrerer en brukerprofil	
Beskrivelse	Brukeren velger å registrere en brukerprofil.	
Aktører	Bruker	
Bruksmønster forbindelser	Opprette brukermodell	
Miljø	Mobiltelefon, Server	
Pre-betingelser	Brukeren har installert applikasjonen på mobilen.	
Post-betingelser	Bruker har fått opprettet en brukerprofil med sine personopplysninger.	
Normal flyt	1	Brukeren velger registrer brukerprofil i applikasjonen.
	2	Systemet viser et registreringsskjema hvor for personopplysninger.
	3	Bruker fyller ut skjemaet og velger å lagre.
	4	Systemet validerer feltene i skjemaet.
	5	Systemet lagrer opplysningene i BrukerDB.
	6	Systemet viser en melding om at data ble lagret og viser de registrerte dataene.
Alternativ flyt	4.a	Systemet finner feil i innfyllingen av brukerdata.
	.1	Systemet ber brukeren om å rette opp.
	5.b	Systemet får ikke lagret brukerdataene i systemet.
	.1	Systemet gir brukeren beskjed om dette, og at brukerdata lagres i enhetens persistente lagringsplass(Record Management System (RMS)).

Tabell 5.4: Bruksmønster - Bruker registrerer en brukerprofil.

5.3.1.4 Registrere en avtale/hendelse

Bruksmønsterdiagram *Registrere en avtale/hendelse* som vises i 5.7, og medfølgende Tabell 5.5 viser hvordan registrering av en ny avtale foregår.

Bruksmønster 4	Bruker registrer en ny avtale/hendelse
Beskrivelse	Brukeren skal registrere en ny avtale/hendelse.

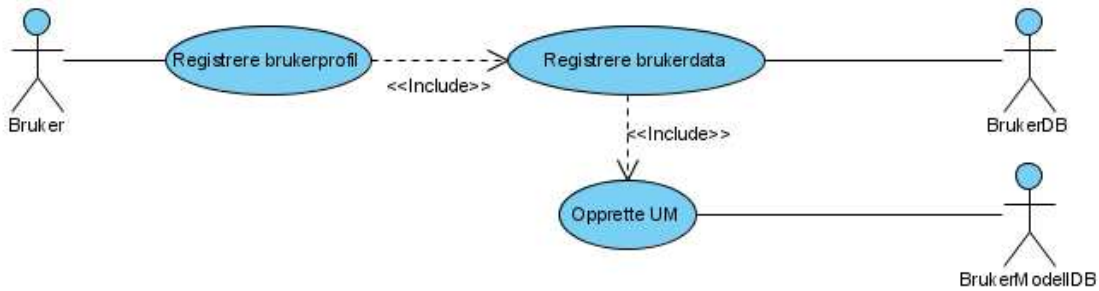
Fortsetter på neste side.

Tabell 5.5 – forts. fra forrige side.		
Bruksmønster 4	Bruker registrer en ny avtale/hendelse	
Aktører	Bruker	
Bruksmønster forbindelser		
Miljø	Mobiltelefon, Server	
Pre-betingelser	Brukeren har installert applikasjonen på mobilen og har registrert en brukerprofil.	
Post-betingelser	Bruker har fått registrert en ny avtale/hendelse.	
Normal flyt	1	Brukeren velger registrer avtale/hendelse i applikasjonen.
	2	Systemet viser en registreringsform hvor brukeren kan fylle inn informasjon vedrørende avtalen/hendelsen og avslutter med å trykk OK.
	3	Systemet validerer hvert felt i avtaleregistreringen.
	4	Systemet lagrer avtalen i mobilens persistente lager.
	5	Systemet lagrer avtalen i KalenderDB.
	6	Systemet viser en melding om at data ble lagret og viser oversikten over registrerte avtaler.
Alternativ flyt	3.a	Systemet finner feil i innfyllingen av brukerdata.
	.1	Systemet ber brukeren om å rette opp.
	5.b	Systemet får ikke lagret avtalen i systemet.
	.1	Systemet gir brukeren beskjed om dette, og at brukerdata lagres i enhetens persistente lagringsplass(RMS).

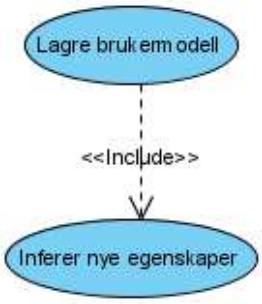
Tabell 5.5: Bruksmønster - Bruker registrerer en ny avtale/hendelse.

5.3.1.5 Opprette en brukermodell

Bruksmønsterdiagram *Registrere en avtale/hendelse* vises i Figur 5.8, og medfølgende Tabell 5.6 viser hvordan oppretting av en ny brukermodell foregår.



Figur 5.7: Bruksmønsterdiagram - Bruker registrer en ny avtale/hendelse.



Figur 5.8: Bruksmønsterdiagram - Opprette en ny brukermodell.

Bruksmønster 5	Opprette en ny brukermodell	
Beskrivelse	Brukeren har registrert en brukerprofil og systemet oppretter en brukermodell.	
Aktører		
Bruksmønster forbindelser	Bruker registrer en ny avtale/hendelse	
Miljø	Server	
Pre-betingelser	Systemet har registrert en ny bruker i brukerdatabasen.	
Post-betingelser	Systemet har opprettet en brukermodell for den nye brukeren.	
Normal flyt	1	Systemet trekker ut informasjon om bruker som skal være med i brukermodellen.
	2	Systemet kjører data om bruker og stereotyper gjennom prolog-motoren og oppdaterer brukerdata.
	3	Systemet lagrer brukerdata i brukermodelldatabasen.
Alternativ flyt	3.a	Systemet får ikke opprettet en ny brukermodell.

Fortsetter på neste side.

Tabell 5.6 – forts. fra forrige side.		
Bruksmønster 5	Opprette en ny brukermodell	
	.1	Systemet gir brukeren beskjed om feil under lagring.

Tabell 5.6: Bruksmønster - Opprette en ny brukermodell.

Bruksmønsterene har belyst hvilken funksjonalitet systemet skal ha, og gir en del informasjon om hva som blir de endelige kravene til systemet.

Forskningen på tidligere system, tilgjengelig teknologi og bruksmønsterene i Steg 1 har avdekket kravene til *Intelligent Helper*

5.3.2 Kravspesifikasjon

Denne delen tar for seg kravene for IH. Kravene er delt opp i *funksjonelle* og *ikke-funksjonelle krav.*

5.3.2.1 Funksjonelle krav

De funksjonelle kravene(FR). Kravene gir et grunnlag for videre design av agent-systemet.

Funksjonelle krav	
ID	**Beskrivelse**
FR1	Det skal være mulig å søke blant brukere som registreres av Cordis RadioEye™
FR2	Det skal være mulig å innhente posisjoner til et sett valgte brukere.
FR3	Det skal være mulig å vise brukere på en kartrepresentasjon.
FR4	Det skal være mulig å registrere en brukerprofil.
FR5	Det skal være mulig å endre sin egen brukerprofil.
FR6	Det skal være mulig å slette sin egen brukerprofil.
FR7	Det skal være mulig å registrere en avtale/hendelse.
FR8	Det skal være mulig å endre en egen avtale/hendelse.
FR9	Det skal være mulig å slette en egen avtale/hendelse.
FR10	Det skal opprettes en brukermodell ved registrering av brukerprofil.
FR11	Brukermodell skal oppdateres ved endringer av profil.
FR12	Brukermodellen skal oppdateres med nye egenskaper fra stereotyper.

Tabell 5.7: Oversikt over funksjonelle krav.

5.3.2.2 Ikke-funksjonelle krav

Det er noen ikke-funksjonelle krav(IFK) i systemet som styrer oppgaven i stor grad, og påvirker systemets utforming.

Ikke-funksjonelle krav	
ID	**Beskrivelse**
IFK1	Det skal benyttes JADE plattform versjon 3.4.1 for agentutvikling.
IFK2	Det skal benyttes Cordis RadioEye™ som posisjoneringsteknologi.
IFK3	Det skal skal benyttes LEAP add-on for agenter på mobile enheter.

Tabell 5.8: Oversikt over ikke-funksjonelle krav.

5.3.3 Steg 2 - Agent identifikasjon

Metodens neste steg er å identifisere de agenter som umiddelbart og naturlig peker seg ut. Kriteriene som benyttes for å identifisere de første agentene er:

- Legg til en agent per bruker/enhet(device)
- Legg til en bruker per ressurs(inkl. legat-software)

Etter disse stegene ble det utpekt fire agenter for å dekke følgende brukere, enheter og ressurser:

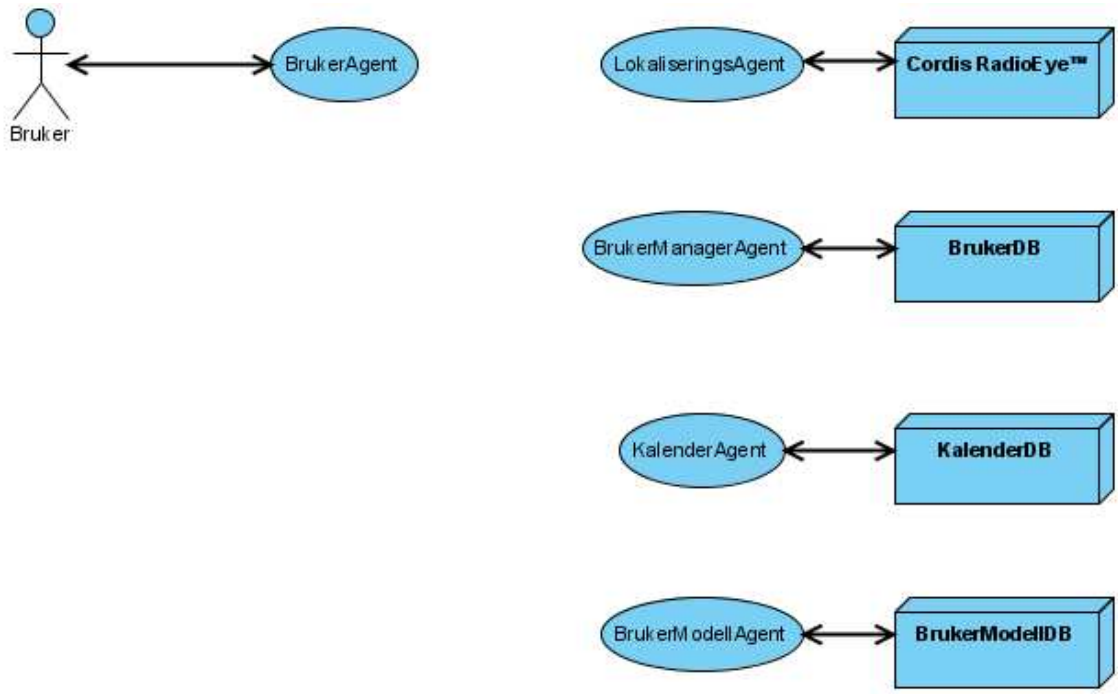
- **Brukere**; gjester og registrerte brukere.
- **Cordis RadioEye™**; kommunikasjon med CRE interface.
- **Brukerdatabase**; brukerdatabase som holder informasjon om brukeren
- **Kalenderdatabase**; database for avtaler og hendelser.
- **BrukerModelldatabase**; database for brukermodeller.

En innledende artifakt ble produsert etter dette steget og vises i Figur 5.9.

Artifaktet viser nå fire typer elementer og hvor symbolene er:

- **Agent typer**; identifiserte agenter.
- **Personer**; menneskelig interaksjon.
- **Ressurser**; eksterne systemer som interagerer med systemet.
- **Relasjoner**; viser hvilke elementer som interagerer med hverandre(agent-til-agent avventes til et senere).

Ulikt UML notasjon skiller man mellom mennesker og eksterne systemer, fordi i interaksjon mellom mennesker og system oppstår det flere problemer som må håndteres.



Figur 5.9: Agentidentifikasjon

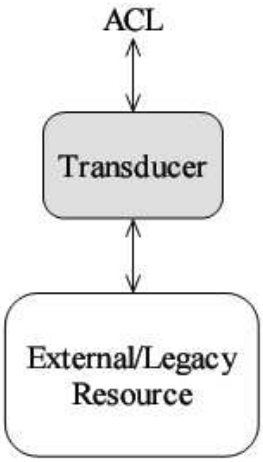
Håndteringen av interaksjon mot mennesker eller eksterne systemer kan håndteres på ulike måter. I [26] presenteres det tre ulike teknikker for dette, som er:

- **Transducer(transformat) agent**; en agent i systemet som fungerer som grensesnitt mellom eksterne ressurser og andre agenter i systemet.
- **"Wrapper" kode**; hvis kode på ressurs er tilgjengelig, legge inn kode for kommunikasjon.
- **Kode omskrivning**; skrive om koden i ressursen slik at den kan kommunisere med agentsystemet.

Teknikken som skal benyttes dette systemet er for tre av agentene transformat(transducer) teknikken, hvor det opprettes en agent for hver eksterne ressurs. Dette er fordi det er den enkleste måten å benytte ressursene, og fordi tilgang til applikasjonsspesifikke data på ressursene ikke er en mulighet. I dette tilfellet blir det å lage en agent som håndterer forespørsler til CRE, mot database for brukerinformasjon, kalenderdatabasen og brukermodelldatabasen. Figur 5.10 viser en transformat tilnærming.

BrukerModellAgent kommer i tillegg til å håndtere lagring, inneholde en **Amzi Prolog** komponent, som håndterer regler i Prolog [3]. Denne blir benyttet for å inferere nye egenskaper til brukermodellene, fra et sett stereotyper som er lagret i systemet. Prolog-motoren kommer til å motta meldinger fra KalenderAgent, om hendelser som er brukere finner interessante. De stikkord(**tags**) som er lagret om

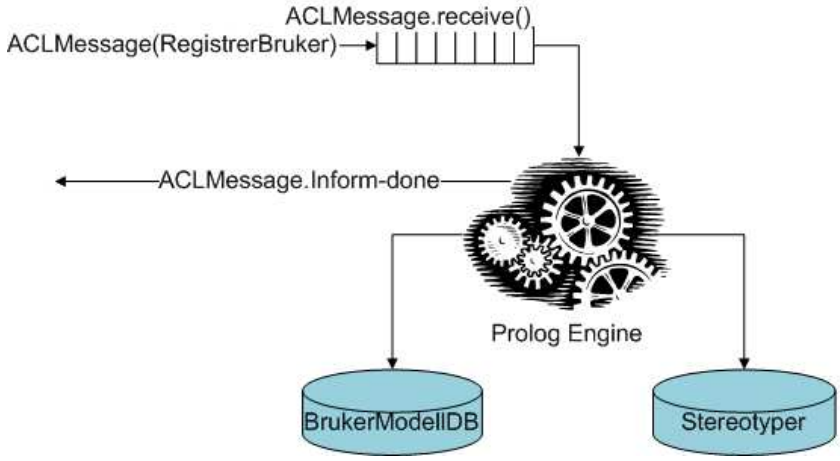
[3] http://www.swi-prolog.org/



Figur 5.10: Transformat teknikk for eksterne ressurser i agentsystem.

hendelse, vil bli regnet for å være interessant for brukeren og brukeremodellen oppdateres.

Prolog-motoren slik den er tenkt i IH er vist i Figur 5.11.



Figur 5.11: En modell av Prolog-motor.

Dette artifaktet blir oppdatert etter hvert som man går videre i metoden, helt frem til steg 5.

5.3.4 Steg 3 - Ansvarsidentifikasjon

I dette steget produseres en innledende liste over de ansvarsoppgavene hver agent skal ha, og produserer et innledende artifakt i form av en "ansvarstabell". Reglene for å utarbeide denne tabellen er:

- **Utled** de innledende ansvar fra bruksmønsterene i steg 1.

- **Begynn** med de agentene hvor ansvarsområdene er klarest og avvent med resten til senere steg.

Ut fra denne beskrivelsen ble en tabell opprettet som beskrev de forskjellige oppgavene agentene i systemet skulle ha ansvar for. Tabell 5.9 viser artifaktet som kom ut av dette steget.

Agent Type	Ansvarsbeskrivelse	
BrukerAgent	1	La brukeren søke blant online brukere.
	2	Innhente posisjonsdata fra valgte online brukere.
	3	Vise valgte brukere på kart.
	4	La brukeren registrere sin brukerprofil.
	5	La brukeren endre sin brukerprofil.
	6	La brukeren registrere en avtale/hendelse.
	7	La brukeren endre/slette en avtale/hendelse.
	8	Vise brukeren registrerte avtaler/hendelser.
LokaliseringsAgent	1	Håndtere forespørsler om å innhente posisjonsdata fra CRE.
	2	Sende posisjonsdata av valgte brukere til BrukerAgent, KalenderAgent på forespørsel.
	3	Sende posisjonsdata av samtlige posisjoner registrert av CRE til BrukerAgent og KalenderAgent.
BrukerManagerAgent	1	Håndtere forespørsler fra BrukerAgent og KalenderAgent.
	2	Registrere en brukerprofil for BrukerAgent.
	3	Endre en brukerprofil for BrukerAgent.
	4	Opprette en brukermodell ved registrering av ny profil.
	5	Oppdatere brukermodell ved endring av brukerprofil.
	6	Oppdatere brukermodell ved endring av avtaler/hendelser fra KalenderAgent.
KalenderAgent	1	Håndtere forespørsler fra BrukerAgent og BrukerModellAgent.
	2	Registrere en avtale/hendelse for BrukerAgent.
	3	Endre en avtale/hendelse for BrukerAgent.
	4	Sende oversikt over egne avtaler til BrukerAgent.
	5	Sende informasjon om valgte avtaler til BrukerAgent.
	6	Håndtere avtaler som inntreffer.

Fortsetter på neste side.

Tabell 5.9 – forts. fra forrige side.		
Agent Type	Ansvarsbeskrivelse	
	7	Innhente avtale-/hendelse-avhengig posisjonsdata fra LokasjonsAgent.
	8	Varsle BrukerAgent om avtaler som inntreffer.
	9	Sjekke **PUBLIC** hendelser for å se om brukere skal få informasjon om en hendelse på mottatt info bra BrukerModellAgent.
BrukerModellAgent	1	Håndtere forespørsler fra BrukerManagerAgent og KalenderAgent
	2	Opprette ny brukermodell på forespørsel fra BrukerManagerAgent.
	3	Endre en brukers brukermodell med mottatt info fra BrukerManagerAgent eller KalenderAgent.
	4	Varsle KalenderAgent om mulige nye interesser til en bruker.
	5	Slette en brukers brukermodell på forespørsel fra BrukerManagerAgent.

Tabell 5.9: Ansvarstabell.

Arbeidet med å utarbeide tabellen og trekke frem ansvarsområdene til agentene leder videre frem til neste steg.

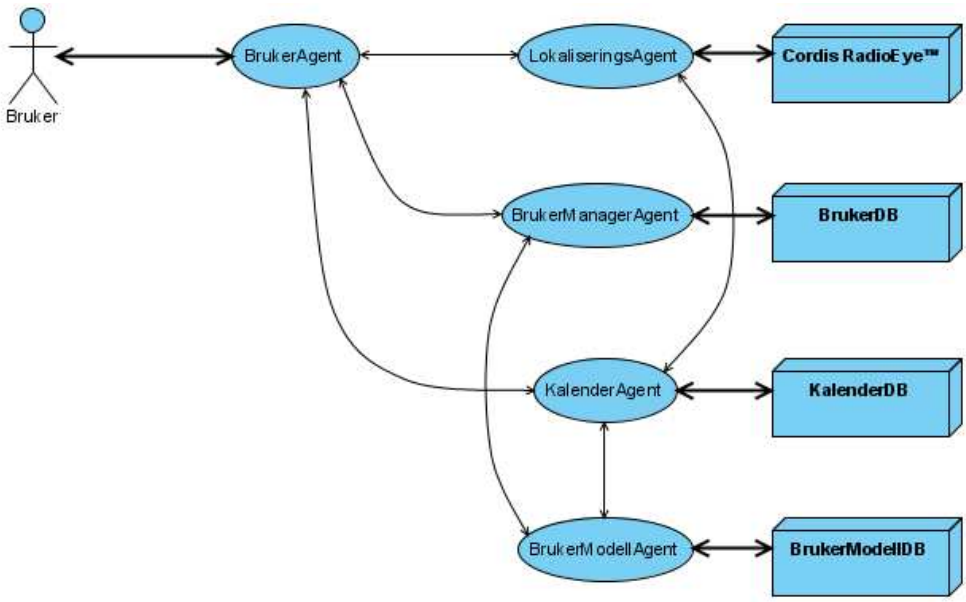
5.3.5 Steg 4 - Identifisering av relasjoner

Relasjoner mellom agenter og de eksterne ressursene blir utarbeidet og representert i dette steget. Dette gjøres ved å oppdatere artifaktet fra Steg 2, og legge til relasjoner,som vist i Figur 5.12.

5.3.6 Steg 5 - Agentraffinering

Dette steget ser over agentene som er definert i steg 2, og dette gjøres etter definerte steg i metoden. Det som tas i betraktning er:

- **Støtte**; hvilken informasjon trenger agentene for å utføre sine ansvarsområder, og hvordan, når og hvor fins denne informasjonen.
- **Oppdagelse**; hvordan agenter med relasjonskoblinger finner hverandre.
- **Administrering og monitorering**; skal systemet holde styr på agentene, og eventuellt starte/stoppe agenter på kommando.



Figur 5.12: Identifisering av relasjoner.

5.3.6.1 Støtte

Dette punktet ser på hvilken informasjonen de ulike agentene trenger, og om det er naturlig å legge til nye agenter for å skaffe til veie denne informasjonen blir vurdert. Agentene som er identifisert vil kún håndtere de ressursene de er tildelt og ingen tilleggsinformasjon utenom dette kreves. Det er derfor ikke behov for å legge til noen flere agenter i dette systemet.

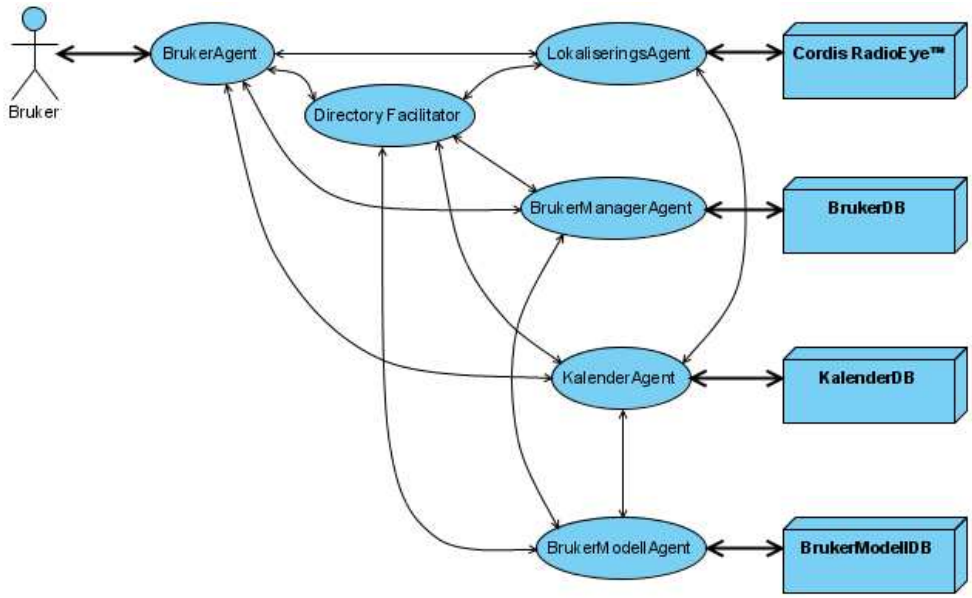
5.3.6.2 Oppdagelse

Navnet på en agent i JADE må være unikt. Kommunikasjon mellom agenter skjer ved at én agent sender en melding til en annen ved å legge til mottakers unike navn i en liste over mottakere i meldingen. For å gjøre dette må senderagenten vite det unike navnet på mottakeren.

Oppdagelsen av tjenester og agentene som tilbyr dem, kan gjøres på ulike måter. I enkelte system er det tilstrekkelig å kun bruke navnet på agentene, og disse må da være implementert i agenten.

En alternativ teknikk er en *gule sider* agent, DF, som sørger for oppdagelse av tjenester og agenter. Denne teknikken adopteres her, da det gir en mer dynamisk kobling mellom agentene.

Dette steget fører til at en oppdatering av artifaktet fra forrige steg, hvor det nå legges til *Directory Facilitator*. Relasjonene oppdateres og et nytt artifakt blir produsert, som vist i Figur 5.13.



Figur 5.13: Raffinering av agentdiagram.

5.3.6.3 Administrering og monitorering

Administrasjon av systemet kan være punkt for videreutvikling av dette systemet. En mulighet er å legge til administrere agenter for lokalisering og muligens brukermanagement hvis det er mange BrukerAgent'er som kjøres. Slik vil en kunne takle stor pågang på de tjenestene de tilbyr og sørge for hurtigere respons for brukerene.

Etter en eventuell endring av agentdiagrammet i dette steget vil det være en naturlig endring i ansvarstabellen 5.9 og agentdiagrammet. Endringen i 5.13 gjør at ansvarstabellen blir endret noe for å få med registrering av tjenester for tilbydere og søk etter tjenester for brukere av disse. Den oppdaterte tabellen vises i appendiks i tabellen 1.

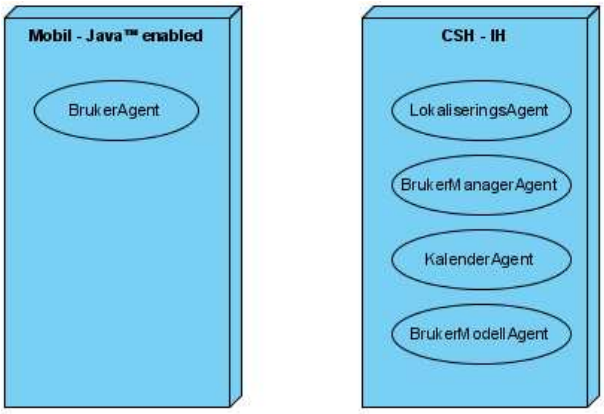
5.3.7 Steg 6 - Agent utplassering

Dette artifaktet viser i hvilket fysisk kjøremiljø agentene skal være, og representeres ved diagrammet på figur 5.14.

Dette var siste steg i **agentmetodens** analysefase. Metoden fortsetter i neste seksjon med *Design*, som består av 10 steg.

5.4 Design

Denne seksjonen tar for seg design av systemet fra kravene spesifisert i analysefasen.



Figur 5.14: Plassering av agentene i systemet.

Metoden spesifiserer at analyse og designfasen ikke er en lukkede faser og man kan under design gjøre endringer på steg i, eller artifakter fra, analysefasen.

5.4.1 Steg 1 - Agent splitting/fusjon/omdøping

Det første steget i designfasen er å se over agentdiagrammet og de ulike agentene. Metoden lister opp 6 regler som skal avgjøre om de foreløpige agentene skal splittes opp, fusjoneres sammen eller omdøpes. De ulike reglene er:

1. **Data gjentakelse**; unngå flere agenter som trenger mye av de samme dataene.
2. **Kode gjentakelse**; unngå gjentakelse av kode for å aksessere eksterne ressurser.
3. **Unngå unødig splitting**; for mange agenter øker kompleksitet og reduserer effektiviteten i systemet.
4. **Plassering av agenter**; der hvor det er to typer funksjonalitet, som trengs på to ulike maskiner, skal splittes i ulike agenter.
5. **Reduser kompleksitet**; for store og komplekse agenter gjør systemet vanskelig å vedlikeholde.
6. **Teknikk for eksterne ressurser**; benytte transformator teknikk istedet for ev. wrapper teknikk hvor det vil være naturlig å dele opp i flere agenter. Dette vil ikke være mulig når man benytter wrapper teknikken.

I IH vil noe data bli brukt av flere agenter. Dette er brukerinformasjon som BrukerModellAgent skal lagre, samt inferere nye egenskaper, fra stereotyper. Denne informasjonen vil bli brukt av KalenderAgent, men kun nå den blir tilsendt av BrukerModellAgent. KalenderAgent vil benytte brukerinformasjon for å finne ut hvilke hendelser som kan presenteres for brukeren.

Dette gjør ikke at noen fusjon av agentene blir gjort, da de har svært ulike roller i systemet. Agentene vil komme til å bli unødig komplekse ved en slik fusjon.

Videre kunne agenten BrukerModellAgent vært splittet opp slik at én agent håndterte lagring av brukermodeller, mens en agent håndterte Prolog-motoren. Denne løsningen er ikke valgt, da det er regnet for unødvendig og behovet for en ekstra agent er vurdert som liten.

Etter en gjennomgang av reglene og systemet ble det ikke funnet noen grunn til endringer i dette steget.

5.4.2 Steg 2 - Interaksjons-spesifikasjon

Dette steget tar for seg den ansvarsfordelingen som ble gjort i analysefasen, i tabellen 5.9. For hver agent blir ansvarstabellen gjennomgått, sammen med de relasjoner den har til andre agenter. Ut fra dette ble det produsert en *interaksjonstabell* hvor det spesifiseres hvordan samhandlingen mellom agentene skal foregå. Kolonnene i tabellen består av følgende:

- **Interaksjon**; et beskrivende navn for interaksjonen.
- **Ansvar**; hvem er ansvarlig og starter samhandlingen.
- **IP**; Interaksjonsprotokoll - hvilken interaksjonsprotokoll skal benyttes.
- **Rolle**; hvilken rolle har agenten i interaksjonsprotokollen.
- **Med**; den komplementære agent som samhandles med.
- **Når**; betingelsen som utløser denne interaksjonen.

Tabell 5.10 viser interaksjonstabellen for BrukerAgent.

Interakasjons-spesifikasjon	BrukerAgent				
Interaksjon	Ansvar	IP	Rolle	Med	Når
Søke Online-brukere	1	FIPA-REQUEST	I	LokaliseringsAgent	Bruker ber om å få søke blant Online-brukere
Hent posisjoner	2	FIPA-REQUEST	I	LokaliseringsAgent	BrukerAgent ber om å få posisjoner til valgte brukere
Registrere brukerprofil	5	FIPA-REQUEST	I	BrukerManagerAgent	Bruker vil registrere en brukerprofil
Oppdatere brukerprofil	6	FIPA-REQUEST	I	BrukerManagerAgent	Bruker oppdaterer sin brukerprofil
Registrering av avtale/hendelse	7	FIPA-REQUEST	I	KalenderAgent	Bruker vil registrere avtale/hendelse
Endring av avtale/hendelse	8	FIPA-REQUEST	I	KalenderAgent	Bruker vil endre avtale/hendelse
Vise avtaler/hendelser	9	FIPA-REQUEST	I	KalenderAgent	Bruker ber om å få se sine avtale/hendelse
Søke tjenestetil-bydere	10	FIPA-REQUEST	I	Directory Facilitator	BrukerAgent søker opp tjenestetilbydere

Tabell 5.10: Interaksjonstabell for BrukerAgent.

Disse tabellene som kommer ut av dette steget blir gjennomgått og endret flere ganger, ettersom forståelsen for agentsamhandlingen og systemet bygger seg opp. Tabellene for de resterende agentene er lagt med for steg 4, hvor den er utvidet med en ekstra kolonne.

5.4.3 Steg 3 - Definisjon av Ad-Hoc interaksjonprotokoller

Dette steget tar for seg eventuelle behov for interaksjoner som ikke lar seg gjennomføre med av standardiserte FIPA IP. De eksisterende interaksjonsprotokollene definert av FIPA er tilstrekkelig for systemet og ingen endringer blir gjort i dette steget.

5.4.4 Steg 4 - Message Templates

Message templates blir lagt til for hver interaksjon mellom agentene, for å sikre at agentene mottar og innhenter riktige meldinger. I dette steget er det også definerte et sett regler for å sikre at man ender opp med et godt valg av template. I de fleste tilfellene benyttes en konversasjons-id som blir opprettet hver gang en IP startes, og som håndteres av IP.

Tabell i interaksjons-spesifikasjonen blir utvidet med en kolonne til, hvor valgte template for hver interaksjon beskrives. Tabell 2 for BrukerAgent ligger i appendiks, hvor også de utvidede interaksjonstabeller for LokaliseringsAgent, vist i Tabell 3, BrukerManagerAgent i Tabell 4, KalenderAgent i Tabell 5 og BrukerModellAgent i Tabell 6 er lagt med.

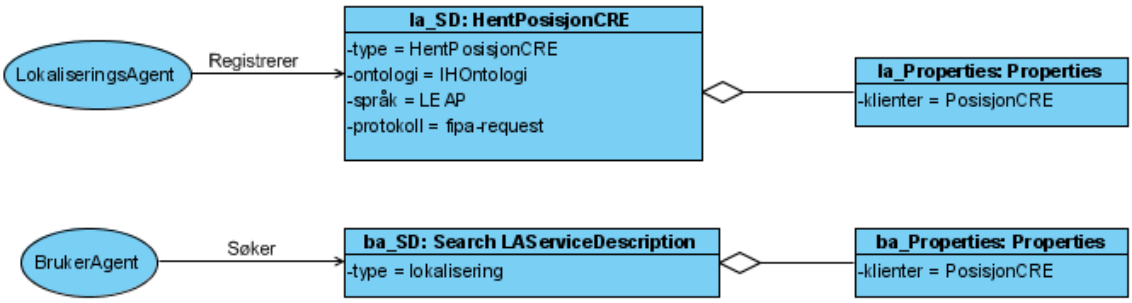
5.4.5 Steg 5 - Beskrivelse for registrering/søk

Som tidligere nevnt benyttes ontologien for å registrere tjenester, og søke blant tjenester hos DF. Metoden foreslår en representasjon av registrering og søk etter tjenester i agentsystem, og denne denne representasjonen benytte også i dette systemet. Figur 5.15 viser et utdrag av artifaktet som kommer ut av dette steget. Et større utdrag presenteres i Appendiks 8.3, i Figur 1.

5.4.6 Steg 6 - Interaksjon med agentressurser

Eksterne ressurser blir kategorisert i to ulike kategorier;

- Passive; ressurser com forandrer status kun etter en handling utført av systemets kontrollerende agent.
- Aktive; ressurser som kan forandre sin status uavhengig av systemets kontrollerende agent.



Figur 5.15: Steg 5 - Tjeneste registrering og søk hos DF.

Dette systemet kommer til å benytte både passive og aktive ressurser, og ulike hensyn blir tatt for å håndtere dette.

5.4.6.1 Passive ressurser

i dette systemet blir brukerdatabasen, kalenderdatabasen, og prologmaskinen. Der kommer det til å være èn kontrollerende agent per ressurs, som gjør at vanlige Java teknikker blir tilstrekkelig.

subsubsectionAktive ressurser blir posisjoneringer fra CRE. Metoden beskriver en tilnærming til håndteringen av aktive ressurser med et sett regler. LokaliseringsAgent kommer til å hente data fra CRE på forespørsel fra andre agenter, for å redusere unødig sending av data til og fra CRE.

Arkitekturen for systemet kan vises ved å vise hovedkomponentene i systemet, hvor hovedkomponentene viser den logiske arkitekturen.

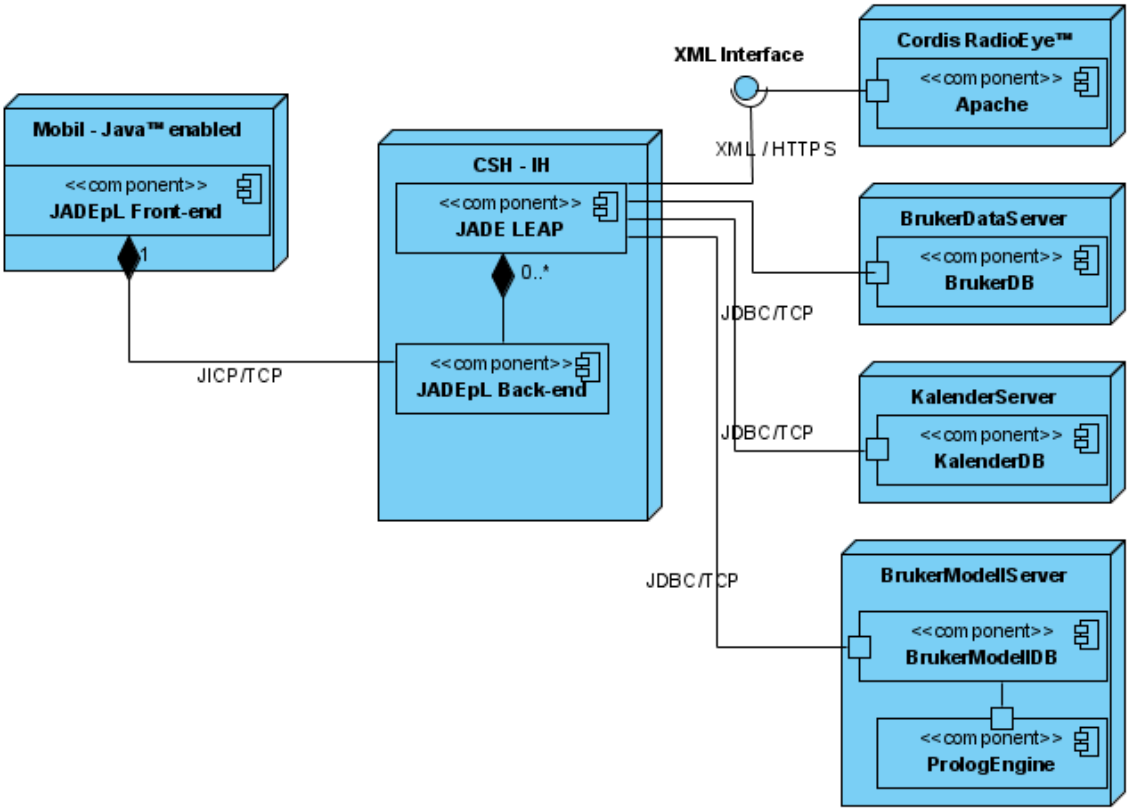
Dette komponentdiagrammet er ikke en del av metodens artifakter, men den presenteres allikevel fordi den er beskrivende for systemets hovedkomponenter og ressurser i det totale systemet.

I prototypen blir komponentene *BrukerDataServer*, *KalenderServer* og *BrukerModellServer* fysisk på én servermaskin.

5.4.7 Steg 7 - Agent-Bruker interaksjon

Interaksjon mellom agent og bruker på et "lokalt" GUI(grafisk brukergrensesnitt [4]) vil ofte arbeide på de samme datene. Ofte krever agentens oppgaver ulike datastrukturer enn en kanskje ønsker å representere grafisk for brukeren. Duplisering av data kan fort bli resultatet hvis en er nødt å legge til en ny datastruktur av samme data før en grafisk representasjon. Metoden presenterer en mulig løsning, basert på *Model-View-Controller* arkitekturen, og et sett regler:

[4] Graphical User Interface



Figur 5.16: Systemets hovedkomponenter.

- Lagre data i agenten i strukturer som er passende for å utføre agentens oppgaver.
- Synkroniser disse datene.
- Implementer interface til modell og link disse til relaterte grafiske klasser i GUI modulen.
- Definer passende klasser for presentasjon av data, når det er nødvendig.

5.4.8 Steg 8 - Indre agent oppførsel

Agent behaviours, eller agentoppførsel, er hva agentene kan utføre av handlinger. I dette steget ser vi på ansvarstabellen, vist i Tabell 5.9, i analysefasen, og oppretter de nødvendige oppførslene utifra tabellen. Igjen presenterer metoden et sett regler for å finne riktige implementasjoner:

- **Ansvar**; Relatert til en interaksjon i interaksjonstabellen, fra Steg 2 i designfasen. Her skal det velges den JADE klassen som implementerer riktig interaksjonsprotokoll og rolle for den gjeldende interaksjonen.

For å implementere oppførsler til agenter, finnes det et sett klasser som er ferdig definert i JADE. Klassene skilles opp i to hovedgrupper, med *primitive* og

sammensatte oppførsler. Samtlige oppførsler er underklasser *jade.core.Behaviour* klassen. Under vises de mest typiske oppførslene under jade.core.Behaviour:

Primitive oppførsler er oppførslen *SimpleBehaviour* og direkte subklasser av denne. Klassen SimpleBehaviour kan benyttes i seg selv eller en av subklassenene direkte under den. Hver og en har ulike måter å kjøre på:

- **OneShotBehaviour**; kjøres èn gang og terminerer.
- **TickerBehaviour**; syklisk oppførsel som kjøres periodisk etter spesifisert intervall.
- **WakerBehaviour**; kjører èn gang på et spesifisert tidspunkt.
- **CyclicBehaviour**; syklisk oppførsel som er aktiv så lenge agenten lever.
- **SenderBehaviour**; sender en ACL melding og terminerer.
- **ReceiverBehaviour**; kjører èn gang når en type melding er mottat, eller etter et definert tidsavbrudd.
- **SimpleAchieveREInitiator**; implementasjon som imiterer FIPA-Request protokoll, hvor denne er initiatoren.
- **SimpleAchieveREResponder**; implementasjon av FIPA-Request protokoll, hvor denne er responder.

Sammensatte oppførsler er klassen **CompositeBehaviour** og dens subklasser. Sammensatte oppførsler består av flere barn oppførsler som legges til en sammensatt oppførsel. Klassen CompositeBehaviour vil normalt ikke blir direkte extended, men en av heller en av underklassene. Direkte subklasser av CompositeBehaviour er:

- **ParallelBehaviour**; kontrollerer et sett oppførsler som eksekveres parallelt.
- **SerialBehaviour**;
  - **FSMBehaviour**; en meget kompleks sammensatt oppgave som setter opp under-oppgaver etter samme prinsipp som en *Finite State Machine* [5]
  - **SequentialBehaviour**; en oppførsel som eksekverer sine barn-oppførsler(childer behaviours) én etter en og terminerer når siste barn-oppførsel er ferdig.

Sammensatte oppførsler er designet for å dele opp komplekse agentoppgaver i flere mindre oppgaver.

[5]Finite State Machine er en software/hardware entitet som, under arbeid, kan modelleres som en sekvens av tilstander, slik at entiteten er i kun én tilstand til enhver tid [26].

5.4.8.1 Systemets oppførsler

Første punkt i ansvarstabellen til BrukerAgent og samsvarende i interaksjonstabellen, er en interaksjon med LokaliseringsAgent. BrukerAgent kjøres på den mobile enheten. Formålet med interaksjonen er å innhente samtlige posisjoner CRE registrerer. Dette er en FIPA "REQUEST", hvor BrukerAgent er initiator, og oppførselen som passer til dette formålet er *SimpleAchieveREInitator*.

Artifaktet produsert i dette steget, er ikke en del av metoden, men presenteres fordi det gir en oversikt over resultatet av dette steget. Tabell 5.11 viser en de innledende oppførslene satt opp i dette steget. I tabellen vises agenten, nummeret som korresponderer med interaksjonstabellen, i Tabell 5.10 for BrukerAgent, og tabellene i Appendiks 8.3 for resten av agentene.

Agent	**Inter.spek**	**Oppførsel**
BrukerAgent	1	SimpleAchieveREInitiator
	2	TickerBehaviour
		& SimpleAchieveREInitiator
	5	SimpleAchieveREInitiator
	6	SimpleAchieveREInitiator
	7	SimpleAchieveREInitiator
	8	SimpleAchieveREInitiator
	9	SimpleAchieveREInitiator
	10	OneShotBehaviour
LokaliseringsAgent	2	SimpleAchieveREResponder
	3	SimpleAchieveREResponder
	4	OneShotBehaviour
BrukerManagerAgent	2	SimpleAchieveREResponder
	3	SimpleAchieveREResponder
	6	SimpleAchieveREResponder
	7	OneShotBehaviour
KalenderAgent	2	SimpleAchieveREResponder
	3	SimpleAchieveREResponder
	4	SimpleAchieveREResponder
	5	SimpleAchieveREResponder
	6	SimpleAchieveREResponder
	7	SimpleAchieveREResponder
	8	SimpleAchieveREResponder
	9	SimpleAchieveREResponder
	10	OneShotBehaviour
BrukerModellAgent	2	SimpleAchieveREResponder
	3	SimpleAchieveREResponder
	4	SimpleAchieveREResponder
	5	OneShotBehaviour

Fortsetter på neste side.

Tabell 5.11 – forts. fra forrige side.		
Agent	**Inter.spek**	**Oppførsel**

Tabell 5.11: Innledende oppførsler.

5.4.9 Steg 9 - Definere en ontologi

Ontologien i dette systemet handler blant annet om brukere av systemet, brukerprofiler, brukermodeller, avtaler, posisjoner og trådløse klienter. Videre om de handlinger(AgentActions) som agentene i systemet kan utføre.

- **Konsepter**; er entiteter i agentenes miljø.
- **Predikater**; er relasjoner mellom entiteter.
- **Agent handlinger**; er de handlinger agentene kan utføre.

5.4.9.1 Konsepter

Konsepter er representert som komplekse templates. Konseptene beskrives med navn og et eller flere felt felt, som kan ha ulike primitive verdier(String, int, long). Konsepter i dette systemet blir presenter med en tabell som viser navnet på konseptet, felt og hvilken type det krever. Tabell 5.12 viser konseptet *Bruker*, *WlanKlient* og *Posisjon* i ontologien.

Konsept	**Felt**
Bruker	
	- brukerID(int)
	- brukerNavn(String)
	- passord(String)
	- forNavn(String)
	- etterNavn(String)
	- fodselsdag(String)
	- kjonn(String)
	- tittel(String)
	- tlfArbeid(int)
	- tlfMobil(int)
	- epost(String)
	- accessLevel(int)
	- viewLevel(int)
	- modified(String)
	- brukerKlient(WlanKlient)
WlanKlient	
	- mac(String)
	- wlanKlientPosisjon(Posisjon)

Posisjon	
	- x(int)
	- y(int)
	- tidspunkt(long)

Tabell 5.12: Konsepter i ontologien.

Konseptene som vises i Tabell 5.12 viser hvordan konseptet **Bruker** er bygget opp av andre elementer i ontologien. **Bruker** konseptet inneholder konseptet **WlanKlient** som igjen inneholder **Posisjon**.

5.4.9.2 Predikater

Predikater viser relasjoner som kan være enten *sann* eller *falsk*. De representeres med et navn og en eller flere felt(slots) som beskriver relasjonen, hvor feltene har restriksjoner på type data. Et eksempel på et predikat i ontologien er vist i tabellen 5.13.

Predikat	**Felt**
Tilhorer	
	- brukerID(int)
	- romID(int)

Tabell 5.13: Predikat i ontologien.

5.4.9.3 Agent handlinger

Agent handlinger er også representert som templates, hvor et navn, eventuelle krav om parameterverdier eller en spesifikasjon av en returverdi beskrives. I Tabell 5.14 vises to eksempel på agent handlinger.

Agent handling	**Felt**
HentListeCRE	
	- posKlienter(List) (optional)
RegistrerBrukerInfo	
	- brukerInfo(List)

Tabell 5.14: Agenthandling i ontologien.

5.4.10 Steg 10 - Valg av innholdsspråk

JADE rammeverket støtter 3 forskjellige innholdsspråk, som er:

- **FIPA-SL**; kodet slik at det er leselig for mennesker.
- **LEAP**; en lettvekt binær koding, utviklet for små mobile enheter.
- **Java Codec**; designet for effektiv meldingsutveksling mellom agenter på samme plattform.

Valget av innholdsspråk for dette systemet blir relativt enkelt, da det må sørge for så lite data for overføring som mulig. Derfor blir innholdsspråket LEAP, utviklet for kommunikasjon mellom ressursbegrensede enheter, riktig for dette systemet.

Dette var siste steget i agentmetoden og neste kapittel tar for seg implementasjon av prototypen til *Intelligent Helper*. Metoden dekker, som tidligere nevnt, verken implementasjon eller testing av agentsystemet.

Kapittel 6

Implementasjon

Dette kapittelet dokumenterer implementasjonen av *Intelligent Helper*.

Seksjon 6.1 tar for seg verktøy og språk benyttet i implementasjonen.

Seksjon 6.2 presenterer ontologien i IH.

Seksjon 6.3 presenterer implementasjonen av agentene som er på serveren. Her presenteres implementasjon av serveragentene *LokaliseringsAgent* og *BrukerManagerAgent*. Funksjonene til *KalenderAgent* og *BrukerModellAgent* er ikke implementert, men noen designvalg blir diskutert.

Seksjon 6.4 presenterer implementasjonen av agenten på mobilen.

6.1 Utviklingsverktøy og programmeringsspråk

Denne seksjonen gir en forklaring på verktøy og språk som er benyttet under implementasjonen.

6.1.1 Programmeringsspråk

Implementeringen av IH skjer i programmeringsspråket Java. Som tidligere nevnt benytter JADEpL seg av J2ME versjonen av Java familien og tilbyr et API for utvikling av agenter til JADE rammeverket.

Det hadde vært mulig å implementere applikasjonen i Microsoft .NET, da JADEpL støtter dette. Grunnen til at Java er valgt til implementasjon er fordi applikasjonen skal fungere på mobiler, og det er et større antall mobiler som støtter Java. Dette legger mindre begrensning på hvilken mobil som kan benytte applikasjonen. Størsteparten av informasjonen og diskusjoner i Forum om JADE, omhandler implementasjoner gjort i Java, og denne støtten var verdifull under implementasjonen.

6.1.2 Protégé

Ontologien som ble designet for IH ble konstruert med verktøyet *Protégé* [1] Ontologien består av konsepter, predikater og agenthandlinger. En ontologi *beangenerator* plugin [37] for Protégé konverterte ontologien fra et Protégé prosjekt til Java kode. Dette tillegget var levert med et Protégé prosjekt, som ble inkludert i et nyopprettet prosjekt. Det inkluderte prosjektet inneholdt en abstrakt JADE ontologi, med klassene *Concept*, *Predicate* og *AgentAction*, og konstruksjonen av ontologier ble gjort ved å subklasse de abstrakte JADE klassene.

Pluginen som konverterte ontologien fra Protégé prosjektfiler til Java filer, hadde små feil i konverteringen. Dette gjorde at Protégé ikke ble benyttet etter at den innledende utviklingen av ontologien var ferdig. I steden ble endringer gjort manuelt i den konverterte ontologien.

6.2 Ontologi

Ontologien til IH heter **IHOntologi**, og består av totalt 10 konsepter, 1 predikat og 15 agenthandlinger.

Konseptene i IH omhandler blant annet **Bruker**, **WlanKlient** og **Posisjon**. Noen av konseptene i ontologien er ikke benyttet i prototypen, men var en del av designet og ble derfor tidlig implementert. Konseptene og det ene predikatet **Tilhorer**, vises i Appendiks 8.3, Figur 2, 2. Agenthandlinger i ontologien er også plassert i Appendiks 8.3, i Figur 4, 5 og 6.

Ontologien spesifiserer agenthandlinger både i vokabularet til ontologien som en *String*, og som en klasse med de parameterene som utgjør agenthandlingen. For agenthandlingen **HentPosisjonCRE** er det bare ett parameter av typen **jade.util.List**. Det holder **WlanKlient** konsepter fra ontologien, en for hver klient som skal posisjoneres og kommuniseres mellom agentene.

6.3 Server

Denne seksjonen tar for seg agenter på serveren. Innledningsvis introduseres bruken av tjenesteregistrering og oppdagelse JADEpL tilbyr, og noen funksjoner som er felles for agentene.

6.3.1 Administrasjon av tjenester

Agentenes løse kobling er noe av styrken med agentbasert utvikling. Den sentrale oppføringen av tjenester hindrer overflødig kommunikasjon, som ville vært

[1] Protégé er en gratis, Java basert med åpen kildekode, ontologi-editor og rammeverk for kunnskapsbaser. http://protege.stanford.edu/

nødvendig hvis agentene måtte søke eller meddele sine tjenester til samtlige agenter.

Agentene har noen felles funksjoner, som håndterer viktige aspekter med agentarkitekturen. Funksjonene omhandler agentenes tjenester og hvordan de registrerer seg hos DF.

6.3.1.1 Registrering av tjenester

Agentene implementert i IH registrerer tjenester gjennom oppførselen **RegisterServiceDF**, som er en **OneShotBehaviour**.

Agenten vil gi en beskrivelse av seg selv ved å angi sin AID, ontologien den benytter og språket(*LEAP*). Det er også mulig å registrere interaksjonsprotokoll på agentbeskrivelsen, men dette er ikke gjort for noen av agentene i IH.

Beskrivelsen av tjenestene blir gjort ved å opprette et Java objekt, *ServiceDescription*, for hver tjeneste. Beskrivelsen av tjenestene er mer utfyllende enn agentbeskrivelsen, slik at brukeren vet hva tjenesten forventer når en forespørsel om å uføre en handling sendes.

Hver agent på serveren tilbyr tjenester i IH og vil registrere seg. Et eksempel på dette er **LokaliseringsAgent**, som registrerer tjenestene *HentPosisjonCRE* og *HentListeCRE*, som er agenthandlinger. Agenthandlingene som er spesifisert i ontologien vil være tilgjengelig for samtlige agenter som har den samme ontologien.

6.3.1.2 Søk etter tjenester

Søk etter tjenester hos DF er implementert i nesten alle agentene, inkludert **BrukerAgent** på mobilen, men ikke i **LokaliseringsAgent**. Dette er fordi LokaliseringsAgent ikke innleder noen konversasjoner.

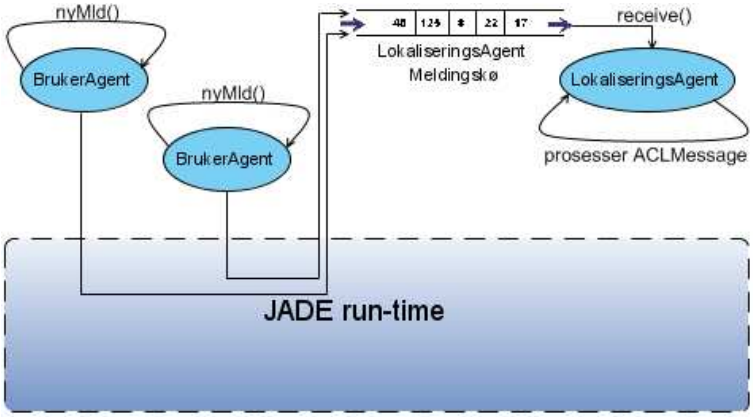
Når det søkes etter agenter opprettes et *array* [2] av samme typen objekt som blir brukt under registreringen av tjenester, da dette er returverdien som mottas når det søkes hos DF. Resultatet traverseres og man ser om den agentbeskrivelsen man søker, for eksempel *HentPosisjonCRE*, er registrert hos DF. Blir tjenesten funnet lagres AID til den tilbydene agenten, slik at forespørsler kan benytte denne id'en.

6.3.1.3 Opprydning i meldingskø

Samtlige agenter i serverpakken har blitt utstyrt med en oppførsel som rydder opp i meldingskøen. Dette hindrer meldingskøen i å bli overbelastet og stoppe all kommunikasjon med agenten. Meldingene som blir håndtert sist vil stort sett få

[2] Array er en ordnet datastruktur av dataelementer, som aksesseres via en indeks.

svar for sent, og vil forsøke å sende på nytt. Meldingskøen kan få en opphopning fordi mottakeren ofte må gjøre en del prosessering ved mottak av en melding og den forespørselen meldingen inneholder. Samtidig kan den få mange meldinger på samme tid inn i meldingskøen, fra ulike agenter i systemet. Figur 6.1 viser hvordan meldinger går fra avsender(initiator) og til meldingskøen hos mottaker.



Figur 6.1: Kommunikasjon mellom senderagenter og en mottaker.

6.3.2 LokaliseringsAgent

Denne agenten er en *transformat* agent og håndterer kommunikasjon med den eksterne ressursen CRE.

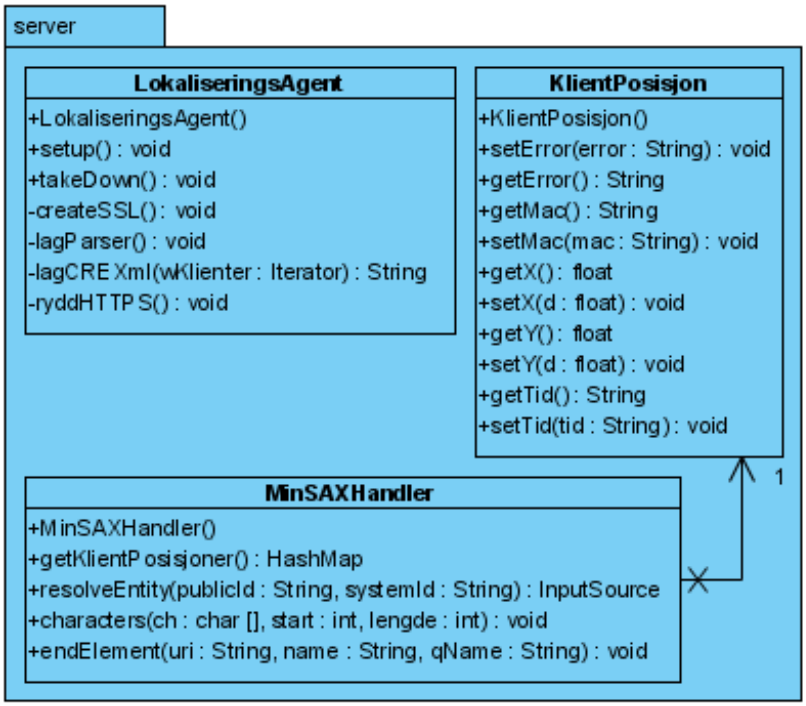
Figur 6.2 viser klassediagrammet til LokaliseringsAgent. Konseptene og agenthandlingene som er i bruk fra ontologien vises i Appendiks 8.3, agenthandlingene i Figur 4, og Figur 2 for konsepter.

Agenten vil starte opp med å opprette diverse objekt(*instanser*) [3] som trengs under agentens levetid. Den oppretter et objekt av typen **IHOntologi**, og et objekt av en *ContentManager*, som er en klasse som håndterer de språkene og ontologiene agenter benytter.

ContentManager benyttes også for å fylle en ACL melding med objekter fra ontologien, med metoden *fillContent*. Den benytter seg da av språket og ontologien som spesifiserer hvordan en de enkelte objektene skal oversettes før sending. I *JADEpL* er det klassen **ACLMessage** som implementerer ACL meldinger, og følger *FIPA* direktiver. Oppretting av disse objektene er felles for samtlige agenter.

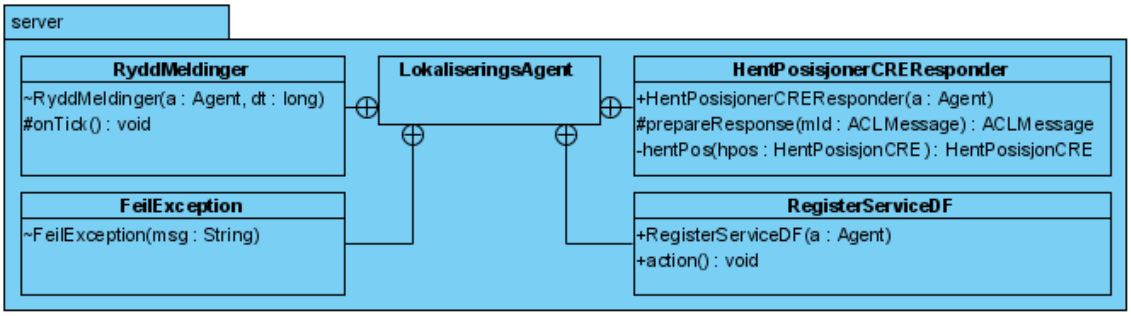
LokaliseringsAgent vil under oppstart også legge til de oppførsler som skal kjøre og behandle forespørsler fra andre agenter. Agenten benytter metoden *addBehaviour* fra klassen **jade.core.Agent**, som er agentens superklasse, for å starte oppførselene. Oppførselene i LokaliseringsAgent er **RegisterServiceDF**, **HentPosisjonerCREResponder** og **RyddMeldinger**. Disse oppførselene er imple-

[3] En instans oppretter et objekt av en klasse.



Figur 6.2: Klassediagram til LokaliseringsAgent.

mentert som subklasser av henholdsvis *OneShotBehaviour*, *SimpleAchieveREResponder* og *TickerBehaviour*. Figur 6.3 viser indre klasser som utgjør oppførselene til LokaliseringsAgent.



Figur 6.3: Interne klasser i LokaliseringsAgent.

For at LokaliseringsAgent skal kommunisere med CRE innebærer det oppretting av en sikker forbindelse. Dette gjøres ved å benytte HTTPS via TCP, og som tidligere nevnt har Java støtte for HTTPS og håndtering av PKI for autentisering.

Oppkoblingen til CRE skjer først når agenten mottar en FIPA-Request ACL melding, som enten ber om en *HentPosisjonCRE* eller *HentListeCRE*. Siden sending over HTTPS må autentiseres blir det opprettet en sikker tilkobling mot CRE, og dette gjøres kun ved første oppkobling for å sikre at sending gjøres så raskt som mulig. En oppførsel som legges i agentens *kjøreplan*, vil kjøre uten avbrytelser, til den er ferdig. Dette blokkerer agentens andre oppførsler og må derfor gjøres

så rask som mulig.

Det er metoden **HentPosisjonerCREResponder**, en subklasse av **SimpleAchieveREResponder**, som håndterer forespørsler om posisjonering. Metoden henter ut innholdet i ACL meldingen og sjekker hvilken agenthandling som er med, som kan være av typen **HentPosisjonCRE** eller **HentListeCRE**. HentListeCRE er en "tom" agenthandling som identifiserer at LokaliseringsAgent skal hente samtlige posisjoner som CRE registrere, og returnere dette. Hvis det derimot er en HentPosisjonCRE, er det mulig at det er med en liste med **WlanKlient**, som er et konsept i ontologien, og LokaliseringsAgent må hente kun de forespurte klientene. Under presenteres litt om sending og mottak av XML gjort i LokaliseringsAgent.

Sending av XML Når LokaliseringsAgent mottar en forespørsel om posisjonering via en ACL melding kan det følge med et antall klienter som ønskes posisjonert. Klientene er av typen WlanKlient, og gjør at LokaliseringsAgent mottar en forespørsel som den kan håndtere. Agenten oppretter gjennom metoden **lagCREXml**, et XML **wpp_init** dokument, som CRE forstår, og legger til en **lir** for hver klient. Det opprettede dokumentet returneres fra metoden, og sendes til CRE.

Mottak av XML Et viktig valg som ble gjort i analysen var å gjøre parsingen av XML fra CRE på serveren, for så å kommunisere resultatet videre. Det hadde vært mulig å gjøre parsing av XML på mobiltelefonen, men som nevnt tidligere er formålet med LokaliseringsAgent å fjerne den belastningen fra mobilapplikasjonen.

For å håndtere parsing av XML dokumenter fra CRE ble det benyttet *Xerces* [4], en parser basert på Java.

Klassen **MinSAXHandler** subklasser *DefaultHandler*, og denne får tilgang til elementenen i XML dokumentet. Her hentes posisjoner, tidspunkt, og MAC adressen ut for hver posisjonerte WLAN klient som er mottatt i dokumentet. En datastruktur holder på objekter av klassen **KlientPosisjon**, som er en klasse som holder på posisjoneringsdataene.

Når MinSAXHandler har parset XML dokumentet hentes alle KlientPosisjon objektene ut og oppretter en WlanKlient for hver, og en ACL melding sendes tilbake til initiatoren.

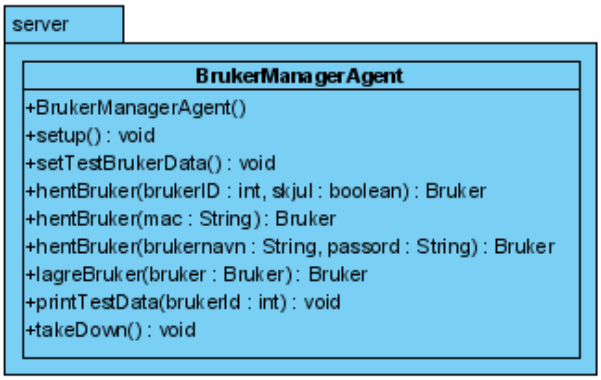
Hvis LokaliseringsAgent ikke forstår innholdet i meldingen som er mottatt, eller ikke finner noen registrerte posisjoner fra CRE, vil en agenthandling *Informer*, sendes i retur. Innholdet i Informer er en **int** [5] verdi som er spesifisert i ontologien. Hvis LokaliseringsAgent ikke forstår meldingen som er mottatt vil den sette *performative*, intensjonen med meldingen, til *ACLMessage.FAILURE*. Klassen ACLMessage har samtlige performativene inkludert, og benyttes for å sette intensjonen. Hvis agenten derimot utfører en sjekk mot CRE, men ingen posisjoner blir funnet, blir en *Informer* agenthandling sent i retur. Informer handlingen vil merkes med en int som har navnet **IHOntology.INGEN_FUNNET**.

[4] http://xerces.apache.org/xerces-j/
[5] int er en primitiv tallverdi i Java.

6.3.3 BrukerManagerAgent

BrukerManagerAgent er en implementasjon av agenten mellom brukerdatabasen og andre agenter, som spesifisert i analysen. Prototypen er kun utstyrt med testdata og ikke en operativ database.

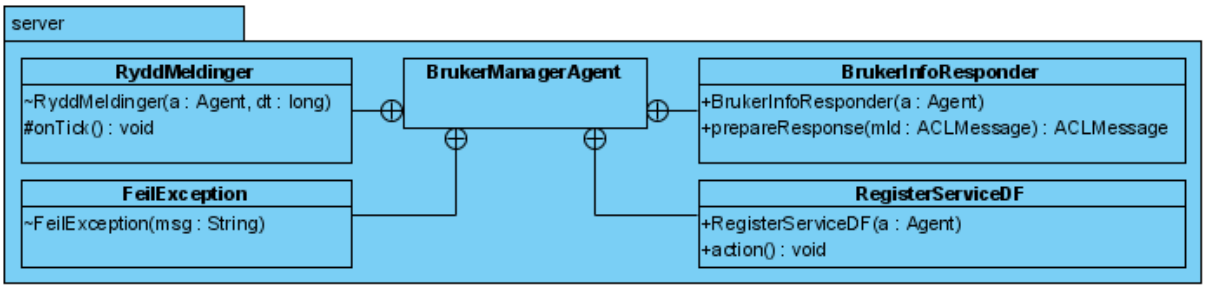
Klassediagrammet til agenten vises i Figur 6.4. Klassediagram for *Bruker*, *WlanKlient* og *Posisjon* fra ontologien, ligger i Appendiks 8.3. Figur 2 viser Bruker, og Figur 3 viser WlanKlient.



Figur 6.4: Klassediagram til BrukerManagerAgent.

Agenten håndterer FIPA-Request forespørsler fra **BrukerAgent**, som ønsker å få utført en av agenthandlingene *RegistrerBrukerProfil*, *EndreBrukerProfil* eller en annen av de 6 agenthandlingene som omhandler brukerhåndtering. Agenthandlingene til BrukerManagerAgent vises som klassediagram i Appendiks 8.3, i Figur 5.

I BrukerManagerAgent er det den indre klassen **BrukerInfoResponder** som håndterer innkommende FIPA-Request ACL meldinger fra andre agenter og sjekker hvilken agenthandling som anmodes utført. Figur 6.5 viser indre klasser som utgjør oppførselene til BrukerManagerAgent.



Figur 6.5: Interne klasser i BrukerManagerAgent.

Hvis en agent ønsker å utføre en agenthandling som involverer brukerprofilen i databasen, sjekkes det for brukernavn og passord. Agenten henter ut brukerprofi-

len til en bruker gjennom metoden **hentBruker**, som tar brukernavn og passord som parameter.

Lagring av en ny brukerprofil håndteres i metoden **lagreBruker**, og den sjekker om *Bruker*, et objekt fra ontologien, inneholder en *brukerID*. Hvis den mangler lagres en ny profil, med en ny brukerID, og et oppdatert *Bruker* objekt sendes i retur.

Endring av en brukerprofil vil benytte seg av samme metode, men da vil Bruker objektet ha med en brukerID, og databasen oppdateres.

Ved registrering av en ny profil vil tilgangsnivået, **AccessLevel**, som er et parameter i Bruker og i databasen, settes slik at brukeren får tilgang til de ekstra funksjoner som en registrert bruker skal ha.

BrukerManagerAgent er eneste agent som har tilgang til informasjon om brukere. Denne agenten håndterer derfor agenthandlingen **HentBrukerInfo**, som ber om et utvalg brukere ut fra en liste med **WlanKlient** objekt. Fra denne listen henter agenten ut samtlige WlanKlient objekt og fra dem, MAC-adressene. Deretter går den gjennom databasen og finner frem informasjon om eieren av den aktuelle MAC-adressen, hvis den MAC adressen er registrert på en bruker. Før samlingen med Bruker objekt hentet fra databasen blir sendt i retur, blir brukernavn og passord slettet fra objektet, slik at det ikke distribueres over nettverket. Svaret på denne handlingen blir **SendBrukere**, som inneholder en liste med Bruker objekt.

6.3.4 KalenderAgent & BrukerModellAgent

Det meste av logikken til KalenderAgent og BrukerModellAgent er ikke implementert, og de har kun basisfunksjonene for registrering til DF.

Intensjonen med KalenderAgent var å administrere avtaler og hendelser for brukere. Agenten skulle også ved pålogging til systemet, samt når en hendelse ble registrert, gå gjennom brukerens interesser og se om noen matchet beskrivelsen av hendelsen.

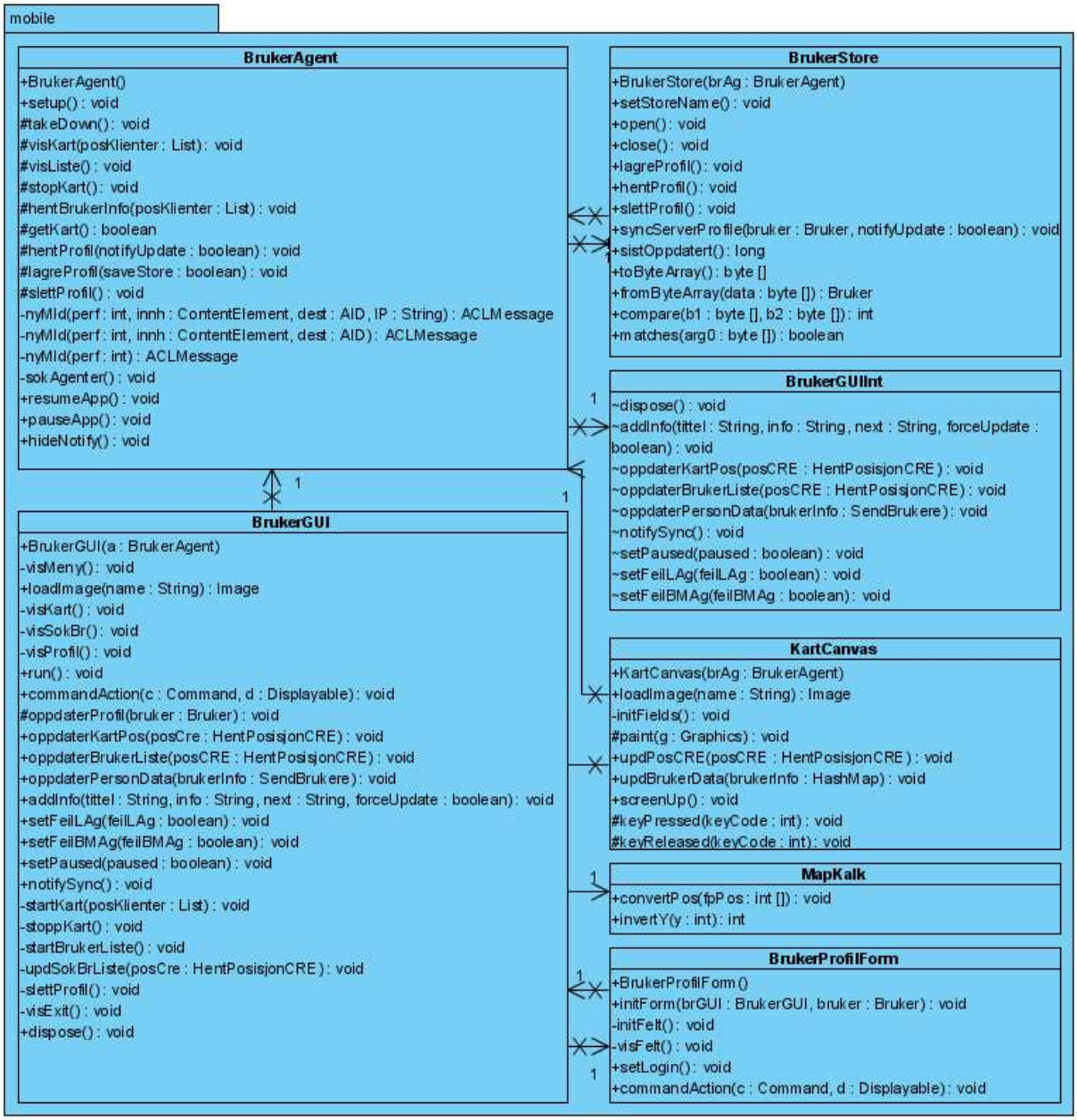
Felt i en **Avtale**, som den er implementert i ontologien, er *avtaleID*, *brukerID*, *tittel*, *tekst*, *tags*, *fratid*, *tiltid*, *viewlevel*, *modified* og *brukerInfo*. Listen **brukerInfo** består av **Bruker** fra ontologien, og er for avtaler som er mellom 2 eller flere brukere. Feltet **tags** er ment for å inneholde stikkord som beskriver hendelsen, slik at hendelsen kan matches mot interesser til brukere.

For å tilrettelegge for annonsering av hendelsen ville brukeren, under registrering, merke hendelsen med en verdi *public* eller *private*. Slik kunne felles hendelser annonseres over systemet, til samtlige registrerte brukere, eller kun for privat bruk.

BrukerModellAgent skulle implementere Prolog-motoren for å oppdatere brukermodellene til brukere, med nye egenskaper fra stereotyper.

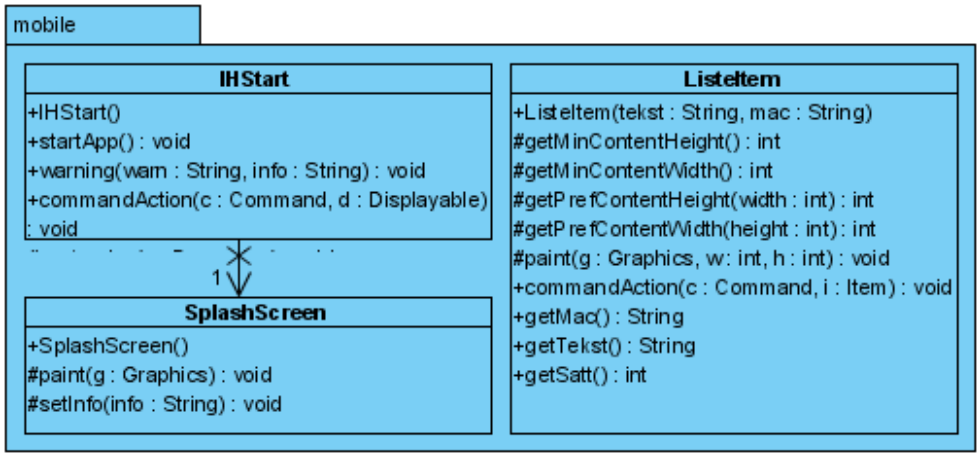
6.4 Mobil

mobile pakken inneholder klassene til mobil-agenten og er gjort opp av 10 egne klasser. I tillegg har klassen **BrukerAgent** flere indre klasser som blant annet er oppførslene til agenten. Klassediagram for BrukerAgent vises i Figur 6.6, hvor agentens hovedklasse og dens relasjoner er spesifisert. Figur 6.7 viser oppstarts-klassen **IHStart** sammen med to tilleggsklasser.



Figur 6.6: Klassediagram til BrukerAgent.

IH på mobilen startes i klassen **IHStart**. Denne klassen er en utvidelse av klassen **MicroBoot**, som er en JADEpL klasse. Den sørger for å starte en **FrontEnd** Container på mobilen og kobler seg opp til **Main-Container** på serveren.



Figur 6.7: Klassediagram til BrukerAgent.

Hvis denne klassen ikke klarer å koble seg opp til serveren vil ikke applikasjonen starte. JADEpL leverer **jade.util.leap.OutputViewer**, en MIDlet som lar brukeren se på rapporterte meldinger, og deriblant feilmeldinger.

IHStart håndterer innloggingen til IH, og viser fire felt som skal benyttes videre i **BrukerAgent**.

- **brukernavn**(valgfritt). Uten dette(og passord) vil brukeren allikevel bli pålogget og betraktes som gjest.
- **passord**(valgfritt). Hvis brukernavn er inntastet er dette feltet påkrevd. Begge feltene brukernavn og passord tillater 8 tegn.
- **oppdateringsfrekvens**(valgfritt). Dette feltet angir hvor ofte agenten vil oppdatere kartvisningen. Minste tillatte verdi er 2 sekunder.
- **MAC**(påkrevd). Denne adressen blir agentens ID(AID) og må være med for at IH skal fungere som tenkt. Det stilles derfor krav til innfyllingen og hvis brukeren har for få tegn i MAC feltet blir dette varslet på skjerm og en forklarende tekst blir vist. Når applikasjonen logger seg på(AP), vil agenten bli stoppet hvis AID ikke er unik, varsler brukeren om dette, og inntasting må korrigeres.

Klassen **BrukerAgent** som subklasser **jade.core.Agent**, er mobilapplikasjonens hovedklasse. Den inneholder de oppførslene som gjør at agenten kan fullføre sine oppgaver i systemet, og oppretter en instans av mobilens grafiske grensesnitt.

IH-mobile oppstart

Når applikasjonen starter opp opprettes samme objekter som LokaliseringsAgent, og resten av agentene, for å kunne kommunisere med riktig språk og ontologi. Videre benytter samtlige agenter seg av et verktøy for å håndtere feilmeldinger og andre meldinger som kan være viktige å lagre. Det er som tidligere nevnt *OutputViewer*, og den kan ta imot feilmeldinger som "kastes" [6] i agenten. For

[6]Melding som kastes etter en uventet hendelse.

mobilapplikasjonen er denne meget nyttig da den kan gi brukeren en indikasjon på hva som er feil, hvis applikasjonen ikke fungerer som den skal.

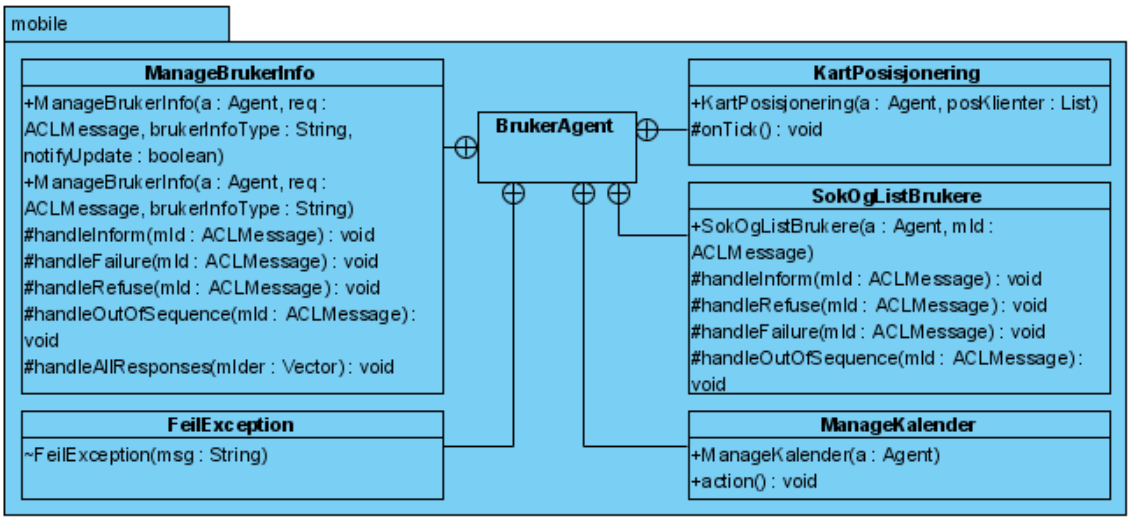
Metoden **sokAgenter** kjøres ved oppstart og søker etter de agentene som tilbyr tjenestene BrukerAgent har behov for.

BrukerAgent oppretter også et objekt av klassen **BrukerStore**, som er mobilapplikasjonens persistente minne, en **RecordStore**. Den benytter seg av et API for lagring på MIDP telefoner. Klassen holder også på et objekt av klassen **Bruker** fra ontologien.

BrukerStore sørger for åpning og lukking av minnet, samt lagring, henting og sletting av data.

Når applikasjonen starter vil det bli hentet ut den lokalt lagrede profilen til brukeren, og lagt i et objekt av typen Bruker. Dette vil kun skje hvis brukeren har fyllt inn brukernavn og passord, og at dette stemmer med det som er lagret på mobilen.

Ved lagring av en brukerprofil vil **modified** parameteret i objektet Bruker oppdateres, for å kunne synkronisere med IH sentralt. Synkronisering skjer ved at dette parameteret i brukerens profil på serveren blir sjekket mot de lokalt lagrede dataene. Brukerens data hentes ved oppstart av mobilapplikasjonen ved å legge til oppførselen **ManageBrukerInfo**, som må initieres med en ACL melding. I denne meldingen legges det et objekt av agenthandlingen **HentBrukerProfil** fra ontologien. Agenthandlingen har tatt med seg brukerenes Bruker objekt, som skal brukers på serveren for å autentisere forespørselen. Objektet av typen Bruker som kommer i retur blir sjekket mot de lokalt lagrede dataene, og hvis det var nyere data på serveren vil dette bli brukt på mobilen. Passord og Brukerenavn som ble tatt bort ved sending blir først lagt med i det nye objektet Bruker. Figur 6.8 viser indre klasser som utgjør oppførselene til BrukerAgent.



Figur 6.8: Interne klasser i BrukerAgent.

Ved å håndtere brukerinfo på denne måten blir begrensningen til å måtte benytte

én mobil tatt bort. Den siste profilen på serveren vil oppdatere den nye mobilen.

En viss funksjonalitet er mulig for brukere som ikke har registrert en gyldig profil. For brukere som logger seg på med kun en MAC adresse, blir funksjoner for søk etter online brukere, og kartvisning tilgjengelig.

Det er klassen **BrukerGUI** som håndterer brukergrensesnittet i applikasjonen, og BrukerAgent benytter seg av dette gjennom interfacet **BrukerGUIInt**. BrukerGUI oppretter en meny utifra hvorvidt brukeren er en registrert bruker eller en gjest(uten brukernavn/passord). Menyen som vises gir valgene *Vis kart*, *Søk online brukere*, *Registrer profil*.

Søk online brukere

Valget "Søk online brukere", kaller i BrukerAgent en metode **visListe**, som første sjekker at den har funnet en agent av typen LokaliseringsAgent, som tilbyr tjenesten **HentListeCRE**. Hvis den ikke har funnet en slik agent, blir brukeren varslet og metoden *sokAgenter* kjøres igjen.

Hvis en LokaliseringsAgent er funnet starter BrukerAgent oppførselen **SokOgListBrukere**. som subklasser **SimpleAchieveREInitiator**. Dette er en klasse som implementerer FIPA-Request interaksjonsprotokollen. Oppførselen startes med en ACL melding som parameter, hvor det er lagt til en **HentListeCRE** agenthandling for å få samtlige klienter som er posisjonert av CRE.

Hvis LokaliseringsAgent finner posisjonerte brukere mottar BrukerAgent en **HentPosisjonCRE** som inneholder de klienter som er funnet. Hvis det derimot ikke rapporteres noen posisjonerte brukere, vil brukere få opp en melding som varsler dette, før applikasjonen viser hovedmenyen.

Metoden **oppdaterBrukerListe** i BrukerGUI tar imot posisjonerte klienter, som inneholder objekter av typen **WlanKlient**, med MAC-adresse som innhold. Metoden går gjennom samtlige objekter og sjekker om det finnes brukerinfo for hver posisjonerte MAC-adresse tilgjengelig på mobilen. Hvis ikke blir det hentet fra BrukerManagerAgen, gjennom oppførselen ManageBrukerInfo, med agenthandlingen **HentBrukerInfo** som parameter. Når brukerinfo er mottat kan listen vises for brukeren, og dette gjøres ved å legge de posisjonerte brukerene inn i en liste.

For å søke ut en enkelt person iblant mange ansatte, ble listen implementert med en søkefunksjon som tar bort brukere i listen ettersom det tastes inn bokstaver i søkefeltet. Det sjekkes både på fornavn og etternavn i listen, for å gjøre det lettest mulig for brukeren.

Når den, eller de brukerene som ønskes posisjonert er funnet i listen, merkes de i listen, og brukeren godkjenner for å gå videre. Dette gjør at WlanKlient objektene for de brukerene som ble merket i listen, blir lagt til i en HentPosisjonCRE, som legges ved når oppførselen **KartPosisjonering** startes. Denne oppførselen er av typen **TickerBehaviour**, som er metoden som kjøres i en intervall spesifisert av parameteret fyllt inn under innloggingen. Denne oppførselen legger internt til en SimpleAchieveREInitiator, og sender en ACL melding til LokaliseringsAgent,

med de utvalgte WlanKlient objektene.

Posisjoner blir mottatt fra LokaliseringsAgent og kartrepresentasjonen som nå allerede er startet viser de utvalgte brukerene. Brukerdata om de utvalgte brukerene er allerede på mobilen og ingen forespørsler går til BrukerManagerAgent.

Det er klassen **KartCanvas** som håndterer kartrepresentasjonen og legger til brukere. Den tegner opp ved å gå gjennom samtlige WlanKlient objekt som den har mottatt, og henter ut brukerinfo fra den lagrede informasjonen om brukerene.

Hvis brukeren av applikasjonen har valgt kun én person i listen, vil en pil indikere i hvilken retning den ene valgte brukeren befinner seg. Skjermen til en mobil er så liten at en representasjon av hele Den Intelligente Korridor vil være lite hensiktsmessig. Hvis da destinasjonen befinner seg et stykke bortenfor vil figuren som indikerer brukeren bli utenfor skjermvisningen. Visning av en pil som indikator skal gjøre det lettere og sikre at brukeren føler seg trygg.

Kartrepresentasjonen kan som nevnt startes uten å søke ut personer i søkelisten, og da vil samtlige registrerte brukere vises. For at applikasjonen skal vise navn til posisjonerte brukere, blir listen av WlanKlient objekt som blir mottatt fra LokaliseringsAgent gjennomgått, og sjekket mot den lagrede brukerinfoen. Hvis noen mangler sendes det en forespørsel til BrukerManagerAgent, og nye posisjonerte brukere blir vist med navn.

Y aksen på mobiler er motsatt vei i forhold til det som er vanlig, slik at motatt posisjon fra CRE må inverteres i Y aksen. Forholdet mellom posisjonene mottatt fra CRE må også skaleres til kartrepresentasjonen på mobilen, og dette er det klassen **MapKalk** som håndterer. Forholdene som må tas i betraktning er det lokale koordinatsystmet til CRE, den totale høyden og bredden på kartet i mobilen, og hvilken forskyvning kartet har i forhold til radioøyet.

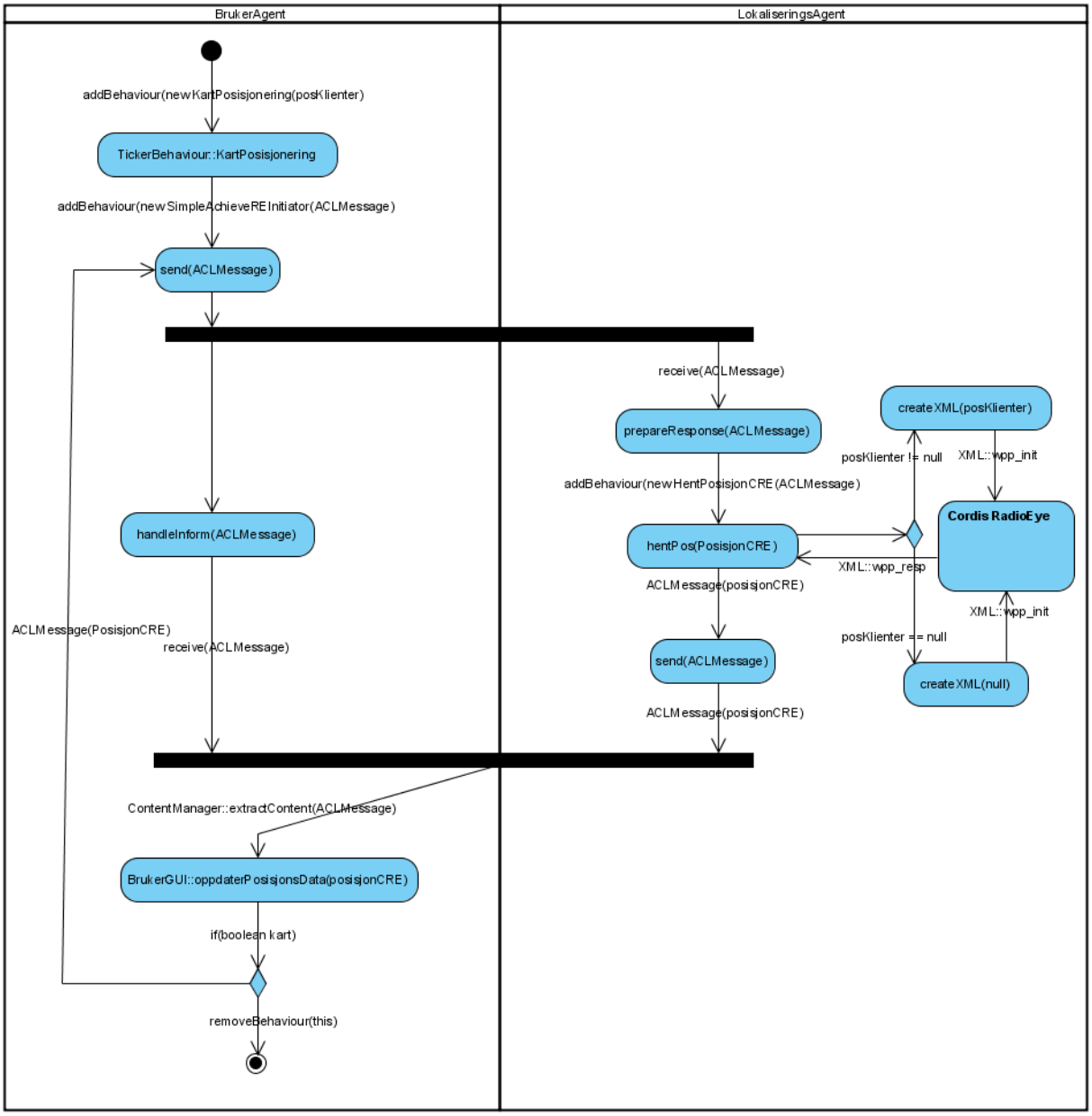
Oppførselen KartPosisjonering som oppdaterer posisjoneringsdata til kartet, vil kjøre så lenge en *boolsk* [7] verdi er sann. Når brukeren velger å avsluttet kartrepresentasjon, vil også oppførselen stoppe sin kommunikasjon med LokaliseringsAgent.

Hvis BrukerAgent mister kontakt med LokaliseringsAgent eller BrukerManagerAgent under kartvisningen blir dette varslet brukeren gjennom informasjon som kommer opp på kartrepresentasjonen. De siste brukerdata vil fortsatt være tilgjengelig, men alt etter hvilken agent på serveren som mangler, vil posisjonene eller brukerinfo ikke bli oppdatert.

Aktivitetsdiagrammet i Figur 6.9, viser hvordan BrukerAgent og LokaliseringsAgent samspiller for å få oppdatert mobilens kartvisning.

Registrere profil For å bli en registrert bruker i IH må en brukerprofil registreres. Dette er implementert ved å lage et registreringsskjema i mobilapplikasjonen, som innhenter de data som trengs om brukeren. Skjemaet er implementert som subklasse av en MIDP *Form* i mobilen, og heter **BrukerProfilForm**. Den legger

[7] Verdi som kan være enten sann eller falsk.



Figur 6.9: Aktivitetsdiagram for kartvisning.

til eventuell data som brukeren har registrert fra før, eller viser en blank form hvis brukeren ikke har registrert en profil. Formen inneholder de felt som er i et Bruker objekt i ontologien.

Kapittel 7

Demonstrasjon

Dette kapittelet viser en demonstrasjon av *Intelligent Helper*.

Seksjon 7.1 viser en demonstrasjonen av IH. Det blir gjort gjennom å følge flere scenario for å vise funksjonaliteten til *Intelligent Helper*.

Seksjon 7.2 viser en evaluering av design og implementasjon.

7.1 Scenario

Intensjonen med IH var å gi bistand til besøkende og ansatte i *Den Intelligente Korridor*, i 3. etasje ved Institutt for Datateknikk og Informasjonsvitenskap. Det er benyttet testdata for posisjoner i scenarioene. Navnene i figurene er noen ansatte, samt oppdiktede karakterer.

Følgende scenario viser hvordan en gjest som ankommer 3. etasje ved IDI kan bli assistert av IH.

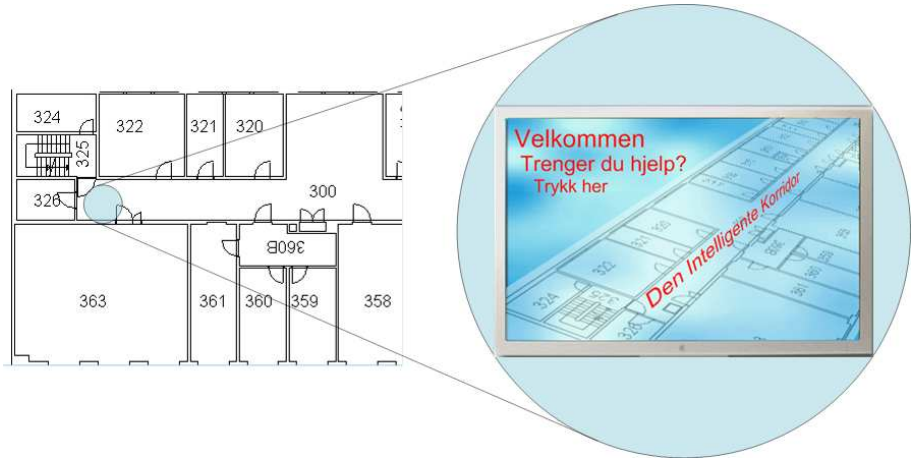
7.1.1 Besøkende

Kjell, en student ved NTNU, har et problem med en oppgave i faget, "Kunstig intelligens", som han følger. Kjell har som mål å finne Agnar, en professor som underviser i dette faget. Kjell har aldri vært i 3. etasjen ved IDI før, så han vet ikke hvor han skal finne Agnar der. Ved et mirakel finner Kjell trappen som går opp til 3. etasje og kommer inn i ***Den Intelligente Korridor****. Det første Kjell ser foran seg når han går inn i gjennom døren fra trappeoppgangen er en LCD* [1] *skjerm montert på veggen, hvor det står en velkomsttekst (se Figur 7.1)* [2]*.*

Kjell ser litt rundt seg og ser at nærmeste dør bak seg er kontoret til Agnar, og banker på. Ingen svarer, og siden Kjell har det travelt trykker han på den

[1] Liquid Crystal Display, en type flatskjerm.

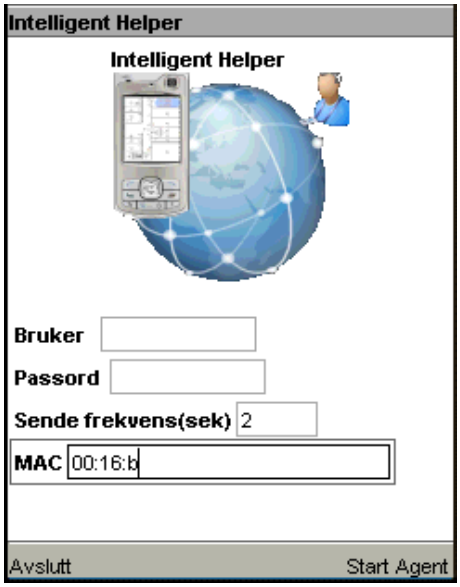
[2] Ikke en del av IH, men en LCD skjerm er montert i 3.etasje, som har andre funksjoner



Figur 7.1: *Intelligent Helper* velkomstsone.

trykkfølsomme skjermen i gangen, hvor skjermen forandrer innhold og viser en meny. Ett av valgene er en beskrivelse for hvordan en kan laste ned mobilapplikasjon Intelligent Helper, som tilbyr en søkefunksjon etter ansatte og en kartrepresentasjon som guider frem til målet. En liten beskrivelse av applikasjonen vises. Kjell laster ned IH og trykker ***OK*** *for å akseptere installasjonen på mobilen.*

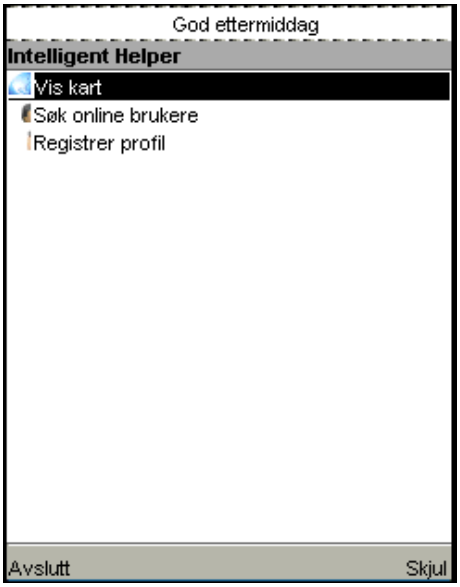
Straks får han opp innloggingen til IH, og siden mobilen hans har WLAN innebygd, fyller han inn MAC-adressen (se Figur 7.2). Kjell fyller ikke inn brukernavn eller passord da han skal kun søke opp Agnar og benytte kartrepresentasjonen.



Figur 7.2: *Intelligent Helper* login.

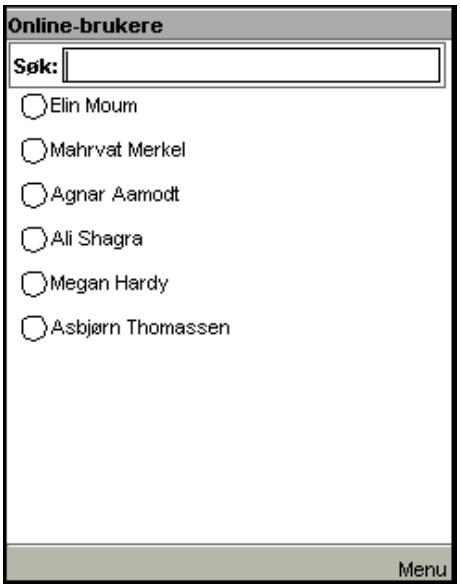
Menyen han får opp i IH gir mulighet for å søke etter brukere, vise en kartrepresentasjon og registrere en ny profil (se Figur 7.3).

Kjell trykker på valget "Søk online brukere" i menyen og IH henter de ansatte og registrerte brukere CRE registrerer (se Figur 7.4). Listen over ansatte er lang og



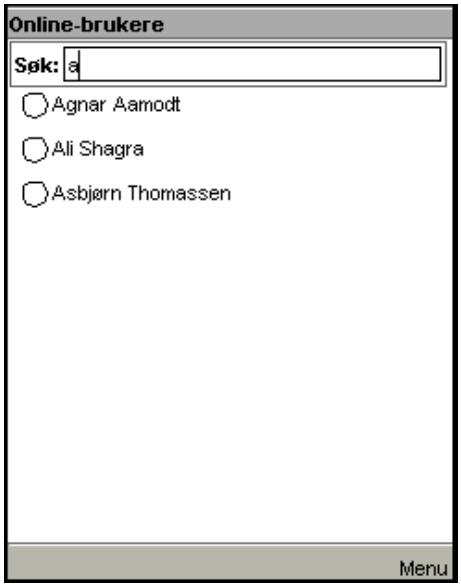
Figur 7.3: *Intelligent Helper* - meny for uregistrerte brukere.

han benytter seg av søkefunksjonen for å finne frem til Agnar. Mens han taster inn bokstavene i etternavnet til Agnar, reduseres listen etter hvert og viser de som passer til søket (se Figur 7.5). Raskt ender han opp med bare Agnar og markerer ham i listen (se Figur 7.6).

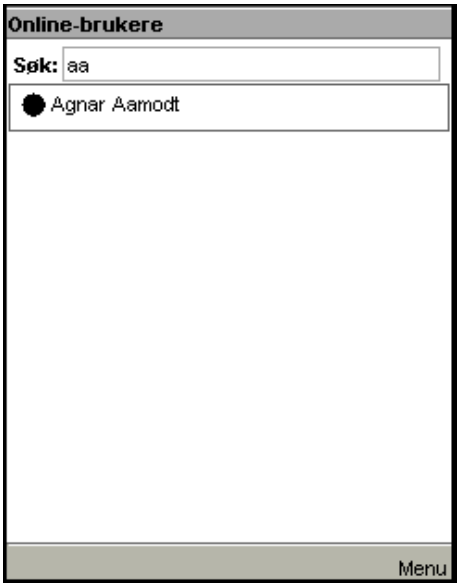


Figur 7.4: IH - søkefunksjon

Deretter trykker han ***OK****, og IH viser kartet over Den Intelligente Korridor og hvor Agnar befinner seg (se Figur 7.7). Når Kjell beveger seg i retning Agnar oppdaterer IH kartet med nye posisjoner til både Agnar og Kjell, slik at Kjell føler seg hele tiden trygg på at han beveger i riktig retning (se Figur 7.8). Kjell går frem til kontoret hvor Agnar er avmerket i IH og indikatoren viser retningsendringen*

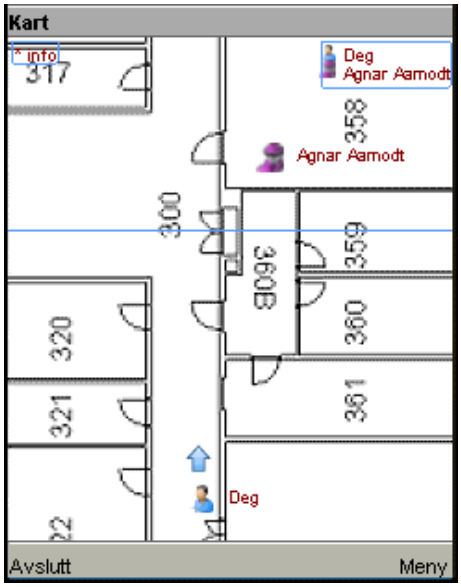


Figur 7.5: IH - søkefunksjon



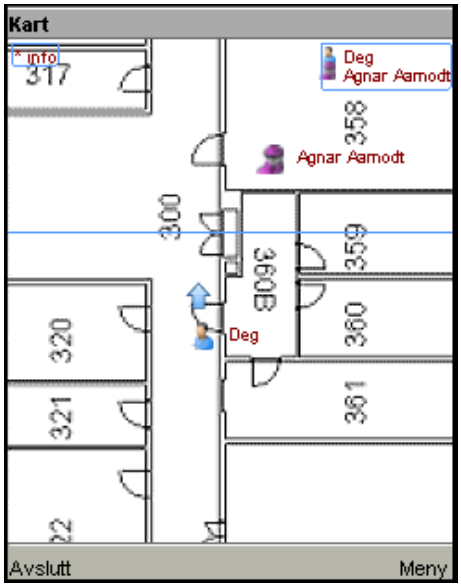
Figur 7.6: IH - søkefunksjon

han må ta (se Figur 7.9).

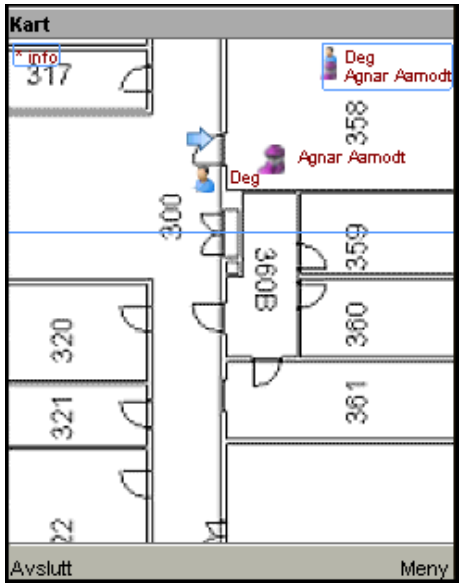


Figur 7.7: IH guider besøkende til destinasjon.

Dette scenarioet viste hvordan gjester, slik som Kjell, kan bli assistert ved hjelp av en oversiktlig og informativ applikasjon som IH. Neste scenario viser hvordan brukere som registrerer seg vil få utvidet funksjonalitet i IH.



Figur 7.8: IH guider besøkende - Steg 2.



Figur 7.9: IH guider besøkende - Steg 3.

7.1.2 Registrerte brukere

Hilde Melkersen er nylig ansatt ved IDI og prøver å komme seg til rette der hvor hun har kontor i 3.etasje, IDI bygget. Hun er ny og ukjent med rutiner og hvordan kollegaene hennes fungerer i løpet av en dag.

Hilde har hørt om det nye agentbaserte systemet som er satt opp i Den Intelligente Korridor, som blant annet viser hvor kollegaene befinner seg. Hilde har sett at kollegaene bruker å ha lunsj i en sofakrok litt lengre bort i korridoren, og finner ut at hun skal benytte seg av IH for å se når det passer best å ta lunsj. Hilde finner web-adressen hvor hun kan laste ned IH og henter den ned.

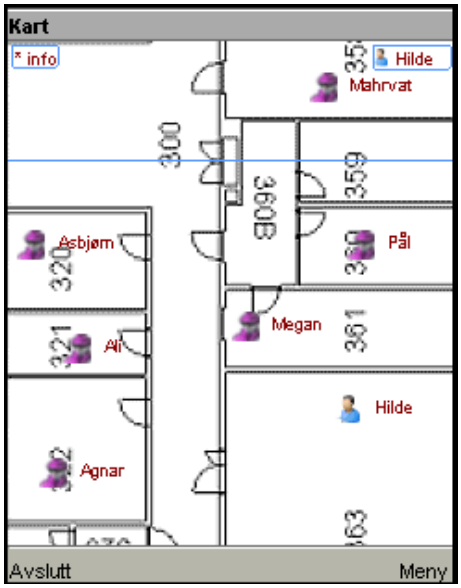
Hilde husker på å legge inn MAC-adressen når hun starter applikasjonen og går videre til registrering i menyen (se Figur 7.10). Hilde fyllte ikke inn brukernavn og passord, noe som er påkrevd ved registrering, og blir varslet om dette (se Figur 7.11) når hun forsøker å lagre profilen sin. Hun fyller inn i feltene nederst på siden, lagrer og får en melding om at profilen hennes er registrert.

Siden Hilde ønsker å finne ut når noen samler seg i sofaområdet velger hun å benytte kartvisning uten å søke opp noen i søkefunksjonen. Da vises samtlige som registreres av CRE. Da hun tenker at mobilen kommer til å være koblet opp til nettverket en stund, velger hun å benytte seg av feltet i innloggingen hvor hun kan bestemme oppdateringshastigheten. Hun setter så på kartvisningen, hvor den oppdaterer seg hvert tjuende sekund, som hun selv har satt (se Figur 7.12). Hilde tenker at dette også kan være en god mulighet til å få med seg navnene på de nye kollegaene, og benytter seg av valget i IH hvor kartrepresentasjonen også viser etternavnet (se Figur 7.13).

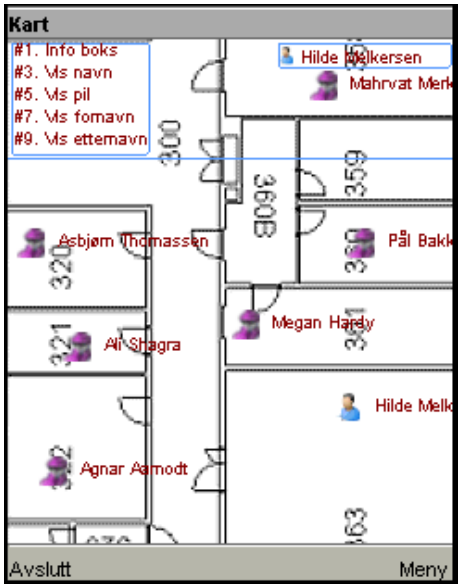
Etter et kvarter begynner det å samle seg noen i sofakroken og Hilde ser med glede at det er noen kjente der. Det er Mahrvat som allerede befinner seg i sofaområdet

Figur 7.10: IH - registrering av profil.

Figur 7.11: IH - varsler mangler i registrering.



Figur 7.12: IH - valg i kartrepresentasjon.

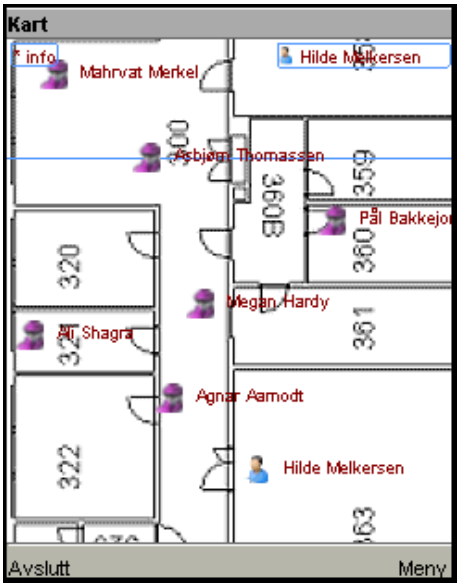


Figur 7.13: Søkefunksjon

og Megan ser ut til å være på vei dit (se Figur 7.14). Hilde tar med seg matpakken sin og rusler bortover gangen, og hun glemmer ikke å ta med seg sin Intelligent Helper.

7.2 Evaluering av design og implementasjon

Denne seksjonen evaluerer design og implementasjon av IH. Noen styrker og svakheter, samt mulige løsninger blir presentert.



Figur 7.14: Scenario 2.

7.2.1 Oppnåelse av funksjonelle krav

I analysen ble det spesifisert 15 krav for IH, hvor de funksjonelle kravene utgjorde 12 av dem. De funksjonelle kravene vises i Tabell 7.1 sammen med en status, som viser oppfyllelsen av de enkelte kravene.

Status for funksjonelle krav		
ID	**Beskrivelse**	**Status**
FR1	Det skal være mulig å søke blant brukere som registreres av Cordis RadioEye™	Implementert
FR2	Det skal være mulig å innhente posisjoner til et sett valgte brukere.	Implementert
FR3	Det skal være mulig å vise et sett utvalgte brukere på en kartrepresentasjon.	Implementert
FR4	Det skal være mulig å registrere en brukerprofil.	Implementert
FR5	Det skal være mulig å endre sin egen brukerprofil.	Implementert
FR6	Det skal være mulig å slette sin egen brukerprofil.	Implementert
FR7	Det skal være mulig å registrere en avtale/hendelse.	Ikke implementert

Fortsetter på neste side.

Tabell 7.1 – forts. fra forrige side.		
Status for funksjonelle krav		
FR8	Det skal være mulig å endre en egen avtale/hendelse.	Ikke implementert
FR9	Det skal være mulig å slette en egen avtale/hendelse.	Ikke implementert
FR10	Det skal opprettes en brukermodell ved registrering av brukerprofil.	Ikke implementert
FR11	Brukermodell skal oppdateres ved endringer av profil.	Ikke implementert
FR12	Brukermodellen skal oppdateres med nye egenskaper fra stereotyper.	Ikke implementert

Tabell 7.1: Oversikt over funksjonelle krav.

En viktig moment for systemet var å tilby hjelp til besøkende som ankom Den Intelligente Korridor, ved IDI. Dette ble derfor prioritert høyest, og kravene som omhandlet dette er ble derfor påbegynt først.

Funksjonen som lar brukere og besøkende søke blandt online ansatte(FR1), ble en utfordring å få til på mobiltelefonen, men den ferdige implementerte løsningen ble et godt resultat. Den gir en enkel og oversiktlig søkefunksjon. Listen ble utført slik at brukere i listen kunne bli valgt for en videre kartvisning. Et utvalg brukere ble sendt til serveren og en forespørsel ble sendt til CRE(FR2).

Det var ikke mulig å hente ut data fra CRE ved å benytte vanlig kommunikasjon over HTTP. Dette var på grunn av noen tjenester som ikke var startet på CRE, tok det en del tid før denne ressursen var tilgjengelig for IH. Løsningen som ble valgt her var å benytte HTTPS kommunikasjon, og dette forsinket implementasjonen da dette krevde en del tillegg til IH. Etter hvert fungerte CRE som posisjoneringsverktøy bra, og kommunikasjon mellom IH og CRE fungerte feilfritt. Dette gjorde implementasjonen av IH gjennomførbar.

Mobilen ble utstyrt med et kart over Den Intelligente Korridor(FR3). Mobilen håndterte kartrepresentasjonene meget tilfredsstillende, selv med begrensede ressurser. Skjermen til mobilen er liten og dette gjorde at kartvisningen måtte begrenses med informasjon.

Brukere kan registrere seg på mobilen(FR4), og en profil blir opprettet i systemet. Dette ble gjort både på server og lokalt i mobilen.

Endring(FR5) og sletting(FR6) av profilen til brukeren ble også implementert, og dette måtte synkroniseres på server og mobil.

På grunn av problemer med teknologien som utgjør IH, tok oppsett og konfigurasjon av programvare og CRE en del tid. Mye av teknologien var også ukjent, og sammen førte dette til at implementasjon tok lengre tid enn for-

ventet. Implementasjonen fikk derfor ikke oppfylt kravet som omhandler lagring(FR7)/endring(FR8) og sletting(FR9) av avtaler og hendelser i CRE. Disse funksjonene var ment for å tilføre funksjonalitet til ansatte og registrerte brukere av systemet.

Logikk til brukermodeller ble heller ikke implementert, og da heller ikke kravene FR10, FR11 og FR12. Tiden som var igjen etter implementasjon tillot ikke dette. Dette ville blitt en spennende tilførelsel til systemet, og det er derfor et tap for IH.

7.2.2 Oppnåelse av ikke-funksjonelle krav

De ikke-funksjonelle kravene til la føringer for design og implementasjon av IH. Kravene vises i Tabell 7.2, sammen med status for oppnåelse.

Status for ikke-funksjonelle krav		
ID	**Beskrivelse**	**Status**
IFK1	Det skal benyttes JADE plattform versjon 3.4.1 for agentutvikling.	Oppfyllt
IFK2	Det skal benyttes Cordis RadioEye^{TM} som posisjoneringsteknologi.	Oppfyllt
IFK3	Det skal skal benyttes LEAP add-on for agenter på mobile enheter.	Oppfyllt

Tabell 7.2: Oversikt over ikke-funksjonelle krav.

Systemet ble utviklet på JADE plattformen, med den spesifiserte versjonen(IFK1), og tillegget LEAP ble også satt opp(IFK3). CRE ble etter noen oppstartsproblemer, benyttet som en ressurs for IH(IFK2).

7.2.3 Styrker & svakheter

Ved å implementere systemet som et agentsystem, medfører det en modulbasert oppbygning. Systemets agenter kunne stoppes og startes, uten at hele systemet måtte tas ned. Den løse koblingen mellom agentene, gjorde at en gradvis degrasjon av systemets funksjoner ble mer naturlig. Agentene forventet ikke at samtlige tjenester var tilgjenglig til enhver tid. BrukerAgent ville, som nevnt tidligere, fungere hvis en agent på serveren falt bort. Hvis BrukerManagerAgent falt bort mens kartfunksjonen kjørtes, ville brukeren fortsatt ha informasjon om brukere som allerede var posisjonert. Nye brukere som ble registrert, ville bli vist på kart, men uten brukerinformasjon.

En svakhet med agentsystemet var at under kjøring av IH kunne en agentid(AID) bli opptatt på serveren. Dette var MAC-adresse til brukeren, og den måtte være et unik på serveren. Dette kunne skje hvis applikasjonen ble stoppet brått og agenten ikke fikk meldt ifra til AMS om at den logget seg av systemet.

En svakhet med en å benytte en mobil, er den begrensede størrelsen på skjermen.

Dette medførte at kartet som ble vist på mobilen, sammen med tekst for navn og informasjon, ble noe smått og utydelig. Bilde over Den Intelligente Korridor, som ble vist under kartvisning, viste alle detaljer med etasjen, og dette ble forstyrrende for resten av informasjonen.

Brukeren må selv taste inn MAC-adressen ved pålogging til IH. API støttet ikke uthenting av dette fra mobilen, og det ikke ble funnet noen god måte å løse dette på.

Grunnet noen problemer under konfirgurasjonen ble ikke det lokale koordinatsystemet riktig i forhold til Den Intelligente Korridor. Posisjoner til brukerene og systemet fungerte allikevel slik som det var ment.

7.2.4 Mulige forbedringer

Selv om agentene i IH håndterte all kommunikasjon mellom server og mobil uten problemer, er dette fortsatt testet i liten skala. Tilleggsfunksjoner for administrasjon av agenter på serveren kunne vært lagt til. Ved stor pågang i meldinger til én agent, kunne nye agenter av samme type blitt startet, og belastningen distribuert blant disse. CRE håndterer opptil 1000 posisjoner i sekunder [1], så hvis agentene ble en flaskehals, kunne flere agenter av typen LokaliseringsAgent håndtert pågangen. Agenten er opptatt så lenge en oppførsel kjøres, så den kan kun behandle én forespørsel i gangen. En endring i måten LokaliseringsAgent håndterer forespørsler på, ved å avvente henting av posisjoner og samle opp flere forespørsler, ville kanskje vært en mulig løsning på det samme problemet.

I implementasjonen ble ikke de ansattes kontor lagt ved som et felt i registreringen i profilen, da denne funksjonen ikke var med designet. Dette ville vært et nyttig tillegg, da denne informasjonen kunne gjort det mulig å vise en besøkende kontoret til den ansatte. Besøkende kunne da bli guidet til riktig kontor, selv om den ansatte ikke hadde slått på sin mobil, for posisjonering.

IH kan relativt enkelt utvides til å støtte andre mobile enheter enn MIDP/CLDC mobiler. JADEpL kan konfigurere ferdig utviklede agentklasser til både CDC konfigurasjonen, og også Microsoft .NET. Dette ville gjøre IH tilgjengelig for et større antall brukere.

I stedet for å merke hendelser i IH med *public* eller *private*, kunne vært mulig å spesifisere hvilke "grupper" en slik hendelse skulle innebefatte. Grupper kunne vært for eksempel *lærer*, *student* eller *IT2702*, for et AI fag. Slik kunne informasjon blitt begrenset til enkelte grupper, og ikke bare en grov inndeling som *public/private*.

Kapittel 8

Konklusjon

Dette kapittelet konkluderer oppgaven og arbeidet som er utført.

Seksjon 8.1 ser på måloppnåelsen og hva oppgaven har bidratt med.

Seksjon 8.2 evaluerer arbeidet som er utført og arbeidsprosessen.

Seksjon 8.3 gir forslag til videre arbeid.

8.1 Måloppnåelse

Oppgavens hovedmål var å designe og implementere et system som benyttet seg av kontekst, for å tilby funksjonalitet til besøkende og ansatte.

Målet for oppgaven ble videre oppdelt i delmål:

- **Delmål 1**: Definere krav til systemet som skal designes
- **Delmål 2**: Design av et agentsystem, basert på agentrammeverket JADE.
- **Delmål 3**: Benytte kontekstinformasjon i systemet.
- **Delmål 4**: Benytte brukermodeller i systemet.
- **Delmål 5**: Implementere, demonstrere, og evaluere en prototyp av det modellerte systemet.

De følgende seksjonene ser på hvert av delmålene.

8.1.1 Delmål 1

En innledende analyse av problemet, støttet av relatert forskning, førte til at en kravspesifikasjon ble utarbeidet. Denne kravspesifikasjonen omhandlet funksjonenen for IH, og kravene ble spesifisert for hver av hovedfunksjonalitetene i systemet. Kravene ble presentert som bruksmønsterdiagram, tekstet brukesmønster og i en tabell som oppsummerte samtlige krav.

Delmål 1 anses derfor som oppfyllt.

8.1.2 Delmål 2

Systemet IH ble designet basert på JADE rammeverket, og metoden som ble benyttet baserte seg på samme rammeverk.

Metoden var ny og fortsatt under utvikling. Den manglet deler av fasene, henholdsvis første fase planlegging, og den siste fasen, implementasjon. Dette gjorde at deler av utviklingen måtte støtte seg til tradisjonelle teknikker. Den hjelpen som metoden ville gitt under implementasjon og testing, kunne ha bidratt til at en større del av design hadde blitt realisert.

En overordnet planlegging av systemet ble gjort, før metoden, og stegene i analyse og design ble fulgt. Dette førte til et ferdig design av et agentsystem som baserte seg på rammeverket JADE.

Delmål 2 anses derfor som oppfyllt.

8.1.3 Delmål 3

Design av IH viser hvordan en kartrepresentasjon skal hjelpe besøkende og ansatte ved Den Intelligente Korridor. Dette gjøres ved å benytte lokasjon fra CRE. Dette innebar at en utforskning av kommunikasjonen med CRE måtte gjennomføres, og den nødvendige konfigurasjon måtte gjennomføres. Lokasjonsdata fra CRE er den kontekstinformasjon systemet var ment å benytte seg av.

Delmål 3 anses derfor som oppfyllt.

8.1.4 Delmål 4

Teori om brukermodeller, og et design som benyttet seg av brukermodeller ble presentert. Denne funksjonaliteten var ment for å gjøre IH mer rettet mot ansatte og tilby distribusjon av informasjon, gjennom registrerte hendelser.

Brukermodeller ble ikke fullstendig gjennomført da de ikke ble i prototypen.

Delmål 4 er delvis oppfyllt.

8.1.5 Delmål 5

En prototype av IH ble implementert. Det bestod av fire agenter på en server, som leverte den funksjonalitet som var spesifisert for IH. To av agentene ble ble ikke fullført i prototypen, men systemet viser fortsatt sine hovedfunksjoner.

Én agent ble designet for å kjøre på en mobiltelefon, og den ble implementert ved å benytte JADEpL, et tillegg til JADE.

Demonstrasjonen av IH ble gjennomført, og viste hvordan den kunne assistere besøkende og ansatte.

En evaluering av det implementerte systemet ble gjennomført, hvor styrken og svakheten til IH ble vurdert.

Delmål 5 anses derfor som oppfyllt.

8.2 Evaluering

Utviklingen av *Intelligent Helper* til *Den Intelligente Korridor* ble en krevende og meget lærerik utviklingsprosess.

Dette arbeidet har vist hvordan et system kan benytte seg av kontekstinformasjon, for å assistere besøkende og ansatte ved Den Intelligente Korridor. Det er vist hvordan et slikt system kan realiseres ved hjelp av et agentsystem.

En feilbedømmelse som ble gjort under arbeidet med oppgaven, var å begynne for sent med implementeringen. Mange nye teknologier og rammeverk gjorde implementasjonen vanskelig. Hvis den kunnskapen, som er tillært under implementering, hadde vært tilgjengelig tidligere, ville en større del av systemet blitt realisert. Dette gjorde at ikke alle deler av oppgaven ble testet.

Arbeidet har vist hvordan en agentarkitektur kan gjennomføre de forskjellige oppgavene i systemet. Demonstrasjonen viser hvordan systemet gir assistanse til besøkende og ansatte, og at det kan ha en realistisk nytteverdi.

Gjennomførelsen av systemet var også ment for å teste metoden for utvikling av agentsystem. Utviklingen av IH viser hvordan metoden fører frem til et design, og sørger for at resultatet blir et fungerende system.

8.3 Videre arbeid

Bruk av kontekst har et stort utnyttelsespotensiale i applikasjonsutvikling, og utforskning av dette domenet er både viktig og interessant.

Kontekstsensitive applikasjoner har store muligheter for å gi økt funksjonalitet for brukere. Mye informasjon kan trekkes ut av konteksten til en bruker, hvor bare tid og lokasjon alene gir mange muligheter. Dette arbeidet har vært med på å belyse hvordan lokasjon kan benyttes i et miljø, slik som Den Intelligente Korridor.

Videre arbeid kan se på flere måter å benytte seg av kontekst. Et fokus kan være å se på hvordan kontekst kan benyttes for å utføre proaktive handlinger på vegne av en bruker. Små oppgaver som å slå på en brukers PC, slå på lyset på kontoret,

eller det motsatte, alt ettersom om brukeren kommer eller går, kan være mulige tillegg til en applikasjon som IH.

En av de mest interessante områdene vil være å se hvordan brukermodeller faktisk fungerer i en slik sammenheng. Hvilke oppgaver kan utføres på vegne av en bruker, og hvordan vil en dette bli mottatt av brukeren.

Hvordan mobilapplikasjonen i IH kan distribueres ble ikke tatt i betraktning i oppgaven. For at et system som IH skal være brukervennlig, må distribusjon av mobilklienten foregå på en enkel måte. Videre arbeid kan se på hvordan mobilapplikasjonen kan bli lastet ned til brukeren, slik at den assistansen IH kan tilføre blir komplett.

Bibliografi

[1] R. C. AS, *Cordis RadioEye^{TM} Localisation Technology* (2003). www.radionor.no/Downloads/radionor_white_paper.pdf.

[2] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa, *JADE Programmer's Guide* (2006). jade.tilab.com/doc/programmersguide.pdf.

[3] F. Bergenti and A. Poggi, *LEAP: A FIPA Platform for Handheld and Mobile Devices* (2001). citeseer.ist.psu.edu/bergenfi01leap.html.

[4] I. Bluetooth SIG, *Bluetooth.com — The Official Bluetooth® Wireless Info Site* (2007). www.bluetooth.com/bluetooth/.

[5] J. Board, *JADE Security Guide* (2005). jade.cselt.it/doc/tutorials/JADE_Security.pdf.

[6] G. Caire and F. Pieri, *LEAP USER GUIDE* (2006). jade.tilab.com/doc/LEAPUserGuide.pdf.

[7] G. Caire, G. Rimassa, and F. Bellifemine, 'JADE: a versatile run-time for distributed applications on mobile terminals and networks', *2004 IEEE International Conference on Systems, Man and Cybernetics* **2** (2004), 1882–1888. ieeexplore.ieee.org/iel5/9622/30422/01399943.pdf?arnumber=1399943.

[8] G. Chen and D. Kotz, *Solar: An Open Platform for Context-Aware Mobile Applications* (2002). citeseer.ist.psu.edu/article/chen02solar.html.

[9] A. K. Dey and G. D. Abowd, *Towards a Better Understanding of Context and Context-Awareness* (1999). ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf.

[10] F. for Intelligent Physical Agents, *Agent Communication Language Specifications* (2002). www.fipa.org/repository/aclspecs.html.

[11] _____, *FIPA Agent Message Transport Service Specification* (2002). www.fipa.org/specs/fipa00067/SC00067F.html.

[12] _____, *FIPA Communicative Act Library Specifications* (2002). www.fipa.org/repository/aclspecs.html.

[13] _____, *FIPA Request Interaction Protocol Specifications* (2002). www.fipa.org/repository/aclspecs.html.

[14] _____, *JADE A White Paper* (2003). jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf.

[15] _____, *FIPA Agent Management Specification* (2004). www.fipa.org/specs/fipa00023/SC00023K.html.

[16] T. R. Gruber, 'A translation approach to portable ontology specifications', *Knowl. Acquis.* **5** (1993), no. 2, 199–220. portal.acm.org/citation.cfm?id=173747.

[17] J. Hendler, *Is There an Intelligent Agent in Your Future?* (Web, 1999). www.nature.com/nature/webmatters/agents/agents.html.

[18] V. Holmstedt, *Objektorientert Systemutvikling og UML* (Fagbokforlaget Vigmostad & Bjørke AS, 2004). ISBN 82-450-0128-7.

[19] E. Inc., *Ekahau Positioning Engine 4.0*. www.ekahau.com.

[20] V. R. Lesser, 'Multiagent systems: an emerging subdiscipline of AI', *ACM Comput. Surv.* **27** (1995), no. 23, 340–342. doi.acm.org/10.1145/212094.212121.

[21] P. Maes, 'Artificial Life Meets Entertainment: Lifelike Autonomous Agents', *Commun. ACM* **38** (1995), no. 11, 108–114. doi.acm.org/10.1145/219717.219808.

[22] R. Mahy, *Location Types and their relations* (Web, 1999). www3.ietf.org/proceedings/00mar/slides/spatial-location-00mar/sld001.htm.

[23] A. Mathias, 'SmartReminder: A Case Study on Context-Sensitive Applications', *Tech. Report* **TR2001-392** (Dartmouth College, Hanover, NH, 2001). citeseer.ist.psu.edu/mathias01smartreminder.html.

[24] M. E. Müller, *Inducing conceptual user models* (Ph.D. thesis, Institute of Cognitive Science, University of Osnabrück, 2002). citeseer.ist.psu.edu/muller99inducing.html.

[25] J. S. D. Network, *Java* (2007). java.sun.com/javame/technology/index.jsp.

[26] M. Nikraz, G. Caire, and P. A. Bahri, *A Methodology for the Analysis and Design of Multi-Agent Systems using JADE* (2006). jade.tilab.com/doc/JADE_methodology_website_version.pdf.

[27] U. of Helsinki, *Context Recognition by User Situation Data Analysis (Context)* (2005). www.cs.helsinki.fi/group/context/.

[28] M. Papasimeon and C. Heinze, *Specifying Agent Behaviour with Use Cases* (2000). www.agents.org.au/20000225Heinze-AgentUseCases_4.pdf.

[29] E. Rich, 'Users are Individuals: Individualizing User Models.', *International Journal of Man-Machine Studies* **18** (1983), no. 3, 199–214. dblp.uni-trier.de.

[30] _____, 'User modeling via stereotypes', *Cognitive Science 3* (1998), 329–342. portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=286035.

[31] Åsa Hammar, *Hukommelse PS 101 grunnfag psykologi* (Web). www.uib.no/psyfa/studentinformasjon/Grunnfag/Grunnfag/Forelesningsnotater/hammar/hosten

[32] D. Salber, A. K. Dey, and G. D. Abowd, 'The Context Toolkit: Aiding the Development of Context-Enabled Applications', in *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems* (ACM Press, New York, NY, USA, 1999), 434–441. doi.acm.org/10.1145/302979.303126.

[33] B. Strømsholm, *Fjernanalyse og geografiske informasjonssystemer* (Web, 2001). www.romteknologi.no/books/33/13.html.

[34] I. Sun Microsystems, *Java Security Overview*. java.sun.com/javase/6/docs/technotes/guides/security/overview/jsoverview.html.

[35] K. P. Sycara, 'Multiagent systems', *AI Magazine* **10** (1998), no. 2, 79–92. www.aaai.org/AITopics/assets/PDF/AIMag19-02-2-article.pdf.

[36] T. I. S. A. L. T. R. I. C. M. University, *Multi-Agent Systems* (Web). www.cs.cmu.edu/~softagents/multi.html.

[37] C. Van Aart, *OntologyBeanGenerator* (Web). protege.cim3.net/cgi-bin/wiki.pl?OntologyBeanGenerator.

[38] B. Veseth, *Nordmenn mer avanserte mobilbrukere* (2007). www.tns-gallup.no/index.asp?did=207173&ARGUMENT=&title=Nordmenn%20mer%20avanserte%20mob

[39] A. I. Wang, C.-F. Sørensen, S. Brede, H. Servold, and S. Gimre, *Development of Location-Aware Applications: The Nidaros framework* (2005). www.idi.ntnu.no/grupper/su/publ/alfw/mobis2005-nidaros-framework.pdf.

[40] M. Weiser, 'The Computer for the 21st Century', *Scientific American* **9** (1991), no. 265, 94–104. www.ubiq.com/hypertext/weiser/SciAmDraft3.html.

[41] _____, *Ubiquitous Computing* (Technical Conference, January 1995). www.ubiq.com/hypertext/weiser/Usenix95Slides.ps.

[42] E. Wilson, *Consilience : The Unity of Knowledge (Vintage)* (Vintage, March 1999). www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20\&path=ASIN/067976867X.

[43] M. Wooldridge, *An Introduction to MultiAgent Systems* (John Wiley & Sons, Chichester, England, 2001). ISBN:0 471 49691X.

[44] M. Wooldridge and N. R. Jennings, 'Intelligent Agents: Theory and Practice', *Knowledge Engineering Review* **10** (1995), no. 2, 115–152. citeseer.ist.psu.edu/article/wooldridge95intelligent.html.

Ansvarstabell

Ansvarstabell

Agent Type	Raffinert ansvarsbeskrivelse	
BrukerAgent	1	La brukeren søke blant online brukere.
	2	Innhente posisjonsdata fra valgte online brukere.
	3	Vise valgte brukere på kart.
	4	La brukeren registrere sin brukerprofil.
	5	La brukeren endre sin brukerprofil.
	6	La brukeren registrere en avtale/hendelse.
	7	La brukeren endre/slette en avtale/hendelse.
	8	Vise brukeren registrerte avtaler/hendelser.
	9	Søke etter tjenestetilbydere hos Directory Facilitator.
LokaliseringsAgent	1	Håndtere forespørsler om å innhente posisjonsdata fra Cordis RadioEye ™.
	2	Sende posisjonsdata av valgte brukere til BrukerAgent, KalenderAgent på forespørsel.
	3	Sende posisjonsdata av samtlige posisjoner registrert av CRE for til BrukerAgent, KalenderAgent.
	4	Registrere tjenester og informasjon hos Directory Facilitator.
BrukerManagerAgent	1	Håndtere forespørsler fra BrukerAgent og KalenderAgent.
	2	Registrere en brukerprofil for BrukerAgent.
	3	Endre en brukerprofil for BrukerAgent.
	4	Opprette en brukermodell ved registrering av ny profil.
	5	Oppdatere brukermodell ved endring av brukerprofil.
	6	Oppdatere brukermodell ved endring av avtaler/hendelser fra KalenderAgent.

Fortsetter på neste side.

Tabell 1 – forts. fra forrige side.		
Agent Type	Raffinert ansvarsbeskrivelse	
	7	Registrere tjenester og informasjon hos Directory Facilitator.
KalenderAgent	1	Håndtere forespørsler fra BrukerAgent.
	2	Registrere en avtale/hendelse for BrukerAgent.
	3	Endre en avtale/hendelse for BrukerAgent.
	4	Sende oversikt over egne avtaler til BrukerAgent.
	5	Sende informasjon om valgte avtaler til BrukerAgent.
	6	Håndtere avtaler som inntreffer.
	7	Innhente avtale-/hendelse-avhengig posisjonsdata fra LokasjonsAgent.
	8	Varsle BrukerAgent om avtaler som inntreffer.
	9	Registrere tjenester og informasjon hos DF.
	10	Søke etter tjenestetilbydere hos DF.

Tabell 1: Ansvarstabell

Interaksjons-spesifikasjon

Interaksjonstabell for BrukerAgent

Interakasjons-spesifikasjon	BrukerAgent					
Interaksjon	Ansvar	IP	Rolle	Med	Når	Templates
Søke Online-brukere	1	FIPA-REQUEST	I	LokaliseringsAgent	Bruker starter søk blandt Online-brukere	Conv-Id
Hent posisjoner	2	FIPA-REQUEST	I	LokaliseringsAgent	BrukerAgent ber om å få posisjoner til valgte brukere	Conv-Id
Registrere bruker-profil	4	FIPA-REQUEST	I	BrukerManagerAgent	Bruker vil registrere en bru-kerprofil	Conv-Id
Oppdatere bruker-profil	5	FIPA-REQUEST	I	BrukerManagerAgent	Bruker oppdaterer sin bru-kerprofil	Conv-Id
Registrering av av-tale/hendelse	6	FIPA-REQUEST	I	KalenderAgent	Bruker vil registrere avta-le/hendelse	Conv-Id
Endring av avta-le/hendelse	7	FIPA-REQUEST	I	KalenderAgent	Bruker vil endre avta-le/hendelse	Conv-Id
Vise avtaler/hen-delser	8	FIPA-REQUEST	I	KalenderAgent	Bruker ber om å få se sine avtaler/hendelser	Conv-Id
Søke tjenestetilby-dere	9	FIPA-REQUEST	I	Directory Facilitator	BrukerAgent søker opp tje-nestetilbydere	Conv-Id

Tabell 2: Interaksjonstabell for BrukerAgent

Interaksjonstabell for LokaliseringsAgent

Interakasjons-spesifikasjon	LokaliseringsAgent					
Interaksjon	Ansvar	IP	Rolle	Med	Når	Template
Send posisjonsdata til valgte brukere	2	FIPA-REQUEST	R	BrukerAgent	BrukerAgent ber om posisjoner til et sett brukere.	Perf=REQUEST
Alle online-brukere	3	FIPA-REQUEST	R	BrukerAgent	BrukerAgent ber om alle online-brukere.	Perf=REQUEST
Registrere tjenester	4	FIPA-REQUEST	I	Directory Facilitator	Registrerer sine tjenester ved oppstart.	Conv-Id

Tabell 3: Interaksjonstabell for LokaliseringsAgent

Interaksjonstabell for BrukerManagerAgent

Interakasjons-spesifikasjon	BrukerManagerAgent					
Interaksjon	Ansvar	IP	Rolle	Med	Når	Template
Profilregistrering	2	FIPA-REQUEST	R	BrukerAgent	Bruker har registrert en brukerprofil og brukeragent ber om lagring.	Perf=REQUEST
Profilendring	3	FIPA-REQUEST	R	BrukerAgent	Bruker vil endre sin brukerprofil.	Perf=REQUEST
Registrere brukerprofil	6	FIPA-REQUEST	I	BrukerManagerAgent	Bruker har endret en avtale/hendelse.	Perf=REQUEST
Registrere tjenester	7	FIPA-REQUEST	I	Directory Facilitator	Registrerer sine tjenester ved oppstart.	Conv-Id

Tabell 4: Interaksjonstabell for BrukerManagerAgent

Interaksjonstabell for KalenderAgent

Interakasjons-spesifikasjon	KalenderAgent					
Interaksjon	Ansvar	IP	Rolle	Med	Når	Template
Registrere avtale/hendelse	2	FIPA-REQUEST	R	BrukerAgent	Bruker vil registrere en avtale/hendelse.	Perf=REQUEST
Endre avtale/hendelse	3	FIPA-REQUEST	R	BrukerAgent	Bruker vil endre avtale/hendelse.	Perf=REQUEST
Vise alle avtaler/hendelser	4	FIPA-REQUEST	R	BrukerAgent	Bruker vil se alle avtaler/hendelser.	Perf=REQUEST
Vise utvalgte avtaler/hendelser	5	FIPA-REQUEST	R	BrukerAgent	Bruker vil se et utvalg avtaler/hendelser.	Perf=REQUEST
Innhente posisjonsdata	7	FIPA-REQUEST	I	LokaliseringsAgent	KalenderAgent trenger posisjonsdata for ev. fyring av hendelser.	Perf=REQUEST
Varsle om avtale/hendelse	8	FIPA-INFORM	I	BrukerAgent	Avtale/hendelse fyres og varsler BrukerAgent	Perf=REQUEST
Registrere tjenester	9	FIPA-REQUEST	I	Directory Facilitator	Registrerer sine tjenester ved oppstart.	Conv-Id

Tabell 5: Interaksjonstabell for KalenderAgent

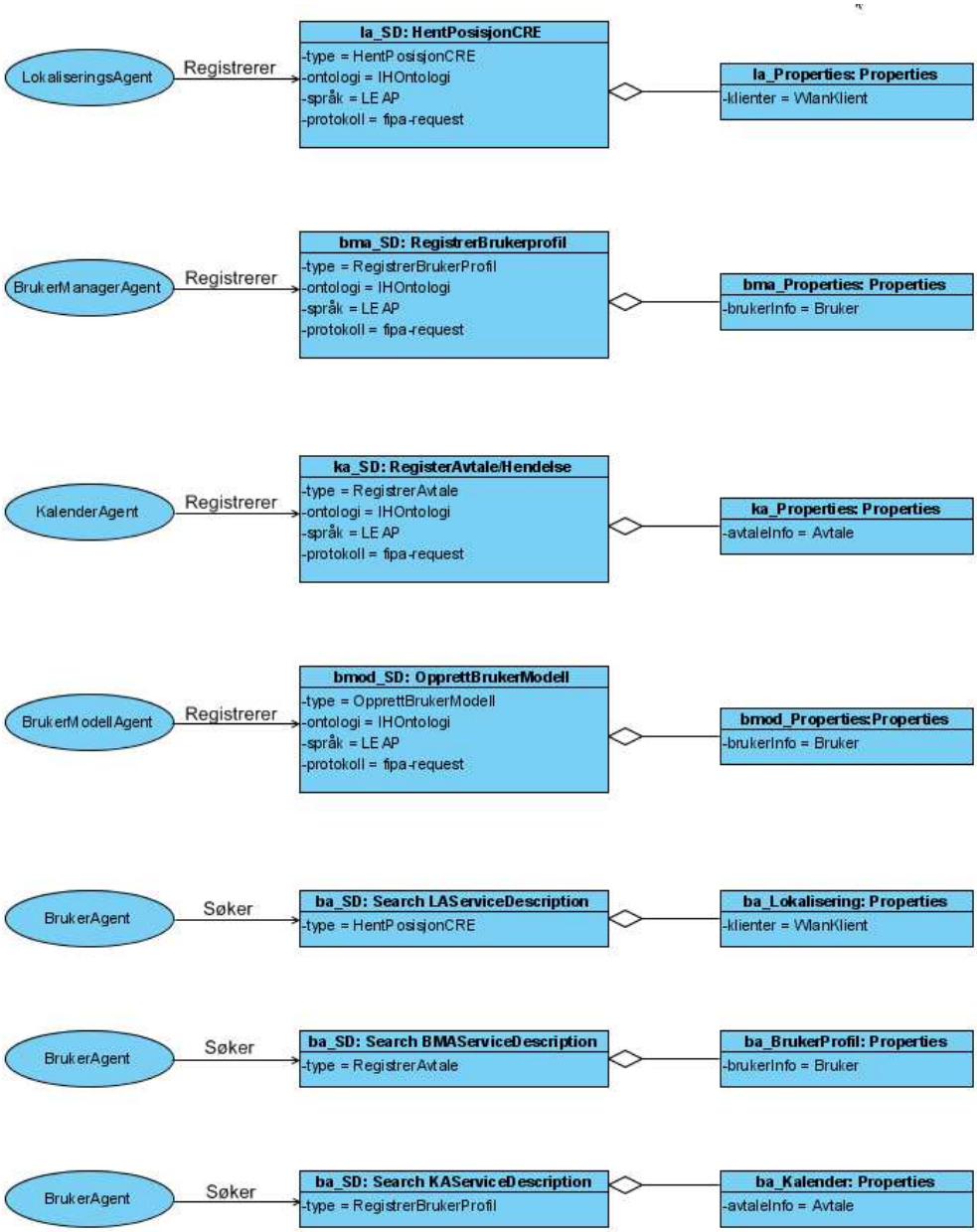
Interaksjonstabell for BrukerModellAgent

Interakasjons-spesifikasjon	BrukerModellAgent					
Interaksjon	Ansvar	IP	Rolle	Med	Når	Template
Opprette brukermodell	2	FIPA-REQUEST	R	BrukerManagerAgent	BrukerManagerAgent gir beskjed og info om en ny bruker.	Perf=REQUEST
Endre brukermodell	3	FIPA-REQUEST	R	BrukerManagerAgent	BrukerManagerAgent gir beskjed om en oppdatert brukerprofil.	Perf=REQUEST
Slette brukermodell	4	FIPA-REQUEST	R	BrukerManagerAgent	BrukerManagerAgent ber om å slette brukermodell til en bruker.	Perf=REQUEST
Registrere tjenester	9	FIPA-REQUEST	I	Directory Facilitator	Registrerer sine tjenester ved oppstart.	Conv-Id

Tabell 6: Interaksjonstabell for BrukerModellAgent

Tjenestebeskrivelse

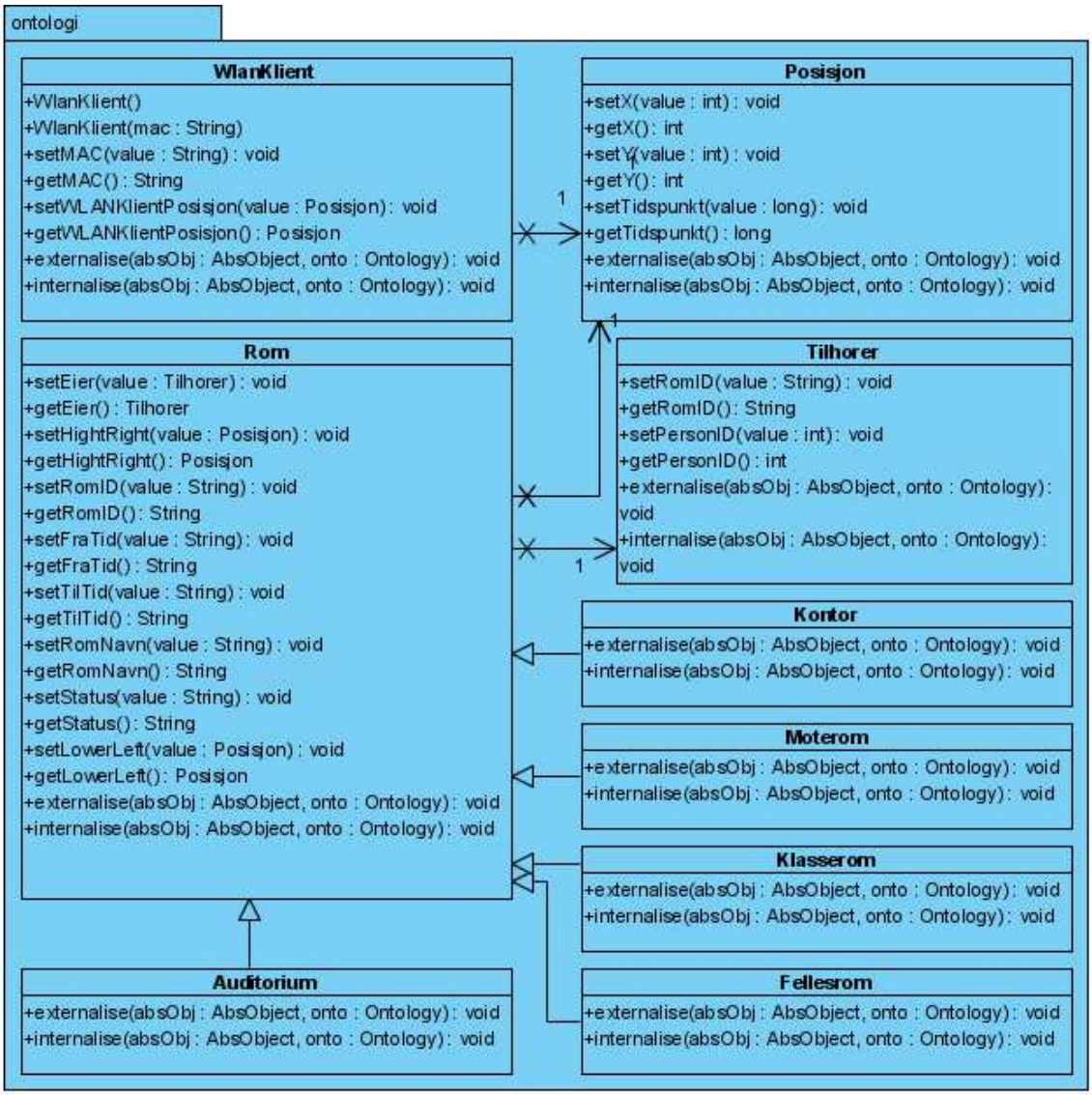
Tjenestebeskrivelse



Figur 1: Registrering av, og søk etter, tjenester hos DF.

Ontologi

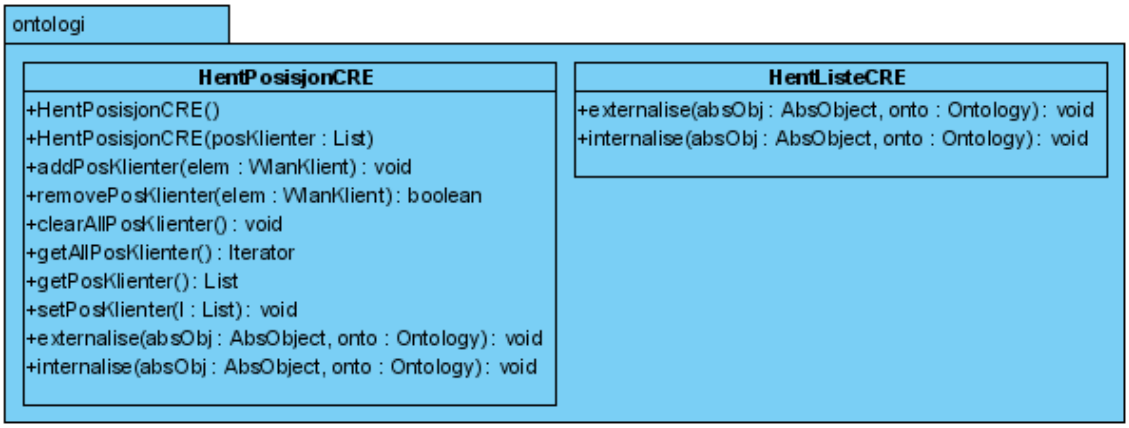
Konsepter



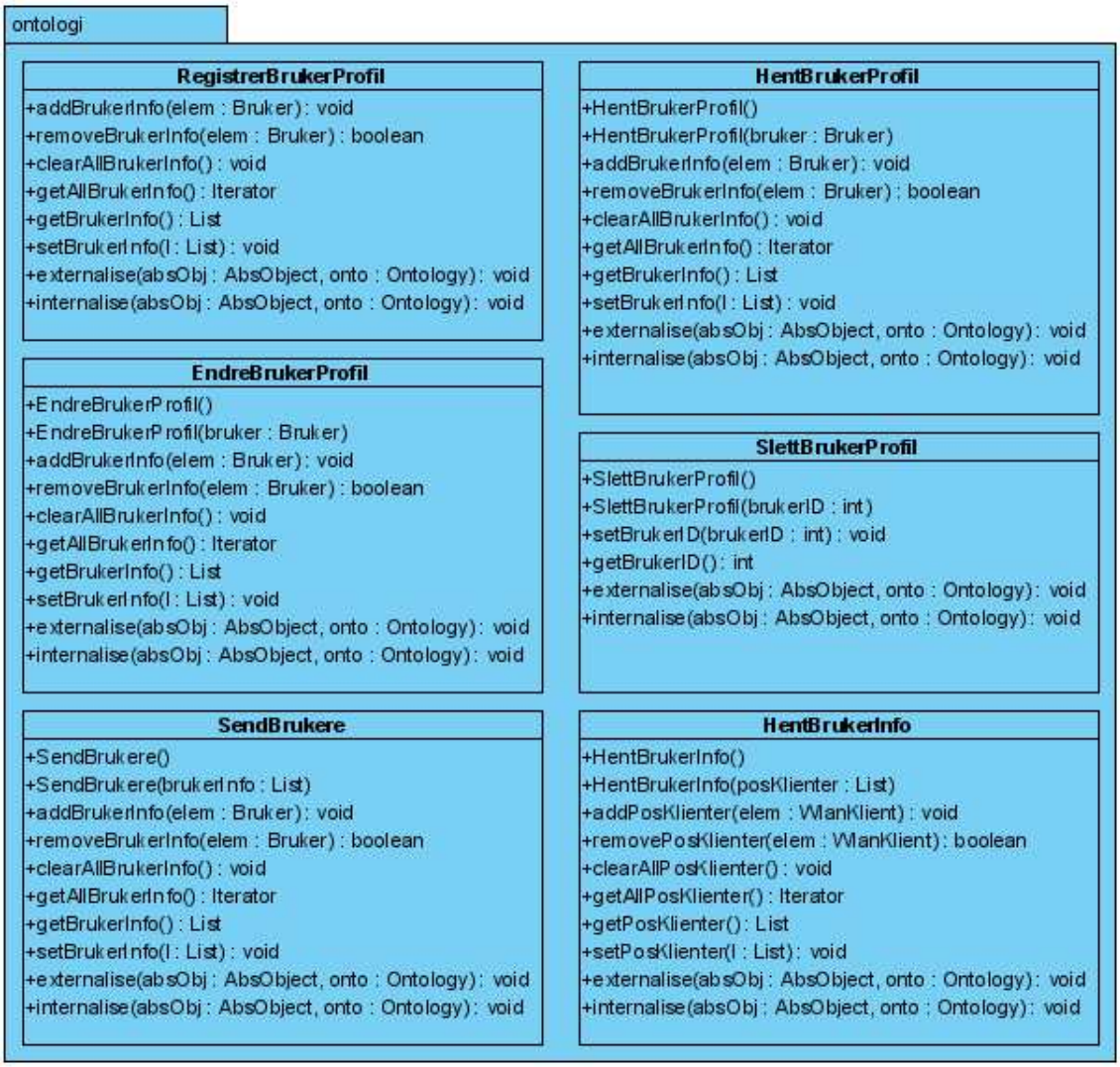
Figur 2: Konsepter i IH.

Figur 3: Konsepter i IH.

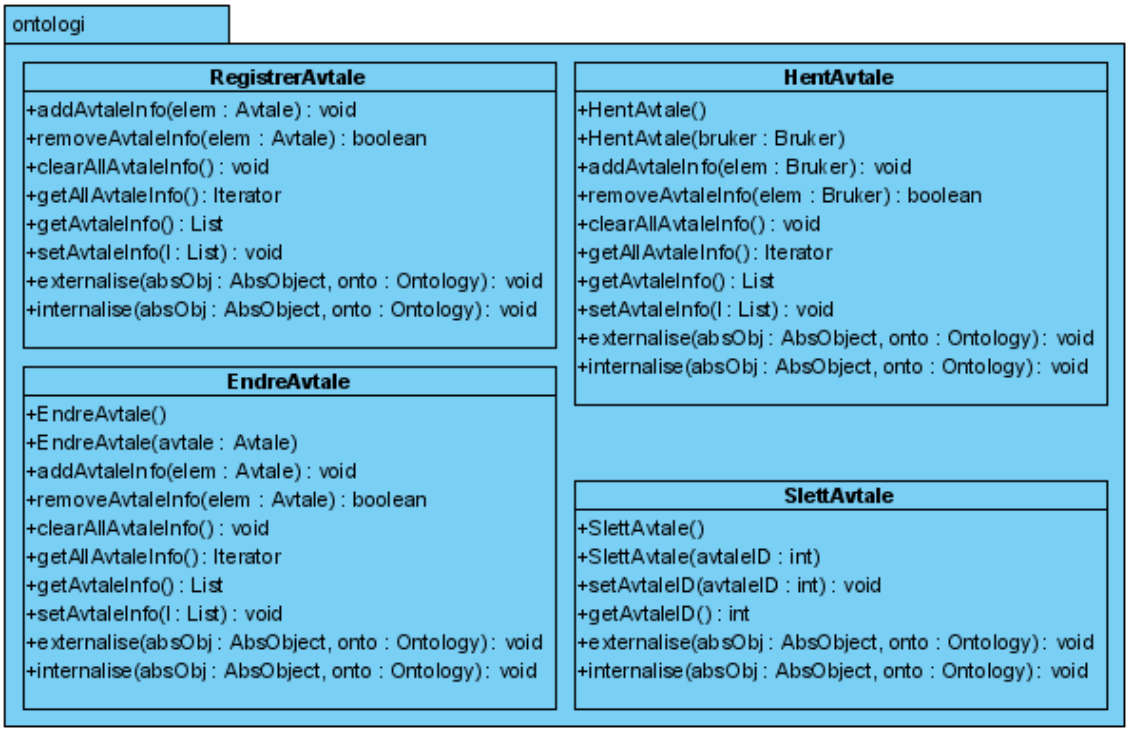
Agenthandlinger



Figur 4: Agenthandlinger i ontologien for posisjonering i IH.



Figur 5: Agenthandlinger i ontologien for brukerhåndtering i IH



Figur 6: Agenthandlinger i ontologien for avtalehåndtering i IH.