Amanda: En arkitektur for adaptiv og autonom søking styrt av brukeradferd

Bjørn Remseth
Universitetet i Oslo
Institutt for Informatikk
Epost: rmz@rmz.priv.no

30. oktober 2003

Innhold

Kapittel 1

Innledning

1.1 Problemstilling

I denne oppgaven behandles problemstillingen:

> *En svært vanlig problemstilling for brukere av datasystemer er å søke etter relevant informasjon i forhold til en kontekst. Ofte er konteksten gitt i forbindelse med arbeidsoppgaver brukerne er i ferd med å løse. Det er ønskelig å ha datasystemer som støtter en slik søkeprosess - uten at brukerne selv må styre teknologien for søkingen, og slik at søkingen legger beslag på et minimum av brukerens oppmerksomhet. Problemstillingen for denne oppgaven er å beskrive en hensiktsmessig arkitektur som muliggjør slike systemer.*

1.2 Metode

Teknologisk tilnærming

Det beskrives en arkitektur, kalt "Amanda", som besvarer problemstillingen. Amanda er en arkitektur jeg har utviklet i forbindelse med denne oppgaven. Amanda brukes til automatisk søking i store informasjonsmengder der søkekriteriene hovedsaklig produseres ut fra observasjoner av brukerens atferd. Som en del av arbeidet med arkitekturen er det implementert et system (også kalt "Amanda") som realiserer deler av arkitekturen.

Amandas mål nås ved å la "søkeagenter" søke på vegne av brukeren; agentene observerer deler av brukerens atferd og et utvalg av dokumenter brukeren arbeider med. Agentene bruker så denne inputten til lage modeller for hva som er gode søk for brukeren i den konteksten hun er i. Agentene får tilbakemeldinger fra brukeren; tilbakemeldingene kan være eksplisitte ved at brukeren velger ut et bestemt forslag fra en agent og rangerer meldingen eller agenten.

Tilbakemeldingene kan også være implisitte ved at agentene måler sin effektivitet gjennom å observere brukerens atferd; dersom en agents forslag leses av brukeren, antas dette å være godt og dersom agentene i brukerens arbeider gjenfinner fragmenter av innhold fra dokumentene som er foreslått, antas det at brukeren har dratt nytte av forslaget selv om ingen eksplisitt tilbakemelding er gitt. Problemer med både falske negative og positive tilbakemeldinger gjør at dette er en fremgangsmåte som må bruke med varsomhet, noe som diskuteres i oppgaven (side 30, om vektorrommodell).

Amanda-arkitekturen er laget for å kunne favne om et meget vidt spektrum av arbeidsoppgaver og informasjonskilder. I løpet av arbeidet med oppgaven er også elementer av arkitekturen implementert. I den forbindelse ble det gjort avgrensninger både med hensyn på informasjonskilder, som er begrenset til å være eksternt tilgjengelige søkemaskiner for søking i lokalt fillager og world wide web, og arbeidsoppgaver som er begrenset til å være enkel tekstbehandling[1]. Med eksternt tilgjengelige søkemaskiner menes her at det innenfor Amanda ikke finnes egne søkemaskiner som søker gjennom dokumentmengder, vi vil for all søking bruke eksterne søkeverktøy for å søke gjennom dokumentmengdene vi er interesserte i.

Arkitekturen er basert på "agenter" som i denne sammenhengen refererer til relativt små programmer som er i stand til å løse avgrensede oppgaver, som å sende et søk til en søkemaskin, presentere et resultat for brukeren, klassifisere et dokument o.s.v. Ingen enkelt agent produserer atferden til hele systemet, det er alltid et samvirke av agenter som produserer eksternt observerbar atferd. Agentene i Amanda utviser både adaptiv og autonom atferd. *Autonome agenter* arbeider i et dynamisk, uforutsigbart miljø hvor de forsøker å tilfredsstille en mengde tidsavhengige mål eller motivasjoner. Agenter er *adaptive* dersom de øker sin kompetanse til å håndtere disse målene ut fra erfaring de gjør seg.

For å formulere og evaluere søk brukes teknikker hentet fra fagområdet "Information Retrieval" (IR). Innen IR er man opptatt av hvordan man kan produsere gode søk med utgangspunkt i en brukers informasjonsbehov. Det er et rikt tilfang og metoder og teknikker å hente i IR, dessverre er det lite med teori som gir gode prediksjoner om hvilke metoder og kalibreringsparametere for disse som i en gitt situasjon best dekker brukerens informasjonsbehov. Mye "fininnstilling" er derfor nødvendig for å oppnå gode søk. I Amanda gjøres denne fininnstillingen av agentene, som ut fra tilbakemeldinger vil søke mot kombinasjoner av basismetoder fra IR og parametere til disse som gir best opplevd nytte for brukeren.

Vår metode kan dermed oppsummeres til å være:

> *Å besvare hovedmålet ved å kombinere teknikker fra autonome adaptive agenter og information retrieval for å produsere kontekstilpassede søk mot informasjonskilder som er aksessert gjennom eksterne søkemotorer.*

[1] Et alternativ som ble vurder var å bruke epost som datagrunnlag. Dette er fortsatt et svært interessant alternativ, men det ble ikke videre forfulgt i denne oppgaven. Bruk av datagrunnlag i Epost nevnes i seksjon 6.4.1 som et mulig videre arbeid.

Problematisering

Vi problematiserer oppgavens mål og teknologiske tilnærming på to vis, gjennom å se på tidligere arbeider, og ved å se på tidligere utarbeidet teori.

Vi har valgt et knippe tidligere arbeider som viser arbeider rundt brukertilpassede søkemaskiner, kontekstilpassede brukergrensesnitt, og "semantic web". Problematiseringen gjøres først ved å presentere fenomener fra litteraturen, så beskrive hvilke elementer fra disse som er tatt inn Amanda. I tillegg indikeres det hvordan disse tidligere arbeidene helt eller delvis kunne vært implementert gjennom vår arkitektur. Gjennom denne problematiseringen ønsker vi både å vise deler av opphavet til vår foreslåtte arkitektur, og dessuten å demonstrere at denne arkitekturen er robust nok til å kunne anvendes på et bredt spektrum av problemer.

Vårt utvalg av teori er hentet fra flere områder: Fra *Information Retrieval* (IR) henter vi teknikker for klassifikasjon og søking, og bruker denne teorien for å begrunne våre valg av teknikker for representasjon av agenter for søk. Vår arkitektur for sammenkobling av agentene både seg imellom og mot omverdenen henter mange elementer fra den eksisterende litteraturen om *autonome adaptive agenter*. Til slutt er det også slik at når et system basert på Amanda er operativt vil det ha en atferd, og det er betimelig å tenke på hvordan man skal evaluere denne atferden. Det finnes noe teori om hvordan man vurderer søkemaskiner, både hentet gjennom Information Retrieval fagfeltet, og hentet fra mer nytteorienterte vurderinger. Vi henter fra begge disse og eksponerer det i teorikapitlet og i kapitlet om tidligere arbeid

Bidrag

Denne oppgavens bidrag er at det beskrives en arkitektur for å autonomt tilpasse søking til brukerens behov, og gjennom dette denne arkitekturen besvares hovedproblemstillingen.

Det gis en konkret implementasjon for deler av arkitekturen. Implementasjonen er brukt for å kontrollere antakelser gjort under designet, og vil kunne fortsette å brukes til dette ettersom designet videreutvikles. Videre skisseres det hvordan arkitekturen kan skaleres opp langs forskjellige dimensjoner: Søkevolum, kompleksitet av søkeuttrykk, datakilder å søke gjennom, og forskjellige slags brukergrensesnitt.

På bakgrunn av funn gjott i litteraturstudiene er det formulert et eksperiment for å måle systemets effektivitet. På grunn av omfanget av arbeidet er eksperimentet ikke gjennomført, men det vil kunne være en naturlig del av en videreføring av arbeidet. Siden eksperimentet representerer et viktig "use case" som lå til grunn for designet både av arkitektur og implementasjon er en beskrivelse av det likevel tatt med i et appendiks.

Takk til

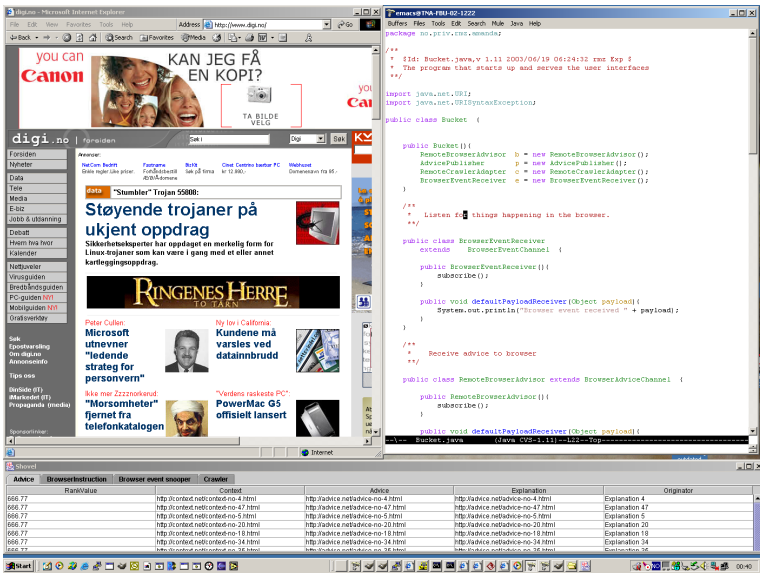
Underveis i arbeidet med oppgaven er det mange som har gitt tilbakemeldinger på større eller mindre deler av den, har vært diskusjonspartnere, har gitt oppmuntring, inspirasjon inspirasjon, eller på annet vis vært nødvendig for at oppgaven i det hele tatt ble ferdig. Tusen takk til alle sammen.

At oppgaven ble som den ble tar jeg naturligvis det fulle og hele ansvar for, selv om det naturligvis er slik at mye av innholdet har blitt til under diskusjon med andre. Under er det listet opp i alle fall noen av dem jeg føler har bidratt sterkt til at oppgaven ble som den ble. Listen under[2] er sikkert alt for kort, så de som føler seg utelatt kan i stillhet føre seg opp, jeg vil ikke protestere. Folkens, jeg er glad i dere: Amund Tveit, Anne Abelsæth, Bjørn Alsberg, Bjørn Borud (den snille), Bjørn Kirkerud, Cynthia Sheng Smith, Elisabeth Bayegan, Erik Bergh, Erik Naggum, Hilde Madsen, Kristen Nygård, Nils Jacob Berland, Ole Bjørn Hessen, Ruben Olsen, Stian Arnesen. Spesiell takk til min veileder Knut Omang som hjalp meg akkurat så mye jeg trengte for å komme meg videre. Den største takk og takknemlighet til min familie som har holdt ut med meg underveis. Mest av alle takk og kjærlighet til min kone May Lill som jeg elsker så høyt.

[2] Sortert alfabetisk etter fornavn.

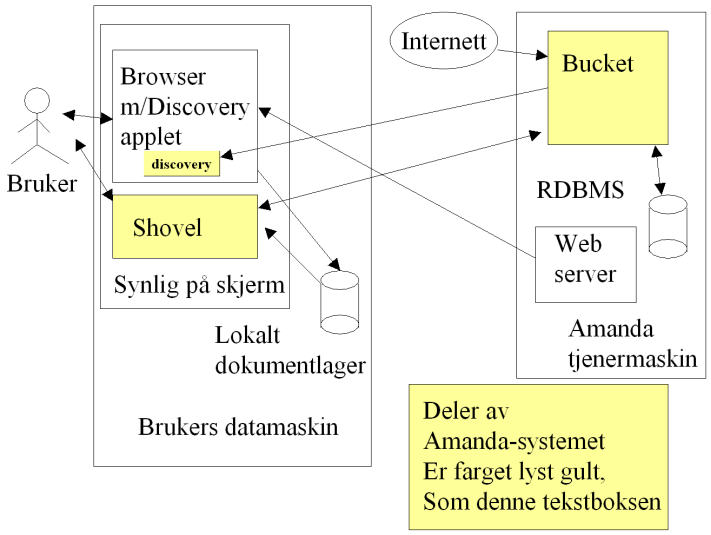
Kapittel 2

Overblikk over implementasjon og arkitektur



Figur 2.1: *Amanda implementasjonens brukergrensesnitt: Øverst til venstre en browser som kjører Amandas "Discovery" applet (appleten er skjult). Nederst er Shovel applikasjonen som gir råd og tar i mot tilbakemeldinger om disse. Oppe til høyre et tekstredigeringsprogram der brukeren arbeider med sin primære arbeidsoppgave.*

Dette kapittelet beskriver Amanda-arkitekturen og den implementerte delen av denne. Det gis overordnede linjer både i arkitekturen og implementasjonen, men detaljene og avveininger rundt design utsettes til kapittel 5.



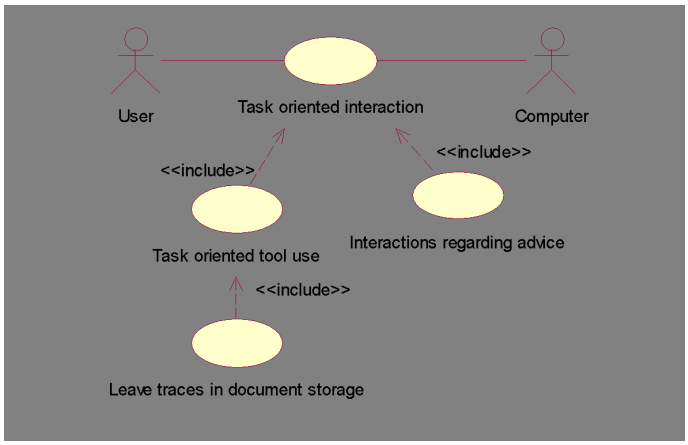
Figur 2.2: *Høynivåstrukturen i det implementerte Amandaesystemet. Pilene viser retning av dataflyt. De brukernære agentene er her plassert i "Shovel" applikasjonen. Bucket fungerer her som en ruter, så det er mulig for Shovel å sende meldinger Discovery selv om de ikke er direkte forbundet med streker.*

2.1 Implementasjon

Samtidig med at designet ble utarbeidet ble deler av det implementert. Figur 2.1 (side 11) viser hvordan implementasjonen ser for en bruker. Implementasjonen er tilpasset en situasjon der brukeren arbeider med redigering av tekstfiler, og får råd om å se på dokumenter som så kan ses på gjennom en browser.

Brukeren opplever bruk av Amanda slik: Hun sitter og taster i vei på dokumentet sitt, og bruker browseren til å se på forskjellige filer som er (mer eller mindre) relevante i forhold til dokumentet hun skriver på. Hele tiden vil Amanda gjennom Shovel-applikasjonen (vist nederst på figur 2.1) gi forslag til dokumenter hun kan se på, og hun kan velge disse dokumentene ved å klikke på forslagene i Shovels vindu. Hun kan også gi direkte tilbakemeldinger om forslagene gjennom Shovel. Shovel vil fra tid til annen lese innholdet i dokumentet brukeren arbeider med, og vil kontinuerlig melde både hva dette er, og hvilke dokumenter brukeren har sett på i browseren.

Implementasjonen er i sin helhet basert på J2SE (Java 2 Standard Edition) versjon 1.4. En kjørende instans består av et javaprogram kalt "Shovel" som implementerer et brukergrensesnitt og flere brukernære agenter, en applet kalt "Discovery" som kan instruere brukerens browser om at et dokument skal ses på. Bakenforliggende agenter (agenter som ikke interagerer direkte med bruk-



Figur 2.3: *Brukerinteraksjon fra arkitekturens perspektiv: Noe interaksjon er direkte oppgaveorientert og noe handler om hjelp og assistanse.*

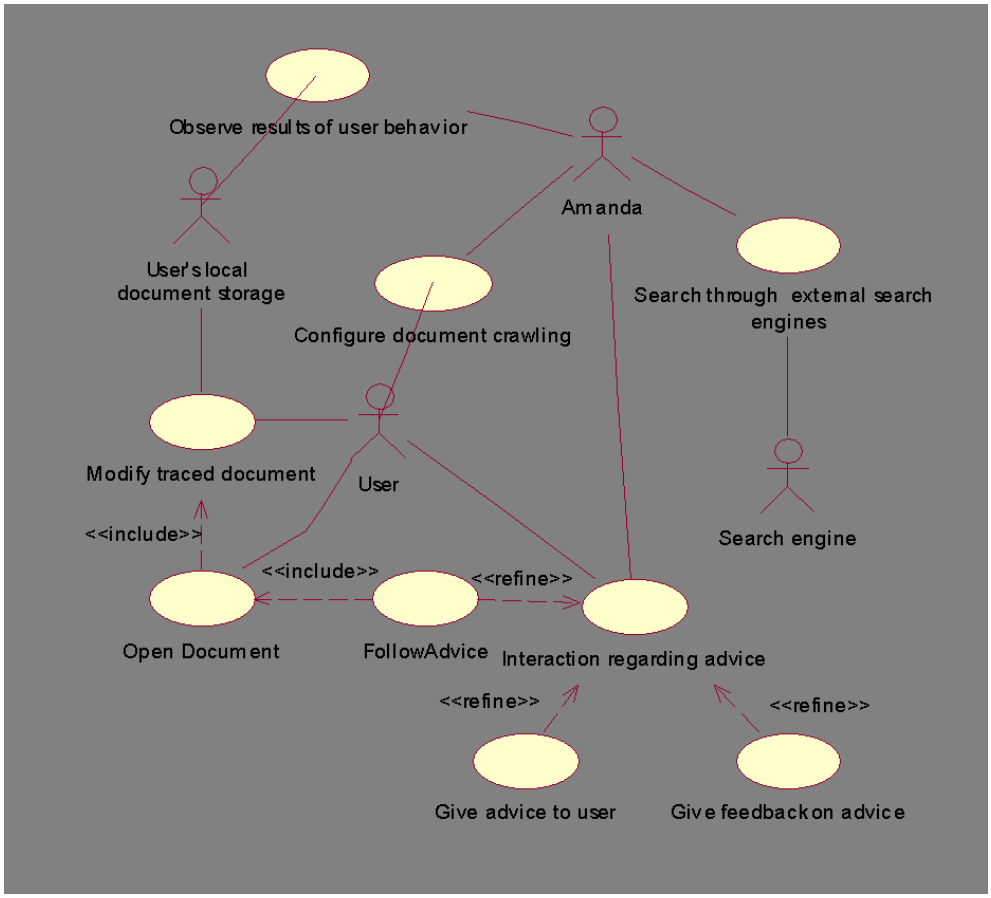
eren eller brukerens filer for å lete etter spor om brukerens atferd) kjører alle i et javaprogram kalt "Bucket" som (delvis) implementerer en agentpopulasjon, og en kontrollstruktur for samvirke mellom agentene i populasjonen. Alle komponentene kommuniserer gjennom et kommunikasjonsrammeverk som er implementert spesielt for Amanda.

I tillegg er det nødvendig med en mekanisme for å levere appleten til browseren. Dersom browseren kjøres på samme maskin som Bucket, kan man hente appleten direkte fra lokalt fillager, ellers vil det være nødvendig med en webtjener på samme maskin som Bucket kjører som leverer fra seg appleten (dette siste er en direkte konsekvens av begrensninger gitt av sikkerhetsmodellen for browseren).

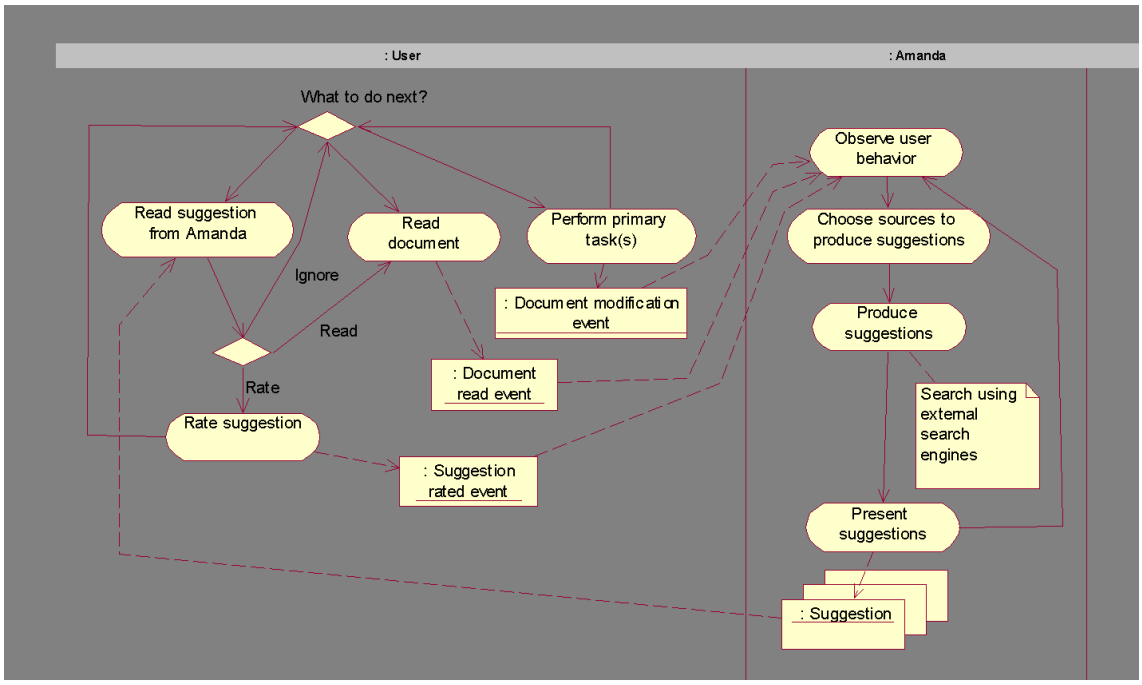
2.2 Arkitekturen

2.2.1 Hva arkitekturen skal brukes til

Amanda assisterer en bruker som arbeider med en eller flere oppgaver. Brukerens interaksjoner med datamaskinen kan som vist i figur 2.3 grovt deles inn i verktøysbruk som er direkte orientert mot oppgavene, og interaksjoner som gir råd med relevans i forhold til oppgavene. Diagammet i figur 2.3 er et UML (Universal Modelling Language) "Use Case" - diagram. Ovalene representerer "use caser", som er oppgaver som løses i felleskap av aktører. Aktører kan både være systemer og mennesker. I dette diagrammet er "user" et menneske, og "computer" et datasystem mennesket har en interaksjon med. Stiplet pil med "include" i hakeparenteser representerer at en beskrivelse inkluderer en annen beskrivelse, så usecaser ordnet med "include" representerer en funksjonell



Figur 2.4: *Noe forfinet brukerinteraksjon. Viser hvilke deler av denne Amanda deltar i. Alle aktører unntatt brukeren er her deler av "computer" fra figur 2.3 fra side 13.*



Figur 2.5: *Dette diagrammet viser aktiviteter i Amanda sett fra brukerens og Amanda-systemets perspektiver.*

dekomponering av en oppgave. Vi vil bruke noe UML i denne oppgaven, og vil kortfattet forklare notasjonselementer første gang de brukes[1].

Amanda arbeider med å gi råd og håndterer overfor brukeren kun interaksjoner i forbindelse med dette.

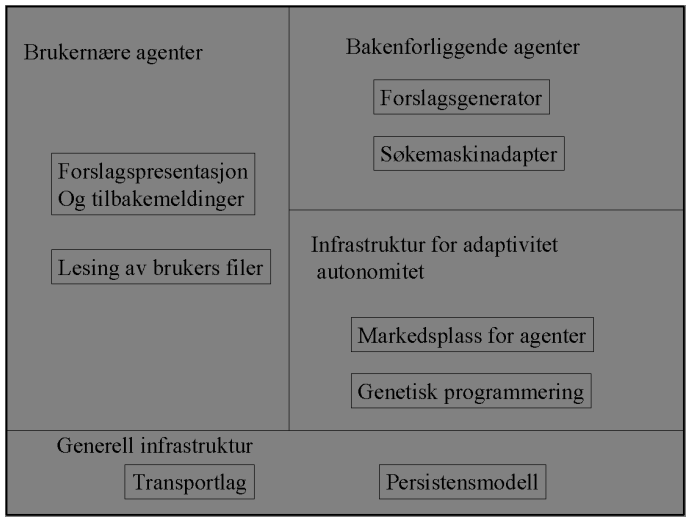
I figur 2.4 vises en noe forfinet versjon av bildet fra figur 2.3. Forfiningene går ut på at interaksjonene angående råd her er vist med Amanda som motpart, ikke en generisk "computer", videre er eksterne søkemaskiner og både eksterne og interne dokumentlagre vist som egne aktører.

Brukerens verktøysbruk vil legge igjen større eller mindre spor i datamaskinens dokumentlager. Amanda observerer disse og bruker dem sammen med interaksjoner angående råd og resultater fra søk via eksterne søkemaskiner for å produsere nye råd til brukeren.

2.2.2 Overordnet arkitekturbeskrivelse

I figur 2.5 kobles brukerens perspektiv på hva som skjer i en interaksjon. Det vises også hvilken informasjon som flyter mellom aktørene. *Brukeren* velger

[1]For mer inngående beskrivelser av UML finnes det etterhvert svært mye litteratur å velge i, personlig vil jeg anbefale Martin Fowlers kompakte bok "UML destilled"[77].



Figur 2.6: *Overordnet inndeling Amandas funksjonelle komponenter.*

hva hun skal gjøre, og avhengig hva hun gjør vil dette produsere spor som plukkes opp av Amanda. Amanda bruker den observerte brukeradferden til å velge agenter som skal produsere forslag. Agentene kan så bruke eksterne søkemaskiner til å produsere forslag som så presenteres så for brukeren. Denne modellen åpner for at brukeren kan arbeide med flere oppgaver samtidig, og at det i parallell, og at det på mange forskjellige vis kan søkes etter forslag. Diagrammet er et UML aktivitetsdiagram. De avrundede firkantene representerer aktiviteter. De heltrukne linjene representerer kontrollflyt, de stiplede linjene representerer dataflyt og firkantene med skarpe hjørner representerer informasjonsobjekter som følger dataflyt. "stabler" av informasjonsobjekter (som "AdviceUpdateEvent") representerer at det kan komme flere informasjonsobjekter av gangen. De vertikale oppdelingene (kalt "svømmebaner") for :User og :Amanda viser en fordeling av aktiviteter mellom aktører; kun aktiviteter utført av Amanda finnes inne i :Amanda-banen. I :User-banen er det samlet aktiviteter utført både av brukeren ("What to do next?" valget og "Perform primary task(s)"), og aktiviteter utført av programmer brukeren interagerer med (f.eks. objektflyten som ut fra noen av oppgavene inn til Amanda). Denne type litt "flytende" beskrivelse er ganske vanlig når man modeller høynivå aktiviteter med UML. Etterhvert som man kommer nærmere implementasjon vil det som regel være et mer finmasket skille mellom de forskjellige systemaktørene. Dette er som senere kan observers direkte i UML modellene som vises i kapittel 5 om implementasjonsdetaljer.

I figur 2.6 vises overordnede funksjonelle komponenter i Amanda. Nesten all kommunikasjon mellom komponentene går gjennom transportlaget, alt som persisteres går gjennompersisteringsmodulen. De øvrige blokkene represen-

terer blokker av funksjonalitet som var naturlig å gruppere sammen under designprosessen.

Vi ser at brukerens forskjellige handlinger gir opphav til forskjelige slags spor, representert som informasjonsobjekter som sendes over til "Observe User Behavior" aktiviteten i Amanda. Basert på inputen fra (og om) brukeren, vil Amanda internt velge ut noen kilder som skal lage forslag til brukeren. Kildene velges ut fra kriterier som Amanda tror vil optimere nytten for brukeren[2]. Kildene gis så anledning til å søke etter dokumenter, og ettersom resultater mottas fra eksterne søkemaskiner prosesseres og rangeres disse, før de presenteres for brukeren.

Internt er Amanda-arkitekturen basert på distribuerte agenter som samvirker for å produsere nytte for brukeren. Med "distribuerte" menes her at agenter kan finnes i flere eksekveringskontekster, på flere fysiske maskiner, innen forskjellige virtuelle maskiner m.v.:

Grovt kan agentene deles i to grupper ut fra hvor direkte de interagerer med brukeren. *Brukernære agenter* observer hva brukeren gjør og presenterer råd og hint som forhåpentligvis passer til de oppgavene brukeren er i gang med. Observasjoner sendes til, og råd og hint mottas fra de bakenforliggende agentene som utfører

Bakenforliggende agenter tar i mot observasjoner fra brukernære agenter, bruker disse for å produsere råd og hint som sendes til brukernære agenter for presentasjon. Bakenforliggende agenter har også kontakt med eksternt tilgjengelige søkemaskiner. Innenfor Amanda finnes det ikke egne søkemaskiner all søking bruke eksterne søkeverktøy for å søke gjennom dokumentmengdene vi er interesserte i.

For å binde sammen alle agentene brukes et transportlag, og agentene kan derfor adressere hverandre gjennom et navnerom som ikke er avhengig av hvilken eksekveringskontekster de er fordelt mellom. Når forslag skal presenteres, blir de publisert som "forslag", og agenter som er interesserte i å presentere dem for brukere lytter ganske enkelt etter "forslag". Dermed behøver ikke avsender vite nøyaktig hvem som lytter, og lytter behøver ikke vite hvem som sender.

Meldingene som sendes over transportlaget er derfor uavhengig av hvilke konkrete brukergrensesnitt eller applikasjoner de brukernære agentene arbeider mot.

Infrastrukturen for å optimere agentenes atferd på lang og kort sikt bruker elementer fra både genetisk programmering og en markedsmetafor. Disse elementene brukes kun av de bakenforliggende agentene. For å persistent lagre tilstand både om agenter og dokumenter er det implementert en persistensmodul. Det samme grensesnittet for persistens brukes både av brukernære og bakenforliggende agenter, men de brukernære agentene implementerer dette ved å delegere persistering til bakenforliggende agenter som igjen implementerer persistering direkte mot en relasjonsdatabase.

[2]Vi tillater oss en viss antropomorfisering i denne beskrivelsen. Dette gjør vi kun for å få flyt i beskrivelsen, og uten at vi for fullt alvor tillegger hverken Amanda eller andre programmer evner til å hverken "tro" eller "føle" på noe vis som minner om hva et menneske gjør.

All logikk for produksjon av forslag og evaluering av tilbakemeldinger ligger i de bakenforliggende agentene.

Kapittel 3

Teori

Teorien som er brukt i denne oppgaven hentes i hovedsak fra to fagområder: Det første er "Information Retrieval" (IR). Vi vil i dette kapitlet overordnet se på problemstillingen Information Retrieval forsøker å løse og hvilke metoder som benyttes. Vi vil underveis påpeke hvordan disse elementene har gått inn i design og implementasjon av Amanda.

I Amanda bruker vi agentteknologi ekstensivtm det andre fagfeltet vi henter teori fra er derfor autonome adaptive agenter. Vi presenterer et omfangsrikt rammeverk for å forstå og beskrive agentadferd, og gir også her direkte koblinger mot elementer i Amanda. Når Amandasystemet er operativt vil det ha en adferd, og det er betimelig å tenke på hvordan man skal evaluere denne adferden. Det siste fagområdet vi henter teori fra er derfor evaluering av søkeverktøy. Dette er en tverrfaglig disiplin som henter elementer både fra Information Retrieval, psykologi, økonomi og annet. Vi vil diskutere noen elementer fra disse som har hatt direkte relevans for arbeidet med Amanda.

definisjon

3.1 Information Retrieval

Vi starter med en definisjon av hovedproblemstillingen som ligger til grunn for fagfeltet Information Retrieval forsøker å løse (hentet fra [57, s. 1]):

> *Information Retrieval (IR) deals with the representation, storage, organization of and access of information items. The representation and organization of the information items should provide the user with easy access to the information in which he is interested.*

Information Retrieval er altså et fagfelt hvor man er opptatt av gjenfinning av informasjon som er relevant i forhold til en brukers informasjonsbehov. IR er ofte betraktet som synonymt med dokumentgjenfinning og tekstgjenfinning, men det finnes IR systemer som finner fram bilder, lyd, eller andre typer ikketekstlig informasjon.

> Undoubtedly, to designers contemplating the unpredictability of the uses and settings of what they design, grappling with context can appear about as attractive as wrestling with a whale: The task looks overwhelming, and the opponent offers few obvious handholds. Context independence, by contrast, appears much less demanding. Moreover, the informational potential of new technologies, which seems to allow them to communicate directly and independently to their users, makes context independence now seem achievable. We, however, have both theoretical and practical reservations.

Figur 3.1: John Seely Browns hjertesukk om vanskene med å forstå kontekst (Fra [33])

Stikkordet er her "relevant", og understreker forskjellen mellom IR og "Data Retrieval": Dersom et spørreuttrykk i et spørrespråk[1] relatert til om en database returnerer ett galt record blant tusen, betyr det at spørresystemet har en alvorlig feil. I et IR system er det mindre viktig at det leveres "gale" data en gang i mellom enn at det faktisk returneres data som oppfattes som nyttige. På den annen side er det også svært utfordrende å lage gode IR systemer, siden det ikke hjelper å returnere tusen svar som er "riktige" i forhold til spørringen som gis, hvis de ikke er relevante for brukeren i den konteksten hun er i. Kontekst er ikke enkelt å modellere, noe som reflekteres i sitatet i figur 3.1, og vi vil i denne oppgaven ikke forsøke å gå dypt inn i problematikken rundt denne. Vår innfallsvinkel vil hele tiden være å forsøke å lage atferd som oppfattes som nyttig, og fokusere sterkt (og kanskje ensidig) på tilpasningsdyktighet i Amanda-arkitekturen: Dette sterkt på bekostning av alle forsøk på noen dyp forståelse av hva kontekst dypest inne er.

I [57] gir Baeza-Yates og Ribeiro-Neto[2] en taksonomi gjengitt i bearbeidet utgave i figur 3.2 for beskrivelse av komponentene i IR systemer. Med utgangspunkt i brukerens oppgaver, deler de inn modellene for IR inn slik. Baeza-Yates observerer så at dokumenter kan observeres på forskjellig vis gjennom "logiske perspektiver[3]". Som eksempler på slike logiske perspektiv nevner de *indekstermer*, der man trekker frem et sett av termer (f.eks. nøkkelord) som skal representere dokumentet, den *fulle teksten*, som er hele teksten slik den forekommer i originalen, og *full tekst med struktur*, hvor man i tillegg til all tekst også er i stand til å skille ut overskrifter, deloverskrifter, avsnitt etc. Det finnes naturligvis mange slike logiske perspektiver, og hvilke man velger å bruke er i stor grad styrt av man ønsker å gjøre med dokumentet. Videre gir Baeza-Yates en tabell (gjengitt i bearbeidet utgave som figur 3.3) der de viser hvordan modeller for gjenfinning oftest er assosiert med bestemte kombinasjoner av logisk perspektiv og brukeroppgave.

[1] Både "query language" og "queryspråk" brukes på norsk synonymt med spørrespråk.

[2] Vi vil i denne teksten referere til [57] dels gjennom direkte dokumentreferanse, og dels med en henvisning til "Baeza-Yates".

[3] I originalen "Logical views". Som hovedregel vil kun norske (eller fornorskede) termer bli benyttet i denne oppgaven, men når et ord er hentet fra engelsk og jeg ikke er helt sikker på om oversettelsen får med alle nyanser fra originalen gjengis originalen i en fotnote, slik som her.

- **Gjenfinning: Ad hoc eller filtrering:**
  - **Klassiske modeller:** De to "mest klassiske" IR modellene er probabilistisk søk og vektormodellene. Det finnes flere moderne varianter av disse. Vi nevner dem kort under, men omtaler dem ikke videre i denne oppgaven:
    - **Mengdeteoretiske:** Fuzzy sets, Utvidet boolsk metode. algebraiske modeller.
    - **Algebraiske:** Generaliserte vektorer, latent semantisk indeks, nevrale nett.
    - **Probabilistiske:** Inferensnett, "belief network".
  - **Strukturerte modeller:** Standardstrukturen for dokumenter er en enkelt liste av termer, men et par varianter av dette er i helt vanlig bruk:
    - **Ikke-overlappende lister:** Hierarkisk dekomponering, hvor hvert nivå av hierarkiet representerer en oppdeling uten overlapp av sin del av dokumentet.
    - **Nærliggende noder:** Nærhet mellom termer eller andre komponenter brukes som utgangspunkt for navigasjon. Spørringer som *Finnes ordet "fotball" mindre enn femti termer unna ordet "mål"?* kan besvares i denne type modell.
- **Browsing:**
  - **Flat:** Termene i dokumentet kommer som en sekvens av symboler uten ytterligere struktur.
  - **Strukturstyrt:** Hierarkisk dokumentstruktur brukes for navigering.
  - **Hypertekst:** Det finnes lenker på tvers av hierarkisk struktur, og på tvers a dokumenter, og lenkene kan følges av en bruker.

Figur 3.2: *Baeza-Yates taksonomi av modeller for "Information Retrieval" brutt ned gjennom brukerens oppgaver (omarbeidet fra original fra [57, s. 21]).*

For å klassifisere modeller for informasjonsgjenfinning introduseres følgend struktur (gjengitt eksakt fra[57, s. 23]):

> **Defintion** *An information retrieval model is a quadruple* $[\mathbf{D}, \mathbf{Q}, F, R(Q_i, D_j)]$ *where:*
>
> 1. $\mathbf{D}$ *is a set composed of logical views (or representations) for the documents in the collection.*
> 2. $\mathbf{Q}$ *is a set composed of logical views (or representations) for the user information needs. Such representations are called* queries*.*
> 3. $\mathcal{F}$ *is a framework for modelling document representations, queries and their relationships.*
> 4. $R(q_i, d_j)$ *is a ranking function which associates a real number with a query* $q_i \in \mathbf{Q}$ *and a document representation* $d_j \in \mathbf{D}$*. Such ranking defines an ordering among the documents with regards to the query* q_i*.*

Noen observasjon man kan gjøre seg om denne "DFQR" -modellen er at den logisk sett frikobler flere forhold: Brukeres perspektiv av dokumentet behøver slett ikke være det samme som den underliggende søkemekanismen forholder seg til (selv om de også godt kan være like). For undertegnede representerer $\mathcal{F}$ og R komponentene kjernen av søkesystemet, mens $\mathbf{D}$ og $\mathbf{Q}$ er adaptere

		Logisk perspektiv på dokumentene		
		Indekstermer	**Full tekst**	**Full tekst og struktur**
Bruker Oppgave	**Gjenfinning**	Klassisk, Mengde-teoretisk, Algebraisk, Probabilistisk	Klassisk, Mengde-teoretisk, Algebraisk, Probabilistisk	Strukturert
	Browsing	Flat	Flat, Hypertekst	Strukturstyrt, Hypertekst

Figur 3.3: *Hvordan gjenfinningsmodeller vanligvis er assosiert med bestemte kombinasjoner av logisk perspektiv og brukeroppgave (bearbeidet fra original i [57, s. 21])*

mot brukergrensesnitt av forskjellig slag. Vi vil noe senere i dette kapitlet se på forskjellige slags søkemekanismer, og disse beskrivelsene vil i all hovedsak konsentrere seg om $\mathcal{F}$ og R.

Vi velger allerede her å presentere frem et designprinsipp for Amanda, nemlig at Amanda i størst mulig grad forsøker å bruke **D** og **Q** komponenter fra elementer som allerede finnes i brukerens brukergrensesnitt, og skjuler i størst mulig grad $\mathcal{F}$ og R fra brukeren. Rasjonalet bak dette prinsippet er hentet direkte fra oppgavens problemstilling, som eksplisitt sier at Amanda for seg selv skal legge beslag på minimum av brukerens oppmerksomhet.

Baeza-Yates velger å dele opp brukerens oppgaver i to brede klasser:

1. *Ad-hoc:* Hvor brukeren har en oppgave, og skal forsøke å finne dokumenter som er relevante til denne oppgaven.

2. *Filtrering:* Hvor oppgaven er konstant, men mengden av dokumenter som er tilgjengelige og relevante endres over tid.

   Et "filter" vil i utgangspunktet kun plukke ut dokumenter som *kan* være interessante. En variasjon av filtrering kalt "ruting" vil i tillegg rangere de filtrerte dokumentene ut ut fra et kriterium som antas å beskrive hvor godt dokumentet passer til brukerens informasjonsbehov.

Baeza-Yates observerer så at enhver DQFR-modell kan brukes til begge disse oppgavene, og at det derfor er slik at modellen for IR i stor grad kan betraktes som uavhengig av bruksområdet.

Dokumentgjenfinning omfatter to relaterte aktiviteter: *indeksering* og *tekstgjenfinning*. Indeksering refererer til metodene som brukes for å lagre dokumentene slik at det skal kunne gjennomføres effektive søk i dem, og søking

er mappingen som avbilder søkeuttrykk over til en mengde dokumenter som er relevante i forhold til spørringen. "Relevans" betyr i en IR kontekst kun at dokumentet handler om det samme som spørringen spør etter, og det på en slik måte som en menneskelig "dommer" ville avgjort dette spørsmålet. Relevans er et noe uklart begrep, og dokumenter kan være mer eller mindre relevante i forhold til en gitt spørring.

Denne svært restriktive, og litt kunstige definisjonen av "relevans" er utfordret av dem som mener at "nytte"[4] eller "verdi" er et langt bedre begrep: Den informasjonen som er slik at den påvirker beslutninger er den som har verdi og derfor er de man ønsker å finne. Et slikt perspektiv reflekterer mer intensjonen i Baeza-Yates definisjon av IRs hovedproblemstilling (gjengitt i 3.1). Vi skal senere (seksjon 4.3 og 4.2) vise til arbeider som bruker et nytteperspektiv i stedet for snevert definert "relevans". Amanda bruker også et nyttebegrep for å kalibrere sine agenter.

En noenlunde standardisert metode for å evaluere søk, og dermed algoritmer som gjennomfører søk, brukt i IR er å undersøke hvordan søket oppfører seg i forhold til et manuelt søk utført under "ideelle forhold". Dette er en arbeidskrevende prosess, siden hvert uttrykk som leveres må evalueres av et menneske. Prosessen er slik: Et menneske lager et søkeuttrykk, og evaluerer så manuelt hvert dokument i korpusen[5] for å bedømme hvorvidt de er eller ikke er relevante i forhold til spørringen. Søkealgoritmen som skal testes bes så om å gjøre det samme, og den leverer da fra seg en liste med dokumenter som den mener er relevante. Ved å sammenligne resultatene fra søket algoritmen gjorde og resultatene fra den manuelle gjennomgangen kan man så beskrive algoritmens ytelse langs to dimensjoner:

Presisjon [6] Andelen av dokumentene som ble presentert av søkealgoritmen som faktisk var relevante.

Gjenfinning [7] Andelen av relevante dokumenter som fantes i korpusen som faktisk ble funnet.

Siden en høy grad av subjektivitet er involvert i evalueringen, og det dessuten kan være store dokumentmengder involvert, er det mye å hente på å samarbeide om evaluering av søkemetoder. Text REtrieval Conference (TREC) gir et forum for å samordne ressurser for evaluering av søkealgoritmer for tekst. Dokumentkorpuser velges fra kjente dokumentsamlinger (f.eks. noen årganger av tidskrifter eller nyhetsartikler). Spørringene lages ved å søke gjennom korpusene etter bestemte emner. Spørringer og korpuser blir distribuert til deltakerne som så bruker sine algoritmer for å returnere rangerte lister av doku-

[4] "Utility".

[5] Webster [75] definerer "Corpus" slik: *a : all the writings or works of a particular kind or on a particular subject; especially : the complete works of an author b : a collection or body of knowledge or evidence; especially : a collection of recorded utterances used as a basis for the descriptive analysis of a language*. Det er vanlig å beskrive en mengde dokumenter brukt i tekstanalyse som en "korpus".

[6] "Precision".

[7] "Recall".

menter i forhold til en gitt spørring. Disse dokumentene blir da evaluert for relevans av den samme personen som skrev spørringen.

Denne evalueringsmetoden er basert på to antakelser. For det første at relevans er det riktige kriteriet for å bedømme et søkesystem. Andre faktorer, slik som kvaliteten på dokumentet, om dokumentet allerede er kjent, hvor mye det koster å finne dokumentet, og hvorvidt spørringen faktisk representerer brukerens informasjonsbehov tas ikke hensyn til.

Den andre antakelsen implisitt i TRECs evalueringsmetoder er at spørringene som testes er representative for spørringer som vil skje under faktisk bruk. Dette er ikke nødvendigvis en gyldig antakelse, siden spørringer som ikke er godt representerte av dokumenter i korpusen, eksplisitt fjernes fra vurderingen. Disse to antakelsene kan oppsummeres slik: Dersom et gjenfinningssystem ikke returnerer noen dokumenter som møter en brukers behov, er det ikke systemets feil så lenge feilen enten er på grunn av en dårlig konstruert spørring, eller dårlige dokumenter i korpusen.

Vi ser altså at tradisjonell IR slik Baeza-Yates beskriver den i stor grad har utelatt mange ikke-funksjonelle betraktninger rundt et praktisk informasjonsgjenfinningssystem, f.eks. hvor lang tid det tar å finne frem til den aktuelle informasjonen og hvordan denne informasjonen skal tilpasses brukerens arbeidskontekst. Slik vi oppfatter det reflekterer denne utelatelsen at dette er en problemstilling som ikke har vært spesielt sentral i IR fagfeltet muligens på grunn av historiske årsaker. Utelatelsen er sikkert hensiktsmessig for mange typer søk, men slett ikke alle. For søkene som gjøres gjennom Amanda er det for eksempel både av stor interesse at resultatene er tilgjengelig for brukeren mens de har relevans i forhold til det brukeren holder på med og at de presenteres på en slik måte at de ikke gjør brukeren mindre produktiv. Vi velger å kommentere dette her for å motivere at selv om tradisjonell IR gir nyttige verktøy for Amanda, er det nødvendig å lete utenfor IR for å finne løsninger på andre helt sentrale problemstillinger som Amanda må løse. Et interessant prosjekt er muligens å utvide DQFR - rammeverket til å inneholde beskrivelser av ikkefunksjonelle behov. Et slikt arbeid faller imidlertid utenfor rammen av denne oppgaven.

3.1.1 Metoder for informasjonsgjenfinning

Vi skal nå se litt nærmere på noen konkrete metoder for informasjonsgjenfinning. Metodene vil bli plassert i DQFR rammeverket fra side 21, og presentasjonen vil være styrt ut fra hva som er relevant overfor Amanda i den forstand at vi vil bruke den til å illustrere enkelte egenskaper av designet. Det imidlertid er så mange metoder både for indeksering og søking så det faller langt utenfor denne hovedfagsoppgavens omfang å kunne beskrive dem alle, vi vil derfor ikke kunne gå i dybden der dette ikke direkte forklarer fenomener med relevans for Amanda.

- **Vektorrom modellen:** I vektorrommodellen representerer både spørringer og dokumenter som vektorer i et "multidimensjonalt informasjonsrom"

(MI). I MI kan er dimensjonene, og dermed indeksene, være ord som forekommer i dokumentet. I tillegg kan andre navgitte egenskaper (f.eks. hvor mange bilder et dokument inneholder) inkluderes. Felles for alle dimensjonene er imidlertid at de avbilder indekser inn i flyttall, og de resulterende disse vektorene brukes så for å representere dokumentene. Søk rangeres gjennom å regne ut en eller annen (vi skal se på flere) avstand mellom en spørrevektor og en dokumentvektor i dette informasjonsrommet. Dette er den modellen Amanda i all hovedsak baserer seg på.

- **Probabilistisk modell:** Innfallsvinkelen som brukes her er å ta utgangspunkt i en mengde dokumenter med målbare egenskaper og kjente klasser, f.eks. "dette er en sportsartikkel" eller" eller "dette er spam-epost". Fra dette utgangspunktet lages en funksjon som avbilder dokumentets egenskaper inn i en sannsynlighetsverdi for at dokumentet er i en av de kjente klassene. Denne probabilistiske henter metoder fra den rikholdige litteraturen for statistisk klassifikasjon og kan gitt passende rammebetingelser (slik som god tid, mye prosesseringskapasitet, gode korpuser av ferdigklassifiserte dokumenter) gir gode resultater også for tekst. Amandas design har infrastruktur som er forberedt for å kunne benytte en probabilistisk modell, men bruker den pr. i dag ikke. Tilpasningene er først og fremst gjort ved å gjøre data tilgjengelige på en form som er anvendelige for probabilistiske algoritmer, hovedsaklig gjennom bruk av "wector" datastrukturen vist i figur 3.5 på side 33.

- **Naturlig språk:** Både vektorroms-modellen og den probabilstisk modellen bruker "lure triks" for å hente ut relevante dokumenter: Overfladiske syntaktiske egenskaper ved dokumentene prosesseres statistiske og brukes som basis for søk og klassifikasjon. Dette skjer uten at det gjøres noe forsøk på å forstå hva som står i dokumentene på noen som helst dyp måte. Naturlig språkprosessering (NLP) sniker seg ikke unna denne jobben, og forsøker å parsere naturlig språk til representasjoner av abstrakt mening. De konseptuelle modellene av spørringer og dokumenter kan så sammenlignes direkte. Dette er en fundamentalt forskjellig fremgangsmåte fra dem som er beskrevet i de to metodene vist over, siden denne metoden forutsetter at man klarer å bygge modeller av *verden*, ikke bare av ordbruk om den.

  Amanda bruker ingen metoder for naturlig språkforståelse. For denne oppgavens tilnærming har jeg vurdert NS som et ikke spesielt fruktbart perspektiv. Å parsere setninger for å "knekke åpen" intern struktur i innebærer tekniske problemer som ligger langt utenfor denne hovedfagsoppgave rekkevidde, og jeg har derfor valgt å ikke forfølge denne innfallsvinkelen videre. Siden jeg ikke dessuten ikke synes at dette var et særlig behagelig valg er det i kapittel 6 om videre arbeid skissert noe om hvordan moduler for forståelse av naturlig språk kan tilpasses Amandas arkitektur.

- **Kunnskapsbaserte innfallsvinkler:** Enkelte ganger kan kunnskap om et

bestemt domene hjelpe til med gjenfinning. For eksempel kan et ekspertsystem hente dokumenter om sykdommer basert på en liste av symptomer. Slike systemer vil bruke kunnskap fra det medisinske for å gjøre en diagnose og hente frem de riktige dokumentene. Andre domener kan ha struktur som kan utnyttes. Amanda bruker ikke nå slike teknikker, men dette virker som relativt overkommelige teknikker å ta i bruk, og den interne representasjonen av dokumenter og egenskaper er forberedt for å kunne bruke slike teknikker.

Kombinasjonen av kunnskapsbaserte systemer og naturlig språkforståelse er interessant. Selv begrensede skritt i retning NS vil kunne fungere som "mediatorer" [6] som automatisk kan gjøre fri tekst anvendelige for analyse av kunnskapsbaserte systemer. Dette er avgjort av interesse, men faller også utenfor rammen av denne hovedfagsoppgaven.

- **Datafusjon:** Datafusjon er en teknikk hvor flere algoritmer og indeksmodeller brukes for å gi forskjellige mengder av relevante dokumenter. Resultatene kombineres så på ett eller annet vis for å gi en liste av de beste dokumentene. Både Rhodes sitt Savant system system[56], Moukas sitt Amalthea [2] og Amandas er eksempler på datafusjons IR systemer.

Probabilistisk modell

Den probabilistiske modellen antar at det finnes en mengde egenskaper man kan observere hos dokumenter. Videre finnes det en mengde predefinerte klasser som dokumenter kan være medlem i. Den probabilistiske modellen forsøker så lage en modell som avbilder de observerte egenskapene inn i en sannsynlighet for at dokumentet er medlem i en av de predefinerte klassene. For å lage modellene brukes det læringsalgoritmer fra statistisk analyse, ofte varianter av en naiv Bayes klassifikator, som så kan kjøres på dokumenter med ukjent klassifikasjon. Resultatet av kjøringen er da for hvert dokument et tall pr. mulige klasse, som viser hvor sannsynlig det er at dokumentet tilhører denne klassen. Standardmåten er så å anta at et dokument kun kan finnes i en klasse, og så alltid velge den klassen som hadde høyest sannsynlighet for å bli valgt.

Litteraturen indikerer at statistisk modellering kan gi god klassifikasjon av meldinger, men det forutsettes at man har en korpus der klassifikasjon allerede er gjennomført, og det kan dessuten være temmelig tidkrevende å produsere klassifikatorer basert på lange egenskapsvektorer i tusenvis av dokumenter.

Rasjonalet for å ta inn probabilistiske klassifikatorer i Amanda var at det virker som om det er en mulig innfallsvinkel for å raskt identifisere dokumenter som passer med brukerens stabile interesser. Dette forutsetter at man har en samling klassifiserte dokumenter som allerede reflekterer denne interessen, men flere slike kilder er enkelt tilgjengelige: En kilde til dokumenter kan være dokumenter i web-browserens "favorites", da disse ofte er hierarkisk klassifisert etter brukerens forgodtbefinnende. Avmerkede "spam" meldinger i en epostleser kan også representere en noen lunde stabil oppfatning om hva

slags meldinger som skal behandles som søppelpost. Dersom brukeren bruker filsystemet som et tematisk arkiv (f.eks. en regel om at filer som handler om "bensinstasjoner" havner i "bensinstasjon-katalogen") utgjør også en slik allerede eksisterende kilde. På bakgrunn av dette rasjonalet ble det et et designmål for Amanda at statistiske klassifikatorer skulle være enkle å implementere. Vi skal her beskrive en av dem, en naiv Bayesisk klassifikator, og så skissere noen designmessige implikasjoner vårt ønske om å kunne implementere denne leder til.

- Vi starter med en mengde kategorier C, hvor enkeltkategoriene er indeksert C_i, og en korpus av dokumenter D der det er kjent hvilken kategori hvert enkelt dokument $d \in D$ har. Korpusen med kjent klassifikasjon kaller vi for kalibreringskorpusen.

- Det finnes en mengde av egenskaper $t \in T$ som dokumentet enten har eller ikke har i større eller mindre grad. Hvor mye av en egenskap et dokument har er i grunnen det samme, så mye hvert bidrag er i intervallet $[0, \ldots, 1]$ og dermed kan tolkes som en sannsynlighet for at egenskapen er tilstede.

  En slik egenskap kan f.eks. være at dokumentet inneholder bestemte ord i et visst forhold til det totale antall ord. Det kan også tenkes at det er andre egenskaper, f.eks. at dokumentet er eller ikke er epost, at det er skrevet av en bestemt forfatter eller liknende.

- Det at et dokument har en egenskap t vil nå bidra til sannsynligheten for at det er i klasse C_i avhengig av hvor unik egenskapen t er for elementer i C_i i kalibreringskorpusen ($P(t|d)$ er sannsynlighetne for at dokumentet d har egenskapen t). For kalibreringskorpusen vil det nå være slik at sannsynligheten for at et dokument d er i kategorien C_i, altså $P(C_i|d)$ er gitt av:

$$P(C_i|d) \quad = \quad \sum_{t \in T} P(C_i|t\, d) P(t|d)$$

  der $P(C_i|t\, d)$ er sannsynligheten for at dokumentet d er i kategori C_i dersom egenskapen t også er til stede.

- Om vi antar uavhengighet av C_i og at d impliserer t d.v.s. at

$$(C_i|td) = P(C_i|t)$$

  får vi:

$$P(C_i|d) \quad = \quad \sum_{t \in T} P(C_i|t) P(t|d)$$

- Ved å bruke Bayes regel[8]
  har vi da:

$$P(C_i|d) \quad = \quad P(C_i|t) \sum_{t \in T} \frac{P(t|C_i)P(t|d)}{P(t)}$$

For å klassifisere velger man den klassen som har høyest sannsynlighet.

Det vi nettopp har vist er strukturen i en bayesisk klassifikator beskrevet av Tokunaga [71, s. 2].

I Baeza-Yates DQFR-rammeverk er det vi nå har beskrevet en $\mathcal{F}$ komponent[9], men det er ennå ikke gitt noen rangeringsfunksjon som kan fortelle oss hvorvidt et dokument er relevant eller ei. Det gis imidlertid en metode for dette ([57, s. 32]). I korthet går den ut på at vi skal finne dokumenter i klassen "relevant". Vi har en slik mengde dokumenter R som er kjent, eller i det minste antatt å være relevante, og vi har da også komplementet til denne mengden $\bar{R}$ som vi antar ikke er relevante. $P(R|d_j)$ er sannsynligheten for at dokumentet d_j er relevant for spørringen q, og $P(\bar{R}|d_j)$ sannsynligheten for at det ikke er det. Likheten mellom dokumentet og spørringen kan da defineres som forholdet: $\text{sim}(d_j, q) = \frac{P(R|d_j)}{P(\bar{R}|d_j)}$ som gjenom Bayes teorem kan beskrives som:

$$\text{sim}(d_j, q) = \frac{P(d_j|R)P(R)}{P(d_j|\bar{R})P(\bar{R})}$$

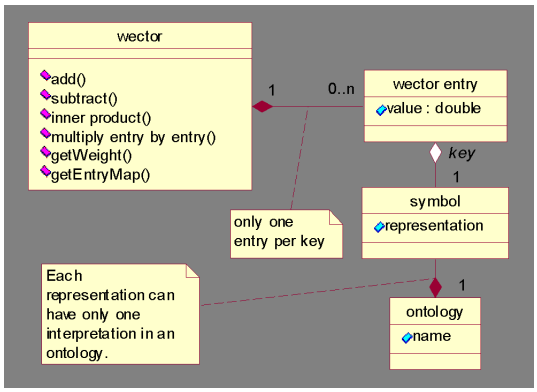
Å implementere en naiv Bayesisk klassifikator i er teknisk ganske enkelt: I essens må man gjøre litt matrisemultiplikasjon hvor indeksmengden kan være meget stor, og hvor verdien for de fleste indeksene er null. Å kjøre klassifikatoren er derfor regnemessig uproblematisk, forutsatt at man har en god implementasjon av "sparse" vektorer (vektorer hvor det kun for en liten andel av indeksene er verdier som er forskjellige fra null). Å finne frem til klassifikasjonsparametrene, altså matrisen det skal multipliseres med kan bli regnemessig ressurskrevende, så det er lite hensiktsmessig å sette strenge sanntidskrav for disse utregningene.

[8] Bayes teorem ([5, s. 94]) sier at dersom $P(C_i|d) = \frac{P(dC_i)}{P(d)}$ og d kun kan være i en av klassene $C_i \in C$, d.v.s. at d er unionen av de disjunkte hendelsene $d\,C_1, \ldots, d\,C_n$, da har vi at $P(d) = \sum_{C_i \in C} P(d\,C_i)$. Hver term i denne summen kan finnes ved $P(dC_j) = P(C_j)P(d|C_j)$, og disse kombineres da til:

$$P(C_i|d) \quad = \quad \frac{P(C_i)P(d|C_i)}{\sum_{C_j \in C} P(C_j)P(d|C_j)}$$

Fremstillingen som er gitt av Bayes teorem er semantisk identisk den som finnes i den oppgitte referansen, men symbolbruken er skrevet om til å være konsistent med resten av fremstillingen i dette kapitlet. Slik frihet til omskrive uten å endre er brukt alle andre steder i denne oppgaven der det referere formelapparater hentet fra andre artikler.

[9] "Framework for modelling document representations, queries and their relationships".



Figur 3.4: *Dette er et UML klassediagram for "wectorer", Amandas sparse-vector implementasjon. "Wector" er en klasse med flere metoder (signaturer bare antydes gjennom tomme parenteser). Hver wector kan ha flere "wector entry" objekter assosiert til seg. Hvert entry har i sin tur en nøkkel assosiert til seg, og hver wektor kan kun inneholde ett entry pr. nøkkel. En nøkkel er et symbol som finnes inne en "ontologi", som vi enn så lenge kan betrakte kun som et navnerom, slik at hver symbolrepresentasjon kan finnes en gang pr. ontologi. "Diamantene" i diagrammene er UMLs "aggregat" notasjon. Fylte diamanter betyr sterk aggregering, som i dette tilfellet betyr at dersom en wektor fjernes, så vil alle entries som var del av den også forsvinne, men dersom et symbol fjernes fra et wector entry så vil symbolet fremdeles finnes, så lenge ontologien symbolet finnes i også eksisterer.*

Ut fra dette resonementet var det åpenbart at Amanda måtte ha en implementasjon av "sparse" vektorer, og det måtte være rom for stor variasjon i termene det kunne indekseres over. Både ord hentet direkte fra dokumentene og andre avledede egenskaper måtte kunne plasseres i en og samme vektor. Amandas dokumenter har derfor assosiert med en egenskapsvektor i en datastruktur kalt "Wector" (vist i figur 3.4). Stavemåten er med vilje gjort litt ustandard for å unngå forvirring om andre typer vektorer. En Wector representerer en total avbildning av indeksmengden (kalt "nøkler") inn i et flyttall. Der en indeks ikke eksplisitt er gitt en avbildning i datastrukturen antas at indeksen avbildes inn i verdien null. Indeksenes navn er igjen sammensatt av to komponenter; et navnerom (kalt en "Ontologi"), og et unikt navn innenfor navnerommet. En ontologi brukes for å representere ord slik de finnes i dokumentet dette kalles i Amanda "standard" ontologien siden den ikke gir noen bestemt informasjon om ordet bortsett fra at det eksisterer som en indeks, og andre ontologier kan brukes for å representere andre egenskaper om dokumentet, f.eks. at et ord er hentet fra et bestemt språk, at det har en bestemt mening (f.eks. en spesifikk medisinsk diagnose) eller liknende. Dersom ord klassifiseres inn i bestemte spesialiserte ontologier indikerer dette gjerne at ordet har en bestemt mening innenfor et bestemt domene, f.eks. medisinsk diagnostikk, noe som ig-

jen indikerer at spesialiserte moduler som representerer kunnskaper om dette domenet kan anvendes på dokumentet. Vi ser dermed at Wektorene åpner for at utvidelser av Amanda i retning kunnskapsbasert IR ikke skal kreve radikale reformuleringer av den basale representasjonen av dokumentenes egenskaper.

Vektorrom modell

Vektorrommodellen (VRM) er en svært robust metode for IR [60]: Gitt en samling dokumenter ser VRM både dokumenter og spørringer som samlinger av ord, og for hvert og lager en n-dimensjonal sparse som representerer dokumentet. Dersom dokumentet har femti forekomster av ordet "hest", inneholder vektoren verdien femti for indeksen "hest". Sammenligninger mellom dokumenter og spørringer overfor dokumenter betraktes da som sammenligninger av vektorer. I matematikken finnes det veldig mange måter å sammenligne vektorer på, og innen IR har det i praksis har det vist seg at kombinasjoner av vektinger basert på termfrekvens, invers dokumentfrekvens og sammenligninger basert på cosinus av vinklene mellom dokumentenes vektorrepresentasjoner har gitt gode resultater. En noen lunde typisk spørring utført i vektorrommodellen ser slik ut:

1. Et input-dokument d er definert som en sekvens av symboler:

$$\begin{aligned} d &= \text{et dokument} \\ \sharp(d) &= \text{Antall symboler i dokument } d \\ d_i &= \text{Ord nr. } i \text{ i dokument } d \\ d &= \langle d_1, \ldots, d_{\sharp(d)} \rangle \end{aligned}$$

2. Stopp-ord slik som "en", "den", "a", "and" o.l. finnes i en statisk ordliste, og alle disse ordene fjernes fra vektoren (tellingen deres settes til null).

3. "stemming" utføres på input-dokumentet. Dette er å fjerne endelser av ord for å finn frem til ordstammer: I stedet for å lagre ordet "hestene" fjernes endelsen "ene" og man sitter igjen med ordstammen "hest". Alle ord som har samme stamme regnes som samme ord i frekvens-vektoren.

$$\begin{aligned} \text{stem}(t) &= \text{Funksjon som fjerner endelser fra ord}(t) \\ \text{stem dokument}(d) &= \langle \text{stem}(d_1), \ldots, \text{stem}(d_{\sharp(d)}) \rangle \end{aligned}$$

Stemming gjør at indeksene som må vedlikeholdes blir mindre, og dessuten variasjoner i endelser ikke gir falske negativer fordi ord med forskjellige endelser antas å være helt ulike ord. Man kan utføre mange typer

stemming. Vi vil nedover ikke eksplisitt si om det den stemmede eller ustemmede versjonen av dokumentet som brukes, siden vektorromsmodellen ikke tar hensyn til dette (med noen få unntak som nevnes eksplisitt). Amanda bruker en meget enkel stemmingsalgoritme som fjerner noen av de mest vanlige endelsene på norsk og engelsk. Algoritmen Amanda bruker kan beskrives med en mengde regulære uttrykk, som alle matcher ord, og så fjerner de delene av ordet som finnes på utsiden av parentesen i det regulære uttrykket (f.eks. "$(\cdot * \cdot\cdot)$ene"). Amandas stemmer er ingen spesielt god stemmingsalgoritme, den inneholder få og litt tilfeldig valgte regler, men den gjør *noe* stemming, og fungerer som en plassholder i systemet til en bedre algoritme kan implementeres. Rhodes [56] bruker to stemmere, en "sterk" og en "svak". Svak stemming er definert som første skritt av Porter-stammeren ([39], også gjengitt i [57, s. 433-436]), og derfor fjerner færre variasjoner. Walker og Jones [40] viste at svak stemming øker gjenfinning betydelig, og svekker ikke presisjon i særlig grad. Sterk stemming ofrer presisjon til fordel for større gjenfinning. Det er ikke mulig å uten videre bruke Porter stemmeren på norske ord, siden stemmeren har hardkodet flere elementer fra engelsk morfologi.

Et problem med å gjennomføre stemming er at dersom man gjør det, blir det vanskeligere å bruke de resulterene vektorene som representerer dokumenter og spørringer til å produsere søkeuttrykk til bruk i f.eks. web-søkemaskiner. Det er derfor ikke opplagt at stemming bør brukes alltid, og om man gjør det, kan det tenkes at det er nyttig å vedlikeholde både en stemmet og en ustemmet versjon av egenskapsvektoren.

4. En parser oversetter teksten til en frekvens-vektor der hver term representerer hvor ofte et bestemt ord forekommer i tekstfragmentet.

$$
\begin{aligned}
TF_{t,j} &= \text{Termfrekvensen av symbol } t \text{ i dokument } j \\
TF_{t,j} &= \sharp_t(j)/\max_l \sharp_l(j) \\
\max_l \sharp_l(j) &= \text{Antall ganger symbolet } l \text{ som forekommer flest} \\
&\quad \text{ganger i } j \text{ forekommer.} \\
\sharp_t(j) &= \text{Antall ganger symbol } t \text{ forekommer i dokument } j \\
\sharp(j) &= \text{Antall symboler i dokument } j
\end{aligned}
$$

5. Så regnes invers dokumentfrekvens ut:

$$
\begin{aligned}
\sharp(p) &= \text{Antall dokumenter i populasjonen } p \\
\sharp_t(p) &= \text{Antall dokumenter i populasjonen } p \text{ som inneholder termen } t \\
IDF_{t,p} &= \text{Invers dokumentfrekvens av term } t \text{ i populasjonen } p \\
IDF_{t,p} &= \log(\sharp(p)/\sharp_t(p))
\end{aligned}
$$

6. Så kombineres så TF og IDF komponentene på ett eller annet vis for å lage en TF/IDF vekting.

$$
\begin{aligned}
WTF_{t,j,p} &= \text{Vektet termfrekvens for term } t \text{ i dokument } j \text{ i populasjon } p \\
WTF_{t,j,p} &= f(TF_{t,j}, IDF_{t,p}) \\
f(\text{tf}, \text{idf}) &= \text{En eller annen funksjon med tall som inputparametere}
\end{aligned}
$$

Dersom TF komponenten gis stor vekt, betyr dette at termer som forekommer ofte i dokumentet gis høy vekt. Dersom IDF komponenten gis høy vekt betyr dette at termer som forekommer lite gis høy vekt. Den kanskje vanligeste (i følge [57, s. 29]) vektingen er ganske enkelt $f(tf, idf) = tf \cdot idf$. En variant som Salton og Bucley bruker (referert av [57, s. 30]) er $f(\text{tf}, \text{idf}) = 1/2 + (1/2 \cdot \text{tf}) \cdot \text{idf}$. Denne varianten gir noe større vekt til termer med høy frekvens, og foreslås brukt for å vektgi termer i spørringer. Disse to eksemplene vises kun for å illustrere at det er variasjon, og at det forekommer mer eller mindre vel motiverte "justeringsparametere" som i praksis har vist seg å gi gode resultater.

7. Det kan nå defineres en likhetsfunksjon bestemmer hvor nære vektorene er innen det n-dimensjonale vektorrommet ved å beregne vinkelen $\theta_{(d,q)}$ mellom de to vektorene. Veldig ofte er likhetsfunksjonen cosinus for for dokumentvektoren d og spørringsvektoren q beregnes slik:

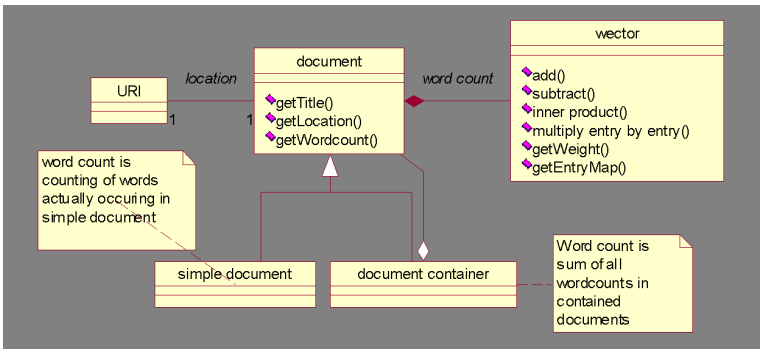
$$
\begin{aligned}
\cos\theta_{(d,q)} &= \frac{d \times q}{|d| \cdot |q|} \\
\text{sim}(q,d) &= \cos\theta_{(d,q)} \\
\text{sim}(q,d) &= \frac{\sum_{i \in T(q) \cup T(d)} WTF_{i,q} \cdot WTF_{i,d}}{\sqrt{\sum_{i \in T(d)} (WTF_{i,d})^2} \cdot \sqrt{\sum_{i \in T(q)} (WTF_{i,j})^2}} \\
T(q) &= \text{Termene vektoren } q \text{ bruker som indekser.}
\end{aligned}
$$

Akkurat denne varianten (som er temmelig ren) er hentet fra [61, s. 243]. Det finnes mange andre.

Til tross for at vektorbasert søk er nærmest triviell i sin enkelhet, spesielt om man sammenlikner med den underliggender rikheten i de språklige uttrykkene somsammenliknes, er metoden svært robust, og kan med de riktige justeringsparametere gi svært gode resultater.

Dette er imidlertid ikke alt: Gjennom vektingsmetodene blir det implisitt plukket ut termer som er representative for dokumentet, og disse termene kan så brukes som input søkealgoritmer som bruker eksterne søkemaskiner til å foreta søk. En metode som benyttes både i Amanda, og av Moukas sitt "Amalthea" system ([2]) er denne:

- Lag TF/iDF vektedede representasjoner av spørringene. Velg populasjon og vektingsmetode med omhu slik at de mest "interessante" ordene får størst vekt.
- Bruk disse ordene til å produsere spørringer mot søkemaskiner.
- Om man ønsker (og har tid), kan man så i tillegg gjennomføre "cosinusbasert" søk på dokumentene levert av søkemaskinen for å rangere søkeresultatene før endelig presentasjon for brukeren.



Figur 3.5: *Oppkobling av dokument med Wektor som representerer ordene som finnes i dokumentet. I likhet med figur 3.4 er også dette et UML klassediagram. Det eneste notasjonselementet som er nytt i dette diagrammet er "spesialisering gjennom subklassing" som er brukt for å vise at et dokument kan spesialiseres på to vis, som en dokumentbeholder og som et enkelt dokument. Å bruke en aggregering fra en subklasse mot en superklasse slik det er gjort her er en direkte anvendelse av "Composite" patternet fra [22, side 163].*

Når eksterne søkemaskiner brukes vil det ofte være slik at det skilles mellom korpusen man søker i og korpusen som brukes for å beregne TF/iDF. Moukas tar utgangspunkt i dokumentene som finnes i en brukers browser hotlist, men gjør ikke bruk av at hotlistens hierarkiske organisering kan brukes for å lage søk som søker etter emner som er kategorisert gjennom hotlistens organisering. Det kan godt tenkes at det i stedet er mer interessant å bruke slike kjente kjente og interessante egenskaper ved korpusen (f.eks. "dette er alle sammen dokumenter som handler om multippel sklerose"), og bruke denne for å finne TF/iDF vekting. En slik kilde til korpuser som kan brukes for å lage modeller relevans kan altså være "hotlists" for web-browsere, eller spor av dokumenter som er lest innenfor "meta-hotlists" av dokumentkilder som man vet har høy kvalitet (f.eks. bestemte tidskrifter brukeren har bestemt seg for er interessante). I disse tilfellener det ikke noe enkelt dokument å finne cosinus mot, men en vektor som representerer f.eks. et gjennomsnitt eller sum av alle dokumentene som er gitt en bestemt klassifikasjon.

Når Amanda overvåker brukerens filer bygges det opp en datastruktur hvor hvert dokument inneholder et Wector-objekt som teller ord i dokumentet (illustrert i figur 3.5. Amanda regner også filkataloger som dokumenter, og vil for katalogene summere alle bidragene fra dokumentene i filene de inneholder og levere dette fra seg som katalogens "word count". Tellingene blir automatisk oppdatert, slik at dersom en endring registreres i et dokument, blir denne propagert til alle kataloger som direkte eller gjennom andre kataloger inneholder det endrede dokumentet. På dette viset vil det fra et dokument alltid være enkelt å finne frem til både en vektor som representerer dokumentet og eventuelt katalogen det forekommer i.

Allan og Salton foreslår i [23] at man ikke kun ser på vektorer som representerer hele dokumentet, men også på vektorer som representerer fragmenter av dokumentet slik som avsnitt, perioder , seksjoner eller liknende. Rasjonalet bak en slik oppdeling er at det kan finnes deler av et større dokumenter som har høy relevans uten at det store dokumentet dermed gir utslag for stor relevans. Det foreslås derfor å bruke en totrinns-metode: La enkeltavsnitt gi bidrag til relevans, og så kombinere disse på en eller annen måte for å finne et bedre relevansmål.

Allan og Salton påpeker at ordtellinger og vektor-representsjon har den sine begrensninger spesielt siden det finnes ord med tvetydige meninger, og at dette kan gi årsak til både falske positive og falske negativer. De presenterer derfor en to-nivås metode der man først ser på lik ordbruk mellom dokumenter, men så krever at individuelle setninger eller avsnitt skal ha svært stor grad av sammenfallenhet for at en match skal kunne antas[10].

En generell metode blir derfor å la teksten deles inn i komponenter slik som avsnitt, seksjoner eller setninger. Hver av disse komponentene kan så brytes inn videre med vektor-representasjon. Allan og Salton påpeker så at om man først gjør en slik nedbrytning, kan man ved å lete etter leksikalsk repetisjon internt i et dokument kan man til en viss grad følge temaer, hvor de oppstår, og hvor de utvikles i dokumentet. Allan og Salton viser så hvordan man kan tegne grafer som viser slike sammenhenger for forskjellige dokumenter. Allan og Salton nevner ikke dette eksplisitt, men det er fullt mulig å tenke seg at man kan la strukturen i slike grafer brukes til å ekstrahere informasjon om hva slags dokument man har for seg: Er det en fortelling, er det en vitenskaplig artikkel, er det en nyhetsartikkel eller hva? Det er også mulig å la en slik teknikk følge temaer på tvers av dokumenter, så man i en mengde dokumenter, som har brutt ned på avsnitt, kan produsere seg "syntetiske dokumenter" som består av temaer som går gjennom flere dokumenter. Disse syntetiske dokumentene kan så brukes til å finne lignende dokumenter ute på nettet, og da ut fra en hypotese om at det finnes temaer som best er dekket av en dokumentsamling, ikke noe enkelt dokument i samlingen. En slik innfallsvinkel blir utforsket noe av Kaszkiel og Zobel i [35], og av Schilder i [61].

[10] De hevder å på denne måten, i oppføringer fra "Funk og Wagnalls" leksikon [21], å under søk etter president John F. Kennedy kunne avvise oppføringen om høyesterettsjustitiarius Anthon M. Kennedy, til tross for at de begge har svært lik utdannelse (Harvard), stillinger med høy status i den amerikanske regjeringen, likt navn etc.

For Amandas vedkommende er det tenkt, men ikke implementert, å bruke en liknende teknikk for å finne frem til dokumenter som ble mer eller mindre direkte siterte. Det tenkes da å ta utgangspunkt i dokumenter som brukeren faktisk ser på, bryte disse opp i fragmenter, og sammenlikne disse fragmentene med fragmenter fra dokumenter brukeren har produsert under arbeidet sitt. Dersom det finnes stor likhet mellom fragmenter brukeren har lest og fragmenter han har skrevet kan Amanda anta (strengt tatt "gjette på") på at brukeren har lest dokumentet og deretter valgt å bruke innholdet fra det ganske direkte. Amanda kan da finne frem til agentene som eventuelt foreslo dokumentet som ble lest og gi denne en "belønning" med en begrunnelse i at brukeren fant det interessant. En slik funksjonalitet kan introduseres ved å utvide dokumentrepresentasjonen vist i figur 3.5 slik at også dokumenter referert til som "simple documents" kan indre hierarkisk struktur på samme måte som "document container" nå har. Det må da introduseres en ny minste atomisk enhet (periode, avsnitt e.l.) som ikke har slik indre struktur.

3.1.2 Hvilken metode er best for Amanda?

Det korte svaret er: "Mange kandidater, ingen klar vinner". Et noe lenger svar er at Amandas sentrale oppgave er å finne relevante referanser. Teknikkene vi bruker er basert på vektor-basert representasjon av dokumenter: Vi teller ord som forekommer i dokumenter, eller deler av dokumenter, og lager vektorer der ordene er indeksene, og antall forekomster er verdi. Med basis i denne representasjonen manipulerer vi så vektoren til den blir anvendelig for effektive søke-algoritmer.

Symbolene som indekseres er i utgangspunktet ord som er funnet i søkeuttrykk og dokumenter, men datastrukturen åpner for at andre symboler fra bestemte ontologier kan brukes, slik at det er mulig å representere at et dokument inneholder termer som har helt bestemte meninger innenfor et domene, også domener som ikke er eksplisitt referert i dokumentet selv. Slik representasjon åpner for at kunnskapsbaserte domene-avgrensede algoritmer kan brukes på dokumentene.

Når vi skal velge blant alle algoritmene som finnes for å finne relevans mellom tekster kodet som vektorer er det et meget stort antall kandidater. Alle funksjoner som tar to vektorer som parametere og avbilder disse på en ordnet mengde dokumenter er i utgangspunktet kandidater til slike algoritmer. Om vi begrenser oss til f.eks. indreprodukter er det fremdeles en stor mengde kandidater, men uansett hvordan vi snur og vender på kandidatmengden er det er det ingen klar vinner blant dem. Det beste man kan si om kandidatene, er at de virker i noen tilfeller, men ikke i andre. Algoritmene er ofte tilpasset alle de beskrankninger problemet de skal løse gir: Dokumentstørrelse, størrelsen av vokabularet, størrelsen av søke-uttrykket o.s.v. [59]. Amandas arkitektur tilbyr derfor innplugging av flere slike algoritmer, og administrasjon av konkurranse mellom dem for å finne hvilken eller hvilke som i en gitt sammenheng er den beste, detaljene om dette er beskrevet i kapittel 5. Denne "prøv og se" holdningen til bruk av de forskjellige algoritmenes søkekompetanser begrunnes og

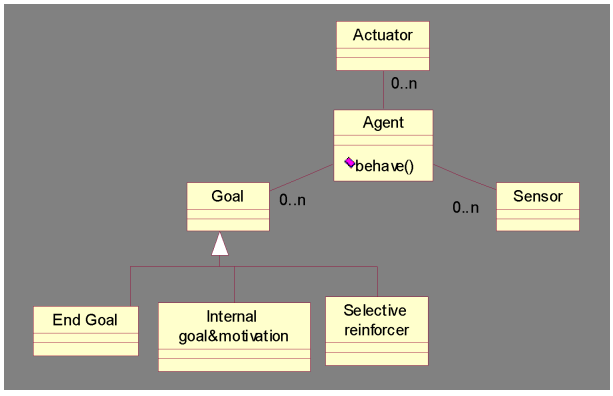
utdypes videre i seksjonen om autonome adaptive agenter.

3.2 Autonome og adaptive agenter

I en oversiktsartikkelen "Modeling Adaptive Autonomous Agents" [42] fra 1994 oppsummerer P. Maes om fremskritt og status rundt temaet autonome adaptive agenter (AAA). Maes fokuserer spesielt på klasser av problemer alle disse bidragene løser, slik at man kan sammenlikne de forskjellige metodene.

Maes artikkel er interessant av flere årsaker: For det første gir den terminologi og begrepsapparat som er nyttig for å forstå den øvrige litteraturen jeg har brukt i denne oppgaven, Alexandros Moukas som skrev [2]var f.eks. en student av Maes, det samme var Rhodes som skrev Remembrence Agent [56], begge disse er systemer vi vil stifte bekjentskap med i kapittel 4 om tidligere arbeider. Dernest er artikkelen interessant fordi den gir et godt rammeverk for å beskrive mitt eget arbeid både med hensyn på problemer jeg forsøkte å løse, og innfallsvinklene jeg valgte. Fremstillingen under følger derfor løseligs Maes struktur, men er begrenset en hel del for å være mest mulig relevant overfor problemstillingene denne oppgaven, kommentarene om anvendelser i Amanda-systemet er naturligvis helt mine egne, det samme er illustrasjonene.

3.2.1 Hva er en adaptiv autonom agent?



Figur 3.6: *En generisk agent. Den har en atferd, den har mål å forholde seg til, sensorer til å observere sine omgivelser og aktuatorer for å påvirke dem.*

En *agent* er et system som forsøker å tilfredsstille en mengde av mål i et komplekst, dynamisk miljø. En agent eksisterer i miljøet: Det kan sanse miljøet gjennom sine sensorer, og det kan utføre handlinger på det gjennom sine aktuatorer (illustrert i figur 3.6).

Maes kategoriserer Artikkelen adresserer en kategori forskning innenfor faagområdet "Kunstig Liv" som arbeider med adaptive autonome agenter. *Autonome agenter* er systemer som fungerer i et dynamisk, uforutsigbart miljø hvor de forsøker å tilfredsstille en mengde tidsavhengige mål eller motivasjoner.

Autonome agenter er også kjent under navn som "atferdsbasert kunstig intelligens (AI), "nedenfra og opp AI", eller "kunstig dyr"[11].

En agents atferd betraktes som styrt av mål, men mål kan så anta mange former:

Slutt-mål, mål som definerer en "slutt-tilstand" for prosessen agenten arbeider med. *Selektiv forsterkning*, som forsterker en motivasjon mot målet agenten forsøker å oppnå og dessuten *interne behov og motivasjoner* som typisk er tilstander som agenten må holde innenfor visse grenser for å kunne fortsette å eksistere (f.eks. oppladet batteri i en robot).

En agent er *autonom* om den opererer autonomt, d.v.s. at den selv bestemmer hvordan den skal forholde seg til sine sensordata og aktuatorer slik at mål nås. Agenten sies å være *adaptiv* dersom den er i stand til å forbedre seg slik at den over tid øker sin kompetanse i til å håndtere sine mål.

3.2.2 Styrende prinsipper

Studiene av AAA er basert på to hovedinnsikter:

1. *Å se på hele systemer endrer problemer på en måte som ofte er fordelaktig.*

   En funksjon som modelleres er en del av en agent. Å bygge systemer på en integrert måte i stedet for å lage separate moduler kan gjøre oppgaven enklere. Et system som lærer kan bruke mindre ressurser på planlegging og kan i stedet kan lagre planer til senere bruk. Et system med sensorer og aktuatorer kan teste miljøet, og har mindre behov for å modellere miljøet for slutninger og ressonementer.

   Et komplett intelligent system er alltid en del av en omgivelse. "Verden er sin beste modell"[14] og sensorer og aktuatorer kan dermed gjøre behovet for mindre for internaliserte modeller ("habitat begrensninger" [29]). For eksempel behøver ikke et system for naturlig språkforståelse å gjøre enhver ytring helt utvetydig hvis det kan gå inn i en diskurs f.eks. ved å stille spørsmål eller å gi enkelte merknader som gradvis vil hjelpe til med å gjøre det entydig hva den andre taleren ønsker å formidle.

   Systemer er også en del av et "samfunn" av andre agenter som i det samme miljøet arbeider med lignende eller relaterte problemer. Det er derfor ikke noe behov for enhver agent å finne ut alt alene. En mobil robot kan f.eks. bruke en strategi med å følge tett inntil en person for å oppnå kompetanse til å navigere i et kontormiljø uten å komme borti ting [44].

[11] "Animat approach" er en forkortelse av "Artificial Animal[2, s. 1]

En konsekvens av denne innsikten er at det innen AAA er et sterkt fokus på å modellere systemer i kontekst. Forskningen har alltid lagt vekt på å løse "ekte" systemer som løser faktiske, men små, problemer i et konkret miljø.

2. Interaksjons-dynamikk kan lede til komplisert observebar atferd.

   Agent-forskning er basert på troen om at ved å skifte mot "interaksjons" domener i motsetning til "komponent" domener, vil det bli enklere å produsere intelligente systemer. Denne ideen kan anvendes på mange nivåer:

   Interaksjon mellom agent og miljø kan lede til "fremvoksende[12]" funksjonalitet. Dette er inspirert fra feltet etologi[13]. Etologer har påpekt at dyrs atferd kun kan forstås, og kun har mening, i forhold til miljøet det lever i og hvor interaksjonsdynamikk mellom komponenter i et sosialt system kan gi opphav til fremvoksende struktur eller funksjonalitet. Maes viser her til Deneouburg [17] [32] som viser hvordan sosiale insekter gjennom å følge enkle regler, kan produsere fremvoksende atferd som å finne stier til matkilder etc. Mange år senere viste Agre [1] at atferd så komplisert som målrettede handlingssekvenser kan modelleres som en fremvoksende egenskap av interaksjonsdynamikken mellom et komplekst miljø og en og en agent. Maes viser også til Matarics "vegg-følgende" robot som ikke har en enkelt komponent som kan sies å være ansvarlig for den "vegg-følgende" egenskapen. I Maes sitt eget arbeide [41] hvor ingen av komponent-modulene er ansvarlig for handlings-utvalg, handlingsutvalget er en fremvoksende atferd ut fra aktiverings/inhiberings dynamikk mellom de primitive komponentene i systemet. Alt dette betyr at den intene kontrollstrukturen i en agent ikke nødvendigvis selv må være komplekse for å produsere kompleks atferd. Det er ofte tilstrekkelig å koble egenskaper i miljøet refleks-mekanismer i agenten hvor agent og miljø gjennom tilbakemeldinger[14] er i stand til å produsere den ønskede atferden. Forståelse av miljø og interaksjon mot dette blir dermed helt sentrale komponenter i forståelse av agentatferd.

Viktige egenskaper ved fremvoksende kompleksitet er at den ofte er robust, fleksibel og mer feil-tolerant enn programmer som er utviklet på trodisjonelt ovenfra-ned vis. Årsaken er at ingen komponenter er sentralt ansvarlig for å produsere kompleksiteten, og ingen av dem i utgangspunktet er mer kritisk enn en annen, så når en av dem bryter sammen, bryter ofte ikke hele systemet sammen, men er i fremdeles i stand til å produsere rimelig atferd. Siden alle

[12] "Emergent".

[13] Webster [75] definerer etologi slik:

1: a branch of knowledge dealing with human ethos and with its formation and evolution.
2: the scientific and objective study of animal behavior especially under natural conditions.

Vi vil i denne oppgaven gå ut fra at tolkning "2" i listen over brukes.

[14] "Feedback".

komponentene interagerer i parallell, er systemet også i stand til å reagere raskere på endringer i miljøet. En typisk ulempe med slik kompleksitet er at den som regel skalerer elendig med problemstørrelse, så selv om en arkitektur fungerer i et begrenset eksempel, vil den som regel feile katastrofalt om man forsøker å plassere den i en kontekst der den må reagere flere størrelsesordner raskere, håndtere mye mer input eller liknende. Skalering av autonom adaptiv agentbasert atferd er fremdeles et forskningsproblem

Når Amandas arkitektur skal beskrives fra et AAA perspektiv, er det naturlig å ta utgangspunkt i hvilke mål, sensorer og aktuatorer som agentene som utgjør Amanda må forholde seg til. Amandas sensorer er rettet dels mot brukerens atferd og dels mot eksterne søkemaskiner. Fra de brukernære agentene som overvåker brukerens atferd, strømmer det en sekvens av observasjoner om hvilke dokumenter brukeren ser på, hvilke tilbakemeldinger brukeren gir om råd som gis og om tilstanden til dokumenter brukeren arbeider på. Aktuatoren som brukes er å gi råd til brukeren. I tillegg finnes det aktuatorer som kan gi søkemotorer instruks om å kjøre søkeuttrykk, og sensorer som kan observere resultatene av slike kjøringer. Noen agenters atferd er hardkodet til å ut fra bestemte observasjoner utføre visse reflekshandlinger, mest typisk i så måte er nok refleksen om å la tilbakemeldinger om forslag føre til at de forslagsgivende agentene får øket sin "godhet", som er et mål for hvor flinke agenene antas å være til å tilfredstille brukerens behov. Agenter kan også be om å få utført oppgaver, som så plukkes opp som mål for andre agenter. Et eksempel er agenter som søker etter dokumenter som er relevante for en bestemt filkatalog, disse vil formulere søk som plasseres som tilbud om oppdrag i en "markesplass" hvor de kan hentes opp av agenter som er i stand til å formulere søk overfor bestemte søkemaskiner. Å utføre et oppdrag på vegne av en annen agent betyr at agenten som utfører oppdraget får tilført noe av oppdragsgiverens "godhet". Hvilke agenter som bidrar til de forskjellige rådene, og i hvilken grad de gjør det lagres, slik at dersom en tilbakemelding gis om et råd kan "godhet" fordeles til alle som har bidratt til å produsere det.

3.2.3 Karakteristiske egenskaper hos autnonome adaptive agenter

Maes trekker så [42, p. 3] frem fem områder der det er klare forskjeller mellom AA (Autonome Agenter) og tradisjonell AI.

1. **Styrke av kompetanse:** I tradisjonell AI har fokuset vært på å demonstrere sterk kompetanse innen et kraftig avgrenset felt, eksempelvis sjakkspill og medisinsk diagnostikk. Til kontrast integrerer en AAA vanligvis mange typer kompetanse. Kompetansene er typisk "lavnivå", som bevegelse, navigasjon, samle objekter o.l. For andre systemer kan det være kompetanse til å by og selge [46] eller utføre en programvare-rutine for en brukergrensesnitt-agent[46].

2. **Åpne v.s. lukkede verdener:** Tradisjonell AI har fokusert på "lukkede" verdener som ikke har noen direkte interaksjon med problemområdet som de koder kunnskap for og løser problemer rundt. Deres forbindelse til miljøet har vært svært kontrollert og indirekte gjennom en menneskelig operatør. I kontrast er en agent er plassert i et rikt miljø og er direkte forbundet til sine problemer gjennom sensorer og aktuatorer. Det kan påvirke eller endre dette domenet gjennom akturatorene. Problemdomenet er typisk svært dynamisk, som igjen betyr at systemet har en begrenset mengde tid til å reagere og at uforutsigbare ting kan skje. Det involverer dessuten typisk også andre agenter (mennesker og/eller andre kunstige agenter).

3. **Sekvensiell problemløsning:** I tradisjonell AI løses typisk ett problem av gangen, ofte er det ikke tidsbegrensninger på løsningen, og systemet trenger ikke bry seg med avbrudd (selv om operatøren kan tenkes å måtte bry seg med slike problemer). I kontrast er en agent er autonom og må selv undersøke miljøet og selv finne ut hva det neste problemet eller målet som skal løses er. Den må løse problemer på en betimelig måte, og typisk må en agent håndtere flere mål som kan stå i innbyrdes konflikt.

4. **Kunnskapsstrukturer:** Tradisjonell AI fokuserer på hvilken kunnskap et system har. AI systemer har deklarative "kunnskapsstrukturer" som modellerer aspekter av domenet som systemets ekspertise skal gjelde. Alle interne strukturer, bortsett fra en interpret, er statiske. Systemet er kun aktivt når et problem blir foreslått av av en menneskelig operatør, og da bruker interpreten statiske kunnskapsstrukturen til å foreslå en løsning til problemet.

   I kontrast til dette fokuserer AAA atferden systemet har når det blir plassert i miljøet. Agentens interne struktur produserer "dynamisk atferd", i kontrast til statisk "kunnskapsstruktur". Agentene må ikke initieres av noe mål gitt av en bruker. Det er mindre viktig at agenten kan svare på spørsmål om sitt problemområde. Det er også mindre viktig at brukeren er i stand til å inspisere de interne strukturene og identifisere hva som er ansvarlig for et bestemt aspekt av den resulterende atferden. For eksempel er det akseptabelt for mål og planer å være fremvoksende observerbare egenskaper[15], som ikke kan attribueres til noen bestemt intern struktur, men som i stedet er resultat av en interaksjon mellom en mengde strukturer og miljøet.

5. **Fokus på utvikling av kunnskapsstrukturer:** I tradisjonell AI er man vanligvis ikke opptatt av hvordan kunnskapsstrukturer utvikles. Strukturene trenger ikke være robuste m.h.p. endrede situasjoner (at ting går i stykker etc.). Innen tradisjonell maskin-læring antas det ofte at en masse bakgrunnsmateriale er tilgjengelig og at denne brukes aktivt av systemet.

[15] "emergent observable properties" i [42].

I kontrast er det innen AAA et sterkt fokus på tilpasning, og på utviklingsstatus. Dette kan ofte bety at systemet forbedrer sin interne representasjon, og dermed sin atferd over tid basert på sin erfaring med det omgivende miljøet. Agenten utforsker aktivt og oppdaterer sine strukturer med en inkrementelle, og induktiv læringsmetode. Dette betyr at designeren tar en mer inkrementell innfallsvinkel til å bygge agenten: Utvikleren utvikler et stadig mer sofistikert system ved å legge mer struktur til et allerede "fungerende" system.

Dette siste er et sterkt element bak måten Amanda er designet på. Jeg vet ikke hva som er de beste måtene å velge ut forslag på, og jeg tviler sterkt på at det noen sinne vil finnes en måte som er "best" for alle mennesker i alle situasjoner. Amanda-arkitekturen er derfor laget for at det skal kunne legges til kompetansemoduler som er i stand til å gi råd i konkurranse med andre moduler, og så skal konkurransen kunne avgjøre hva som til en bruker passer best. For øyeblikket er den eneste brukeren undertegnede, men det er ingen grunn til å tro at et ønske om å kunne utvikle, bytte og prøve forskjellige kompetansemoduler for rådgivning vil være en metode som er mindre nyttig å ha tilgjengelig hvis det blir flere brukere av systemet. Denne erkjennelsen av at så lenge det blir brukt vil Amanda antagelig aldri vil bli noe "ferdig" system er derfor en grunnleggende forutsetning for designet.

Oppgave-orienterte moduler

Agenter betraktes som mengder av kompetanse-moduler (ofte kalt "atferder"). Modulene er ansvarlige for små oppgave-orienterte kompetanser. Hver av modulene er direkte knyttet til sine relevante sensorer og aktuatorer. Moduler kommuniserer mellom hverandre gjennom ekstremt enkle meldinger heller en felles representasjoner av antakelser etc. Kommunikasjonen mellom moduler er nesten alltid av en "kringkastings"-natur men punkt-til-punkt kommunikasjon foregår også. Typiske meldinger handler om aktiverings-energi, enkle aktiverings eller inhiberings-signaler, enkle tokens i begrensede språk. I tillegg kommuniserer moduler "gjennom miljøet" En modul kan påvirke en del av miljøet som kan utløse en annen modul o.s.v.

Oppgave-spesifikke løsninger

Agenter har ikke "generelle" oppgave-uavhengige funksjonelle moduler. Ingen generell persepsjonsmodul, ingen generell planlegger etc. Hver kompetanse modul er ansvarlig for all representasjon, beregning etc. En hindrer-unngåer kan trenge ett bit av informasjon for å representere om et hinder er oppfattet eller ikke innenfor en kritisk avstand. Det kan gjøre noen enkle beregninger for å finne ut hvordan å unngå kollisjon. Kompetanse-moduler er helt selvstendige, sorte bokser. De kan t.o.m. bruke helt andre teknikker (f.eks. annen hardware) for å nå sine mål. En av årsakene til denne innfallsvinkelen er en pessimistisk visjon om at det ikke er mulig å finne noen generell løsning på

"synsproblemet", "planleggingsproblemet" etc. Et syn som også uttrykkes av Minsky [49].

Representasjoners rolle gitt mindre vekt

Liten vekt legges på å modellere miljøet. For det første finnes ingen sentral modell. Det gjøres heller ikke noe forsøk på å integrere informasjonen fra forskjellige sensorer til en enkelt, koherent, objektiv tolkning. De lokaliserte representasjonene for enkelte moduler er ikke relaterte, og kan være inkonsistente eller redundante om man ser dem sammen med representasjoner fra andre moduler.

Desentralisert kontrollstruktur

Alle kompetansemoduler arbeider i parallell. Ingen "kontrollerer" noen andre. Imidlertid finnes en eller annen enkel arbitreringsmetode for å velge en, eller slå sammen multiple aktuatorkommandoer som er i konflikt. Denne arbitreringsmetoden kan være et "winner-take-all-network" [41], eller et hardkodet prioritets-skjema som i [13].

Målrettet atferd er en fremvoksende egenskap

I Agenter er ikke aktivitet modellert som resultatet a en planlagt prosess. Det finnes ingen intern struktur som beskriver systemets "plan". Mange agenter har ikke noen bestemte mål, men er fremdeles drevet mot et bestemt sett av bestemte, innkompilerte mål. I andre arkitekturer har agenten en representasjon av sine mål, som brukes for å modifisere prioritetene mellom de ulike modulene over tid.

En rolle for læring og utvikling

Læring og utvikling ses på som essensielle aspekter ved en autonom agent [76]. Å bygge et adaptivt system som utvikler seg fra et ikke fullt så vellykket system anses som en bedre innfallsvinkel ennå bygge et suksessfullt system som ikke endrer seg når miljøet rundt det gjør det. Læringsalgoritmene implementeres på distribuert vis: Typisk finnes en lignende læringsalgoritme i forskjellige kompetansemoduler. Relatert til læring er redundans: Systemet har ofte flere moduler for en bestemt kompetanse. Erfaring brukes for å finne ut av hvilken av disse modulene som implementerer kompetansen på den mest pålitelige måten, og som derfor bør foretrekkes.

Systemer basert på disse prinsippene har ofte adaptiv robust atferd.

- **De reagerer raskt fordi:**

    1. De har få lag med informasjonsprosessering.
    2. De er mer distribuerte og ofte ikke synkroniserte.

3. De krever mindre kostbare beregninger, og er ikke så utsatt for kombinatoriske eksplosjoner siden de ikke søker så mye.

- **De er robuste fordi:**

1. Ingen av modulene er mer kritiske enn andre.
2. De forsøker ikke å fullt forstå den foreliggende situasjonen (noe som ofte krever mye tid og kan være problematisk uansett).
3. De bruker redundante metoder.
4. De tilpasser seg over tid.

3.2.4 Problemer alle agenter må løse

Hovedproblemene som løses innen AAA er å finne frem til en arkitekturer som gjør at agenten er i stand til å demonstrere adaptiv, robust, og effektiv atferd. Robust betyr her at agenten aldri bryter helt sammen (den demonstrerer gradvis ytelsesreduksjon[16], når komponenter feiler eller noe uforutsett skjer). Effektiv betyr at agenten er i stand til å før eller senere nå sine mål. Spesifikt er det to delproblemer som må løses:

Problemet med handlings-valg: Hvordan kan en agent avgjøre hva den skal gjøre som sitt neste trekk slik at den kan komme seg nærmere mot sine mange tids-varierende mål? Hvordan kan den håndtere muligheter og problemer som oppstår? Hvordan arbitrere mellom mål i innbyrdes konflikt? Hvordan håndtere støy fra sensorer og aktuatorer? Hvordan reagere betimelig?

Problemet med å lære fra erfaring: Hvordan kan en forbedre sin ytelse over tid? Hvordan skal den velge mellom å "utnytte" sine beste handlinger, versus å "utforske" andre muligheter, slik at den kan oppdage bedre måter å nå sine mål på? Hvordan kan den bruke tilbakemeldinger fra verden inn i sine interne atferd-produserende strukturer? Hvordan kan den korrigere "gale" eller ineffektive atferd-produserende strukturer etc.

Maes kaller totaliteten av prinsipper, en organisering og en mengde verktøy, algoritmer og teknikker som støtter disse, for en "arkitektur" for modellering av autonome agenter. Hun hevder så at det nylig har blitt klart at ingen av de arkitekturene som har blitt foreslått er optimale på alle vis. Forskningens mål har derfor blitt å utvikle en forståelse for hvilke arkitekturer som er de enkleste løsningene for en gitt klasse agent-problemer.

Noen eksempler er:

[16] "graceful degradation" i [42].

En mobil robot: Moduler: Gjenkjenne og gå gjennom dører, følge vegger, unngå hindre (gjerne noen redundante), o.s.v. Alle modulene virker i parallell. Et enkelt arbitreringssskjema for å velge mellom moduler: Prioritere å unngå hindringer fremfor å gå gjennom dører, prioritere å gå gjennom dører fremfor å følge vegger. En slik robot planlegger ikke sine handlinger, men fra en observatørs synspunkt, vil det se ut som om roboten opererer på en systematisk, rasjonell måte [13][14].

En interface-agent: Moduler som hver samler inn inn informasjon ved å observere brukeren og ved å føre statistikk om et bestemt aspekt av brukerens atferd. F.eks. kan en modul holde orden på hvilke situasjoner en bruker utfører en utskrift-kommando. Dersom det finnes flere eksperter for flere kommandoer, vil hver av disse vite når de skal bli aktive for å assistere brukeren. Fra en observatørs synspunkt vil det se ut som om systemet "forstår" brukerens intensjoner, som om den vet hvilke oppgaver dokmentproduksjon innebærer. Likevel, handlingssekensen er kun en fremvoksende egenskap av et distribuert system. Systemet vil raskt tilpasse seg endrede brukervaner, vil reagere raskt, vil ikke ha så stor mulighet for å bryte fullstendig sammen etc.

Et prosess-skedulerings system: Oppgaven er å allokere prosesser til prosessorer i sanntid. En løsning foreslått av Malone [74] i hans "Enterprise" system fungerer slik: Systemet er basert på en markeds-metafor. En maskin som produserer en ny prosess, sender ut en "anbudsforespørsel" for oppgaven. Andre maskiner kan svare på forespørselen ved å gi estimater på når de kan bli ferdig, tider som reflekterer deres hastighet, og hvilke andre oppgaver de arbeider med. Maskinen som sendte ut forespørselen vil samle inn budene og sende jobben til den maskinen som hadde det beste tilbudet. Systemet er robust siden ingen maskiner er mer kritiske enn andre (det er ingen sentral scheduler). Maskiner kan bli gjort utilgjengelige under kjøretid, og systemet reagerer godt på denne nye situasjonen ("godt" i den forstand at systemet er i stand til å tilfredsstille sitt overordnede mål som er å få utført oppgaver på de tilgjengelige ressursene). Løsningen er er enkel, og likevel fleksibel m.h.p. hvilke faktorer den tar hensyn til.

3.2.5 Nødvendig funksjonalitet i et agentsystem

Evne til å velge neste handling

> *Problemformulering:* Gitt en agent som har flere tidsvarierende mål, et repertoar av handlinger som kan utføres, og bestemte sensordata, hvilke handlinger skal denne agenten ta for å optimere oppnåelsen av sine mål?

I teorien er det mulig å beregne den optimale utvalgs-policy for en agent som har et bestemt sett av mål og som lever i et deterministisk eller probabilis-

tisk miljø [72]. Dette er umulig[17] å få til for de fleste ekte agenter, siden disse må håndtere begrensinger i ressurser (tid, beregninger, minne), ukomplett eller feilaktig informasjon, dynamisk ikke-deterministisk ikke-probabilistisk miljø, tidsvarierende mål og ukjente og muligens forandringer i sannsynlighetsfordelinger o.s.v.

Noen mål er eksplisitte, andre er implisitte. Implisitte mål er nødvendigvis fastlåste, og kan ikke endres om ikke agenten omprogrammeres. Andre agenter kan ha eksplisitte mål som endres over tid og som har forskjellig intensitet (i motsetning til "av/på").

Siden det er teoretisk svært vanskelig å vise hvilken handlings-sekvens som er optimal for en agent, hvordan evaluerer man da innen feltet en foreslått løsning? Forskere innen AAA er ikke så intereserte i bevisbar optimalitet, som de er interesserte i å se om valget er robust, adaptivt, og om agenten når sine mål innen miljøet og begrensningen som er gitt for den bestemte oppgaven. Blant de tingene som mekanismen som skal velge handlinger bør gjøre er:

- Favorisere handlinger som bidrar til målene. Spesielt bør den favorisere de handlingene som bidrar mest.
- Skal være i stand til å håndtere muligheter og hindre fleksibelt.
- Skal være sanntid (rask nok for det aktuelle miljøet og endringsraten innen dette miljøet).
- Minimere unødvendig bytting frem og tilbake mellom handlinger som bidrar til bestemte mål
- Forbedre seg over tid.
- Demonstrere gradvis ytelsesreduksjon når komponenter bryter sammen.
- Aldri bli låst i en løkke eller deadlock-situasjon, eller få agenten til å blindt forfølge et uoppnåelig mål.
- Viktigst av alt: Være "god nok" for miljøet og oppgaven som skal løses. Så lenge agenten klarer å løse sine mål innen begrensningens som er gitt (tid, kvalitet etc.) antas løsningen å være robust.

For Amandas vedkommende er det spesielt det første av disse hensynene som er problematisk. Hva i all verden er det som bidrar til å hjelpe brukeren best? Vi *vet* at det Amanda er i stand til å sanse er et veldig begrenset utsnitt av hva brukeren faktisk gjør og føler, og antagelig kan det være mye å hente på å utvide spekteret av data som brukes for å kalibrere Amandas oppfatning av hva som er nyttig atferd.

Maes diskuterer flere synspunkter på hvordan man skal evaluere handlingsevaluerings metoder. Diskusjonen understreker hennes poeng om at det ikke er noen konsensus om hvilke metoder som er gode, men også et poeng om at det

[17] i alle fall i praksis, hittil.

er vanskelig å sammenligne handlings-utvalgs metoder. Hun referer til arbeid underveis av Wilson [70] og Littman [38] for å systematisere slike beskrivelser, og gjennom dette oppnå en bedre basis for sammenligning.

Dessverre er det også mange og alvorlige problemer som gjenstår: Svært lite forskning er gjort om naturen av mål, og intensjoner mellom mål. Studier er nødvendig for å finne ut hva slags mål arkitekturene skal støtte, hvor målene kan komme fra, hvordan de endrer seg over tid etc. [69] gir en oppsummering av hva etologi og psykologi har å si om modeller for motivasjon.

Oppskalering til større problemer, er som regel en "komplett katastrofe". Spesielt gjelder dette håndbygde arbitreringsnettverk. Den mest opplagte løsningen er å enten bruke evolusjon [12][18], eller å lære og tilpasse nettverket [43]. Få eksperimenter er gjort langs disse linjene. Videre er det gjort lite forskning er gjort på det å la deler av agenter være gjenbrukbare av andre agenter. Det kunne vært nyttig å la partielle løsninger som har vist seg å virke i en agent kunne bli abstrahert og brukt i en annen agent. F.eks. kunne det vært nyttig om moduler som klarer å følge vegger i en robot kunne abstraheres og brukes i en annen robot med lignende sensorer og miljø. Dynamikken i interaksjon mellom agenter og miljøet og mellom de forskjellige modulene av en agent er dårlig forstått. Noe arbeid er gjort for å forstå dem ut fra et "dynamisk system" - perspektiv, men mye arbeid gjenstår. De færreste av de foreslåtte arkitekturene bryr seg med problemet med å slå sammen kommandoer[19]. Typisk er det kun en modul av gangen som bestemmer hva som blir sendt til en aktuator. Det er ingen generell måte å kombinere output fra flere moduler på, selv om noen spesifikke forslag er gitt [58] [11]. Alle arkitekturene Maes beskriver er at de er fullstendigb desentraliserte og har ikke noen sentral tilstand. En konsekvens er at de kan bli fanget i løkker og deadlock situasjoner ("B-brain" problemet i Minskys terminologi [49]). Til slutt nevnes det at de fleste av arkitekturene ser svært sneversynt på relasjonen mellom persepsjon og handling. Få arkitekturer støtter aktiv eller mål-drevet persepsjon, tar initiativ til egne handling for å få inn mer sensordata etc.

Konsekvensen vi trekker av dette, er å begrense omfanget av oppgaven som skal løses av Amandas agenter. Valget om å bruke eksterne søkemotorer er f.eks. en konsekvens av dette: Det er fullstendig urealistisk å tro at det skal være mulig å la en agentarkitektur raskt skal kunne gjennomføre effektive søk i store dokumentmengder. Å formulere effektive søk er et delproblem som er mer enn stort nok å delegere til agenter.

Evne til å lære fra erfaring

Arkitekturene som fokuserer kun handlingsvalg neglisjerer det å lære fra erfaring. Dette betyr at agentene kun er adaptive i svært begrenset grad: De er i stand til å utnytte uventede situasjoner (muligheter, problemer), men de lærer ikke fra tilbakemeldinger fra miljøet. De blir ikke bedre til å løse sine mål ut fra

[18] Det er dette Amanda gjør.
[19] "Command fusion".

erfaring.

En kategori er foreslått som kan gjøre at handlingsutvalget forbedres over tid. Læring fra er faring er nødvendig for enhver agent som skal oppvise robust, autonom atferd over lange tidsperioder: For det første er det veldig vanskelig å programmere en agent. Det har gjennom praksis blitt vist at det er umulig å korrekt håndkode en kompleks agent eller komme opp med en korrekt spesifikasjon av dens atferd og miljø. Komponenter som bryter sammen kan kreve "reprogrammering" under kjøretid. Adaptiv atferd kan ikke bli sett på som et endelig, statisk punkt. Sann adaptiv atferd er i sin essens dynamisk og kontinuerlig. Problemet med å lære fra erfaring kan derfor formuleres slik:

> *Problemformulering: Gitt en agent med (i) et sett av handlings eller kompetansemoduler (ii) visse sensordata og (iii) multiple tidsvarierende mål, hvordan kan agenten forbedre sitt handlingsvalg over tid basert på erfaring?*

For "sluttmål" eller "attainment mål" vil dette bety at den gjennomsnittlige tiden eller det gjennomsnittlige antall handlinger (eller et hvilket som helst annet mål for kost), for å oppnå målet minsker over tid. For Amandas del vil dette f.eks. kunne bety at det går kortere tid fra en situasjon endrer seg for brukeren, til brukeren er i stand til å direkte benytte forslagene Amanda gir.

Enhver modell for læring må i følge Maes tilfredsstille denne listen av ønskede egenskaper:

- Læringen må være inkrementell. Agenten skal kunne lære av enhver erfaring. Det kan ikke være separatre lærings og "utøvende" faser.
- Læring skal ha en vekting[20] i retning av å lære forhold som er relevant overfor målene. I komplekse, realistiske miljøer kan ikke en agent tillate seg å lære alt det som faktisk kan læres.
- Læringsmodellen bør være i stand til å håndtre støy, probabilistisk miljøer, feil i sensorer etc.
- Læringen bør være selvstyrt[21], agenten må være i stand til å lære autonomt.
- Helst bør læringsmodellen gjøre det mulig å gi agenten noe innebygd kunnskap, slik at den ikke trenger å lære alt fra bunnen, spesielt i situasjoner der kunnskap fra tidligere situasjoner er enkelt tilgjengelig.

Det er tre delproblemer som må løses når man lager en modell for agentlæring:

1. Hvilken seleksjonsmekanisme brukes?

[20] "bias"
[21] "unsupervised"

2. Hvordan lærer systemet? Hvordan lager det "hypoteser" som skal testes, og hvordan velger det hvilke av disse hypotesene som er verd å beholde og eller bruke til å bestemme agentens atferd?

3. Hvordan velger agenten når den skal "utforske" og når den skal "utnytte"? Hvordan velger den om den skal aktivere hva den tror er den mest optimale handlingen for den foreliggende situasjon versus å forsøke en suboptimal handling for å lære og muligens finne en bedre måte å gjøre ting? M.a.o: Hva er en god eksperimenteringsstrategi for en agent.

Det første problemet tilpasses ofte på en naiv og begrenset måte. Mengden av målene som håndteres er svært enkel og er fastlåst over tid. Mer detaljerte eksempler kommer opp når man løser problemen over. F.eks. hver læringsarkitektur må løse "credit assignment" problemet. Hvilken av de tidligere aktiverte handlingene får (delvis) "credit" for visse (ønskelige/uønskede) handlinger som skjer?

Hvordan evaluere forslag til løsning? Problemet er underspesifisert der som man ikke tar hensyn til miljøet, oppgaven og karakteristika av agentene. F.eks. kan en agent med mye minne gjøre lurt i å bruke en minne-intensiv læringsmetode fremfor å gjøre mye generalisering for å komme opp med en konsis representasjon. I noen miljøer er initiell kunnskap enkelt tilgjengelig, som gjør det fordelaktig å gjøre agenten delvis programmerbar (i motsetning til at den må lære alt fra begynnelsen). Hvilket krav som stilles til læring påvirkes også av hvilke krav miljøet stiller: Raskt endrende miljø krever mer aktiv læring enn et statisk miljø.

Maes grupperer arkitekturene for læring inni tre klasser: De forskjelige arkitekturene som er foreslått kan grupperes i tre klasser: "Forsterknings-læring"[22], "klassifikatorsystemer" og modellbyggere.

"Forsterknings-læring"[23]: Ideen er her:

> *Gitt en agent med (i) et sett av handlinger det kan engasjere seg i, (ii) et sett av situasjoner det kan befinne seg i, og (iii) et skalart belønningssignal som mottas når agenten gjør noe; lær en handlings-policy, eller en avbildning fra situasjoner til handlinger, slik at agenten som følger policyen optimerer den kumulerte diskonterte belønningen den mottar over tid.*
>
> I slike algoritmer forsøker agenten å lære for hvert situasjon/handlingspar hva verdien av å ta en handling i denne situasjonen er. Ettersom algoritmen lærer, skapes en todimensjonal matrise av elle mulige kombinasjoner av en situasjon og en handling. Målet for systemet er å oppdatere disse verdiene slik at de konvergerer mot en "maksimum kumulert diskontert belønning". Dette betyr at den belønning systemet får med en gang, sammen med den belønning det får senere regne sammen. Videre er belønninger "diskontert", slik at belønning som gis kort tid inn i fremtiden teller mer enn belønning som gis lenger inn i fremtiden.
>
> Delproblemene som løses er:

[22] "Reinforcement learning"

1. *Handlings-seleksjon:* På et hvert tidspunkt er agenten i en eller annen bestemt situasjon. Gitt denne situasjonen velger den den handlingen som maksimerer verdien (maks kumulert diskontert verdi).
2. *Læringsmetode:* Når en agent utfører en handling kan den motta en belønning (muligens null). Den oppdaterer verdien av situasjon/handling paret det akkurat utnyttet, og øker eller minsker verdien av dette paret til å bedre reflektere den belønningen som ble mottatt.
3. *Utforskings-strategi:* En viss prosentdel av situasjonene velger ikke agenten den handlingen som optimerer belønning, men utfører i stedet en tilfeldig handling, for å samle inn data om andre interessante alternativer.

En attraktiv egenskap ved forsterkningslæring er at dens formelle underlag. Det kan vises at under bestemte betingelser (markovske miljø, uendelig antall prøver), vil agenten konvergere mot en optimal utvalgsstrategi. Dessverre er disse betingelsene sjeldent til stede i ekte, komplekse situasjoner.

Ulemper med forsterkningslæring er de ikke de håndterer ikke tidsvarierende mål. Dersom målene endres må alt læres fra bunnen av. For realistiske applikasjoner er tilstandsrommet så stort at læring tar for mye tid til å være praktisk. Læring skjer kun i "kanten" av tilstandsrommet (kun når en belønning mottas kan systemet begynne å lære om situasjoner som leder til den belønningen), så det tar lang tid å lære lange handlingssekvenser. Modellen antar at agenten til enhver tid vet hvilken situasjon den er i. Det er vanskelig å bygge inn initiell kunnskap i denne typen modell. Modellen kan ikke lære når flere handlinger tas i parallell.

Klassifikator systemer: Klassifikasjonssystemer kan betraktes som et spesialtilfelle av forsterknings-lærings systemer siden agentene forsøker å lære hvordan de kan optimere belønningen de får ut fra handlinger i situasjoner. Ideen er at agenten har regler, kalt "klassifikatorer", og data om hver regels ytelse. På det aller minste holder systemet orden på en verdi som sier "god" regelen er. Delproblemene for en læringsarkitektur håndteres slik:

1. *Handlingsvalg:* Gitt en situasjon beskrevet av sensordata, og intern tilstand herunder tilstanden til klassifikatorene velges klassifikatorer som kan kan matches mot den gjeldende situasjonen. Av alle de matchende klassifikatoren, velges en eller klassifikatorer basert på deres "godhet". Handlingene foreslått av disse klassifikatorene utføres så.
2. *Læringsmetode:* Når en klassifikator kjøres gir de noe av styrken til klassifikatorene som "setter scenen", d.v.s klassifikatorene som var aktive i forrige tidssteg. Dette kalles "bøtte-brigade" -algoritmen[24]

[24] "Bucket-brigade algorithm"[42, s. 26]

og brukes for å løse problemet med credit-fordeling: Når agenten utfører en handling, vil den motta noe belønning. Dersom dette er tilfelle, vil belønningen øke styrken på alle klassifikatorer som akkurat har vært aktivert (sammen med dem som ikke ble aktivert, men som foreslo den samme handlingen). Dette skjemaet garanterer at klassifikatorer som bidrar til en belønning blir belønnet og over tid vil ha høyere styrker enn dem som ikke bidrar, og som et resultat vil bli aktivert oftere.

3. *Utforsknings-strategi:* Antall klassifikatorer er fast. En gang i mellom fjerner agenten de klassifikatorne som har lave styrker og bytter dem ut med mutasjoner og rekomobinasjoner (crossover) av suksessfulle klassifikatorer. På denne måten vil agenten fortsette å utforske og evaluere forskjellige måter å gjøre ting på, samtidig som "gode" løsninger holdes tilgjengelige.

Et interessant aspekt er klassifikator-agenten eksperimenterer på: Den underliggende hypotesen er at man kan finne en bedre løsning til et problem ved å gjøre små endringer på en eksisterende god løsning eller ved å kombinere eksisterende løsninger. En annen fordel av klassifikatorsystemer har er at de kan generalisere over situasjoner såvel som handlinger gjennom "Ikke bry deg" symbol i uttrykkene sine. Dette gjør det mulig for klassifikasjonssystemer å bruk tilstandsrommet på forskjellige abstraksjonsnivåer for å finne den mest abstrakte representasjonen av en klassifikator som er nyttig for et bestemt problem. Klassifikatorsystemer deler mange av problemen til forsterkningslærings agenter. De har problemer med tidsvarierende mål, og problemet med å lære på kanten av tilstandsrommet. I tillegg har de det problemet at de ikke holder styr på alt de har prøvd, så et klassifikatorsystem kan finne på å re-evaluere om og om igjen en klassifikator som den finner å ha for lav styrke, dette fordi den ikke har noen hukommelse for hva som er prøvd.

Amanda er en slags klassifikator-lærer. Agentene som utfører handlinger kan betraktes som "regler" for hvordan handlinger skal utføres, og belønninger fordeles delvis gjennom et "bucket brigade" liknende skjema.

Modellbyggere: En siste klasse agenter som lærer fra erfaring lærer kausale modeller heller enn "policy avbildninger". Agenten bygger opp en probabilistisk modell av hva effektene er av å ta en handling i en bestemt situasjon. Denne kausale modellen kan så brukes av en arbitreringsprosess for å velge hvilken handling som er den mest relevante gitt en bestemt situasjon og en bestemt mengde mål. Handlingsvalg og læring er mye mer frikoblet[25]. Faktisk kan læringsalgoritmen fra en av disse agentene kobles sammen med en annen handlingsvalgsmekanisme.

I de fleste av disse arkitekturene lærer ikke agenten en fullstendig avbildning. I stedet lærer agenten avbildninger fra partielle situasjoner til

[25] "decoupled"

partielle situasjoner. Den avbilder delene av en situasjon som "betyr noe" kombinert med en handling, til en ny resulterende situasjon som "betyr noe". En slik kombinasjon av (i) et sett av kondisjoner (ii) et primitivt (eller kompositt) handing og (iii) en mengde av ventede reslutater (og sannsynligheten til disse) kalles et *skjema* [45].

Modellbyggere lærer med de tre delproblemene slik:

- *Handlingsvalg:* Agenter kan håndtere tidsvarierende, multiple, eksplisitte mål. Gitt en mengde mål og intensiteter beregnes det i runtime hvilke av de lærte skjemaene som er mest relevante for å oppnå målet, såvel som det mest pålitelige. Det er en egen tilordningsprosess for verdi som er frikoblet fra læringsprosessen. Ofte kan denne verditilordningen være slik at den favoriserer moduler som er mer pålitelige. Den kan til og og med gjøre avveininger mellom pålitelig av en handligssekvens og lengden av sekvensen som leder til målene. Typisk vil en "spredende" aktiveringsprosess [45] eller en enkel markerpropagering [18] tilordne disse verdiene gitt noen mål og sensordata.
- *Læringsmetode:* Når en bestemt handling tas, sanser agenten hva som skjer i omgivelsen. Den bruker dette til å lære korrelasjoner mellom bestemte tilstand/handling par og resultater. Etter at en handling tas, vil alle resultatlistene (og deres sannsynligheter) oppdateres til å reflektere det nye eksempelet. Av og til spinner agenten av nye skjemaer fra eksisterende for å kunne representere resultater som er upålitelige eller i konflikt.
- *Utforsknings-strategi:* Utforsknings-strategien for disse arkitekturene varierer, fra rent tilfeldige eksperimenter, til mye raskere teknikker som bruker fokuseringsteknikker. I Maes egen arkitektur [45] er utforskningsteknikken mål-orientert. Agenten gir prioritet til sin eksperimentering på handlinger som virker lovende m.h.p. å nå målet. I det samme systemet er andelen av eksplorasjon v.s. utnytting en fremvoksende egenskap ved systemet. Jo mer som skal læres jo færre eksperimenter utføres.

En av fordelene med modell-lærere er at de kan overføre kunnskap lært i en kontekst til en annen kontekst (f.eks. et annet mål). De lærer fra enhver handling, dette i motsetning til å bare å lære fra handlinger som har vist seg å være direkte relatert til det aktive målet. En konsekvens av dette er at agentene ikke bare lærer i "utkanten" av tilstandsrommet. I tillegg er det mye enklere for en designer å legge inn bakgrunnskunnskap om domenet. Agenten vil være i stand til å korrigere denne kunnskapen om den skulle vise seg å være ukorrekt. Ulempen er at denne type arkitektur tar lenger tid for å velge en handling, siden det ikke er noen direkte avbildning fra situasjoner til "optimale" handlinger.

> Det sies i seksjon 6.4 i kapitlet om videre arbeid litt om hvordan man kunne gått frem for å modellere kontekst og bygget en læringsarkitektur for Amanda basert på dette.

Som for modellene for handlingsvalg, er det mange uløste problemer forbundet med det å lære av erfaring. For alle innfallsvinklene nevnt over gjelder at den beregningsmessig kompleksitet er så høy at komplett skalering til problemer av realistisk størrelse er problematisk med "komplett" menes her at problemet løses fullstendig innenfor en ramme av agenter. I Amanda har vi f.eks. valgt å gå utenfor denne rammen alle steder vi ikke tror at den tilpasningsdyktigheten agentene tilbyr oppveier effektivitetstapet bruk av dem vil medføre. Det kanoniske eksempelet på dette er å sende søking til eksterne søkemaskiner i stedet for å forsøke dette selv.

En av årsakene effektivitetsproblemene er at veldig få algoritmer har interessante "oppmerksomhets"-mekanismer. De fleste algoritmer bruker kun en heuristikk om at tidsmessig nærhet antas å indikere kausalitet.

En annen årsak er at de fleste algoritmene er dårlige til å generalisere over sensordata. For det første er sensordataene kun representert på ett nivå av granularitet, i motsetning til grovere og finere nivåer. Videre utnytter ingen av de foreslåtte algoritmene struktur og likheter som er til stede i sensordata.

Mer arbeid kan gjøres med strategier for å utforske domener. De fleste algoritmene bruker den enkleste mulige strategien: Agentene eksperimenterer en viss andel av tiden, uansett hvor viktig motiver eller behov er, og uansett hvor interessante muligehetene som er tilstede er. Agenten velger også hvilket eksperiment som skal utføres på en tilfeldig måte, i motsetning til å bruke en heuristikk slik som (i) velge handlinger som ikke har vært prøvd på en stund (ii) prøve handlinger som har vært lovende nylig etc.

Det er en mangel på interessante modeller om hvordan læring og persepsjon interagerer. Persepsjonsmodellen som brukes er snever. Sensorddata som agenten forsøker å korreleere med antas uten videre. Systemet kobler ikke mellom læring om handling med læring om persepsjon. Det lærer ikke at det kan gi oppmerksomhet flere eller færre egenskaper. Ideelt burde en agent genere nye egenskaper og kategorier for å oppfatte miljøet sitt basert på hvilke kategorier miljøet krever.

Det er ikke gode modeller for handlingsvalg og læring interagerer. Spesifikt antar modellene at alle primitive handlinger er gitt. Det kunne vært nyttig om primitivene ble lært på basis av hva som er rimelig for miljøet og målene man har for hånden

Vi trenger å forstå bedre hvilken rolle læring har og hvordan det interreagerer med andre fenomener som kulturell læring og adopsjon gjennom evolusjon.

De fleste innfallsvinklene er hentet fra behaviorisme og sammenlignende psykologi, heller en etologi. Mye kunne læres ved å ta bruke en mer etologisk inspirert innfallsvinkel. For eksempel, etologer har vist at dyr har innebygde perioder hvor de er sensitive for å lære bestemte kompetanser. Disse perioden pleier å falle sammen med situasjoner i livet som er optimale for å plukke opp

de kompetansene som skal læres, og på det vise redusere læringsoppgavens kompleksitet.

3.2.6 Oppsummering om autonome agenter

Feltet er foreløpig i et tidlig stadium. Mange demonstrasjoner[26] har vært gjort. Oppmuntrende suksess har vært høstet i områder som tidligere ikke har vært løsbare, eller som kun har vært løsbart på vanskeligere vis. Likevel er det mange problemer som gjenstår.

- Verktøy og teknikker som er foreslått gir ikke tilstrekkelig støtte for å design eller håndbygge komplekse agenter med mange forskjellige mål.
- Læringsteknikkene har beregningsmessige kompleksiteter som gjør dem ubrukelige, for realistiske tidsbegrensninger.
- Vi trenger å forstå klassen av problemer som agenter kan arbeide med, slik at det blir mulig å kritisk sammenligne bestemte arkitekturer og forslag. Det har f.eks. blitt foreslått mange modeller for handlingsvalg, men før vi forstår problemet med handlingsvalg bedre er det vanskelig å vurdere de forskjellige forslagene.
- Flere underliggende prinsipper trenger bedre forståelse: Spesielt må vi forstå og kjenne grensene til fremvoksende atferd. Hvordan globalt ønskelige strukturer eller funksjonalitet bli designet på basis av interaksjoner mellom mange enkle moduler? Under hvilke betingelser er de fremvoksende strukturene stabile, o.s.v.? Noen første steg er gjort, men disse er ikke anvendelige på systemer utover lekeproblem-størrelse.
- Det er en spenning i agent-innfallsvinklen som er uløst. Man har valgt veldig oppgave-orienterte, pragmatiske løsninger. Som et resultat av dette virker agentene mer som en "pose med hacks og triks" enn mengde av mer generelle lover og prinsipper. Betyr dette at feltet vil bevege seg mot en system-engineering disiplin, eller finnes det en sti mot å bli en mer vitenskaplig disiplin? I sin artikkel konkluderer Maes ved å la dette spørsmålet henge i luften.

[26] "proof of concepts".

Kapittel 4

Tidligere arbeid

I dette kapitlet skal vi se på noe arbeider som er relevante for vår problemstilling. Vi vil for alle arbeidene som er både presentere dem ut fra sine egne forutsetninger. Vi vil så beskrive både hvilke elementer som er overført til Amanda, og hvordan Amanda kunne vært brukt enten til å lage systemer med tilsvarende atferd, eller hvordan systemet på annet vis kunne blitt integrert mot Amanda.

Arbeidene vi har valgt å konsentrere oss om er:

"Amalthea" [2]: Et system som foreslår dokumenter som en bruker kan lese. Systemet bruker eksplisitte tilbakemeldinger fra brukeren for å bygge en modell av hva brukerne finner interessant.

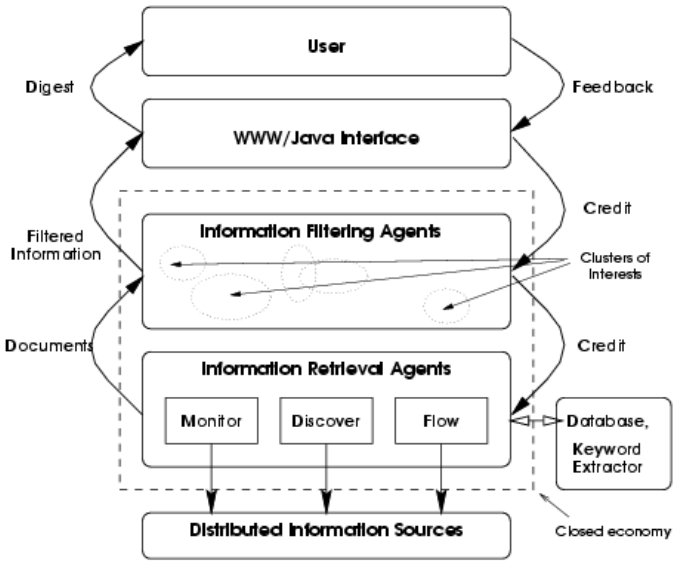
"Inquirus" [25]: Et system som tar eksplisitte søkeuttrykk fra en bruker (på samme måte som en tradisjonell søkemaskin), men så brukers heuristikker både for å reformulere søket, velge hvilken søkemaskin som skal ha søket, og gjør egen relevansranking av resultatene som leveres tilbake .

"Remembrence agent" [56]: Et system som "titter over skulderen" for å se hva brukeren arbeider med, f.eks. i et tekstredigeringsprogram. RA en mengde informasjonskilder som den kontinuerlig forsøker å finne beste match i ut fra det den vet om hva brukeren holder på med der og da .

"Everywhere messaging" [16]: Et system som tar for seg en løsning på problemet "hvordan gi meldinger til brukere på en slik måte at de alltid får med seg alt som er viktig, men aldri blir forstyrret".

"Fast Data Search": En kommersiell søkemaskin laget for å søke i store mengder dokumenter som er lagret i mange forskjellige formater.

"Semantic web": Et initiativ for å standardisere semantiske beskrivelser av innhold i websider.



Figur 4.1: Amaltheas overordnede arkitektur

De tre første arbeidene i listen beskriver fenomener som ligner på dem vi er interesserte i, men i liten "akademisk" skala (forsåvidt i likhet med vårt system). De to siste arbeidene (Fast Data Search" og "Semantic Web") og beskriver systemer som tar mål av seg å kunne gjennomføre søk gjennom alle dokumenter i hele verden, eller i alle fall dem som er lagret i www. Disse to systemene er derfor kilder til inspirasjon for oss som ønsker å lage skalerbare søkesystemer.

4.1 Amalthea

"Amalthea" [2] er et system utviklet av Alexandros Moukas ved MITs Media laboratorium. Systemet søker etter interessant tekst gjennom søkemaskiner, og ved å lytte på nyhetsstrømmer (f.eks. NNTP[34]). Søkingen realiseres gjennom populasjoner av to slags agenter, en for innsamling og en annen for utvalg av tekster. Resultatet blir presentert i et Java/HTML brukergrensesnitt i en browser hvor brukeren kan gi tilbakemelding (i en skala en til fem) om hun synes det som ble presentert var relevant eller ei.

Innhentings-agentene deles igjen i tre kategorier: Overvåkning ("monitor") som sjekker bestemte websider (i essens er dette en liten "crawler"), oppdagelse

("discover"[1] som spør mot søkemotorer og "flow" som lytter på strøm av meldinger (se figur 4.1 for detaljer).

Når dokumentene er hentet inn; vil innhentings-agentene preprosessere dem med en tjeneste som plukker ut nøkklord og forkorter ord til ordstammer ved å fjerne endelser fra ord (en prosess kalt "stemming"). Når dette er gjort transformeres dokumentene til de til en "nøkkelord-vektor". Dette er vektorer der indeksene er ord, og verdiene er tall som i denne vektoren representerer ordets frekvens.

Tilstanden i hver av filter-agentene er en slik vektor, samt et tall som sier hvor "flink" agenten har vært (og derfor antas å være). For å finne ut om agenten tror den er relevant overfor et innkommende dokument, regner den ut cosinus av vinkelen mellom agentens representative vektor, og det innkomne dokumentets vektor, ser om cosinus er over en viss terskelverdi, og om den er det, anbefaler agenten fremvisning av dokumentet. Avhengig av hvor mye over terskelverdien cosinus faktisk er, er tilordner agenten en verdi som beskriver tiltroen den har for denne anbefalingen. Når brukeren så gir tilbakemelding, vil en god tilbakemelding gjøre at agenten blir klassifisert som "flinkere" og en dårlig tilbakemelding vil gjøre den mindre flink. Nøyaktig hvor mye flinkere eller dårligere er også avhengig av hvilken tiltro agenten i utgangspunktet hadde til anbefalingen.

Filter-agentene poster sine forespørsler til hele populasjonen av innhenter-agenter, som så forsøker å hente ut de relevante sidene ut fra ressursene de har fått tilordnet seg (søke-motorer etc.). Innhentings-agenter får sine tilbakemeldinger via de filter-agentene som har produsert anbefalinger, så dersom en innhentings-agent bare leverer resultater til filter-agenter som produserer dårlige resultater, da blir også innhentings-agenten rangert som dårlig. Innhentings-agentene lager seg derfor en liste over hvilke filter-agenter de får best ratinger fra, og vil til en viss grad prioritere å svare på forespørsler fra disse.

Filter-agentene er implementert som "genetiske algoritmer" (se f.eks. [4] for en fin introduksjon) der den representative vektoren er "genomet". Nye agenter kan derfor skapes ved å la flinke agenter"parre" seg med hverandre, d.v.s. at de blander sammen sine representative vektorer på bestemte måter. Agenter med dårlig fitness ikke lov til å parre seg, og til slutt dør de. Hvor mye variasjon som skal påføres agent-populasjonen styres av noen parametere for "total godhet", så dersom resultatene som leveres er i det store og hele dårlige blir det mye parring og død på agentene, og dersom de er gode parrer agentene seg lite. Ytterligere noen tilpasninger er gjort for å sørge for at agenter som aldri leverer resultater til slutt ender opp med å bli betraktet som udyktige, ellers er det hele ganske rett-frem genetisk programmering.

Systemet bootstrappes med å sette inn to hundre filter-agenter og oppdager-agenter. Filter-agentene ble bootstrappet med dokumenter fra brukernes web-browser hotlist, og oppdager-agentene ble laget ved å tilfeldig tilordne forskjellige søkemaskiner til dem fra en forhåndsdefinert liste.

[1] Det er ingen sammenheng mellom Amaltheas "discover" komponent og Amandas "Discovery" komponent utover at navnene ligner.

Som et mål på hvor bra systemet virket målte Moukas systemet totale godhet (fitness), og produserte ut fra målingene diagrammer som viste at fitness gikk jevnt oppover mot noen grenser ved konstante interesser hos brukeren. Dersom brukerens interesser endret seg, sank godheten ved endringstidspunktet, men steg så igjen opp mot det tidligere stasjonære nivået.

Fra Amalthea har vi hentet flere elementer: Den overordne strukturen i både problem og løsning (bruk av java program med agenter som gir råd gjennom en browser). I likhet med Amalthea bruker også vi vektorer med transformerte ord-tellinger for å representre agenters tilstander. Amandas arkitektur er imidlertid mer generell enn Amalthea siden vi både har en rikere intern struktur for tilkobling av brukernære agenter og et rammeverk for agenter som tillater større funksjonell rikdom. Til gjengjeld implementer Amalthea både grensesnitt mot eksterne søkemaskiner og filtere, mens Amanda i dag kun har grensesnitt mot søkemaskiner.

4.2 Inquirus

"Inquirus" [25] er et system utviklet av Eric J. Glover m.fl. ved NEC research institute. Forfatterne klassifiserer systemet som et "meta-søkeverktøy", siden det gjør søking ved å søke gjennom andre søkeverktøy, ikke ved å selv bygge opp søke-indekser direkte mot sider på webben: I en standard søkemaskin har brukeren et informasjonsbehov som formuleres som en spørring. Når spørringen utføres mot en database produseres et resultat, som så ordnes på ett eller annet vis før listen av treff presenteres for brukeren. En meta-søkemaskin sprer søkene ut på flere søkemaskiner, og har i stedet for en enkel ordning av dokumenter en ordning som også kan ta hensyn til hvilken søkemaskin resultatene kom fra.

I Inquirus prosjektet forsøker man å finne søkeresultater som passer best til brukerens informasjonsbehov. Dette gjør man ved å la spørringen styre både hvilke søkemaskiner man bruker for å innhente resultater, og hvordan resultatene ordnes når de presenteres for brukeren.

Dette er imidlertid ikke ren "relevans" -rangering. I Inquirus-prosjektet definerer man relevans som tilstanden som bestemmer hvorvidt et dokument er innen samme emne som forespørselen eller ikke. Ett av problemene ved å rangere innhentet informasjon kun basert på relevans er at brukeren kan ha preferanser over relevante dokumenter som ikke kan uttrykkes. For eksempel; en bruker som søker etter aktuelle hendelser i forbindelse med et nylig inntruffet jordskjelv kan finne to lignende dokumenter. Ett er gårsdagens nyheter, og et annet er dagens. Selv om begge er relevante, kan brukeren tenkes å foretrekke dagens. En måte å løse problemet på er å innføre begrensninger, f.eks. en regel regel som ekskluderer dokumenter som er mer enn en dag gamle, men dette vil ikke gi en regel som ordner de dokumentene som er igjen særlig godt, og dersom det beste dokumentet var en dag og et sekund gammelt, ville det bli ekskludert.

I stedet for å stole på "relevans" brukes derfor et begrep kalt "verdi"[2]. Verdien av et dokument er subjektiv. Brukere med identiske forespørsler kan gi forskjellige verdivurderinger på det samme dokumentet, og den samme brukerens vurderinger kan endres over tid. Når det finnes få relevante dokumenter kan det være aktuelt å vise dem alle til brukeren, men når det finnes hundrer eller tusener blir ordnings-kriterier som tar hensyn til verdi mye mer ønskelige.

- **AverageGrade**: Gjennomsnitt av tre graderingsalgoritmer, FOG, SMOG og FK.
- **GFOG**: En lese-nivå algoritme optimalisert for mindre avanserte dokumenter.
- **WordCount**: Antall ord pr side.
- **WordsPerSection**: Antall ord dividert med antall "seksjoner".
- **Homepage**: Et mål for hvor mye siden ligner en hjemmeside.
- **GenScore**: Et mål som indikerer om siden er en "generell" side, slik som nøkkelord linker eller "ressurser".
- **ResearchPaper**: Et mål som indikerer hvor likt siden er en forskningsartikel. Har den f.eks. et "abstract" og referanser?
- **SectionCount**: Antall seksjoner i en side.
- **Pathlength**: Hvor langt unna toppen i et domene er siden adressert?
- **TopicalRelevance**: Et mål som indikerer hvor mye en side handler om en gitt forespørsel: Attributtet er basert på ord-distanser, mellom hverandre, og fra toppen av dokumentet, i tillegg til antall forekomster av hver term.
- **Summary**: En automatisk generert oppsummering av hvert dokument.
- **DaysOld**: Et estimat av alderen (i dager) av sidens innhold.

Figur 4.2: *Parametere Inquirus ekstraherer fra dokumenter den finner for å utføre rangering før presentasjon overfor brukeren.*

I Inquirus foregår en spørring slik: Brukeren har et informasjonsbehov, som representeres gjennom en forespørsel, og en mengde preferanser. Ut fra en kombinasjon av forespørselens art og preferansene velges det ut en mengde informasjonskilder som antas å være lovende. Noen søkemaskiner er spesielt gode på nyheter, andre er spesielt gode på sport, atter andre er gode på ordboksoppslag. Når en mengde kilder er valgt sendes forespørsler ut, resultater mottas, og de mottatte resultatene parseres. En del parametere ekstraheres fra de mottatte sidene, og er gjengitt i figur 4.2

Til slutt ordnes de innkomne resultatene til presentasjon i en liste. Til hver type informasjonsbehov tilordnes attributene i listen over en vekt. Vekten multipliseres med verdien for de innkomne dokumentene som et indre-produkt av en vekt-vektor og en attributt-vektor, resultatet er et "godhetstall" som kan brukes for å sortere dokumentene. Vektene antas å modellere en verdi-vurdering

[2] "Utility"

som beskrevet over. Vektene er gitt manuelt, men forfatterne skriver at de ønsker å introdusere lærings-algoritmer for å la systemet finne vekter som passer brukerens preferanser overfor et informasjonsbehov. Forfatterne sier at denne metoden er inspirert av en teori kalt "Multi-Attribute Utility Theory" og refererer til boken [36] for detaljer.

I seksjonen om relatert og fremtidig arbeid refererer forfatterne også til en artikkel av Mizarro [50] som en utmerket oppsummering rundt relevans.

Problematiseringen designerne av Inquirus gjorde rundt bruk av Information Retrievals relevansebegrep, og forkastning av denne til fordel for et mer generelt nyttebegrep er vurderinger som også er reflektert i Amandas design (og forsåvidt også i f.eks. Rhodes arbeider slik de er referert noe senere i denne oppgaven). Videre understreket suksessen Inquius hadde gjort med å bruke parametrene i figur 4.2, som åpenbart er produsert av ad-hoc, men gjennomtenkte kompetansemoduler, at det var nødvendig for å åpne for at dokumenter kunne annoteres med et temmelig rikt tilfang av avledede parametere. Alle Inquirus sine parametere kan tolkes som nøkkelord $\mapsto$ tall avbildninger, en fremstilling som Amandas Wector-datastruktur håndterer meget vel. Å skille mellom bidrag fra flere kompetanseenheter er også enkelt, siden hver type enhet kan tilordnes en ontologi som gir tilgang til et navnerom der hvert felt både har en bestemt mening, og ikke kolliderer med andre kompetanseenheters annotasjoner.

Selve funksjonaliteten i Inquirus er det mulig å reimplementere innenfor Amanda-arkitekturen. Inquirus er en meta-søkemaskin med et tradisjonelt søkegrensesnitt. Det er ingen ting i veien for å produsere et slikt brukergrensesnitt, la søkeuttrykkene bli tatt i mot av en preprosesseringsmodul som ut fra søket bestemmer hvilke søkemaskiner som skal velges, og som også presenterer resultatet gjennom tradisjonelt søkemaskingrensesnitt. Inquirus er imidlertid en langt mer statisk arkitektur enn Amanda, og det er for eksempel ikke lagt opp til Inquirus ved å observere brukeratferd over tid skal bli bedre til å levere resultater, så denne type funksjonalitet som Amanda i utgangspunktet har måtte i så tilfelle ha blitt undertrykt.

4.3 "Just In Time Information Retrieval Agents" (JITR)

4.3.1 Oversikt over hva JITR er

Bradley James Rhodes introduserer i sin doktoravhandling [56] begrepet "Just-in-Time Information Retrieval (JITIR) agents". En JITIR observerer kontinuerlig hva en bruker gjør, og miljøet som omgir brukeren og forsøker å presentere informasjon som er nyttig for brukeren. Miljøet som Rhodes "JITIR"er observerer er vanligvis veldig datamaskin-intenst: Epost, en webside brukeren ser på, eller et dokument som blir skrever, men det kan også være brukerens fysiske miljø slik det senses gjennom kameraer, mikrofoner, GPS posisjon-sensorer eller annet. Informasjonen en JITIR leverer kan komme fra hvilket som helst antall av pre-indekserte databaser av dokumenter f.eks. epost-arkiver, notatfiler, eller

dokumenter fra kommersielle databaser og dokumentsamlinger.

Rhodes arbeide har en problemstilling som er svært lik vår, men som vi skal se velger han å bruke en innfallsvinkel som er ganske annerledes enn vår. Hans metode for å evaluere effektiviteten av sine JITIR agenter influerte imidlertid sterkt når det skulle defineres hvilen ramme Amanda skulle kunne brukes innen, og hans eksperimentoppsett har sterkt påvirket designet av eksperimentet beskrevet i appendiks A. Rhodes måte å representere søk bekreftet at vårt valg om å bruke ordvektorer antagelig var et godt valg, og vi har dessuten brukt flere av algoritmene Rhodes bruker som eksempler på søkealgoritmer under arbeidet med implementasjonen.

Rhodes setter frem tre kriterier for at at et system skal kunne være en JITIR:

Proaktivitet: Systemet skal på egen hånd skal hente frem informasjon, brukeren skal ikke trenge å eksplisitt taste inn søkeuttrykk e.l.

Ikke-intrusivt, men tilgjengelig: En JITIR skal presentere informasjon på en slik måte at den kan ignoreres. I motsetning til programmer som "popper opp" alarmer og annet, skal en JITIR la brukeren avgjøre om meldingen som gis skal ignoreres eller ei, og avgjørelsen skal kunne gjøres avhengig av konteksten brukeren er i.

Lokalt kontekstualisert: Systemer med alarmer, slik som filtere som lytter på nyhetstjenester er proaktive, men informasjonen de presenterer er utenfor brukerens lokale kontekst. JITIR henter frem informasjon fra kilder som ikke nødvendigvis selv endrer seg, men en "alarm" kan frembringes dersom brukerens kontekst endrer seg slik at en bestemt epostmelding plutselig blir veldig relevant og verdig en melding.

JITIR har elementer av søkemaskin, alarmer og varslingssystem[3], men ingen av disse systemene har hver for seg alle egenskapene som skal til får å kunne klassifiseres som en JITIR.

- *How does the use of JITIR affect the way people seek out and use information?*
- *How can a JITIR automatically find information that would be useful to a person by sensing that person's current local context?*
- *How should a JITIR present potentially useful information?*

Figur 4.3: Rhodes tre forskningspørsmål, fra [56, s. 19]

De tre forskningsspørsmålene Rhodes stiller seg i doktorgradsarbeidet sitt er gitt i figur 4.3. Rhodes arbeid er interdisiplinært, det første spørsmålet handler om atferdspsykologi og kognitiv etnografi, og målet er å fastslå kvantitative effekter, d.v.s. økningen i mengden informasjon blir sett når man bruker

[3] "Notification systems".

JITIR, og kvantitative effekter, for eksempel om en forfatter bruker direkte sitater i stedet for parafraser. Det andre spørsmålet handler om informasjonsgjenfinning (Information Retrieval - "IR") og datamaskiners evne til å oppfatte hva som er kontekst rundt en bruker. I stedet for å finne opp nok en IR algoritme, bruker han eksisterende algoritmer. Det siste spørsmålet handler design av brukergrenssnitt, men også teorier om kognitiv overbelastning og oppmerksomhet.

4.3.2 Overblikk over de implementerte systemene

I sitt arbeid beskriver Rhodes så tre implementasjoner som alle har forskjellige grensesnitt, men de grunnleggende designbetraktningene er anvendelige på alle tre.

De tre systemene han beskriver er:

Remembrance agent: En JITIR som opererer innenfor EMACS tekstbehandleren. Siden EMACS brukes til mange oppgaver inkludert epostlesing, lesing av nyheter fra nettet og tekstbehandling, er denne versjonen laget for å være spesielt tilpasningsdyktig i forhold til brukerens omgivelser.

Margin Notes: En web-basert agent som automatisk annoterer websider etterhvert som de hentes inn i en browser.

Jimminy: Også kalt "Wearable RA" presenterer informasjon til brukeren gjennom et hodemotert display. Informasjonen som presenteres er basert på det fysiske miljøet han er i, hvor han er, hvem han snakker med, tiden på døgnet etc. Systemet demonstrerer at JITIR kan anvendes også utenfor deskopen.

Alle tre systemene bruker den samme gjenfinnings-backenden som heter "savant". Savant ble utviklet spesielt for forskningsarbeidet til Rhodes, og inneholder både en dokumentindekser og en søkemaskin.

Rhodes beskriver tre eksperimenter:

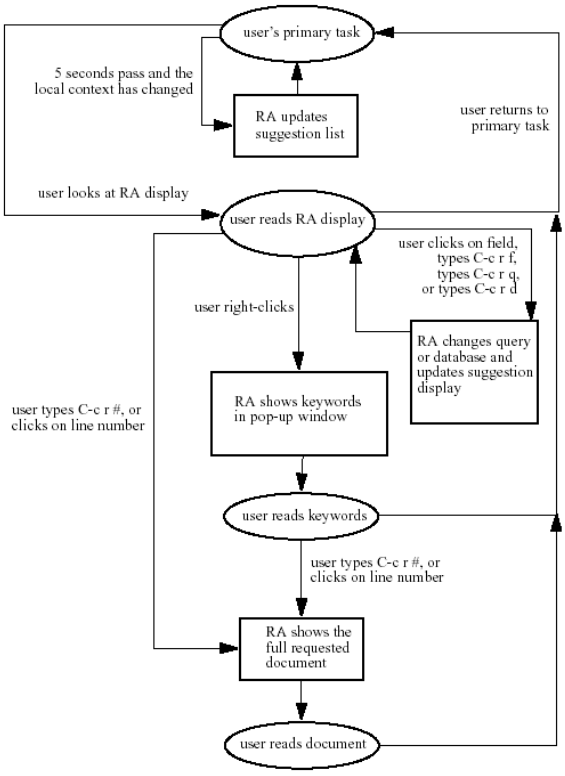
1. Et eksperiment der testsubjektene skulle skrive et essay med få ytre begrensninger gitt. Halvparten av testsubjektene får bruke en JITIR, og den andre halvparten en tradisjonell søkemaskin. Resultatene viser at brukerne som bruker JITIR ser på omtrent tre ganger så mange dokumenter som dem som kun har adgang til en søkemaskin. Subjektene fant også at Remembrance Agent var mer nyttig enn en søkemaskin for oppgaven de var gitt.

2. Det andre eksperimentet studerer forskjellen mellom relevans og nytten av forslag. Resultatene indikerer at antakelser gjort når man evaluerer tradisjonell Information Retrieval, nemlig at relevans impliserer nytte og at valget av database er utenfor skopet av IR algoritmen, ikke er gyldige når de anvendes på JITIR.

3. Det tredje eksperimentet undersøker langtidsbruk, hvor subjekter bruker JITIR over flere måneder. Studien avdekket mange måter JITIR kan være nyttig for en bruker, inkludert å gi informasjon som forandrer hva brukeren hoder på med, støtter opp under det brukeren holder på med, gir kontekst rundt det brukere holder på med. Informasjon som ikke er nyttig til det hun holder på med akkurat da, men kan være nyttig av andre årsaker, f.eks. underholdning.

I tillegg foretok han uformelle intervjuer med alle subjektene for å fange opp inntrykk som ikke ble fanget av spørsmålene som ble stilt til alle.

4.3.3 "Remembrance Agent"



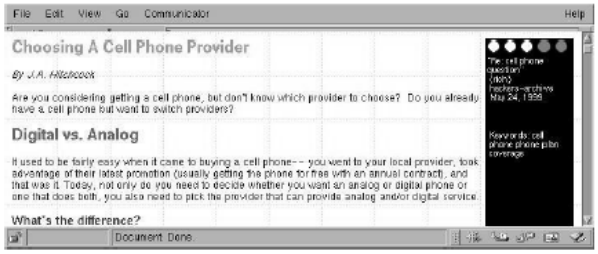
Figur 4.4: *Flytskjema for Remebrance Agent [56, s. 28]*

RA viser kontinuerlig en liste av dokumenter som er relatert til det dokumentet som samtidig vises i et EMACS buffer. Når brukeren skriver, leser epost

eller på annet vis endrer miljøet sitt, blir listen oppdatert med noen sekunders mellomrom. Forslag fra flere databaser vises frem i et eget buffer, hver med noen få linjer. Systemet kan ha regler som sier innenfor hvilke kontekster de forskjellige databasene skal brukes som kilde for informasjon, og hvordan ting presenteres kan være avhengig av hvilke deler av innholdet de antas å være relevante overfor: F.eks. kan de første par linjene kan vise forslag relatert til de siste tyve ordene, mens andre kan vise forslag relatert til de siste fem hundre. Når forslag finnes frem tas det ikke hensyn til brukerens historie, siden RA ikke har noen kunnskap om brukerens tidligere interesser eller om brukeren tidligere allerede har sett et bestemt forslag.

Ved å taste forskjellige tastekombinasjoner kan brukeren velge et foreslått dokument, og få se hele dokumentet.

4.3.4 "Margin Notes"



Figur 4.5: *Flytskjema for Margin Notes [56, s. 31]*

Margin Notes omskriver automatisk websider når de leses inn i en browser, og legger til linker. Når en webside leses inn, legger Margin Notes en sort stripe på høyre side av dokumentet. I likhet med RA sammenligner Margin Notes hver seksjon av dokumentet med preindekserte dokumenter, og avhengig av om ord forekommer i noen av disse kildene og i et avsnitt i teksten som betraktes, legges det på små "annoteringer" i margen til høyre for den aktuelle seksjonen. Margin Notes er implementert som en web-proxy som annoterer de sidene den leverer fra seg.

4.3.5 "Jimini"

"Jimini" er en RA instans som kjører på en skulderbåret datamaskin, og vises frem på et hodebåret tastekommandisplay. Jimini kjører også under EMACS tekstbehandleren, men er tilpasset den reduserte skjermstørrelsen i displayet som brukes. Videre er tastekommandoene tilpasset et bærbart tastatur, og egenskaper i miljøet som senses vises eksplisitt frem (hvilket rom brukeren er i etc.). Dette siste er viktig siden sensordata kan inneholde feil. Brukeren kan også gi

input som viser hva slags sensordata som er viktigst, en prosess som kalles å gi dataene "bias".

Savant

Alle tre brukergrensesnittene bruker den samme backenden kalt "savant" [4]. Når savant gis en spørring, produserer den en rangert liste av preindekserte dokumenter som best matcher spørringen. Savant består av to programmer: *ra-retrieve* som tar i mot en spørring og leverer resultater, og *ra-index* som lager indeksfiler så spørringer kan foretas hurtig. Savant kan indeksere mange slags dokumenter, epost, tekstfiler nyhets og avisartikler etc.

Savant har et "Template matching" system som kan gjenkjenne typer av dokumenter eller forespørsler, kan parsere felt fra hver type og indeksere dem automatisk. For eksempel kan epostmeldinger splittes inn i "subjekt", "mottager" etc. Templatene er hardkodet i Savant, men kan enkelt modifiseres uten å rekompilere kildekoden.

Denne typen komponent er konseptuelt ikke så ulik den Inquirus bruker til for å hente ut attributtene vist i i figur 4.2 på side 59. Forskjellen er altså at atributt-uthentingen brukes under indeksering, ikke under søking. Bruk av komponenter som selektivt henter ut attributter og tilordner dem spesielle meninger har imidlertid vist seg å være nyttig i begge tilfeller.

For søking kan forskjellige likhetsmetrikker brukes på forskjellige felt-typer. For eksempel kan fritekst i en melding og et søkeuttrykk sammenlignes med termfrekvens/invers dokumentfrekvens (tf/idf) algoritmer, tidspunkter sammenlignes med som hvor mye tid det er mellom tidsangivelsene, GPS koordinater med hvor lang avstand det er mellom dem etc. Flere metrikker kan dessuten brukes på de samme datatypene, så f.eks. to vektingsskjemaer eller helt forskjellige algoritmer kan brukes for å sammenligne innholdet i en lagret melding med en tekst som blir skrives.

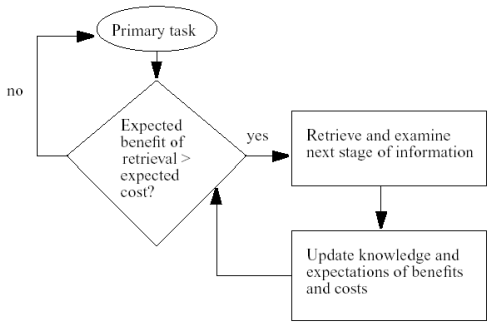
For å fjerne syntaktisk støy som ikke er nødvendig for gjenkjenning, har hver template en samling filtre formulert som perl-lignende regulære uttrykk som matcher tekst som skal fjernes fra feltene før parsering.

4.3.6 Hvordan måle hva som er nyttig med JITR?

For å besvare sitt første forskningsspørsmål ser Rhodes på hva som styrer atferd hos brukere. Figur 4.6 illustrerer hvilken sammenhengen mellom kost og nyttevurderinger som gjøres av en bruker når hun skal finne ut hva hun skal gjøre med en et stykke ny informasjon. Dersom nytten antas å være stor i forhold til innsatsen, havner vurderingen over den stiplede linjen, og det utføres en handling. Havner den under gjøres ingen ting.

I figur 4.7 vises hvilke interaksjoner en bruker vil ha med en generisk JITIR. Prosessen er iterativ, ved hvert steg vil brukeren bevisst eller ubevisst vurdere

[4] Som i forhold til Baeza-Yates terminologi representer Framework og Ranking - elementene av et IR system.



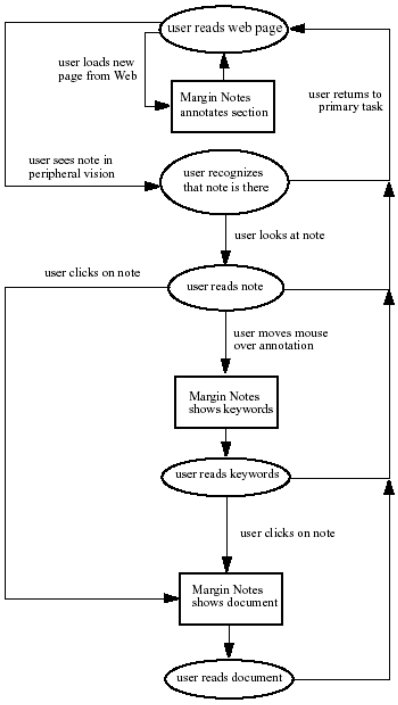
Figur 4.6: *Nytte/arbeidsvurdering hos en bruker [56, s. 42]*

den forventede nytten og kostnader ved å aksessere ny informasjon, og vil handle ut fra dette. For eksempel; dersom en bruker bruker RA for å skrive et prosjektforslag, kan han lure et øyeblikk på avsnittet han nettopp skrev. Han behøver ikke forvente at slik informasjon finnes, men det er verd en kvikk titt ned på RA siden det koster lite å titte. Dersom han ikke ser noen forslag som virker nyttige, vil han gå tilbake til sin primæroppgave, og har brukt under ett sekund på interaksjonen. Hvis han på den annen side oppdager informasjon som virker nyttig, vil han gjøre sine nytte/kost vurderinger en gang til og vurdere hvilke nøkkelord som var assosiert med forslaget, eller å hente frem og lese det fulle dokumentet som foreslås.

Rhodes hevder at innen rammeverket gitt i figur 4.7 kan informasjonen som gis av en JITIR falle i fem kategorier:

- **Falske positive:** Verdiløse resultater.
- **Falske positive:** Informasjonen som gis er allerede kjent.
- **Senket kostnad:** Eksistensen av den informasjonen som gis er kjent, men det er ikke ved innsatsen å leter etter den på annet vis. En JITIR kan f.eks. gi et eksakt sitat slik at en forfatter slipper å parafrasere.
- **Øket forventet nytte:** Informasjonen som gis av agenten er ikke i seg selv nyttig, men den indikerer eksistensen av annen informasjon som kan være verdifull.
- **Redusert kostnad og øket forventet nytte:** Informasjonen som gis var ikke kjent, og den var nyttig.

Dette utfallsrommet representerer en modell for output fra JITR systemet, og dersom man skal lage et adaptivt system (d.v.s. et system som forbedrer seg over tid), kan målet for et slikt system være å øke andelen av de to - tre siste utfallene og minske de to første.



Figur 4.7: *Kontinuerlig beslutningsprosess hos bruker[56, s. 43]*

Systemene Rhodes har produsert viser såvidt jeg kan forstå ikke hver for seg noen adaptiv atferd, men Rhodes forsøker å måle effektene av systemet gjennom intervjuindersøkelser, og finner fra disse at hans system i mange tilfeller oppfattes som godt av brukere. En utfordring når autonome adaptive systemer skal produseres er å kunne gjøre målinger som gir tilsvarende tilbakemeldinger uten at det er nødvendig å etablere en egen eksperimentsituasjon.

4.4 Levering av innhold

I artikkelen "Everywhere Messaging" [16] oppsummerer Chris Schmandt m.fl. noen resultater oppnådd ved MITs Media laboratorium med eksperimentelle systemer for meldingsformidling. Det underliggende målet for arbeide er å lage et system som kan sende meldinger til brukere hvor som helst, på kontoret eller utenfor, men gjør det på slikt vis at meldingene presenteres på et format, og i et slikt utvalg at de oppfattes som berikende for situasjonen.

Forfatterne nevner fire hovedutfordringer for meldingssystemer:

1. Meldingstjenesten må minimere avbrudd samtidig som de sikrer at vik-

tige meldinger kommer frem på et betimelig vis. De nevner et funn i en studie av mobile yrkesutøvere [53] som viser at subjektene i gjennomsnitt bruker ti minutter pr. time på avbrudd. 64 prosent av avbruddene var av en slik art at mottageren fikk en nytte av den, noe som indikerer at totalt forbud mot avbrudd ikke nødvendigvis er formålstjenlig. På den annen side var det også slik at 41 prosent av avbruddene førte til at den avbrutte ikke gjenopptok arbeidet som ble avbrutt, så det virker derfor som om det er et behov for sterkere filtrering dersom man skal tillate ennå større avbruddsfrekvens.

2. Kommunikasjonstjenester må tilpasse seg til endringer i brukeradferd utover dagen.

3. Termen "Everywhere Messaging" nærmest ber om lokasjonsavhengighet, slik at systemets atferd med å prioriterer hvilke input meldinger som har relevans varierer med fysisk plassering, og type aktivitet brukeren er involvert i.

4. Det siste punktet er at meldinger må konkurrere om brukerens oppmerksomhet. Dette er problematisk spesielt når brukergrensesnittene er plassert på bærbare enheter, hvor displayer kan være små, og vanskelige å lese, og aksesseres ofte under dårlige lysforhold. Talegrensesnitt kan være brukbare i et mobilt miljø, men de krever en høyere grad av oppmerksomhet hos mottageren. Vi er av definisjon opptatte når vi blir avbrutt, og brukergrensesnitt må være forståelsesfullt overfor de aktivitetene vi er engasjerte i.

Det presenteres så fire prosjekter som gir noen løsninger på problemene skissert over: *clues* som er en epostfiltreringsagent, *active messenger* som håndterer flere kommunikasjonskanaler og proriterer meldinger slik at de blir levert på den mest passende kanalen, *Nomadic radio* har et rent lydmessig grensesnitt som forsøker å tilpasse en innkommende melding til brukerens oppmerksomhetstilstand, og *comMotion*, som fokuserer på "riktig-tid riktig-sted" meldingsformidling ved å følge fysisk lokasjon med et GPS system, og ved å lære reisemønster.

4.4.1 "Clues"

Clues bruker informasjonskilder slik som epost og kalendere som endres sammen med brukerens planer og aktiviteter, og lager fra spor gitt i kildene dynamiske filtre for å kunne identifisere betimelige meldinger. Opprinnelig ble Clues brukt i et telefonbasert system som leverte epost via stemmesyntese, slik at de viktigste meldingene ble levert først. Ulikt de fleste epostfiltersystemer skiller Clues mellom meldinger som har interesse akkurat i nuet, og meldinger som reflekterer stabile interesser eller sosiale relasjoner hos brukeren. Clues bruker et lite antall statiske regler for å fange langsiktige interesser, og dynamiske filtre for å fange mer kortsiktige interesser.

Clues prosesserer meldinger i steg:

1. Først hentes informasjon ut fra tidsmessig organiserte datakilder. Kalenderentries indekseres med dato og tid, og et brukerdefinert "interessevindu" bestemmer hvilken granularitet som brukes i analysen. Brukere har ofte "todo-lister" og Clues antar at slike lister alltid er oppdaterte og relevante. Sendt epost gir navn på personer man korresponderer mye med, og områder av interesse. Telefoniapplikasjoner gir data om utgående samtaler, som kan assosieres med epostadresser gjennom adresseboka. Et Perlscript produserer en liste med ord og fraser fra det aktuelle tidsvinduet, fra alle informasjonskilder.

2. Neste steg fjernes vanlige ord som "møte" og "epost" ved å anvende en stoppordliste. Clues har også en "kansellerings" egenskap som lager en stoppordliste til hvor ord forblir en kort periode, men perioden øker ettersom ordene blir lagt til kortidslisten igjen, og etter et visst antall kanselleringer går de inn i den permanente stoppordlisten.

3. Den siste Clues gjør er å korrellere informasjonen med ekstraspor fra geografisk organiserte datakilder for å identifisere korrespondenter som geografisk relatert til telefonnummer domenenavn som nylig har kommet inn.

Clues spesifiserer *ikke* hvordan meldinger skal leveres. En "betimelig" melding er relevant til kortsiktige kontekster, noe som indikerer at nytten er større enn kostnaden ved avbruddet. Systemer som Active Messenger og Nomadic Radio bruker denne informasjonen om "korttids-relevans" til å minimere avbrudd.

Clues forsøker å unngå to problemer med desktopbaserte epost - filtreringssystemer nemlig at filterregler er vanskelige å lage for mobile systemer, siden de typisk må lages for hånd eller med en "wizard". Clues genererer regler automatisk. Videre kan brukeren bruke "regelkansellering" for å endre et regelsett. I et mobilt miljø må regler endres oftere for å være oppdaterte, noe som er problematisk siden en mobil bruker pr. definisjon endrer sin geografiske kontekst.

For å finne en fysisk lokalisert kontekst forsøker Clues å bruke records i brukerens kalendersystem. Når Clues finner et lokasjonsspor i kalenderen (gjennom et postnummer, en epostadresse, et telefonnummer eller lignende), leter den etter folk som er i det samme området. For å unngå problemer med geografisk distribuerte domenenavn (brukt i epostadresser), brukes en stoppliste også for domenenavn.

Siden filtre genereres automatisk av clues, trenges det ikke noe brukergrensesnitt for å lage dem. Siden brukere fremdeles trenger å stole på disse "usynlige og automatisk lagde" reglene, er det nylig laget en komponent som gir en forklaring av regelens virkemåte i naturlig språk. Forklaringene lages samtidig med reglene. Det lages to forklaringer ved å bruke en enkel template;

> Messages from the [Locality] area are relevant because you have an [SrcType] entry which lists a phone number with the [AreaCode] area code. The person with this e-mail address has a telephonenumber in or near that area code.

Figur 4.8: *Eksempel på template for forklaring av regel Clues bruker, fra [16, s.664]*

en for personsøkersystemer og en for tale-output. Et eksempel på en slik template er gitt i figur 4.4.1. Når en regel ikke gir god ytelse kan den fjernes fra systemet. Selv når Clues gjør feil er brukere mer positive til systemet når de forstår hvorfor en feil blir gjort.

Brukere av Clues finner at de markerer at mellom tolv og fjorten prosent av meldingene er betimelige. Systemet har vært i bruk i tre år, og brukes som filteragent for andre systemer.

Et system som Clues faller godt inn i Amandas arkitektur. En implementasjon av et Clues-liknende system kunne skje ved å lage brukernære agenter som er i stand til å lese hendelser fra kalenderfiler og lager hendelser av innkommende epost. Selve filtreringen likner svært på den som allerede er tenkt skal skje i Amanda kjernen, men for å direkte kopiere atferden Clues har kan en egen Clues-modul legges til kjernen. En problematikk en Clues-implementasjon reiser, er hvordan skille meldinger som skal gis til forskjellige kanaler: Det er ikke rimelig at de samme meldingene er relevante overfor en mobil bruker som en bruker som sitter stille ved et skrivebord, det er derfor nødvendig å avklare hvordan samvirke mellom regler for relevans i forskjellige typer kontekster skal kunne samvirke. En triviell måte å løse dette problemet på er å opprette separate instanser av Amanda for hver grovt indelte type kontekst (jobbskrivebord, mobil, hjemme, etc.), men dette er antagelig ikke den best tenkelige inndelingen.

4.4.2 "Active Messenger"

AM er en tjeneragent (en agent som kjører på en tjenerdatamaskin, vanligvis ikke på en bærbar eller personlig maskin) som prioriterer meldinger med Clues, og sender dem til tilgjengelige kommunikasjonskanaler slik som personsøkere, faxmaskiner og telefoner. Brukeren oppgir en fil med preferanser, men disse kan modifiseres av agenten. AM kan også sjekke hvorvidt meldinger kommer frem og velge å bruke en alternativ kanal dersom den ikke gjør det, f.eks. sende en personsøkermelding dersom brukeren ikke er logget inn på datamaskinen sin. AM bruker mange kilder for å spore brukere: Parsering av mail-spool filer for å se om meldinger blir lest, "Finger" kommandoen i Unix for å se om brukeren er logget inn etc.

Agenten forsøker å minimere avbrudd ved å velge den mest passende kanalen. Meldinger fra medarbeidere kan f.eks. bli rutinemessig levert når brukeren er innenfor rekkevidde av det trådløse nettet på arbeidsplassen, men uønsket for leveranse hjem med telefon, for eksempel.

Agenten kan modifisere disse reglene ved å hoppe over kanaler som har vist seg å være inaktive.

Ingen av enhetene som brukes for leveranse er direkte i stand til å vite hvor deres geografisk lokasjon er, men AM assosiserer noen kanaler med "virtuelle lokasjoner" slik som "arbeid", "hjemme", "utenfor byen etc. I preferansefila kan brukeren assosiere telefonnummer og navn på datamaskiner med virtuelle lokasjoner. Når brukere ringer inn for høre på voicemail kan f.eks. AM bruke kunnskap om telefonnummeret det ringes fra til å gjette seg til at brukeren er hjemme.

Når artikkelen ble skrevet var det kun to brukere som hadde brukt systemet jevnlig, men de planla å skalere det opp til større mengder brukere for å få mer signifikante data om brukeradferd.

Active Messenger har et ganske statisk responsmønster, så nøyaktig den funksjonaliteten som implementeres er det antagelig ikke særlig ønskelig å implementere i et Amanda-liknende system.

Et AM system kan imidlertid kobles opp mot et Amandasystem gjennom en brukernær agent som kan ta i mot meldinger for leveranse til brukeren og gir tilbakemeldinger om hvorvidt meldingene passet eller ikke.

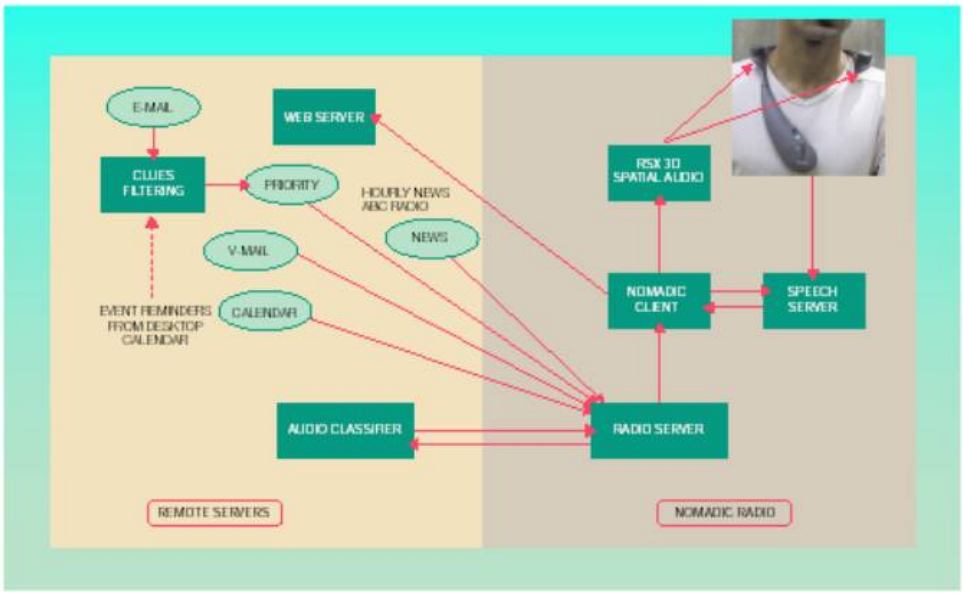
AM systemet har en hardkodet kunnskap om hvilke fysiske lokasjoner, og dermed også hvilke sosiale kontekster brukeren befinner seg i. Dersom en ønsker å lage en AM-liknende agent som mer dynamisk lærer seg å tilpasse meldinger til leveransekanal, kan det være interessant å arbeide innenfor Amanda. AM har også hardkodet kode for å lese forskjellige slags filer (epost, "finger" databaser etc.), og ved en flytting til Amanda vil det være hensiktsmessig å dele opp funksjonaliteten i en inn og en ut-del. Inn-delen genererer hendelser inn mot Amanda (slik som "brukeren ringte nettopp hjemmefra, så han er sikkert hjemme") og ut-delen blir da en brukernær agent som forsøker å hente meldinger som er mest mulig relevant for den.

Det vil da være naturlig å gjøre et arbeid for å finne ut hvordan Amanda kan sortere presentasjon av forslag ut fra mer eller mindre presis informasjon om hvilke kontekster brukeren er i. Denne type funksjonalitet betrakter undertegnede som dårlig forstått, og det trenges ganske sikkert mer forskning for å finne ut hvilke typer modeller som best er i stand til å sortere meldinger inn i passende kontekst.

4.4.3 "Nomadic Radio"

Nomadic Radio (NR) er en lydbasert bærbar plattform som slår sammen mange personlige informasjonstjenester slik som epost, nyhetskringkasting, kalender, alarmer etc. Systemet bruker stemmesyntese og stemmegjenkjenning for interaksjon, og er basert modifisert utgave av en telefonprototyp fra Nortel. Det er to retningsbestemte telefoner på skuldrene, og en retningsbestemt mikrofon på brukerens bryst. Et "rikt bilde" av Nomadic Radio er vist i figur 4.9. Telefonprototypen vises øverst til høyre i figuren.

NR bruker Clues for å filtrere innkomne meldinger. En teknikk for rom-



Figur 4.9: *Nomadic Radio [16, s. 669]*

lig lyd[5] brukes for å plassere lydkilder rundt brukeren, så de høres i distinkte lokasjoner.

NR bruker en modell som veier faktorer i det oppfattede miljøet rundt brukeren mot parametere i meldingen for å avgjøre om meldingen er verd et avbrudd eller ei. Bruksnivået bestemmes gjennom brukerens siste interaksjon med enheten. Interaksjoner som foregikk for kort tid siden vektes høyere. Systemet bruker en skjult Markov modell for å oppdage om brukeren snakker med noen, og vil i så tilfelle ikke avbryte (denne komponenten rapporteres å ha ca. 90 prosent presisjon). Meldinger presenteres med en lydmessig vignett, dersom brukeren ikke gjør noe mer bli den påfølgende meldingen levert. Avhengig av om brukeren velger å høre en melding eller ei vektes brukerens preferanser for den typen meldinger opp eller ned kontinuerlig.

Når en melding sendes inn til en bruker blir den "skalert opp", med anledning til å avbrytes. Når en melding kommer blir brukeren gitt en indikasjon om viktigheten (noe "clues" har funnet ut), så sendes det en lydsnutt med stemmen til avsenderen (hentet fra voicemail fra den samme personen), så gir en stemme en oppsummering av meldingen for å indikere header, emne, etc. Om ønsket kan et par linjer av meldingen leses opp, eller de første sekundene fra en voicemail. En slik oppskalering gir en flytende overgang mellom innkomne meldinger og kontinuerlig spillende lyd.

[5] "Spatial audio"

NR bruker talegjenkjenning og talesyntese og har innkapslet algoritmer for hvordan gjenkjenne kontekst på en slik måte at systemer som kommuniserer med NR kun ser hvilken tilstand den tror brukeren er i. Dette betyr at det å koble seg opp mot NR er en teknisk sett ganske enkel oppgave. I et Amanda-rammeverk kan dette gjøres ved å koble komponenten "Nomadic Client" komponenten opp mot Amandas transportnettverk, la den sende spor om kontekst til Amanda, og hente også som er relevante for brukeren brukeren derfra. Radiotjeneren er det uvisst hvordan man kunne realisert: Den enkelste måten er å sette opp en FM radiosender, men er ikke særlig hensynsfullt m.h.p. båndvidde, en annen måte er å bruke en eller flere digitale multicast/unicast kanaler med lyd og kombinert disse i NomadicClient. Uansett vil selve lydtransporten måtte defineres til å være utenfor Amanda-delen av en realisering.

I sum ser det ut til at funksjonaliteten til NR i stor, muligens grad, muligens totalt, allerede i dag kan realiseres i enkelte moderne mobiltelefoner, så jeg har derfor valgt å antyde noen retninger for mulig fremtidig arbeid om dette i seksjon 6.5.

4.4.4 "comMotion"

"comMotion" (cM) er et kontekstsensitivt system for en mobil eller bærbar plattform, med fokus på brukerens mobilitet fremfor bærbarheten av datamaskinen. når brukeren gjør sine daglige gjøremål, ser en agent på en strøm av GPS koordinater fra brukeren lærer ting om reisemønster og hvilke steder som er besøkt. Jo mer som læres om brukeren jo mer kan leveres til rett tid og rett sted. Kontekstrelevante påminnelser, todo-lister o.s.v. sendes til brukeren trigget av hans lokasjon. Siden mobilitet ofte impliserer at hender og øyne er opptatt, kan systemets kjernefunksjoner aksesseres via tale.

Systemet har en klientapplikasjon som kjører i en bærbar datamaskiner som kommuniserer trådløst over interenettet med servere. Det finnes servere for bestemte oppgaver slik som epost, påminnelser, overføring av innhold fra webben,kart o.l.

Når en bruker har blitt "sett" tre ganger (konfigurerbart) på et nytt sted, spørres brukeren om navnet på stedet, som enten kan gis med en gang eller senere. Ved å gi navn til et sted indikerer hun at sted er viktig. På dette punktet oversettes det geografiske stedet til et "virtuelt" sted, som "hjemme" eller "arbeid", og todo-lister blir umiddelbart assosiert med det.

Når systemet vet hvor brukeren er, fysisk og logisk, kan relevante meldinger gis. Dette kan være "kjøp grønnsaker" når man er i nærheten av en grønnsaksbutikk, en epost med en bokanbefaling når man er i nærheten av et bibliotek eller en bokbutikk. En værmelding når man drar hjemmefra etc.

4.4.5 Oppsummering om levering av innhold

Forfatterne konkluderer med at ukritisk mottak av meldinger alle steder skaper problemer, og dersom mer uttrakt bruk av meldingsmottak og avbruddene de medfører skal bli en suksess, må det løse flere problemer enn det skaper, og de

Challenge	Clues	Active Messenger	Nomadic Radio	comMotion
Minimizing interruption	Prioritizes	Provides channel selection	Monitors conversation	Waits for right time/place
Adaptation	Monitors activity	Monitors user activity over multiple devices	Responds to inactivity	Learns locations
Location awareness	Derives from metro areas (calendar)	Infers virtual location	Uses acoustic classification	Has physical/ geographic location
User interfaces (competing demands)	Prioritizes messages to save time	--	Uses scaled/alerting, voice control	Uses auditory user interface for mobility, voice control

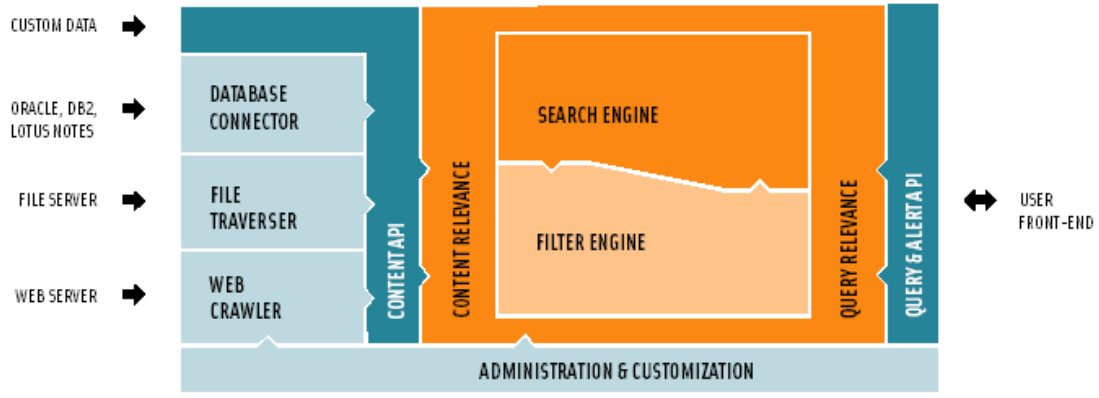
Figur 4.10: *Oppsummering av egenskaper i de beskrevne systemene [16, s. 662]*

forskjellige de fire beskrevne systemene forsøker å løse dette problemet på er angitt i figur 4.10. Clues og Active Messenger har begge vært i bruk i lengre tid og er nyttige for brukerne, som riktignok er svært avanserte brukere, så forfatterne påstår ikke at noen av disse systemene vil vises seg å være nyttige for "vanlige mennesker". Tilpasninger til brukeres behov kan foregå i bakkant i en tjener, som med Clues, eller helt brukernært som i Nomadic Radio. Å gi

Everywhere messaging er interessant, fordi det gjennom sansing av lyd i miljøet (nomadic radio), hvilke kanaler som brukes til lesing (telefon, epost etc.) har vist seg mulig å lage enkle modeler for hvilken kontekst brukeren er i. Det generelle bildet ser ut til å være at det fra hver type enhet vil kunne komme en strøm av "kontekst-spor" som sier "jeg tror brukeren er på bussen", "jeg tror brukeren snakker med noen", "jeg tror brukeren er på kontoret og leser epost" o.s.v.

Slik informasjon kan en kompetansemodul innen Amanda ta imot og bruke til å lage i en probabilistisk modell for hvilken tilstand brukeren er i. Ut fra en slik modell modellen kan man så lage rutingsregler for hvordan og når meldinger kan presenteres. Konseptuelt passer dette inn i Amandas arkitektur, men få eller ingen av de kompetansemodulene som i dag er implementert eller tenkt vil være direkte anvendelige for denne type applikasjon, denne type applikasjon vil imidlertid kunne være et stort og fruktbart område for videre arbeid.

Dersom man velger å gå en slik vei, er det mulig at arbeidene referert i seksjon 6.4 i kapitlet om videre arbeid er mulige steder å starte.



Figur 4.11: Fast Data Search sin overordnede arkitektur, fra [62, s. 2]

4.5 Fast Data Search

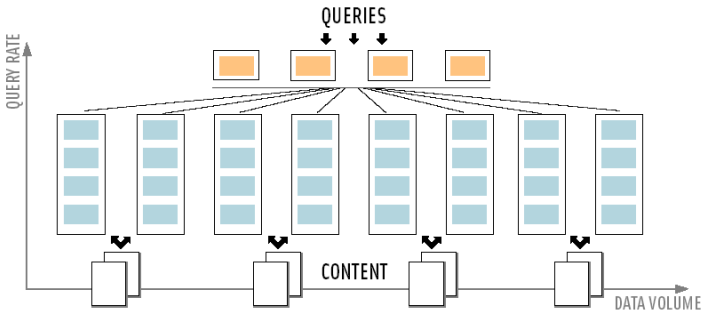
Fast Search and Transfer ASA[63] er et firma som lager søkeprogramvare. Programvaren brukes både i den generelle web-søketjenesten "all the web"[6] og selges til bruk i søketjenester. Vi velger å her beskrive noe av det som er gjort arkitekturmessig av Fast for å kunne håndtere de store volumene både av data og søk som de klarer. Årsaken er at vi som en del av denne oppgaven ønsker å kunne skissere en migreringsvei for Amandasystemet opp til de volumene som en stor og kommersiell søkeprogramvare må forventes å kunne håndtere i dag.

4.5.1 Funksjonell beskrivelse

Den overordnede arkitekturen for Fast Data Search (FDS) versjon 3.2 er vist i figur 4.11. Data som skal inn i systemet kan komme fra mange kilder, men de går alle gjennom noe som i figur 4.11 kalles "content API", som er et sett av programmer som prosesserer innholdet fra et mangfold av formater (html, pdf, tekstbehandlerfiler etc. Totalt 225 forskjellige formater i følge [62, s. 1]) til et enkelt standardisert XML dokumentformat (alt dette i følge [52]). Når dokumentene er lagret på standardiserte formatet blir de sendt inn i en søkemaskin ("search engine") og et sanntidsfilter ("filter engine").

I søkemaskinen blir dokumentene gjort tilgjengelige for søk som kan gjøres etter at dokumentene er lagt inn i søkemaskinens database. I sanntidsfilteret finnes det fra før en mengde søkeuttrykk, og dersom noen av disse matcher det innkomne dokumentet, vil de programkomponentene som har registrert

[6] http://www.alltheweb.com



Figur 4.12: Fastsearch sin arkitektur for skalering, fra [62, s. 2]

søkeuttrykket få beskjed om hvilket av de innkomne dokumentene som utløste treffet.

Innsetting av filteruttrykk til filteret, søkeuttrykk til søkemaskinen og til og presentasjon av søkeresultater fra begge foretas i en bruker-frontend.

Alle komponentene inne i datasearch snakker med en mengde administrasjons og konfigurasjonskomponenter som gjør at det skal finnes et lite antall integrasjonspunkter å sette inn konfigurasjoner samt å hekte inn administrasjons og overvåkningsprogramvare.

Fast datasearch leverer hverken datakilder eller presentasjon, alt dette i stor grad overlatt til kunder. Fast leverer imidlertid et sett med "connectorer" som kan kobles opp mot noen typer datakilder, og de leverer også konsulenttjenester ("Professional Services") som kan hjelpe til med konkret oppkobling mot kundenes datakilder, men selv med disse forbeholdene er det fremdeles kundenes ansvar å gjøre kildene tilgjengelige og få dem til å virke mot Fasts programvare. Dersom det er ønskelig å lage et webgrensesnitt for søk eller levere resultater fra et søkeuttrykk i filteret via SMS, så må Fasts kunder selv skaffe slike grensesnitt tilveie og stå for integrasjon mot Datasearch produktet.

4.5.2 Skaleringsegenskaper

Datasearch har en intern struktur som gjør at det skalerer svært godt både med hensyn på datavolum og antall forespørsler (både gjennom filter og søkemaskin), men ikke kompleksitet av søkeuttrykk (som forblir enkle). Figur 4.12 forsøker å illustrere dette: Figuren har to akser, en for datavolum (øker mot høyre), og en for spørringsvolum (øker oppover).

Figuren er ikke helt klar m.h.p. hvordan skalering foregår men i [52] ble følgende klarlagt:

Datasøk: Når antall forespørsler økes kan det legges på flere "spørreprosessorer". spørreprosessorer fordeler så søkene ut på en mengde av "søkenoder"

som foretar de faktiske søkene i dataene. Dersom man antar at søkenodene alltid er i stand til å finne svar raskt (og hvordan det kan skje beskrives i neste avsnitt), vil økning i antall forespørsler kunne håndteres med en proposjonalt like stor økning antall spørreprosessorer.

For søkenodenes del gjøres skaleringen slik: Når datavolumet øker segmenteres det m.h.p. hvilke ord det søkes mot: Hver "søkenode" håndterer kun en mindre mengde søkeord, og må derfor håndtere en mindre andel av den totale mengden data. Dette begrenser datamengden en søkenode må håndtere og garantere en prosesseringstid.

Når antallet forespørsler øker kan dessuten segmentene dupliseres, og forespørsler til dem fordeles etter en eller annen fordelingsnøkkel, f.eks. "round-robin"[7]. På denne måten begrenses den totale mengden forespørsler en enkelt søkenode er nødt til å besvare pr. tidsenhet.

Antall søkenoder skalerer dermed proposjonalt med produktet av mengde data som indekseres og antall forespørsler som skal besvares pr. tidsenhet.

Sanntidsfilter: Sanntidsfilteret består av en mengde filternoder. Hver node inneholder en mengde søkeuttrykk som hver igjen er koblet til en ekstern komponent gjennom nettet. Denne eksterne komponentent mottar meldinger om hvilket søkeuttrykk som matchet,og hvilket datum som utløste matchen. Filternodene har en øvre grense for hvor mange filteruttrykk de kan håndtere. Denne grensen gitt enten av tilgjengelig minne, eller hvor mye arbeid det er å matche filteruttrykkene. Å prosessere en match krever nemlig at det sendes melding til alle eksterne programkomponenter som har registrert interesse for uttrykket. Denne meldingsutsendelsen kan føre til at ganske store datamengder skal sendes ut, og kan derfor representere en flaskehals. Selve matchingen av søkeuttrykkene representerer ikke en flaskehals da denne foregår i lineær tid gitt av med lengden av det innkomne dokumentet og kun i meget liten grad er avhengig av antall filteruttrykk som filternoden skal håndtere. Denne garantien kan gis siden typen uttrykk det er mulig å søke etter er begrenset, det er f.eks. ikke mulig å søke etter generelle regulære uttrykk.

Når antall filteruttrykk øker håndteres dette ved å opprette flere filternoder, som hver inneholder sin delmengde av den totale filteruttrykksmengden. Alle innkomne dokumenter rutes så gjennom filternodene. Dersom mengden innkomne dokumenter blir så stor at filternodene ikke er i stand til å prosessere dem raskt nok, grupperes nodene i grupper som hver til sammen dekker alle filteruttrykkene. Hver delmengde av filteruttrykk er da dupliserte i hver filternode inn i en gruppe. De innkomne dokumentene fordeles så mellom gruppene av noder f.eks. gjennom round-robin scheduling, gjør at hver node i gruppen ikke får flere innkomne forespørsler enn de klarer å prosessere.

[7] en fordelingsmetode som gjør at alle enhetene får en forespørsel etter tur.

> Det totale antall filternoder skalerer på denne måten med produktet av antall innkomne dokumenter pr. tidsenhet og antall filteruttrykk. I tillegg er skaleringen betinget av hvor mye output som genereres av filteruttrykks-matcher.

Fast datasearch er med arkitekturen som er skissert over i stand til å foreta flere tusen søk pr. sekund inn mot datamengder på flerfoldige terabyte, og det er ingen ting i arkitekturen, slik den er beskrevet, som gir grenser for hverken antall søk pr. tidsenhet eller størrelse på mengde med data som skal søkes gjennom. Det som begrenser er hvilken kvalitet, målt i datamengde, kompleksitet av søkeuttrykk og antall søk pr. tidsenhet, som kunden ønsker, og hva hun ønsker å betale for forskjellige tjenestenivåer.

Både søkenoder og filternoder er koblet sammen gjennom et raskt nettverk som betyr at dataoverføringer internt i systemet aldri blir en flaskehals (unntatt som beskrevet over for filternoder). Nodene er alle feiltolerante i den forstand at det for hver node alltid finnes minst en node som umiddelbart er i stand til å ta over for den ("hot spare" konfigurasjon). Dersom enkeltnoder går ned, vil de resterende nodene derfor alltid være i stand til å bevare eksternt synlig atferd.

Det bør være åpenbart at Fast datasearch er et sofistikert produkt. Arbeidet som er lagt inn i å efektivt hente inn og indeksere store datamengder er ikke trivielt og det er helt urealistisk å tro at det vil være enkelt å gjenta dette, da er det heller grunn til å tro at det vil bli ytterligere konsolidering mellom de allerede etablerte aktørene.

På den annen side, Fasts produkter er basert på å favne store dokument-mengder og på å raskt kunne utføre store mengder av enkle søk mot disse. Sofistikert tilpasning mot brukeres behov er ikke dagens store søkemaskiners styrke. Dette betyr at den type profilering og produksjon av søk som Amanda står for er direkte komplementært i forhold til hva søkemaskinprodusentene gjør: Amanda tar utgangspunkt i komplekse informasjonsbehov og forsøker å levere relevante svar i forhold til disse, også gjennom å bruke de store søke-maskinene. Google, som er en av Fast konkurrenter har eksponert sin ver-densomspennende søkemaskin (www.google.com) gjennom et Web Services grensesnitt [30] som de tar betalt for. Et operativt Amanda-system vil kunne passe godt inn i en slik forretningsmodell, hvor søkemaskiner tar betalt pr. søk, men hvor den totale mengden søk øker i forhold til i dag.

Dagens Amanda-implementasjon bruker "screenscraping" mot søkemask-inenes webgrensesnitt for å sende søk mot Alltheweb og Google. Over "screen-scraperene" brukes et abstrahert Java-interface som gjør det mulig å snakke om "søk mot søkemaskin", så velger man senere hvilken søkemaskin det er snakk om. Screenscraping er en grei måte å få opp prototyper på, men det er i bunn og grunn ikke noen effektiv metode å hente ut informasjon fra søkemaskiner på: For det første kreves det mye manuell programmering for å sende søk til søkemaskiner og motta resultater derfra, dernest er det slik at ikke all inform-asjonen som hentes tilbake fra søkemaskinene blir representert med all inform-asjonen søkemaskinene har tilgjengelig (ofte må man gjette hva f.eks. hvilken

semantisk betydning kursivskrift har, og ofte gjetter man antagelig feil), og til slutt vil screenscraping raskt kunne slutte å virke dersom søkemaskinen velger å endre bare litte grann på hvordan skjermbildene deres ser ut. En langt bedre måte å håndtere søk på ville vært å hatt et stabilt abstrahert grensesnitt mot søkemaskinene, og så implement dette som et adapter mot stabile grensesnitt, for eksempel som Googles web services[30], levert av søkemaskinene. Dersom Amanda eller liknende teknologi noen gang skal kommersialiseres vil jeg betrakte slike stabile grensesnitt på tvers av søkemaskiner som en nødvendig teknologisk forutsetning siden det uten en slik mekanisme vil være svært vanskelig å produsere høyt volum av søk med stor grad av driftsikkerhet.

4.6 "Semantic web"

> "The Semantic Web is a vision: the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse for data across various applications".

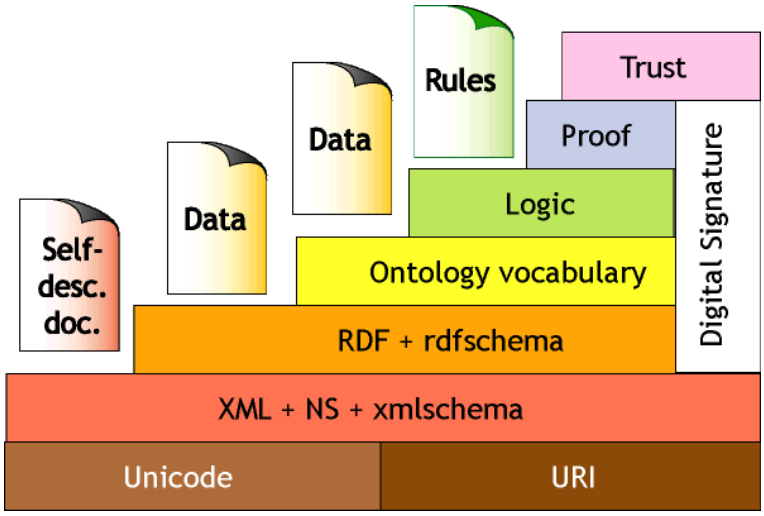
Figur 4.13: *Semantic Web Working Symposiums sin definisjon av hva Semantic Web er (Fra [66])*

Semantic Web (semweb) er et fenomen introdusert av Tim Berners-Lee m.fl. i artiklelen [10]. Semantic Web en videreutvikling av World Wide Web[9] (www). Den originale www inneholdt tekst, bilder, og hyperlenker. Semweb inneholder i tillegg semantikk som beskriver roller og relasjoner for objekter som refereres til inne i dokumentene.

Hvorfor er så Semantic Web relevant for Amanda? For det første representerer det en mulig fremtid for webben, og dermed for det univers av dokumenter som et Amanda-liknende system vil måtte søke i. Videre er det slik at selv om semweb-visjonen ikke blir oppfylt innen overskuelig tid (og det er faktisk fullt mulig), så produserer aktiviteter i forbindelse med semweb en mengde verktøy som er av interesse både fordi de kan brukes til å gjøre enkelte typer kunnskap anvendelige for søking, men også fordi vanskene med å produsere disse verktøyene avdekker underliggende problemer i visjonen om semweb: Hvilke vanskelige grenser finnes det for hva som kan søkes og navigeres gjennom webben?

I figur 4.14 vises en lagdelt fremstilling av elementer som inngår i semweb. Hvert lag bygger på elementene som er introdusert i lagene under. Nederst er URI (Universal Resource Identifier) og Unicode (som er en måte å kode svært mange av alle verdens tegnsett på). Dette er elementer som i dag er godt etablerte deler av den eksisterende www.

Neste lag er XML (Extensible Markup Language)[68], NS (Name Spaces)[67] og xml-schema [27]. Også disse elementene er teknikker som i dag er i aktivt bruk: XML er en standardisert måte å kode hierarkiske datastrukturer på, NS



Figur 4.14: *Hvordan infrastrukturlagene i semweb bygger på hverandre [8, side 14]. For hvert lag i "kaken" finnes det eller skal det finnes bestemte språklige mekanismer som realiserer det.*

[67] er en metode for innen XML kunne navngi elementer XML dokumenter er bygd opp av på tvers av dokumenter (datatyper og annet definert i ett dokument kan brukes av et annet). Xml-schema kan tenkes på som et datadefinisjonsspråk, som gjør det mulig å lage mange vanlige datatyper (records, matriser, etc.). Siste versjon av HTML standarden (XHTML) er beskrevet ved å bruke teknikkene som finnes på dette laget. Det må kunne sies at til og med nivå to i figur 4.14 representerer vel etablerte teknikker.

I Tredje lag finnes RDF (Resource Description Format) og rdfschema. Dette er også eksisterende teknikker som kan brukes for å beskrive meget enkle databaser med nøkkel/record avbildninger. RDF er et generelt verktøy, men det er ikke tatt i vid bruk. Spesialiserte applikasjoner finnes imidlertid, slik som WDSL (Web Services Description Format) [19] som brukes for å beskrive komponenter i Microsofts Web Services arkitektur, og RSS (Rdf Site Summary [54] som brukes til å publisere artikkel-lister fra websider (f.eks. aviser e.l.), slik at lesere kan abonnere på RSS beskrivelser av store mengder websider uten å måtte lese en eneste en av dem).

Over nivå tre er det imidlertid ikke så mye som er ferdig definert, langt mindre implementert, og der det *er* implementert er det ikke mye tatt i bruk. Det er i gang arbeid for å standardisere ontologidefinisjoner [47], men det er i mine øyene for tidlig å si hvor nyttig dette arbeidet vil vise seg å være. Over ontologilaget er det ønskelig å legge nivåer av logikk, bevis og tillit, og på disse nivåene er det ikke stort mer enn visjonen som eksisterer i dag.

Om vi tar utgangspunkt i hva som eksisterer, d.v.s. til og med nivå tre, kan-

skje fire, og ser hvordan fenomener derfra kan kobles opp mot Amanda får vi flere muligheter. På det laveste nivået er sammenkobling allerede skjedd: Amanda bruker Unicode, og alle dokumenter refereres unikt gjennom URIer. Det er et problem her siden lokale filer refereres gjennom "file://" syntaksen, og den gir en referanse som er indirekte gjennom et lokalt filsystem, og det kunne vært hensiktsmessig om også lokale filer gjennom ett eller annet skjema ble gitt globalt unike navn gjennom URIer, men såvidt jeg vet er det ikke i dag noe arbeid i gang som vil kunne oppfylle denne visjonen.

Nivå to er også interessant, siden det blant annet åpner for at alle informasjonsobjekter som flyttes mellom komponenter i Amanda arkitekturen kan kodes på et slikt vis at det er lesbart av komponenter også utenfor Amanda. Dersom det noen sinne blir aktuelt å implementere deler av Amanda utenfor Java, vil det antakelig være hensiktsmessig å omdefinere transportlaget til å kunne frakte standardiserte XML meldinger[8]. En slik innfallsvinkel vil ikke i seg selv utvide funksjonaliteten, men vil kunne gjøre det enklere å integrere forskjellige slags allerede eksisterende kompetansemoduler. I så måte er bare XML nok en (komponent i en) integrasjonsteknologi, og det kan godt tenkes at andre alternativer i praksis viser seg å være bedre.

Nivå tre og over er vanskeligere å forholde seg til, siden det er mindre klart hva vi kan forvente oss fra den siden. Eksisterende tjenester basert på RDF kan naturligvis inkluderes: Å bruke RSS som en kilde til nyheter, som så kan matches av agenter som forsøker å ekstrahere interessante nyheter fra syndikerte websider er absolutt en interessant mulighet. Dersom WDSL brukes til å beskrive tjenester som en bruker kan dra nytte av, og disse kan presenteres overfor brukeren gjennom et passende grensesnitt er også dette en mulig integrasjon. I begge disse to tilfellene ser det imidlertid ut til å være integrasjon mot spesialisert applikasjoner av RDF, ikke søking mot et generelt univers av ressurser beskrevet i RDF. Det ser m.a.o. ikke ut til å på kort sikt være noe håp om at det skal finnes omfattende eksterne søkemotorer som kan brukes for å finne frem til ting som er beskrevet med RDF, og inntil det skjer vil det kreves stor og påviselig nytteverdi før arbeidet med å integrere seg mot RDF kan forsvares (men for noen slike tilfeller, f.eks. overfor RSS finnes antakelig slik nytte i dag).

I det vi har beskrevet nå (xml, navnerom, xml-schema, rdf og ontologier) finnes det apparat for å beskrive mange slags relasjoner mellom fenomener. Det er med dette apparatet fullt gjennomførbart å kode mye informasjon, og legge logikk på toppen av den. Det er imidlertid på dette nivået (nivå fire) og oppover vi møter et stort og fundamentalt problem med semantic web prosjektet:

Siden hvem som helst kan lage et utsagn som påstår hva som helst om hva som helst, er det en fell mulighet at det vil finnes påstander som står i konflikt med hverandre. Hvordan kan dette problemet håndteres? Dette vil ganske sikkert bli komplisert, og vil involvere vurderinger av påliteligheten

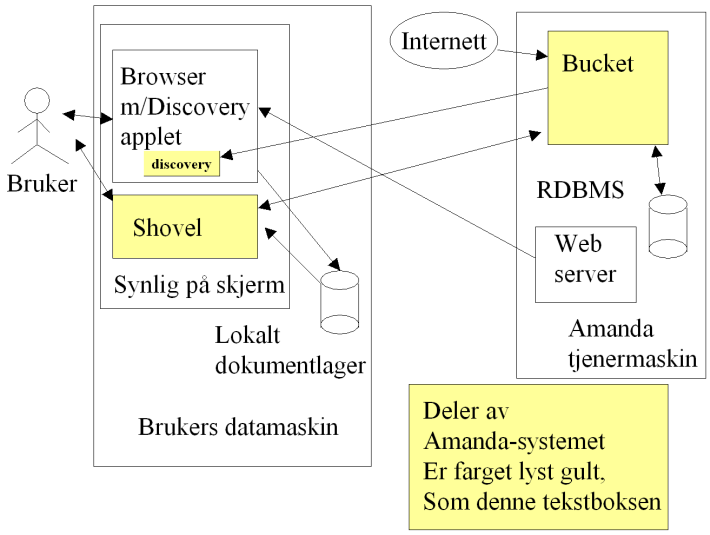
[8]Rent praktisk kunne dette gjøres ved å finne en transportmekanisme, som allerede er i stand til å gjøre dette, f.eks. Web Services, og plugge adaptere for denne mot Amandas eksisterende transportlag.

av kildene, utsagnen, og konsistensen av utsagnene med annen informasjon. Dette problemet oppstår som en konsekvens av webbens natur. I et lukket system kan det konstrueres og kontrollere, og som hovedregel kan en fjerne selvmotsigelser og feil. World Wide Web er ganske enkelt for stor til at det er mulig. I tillegg er den åpen, ikke lukket slik at mer informasjon kan legges til uten at det finnes noen kontroll med innholdet. RDF har ingen som helst mekanismer for å hindre selvmotsigende setninger, skjønt man kan tenke seg algoritmer som vil være i stand til å oppdage dem.

På bakgrunn av dette har jeg valgt å ikke orientere denne oppgavens arbeid mot Semantic web, selv om prosjektet på sikt har potensiale til å bety mye for hvordan søking på webben vil foregå.

Kapittel 5

Design og implementasjon av Amanda



Figur 5.1: *Overordnet "rikt bilde" som viser strukturen av Amanda systemet. Pilene viser retning av dataflyt relevant for Amanda.*

5.1 Overordnet om systemets arkitektur

Dette kapitlet beskriver Amandas design. Designet inneholder både deler som er implementert og deler som ikke er det. Dersom annet ikke er tydelig angitt kan leseren anta at det som er beskrevet også er implementert. Vår beskrivelse vil begynne med et høynivåperpektiv, så vil vi gjennom gradvis forfining ta frem detaljer i designet. Underveis i teksten beskrives hvilke avskjæringer vi gjør i fremstillingen for å få den hensiktsmessig.

Overordnet er Amanda et system som hjelper en bruker som sitter med en webbrowser ("internettleser") å finne frem til dokumenter som er relevante i forhold til det hun ser i nettleseren sin allerede.

Litt mer detaljert kan man si at dette foregår ved at det kjører et program ved siden av nettleseren som gir råd om dokumenter det kan være lurt å se på, og dersom brukeren velger å følge rådene som gis dukker disse dokumentene opp i browseren. Denne virkemåten er beskrevet i figur 5.1.

En detaljering av dette forholdet er at når brukeren bruker browseren til å se på dokumenter hun selv holder på å redigere i eller katalogen disse ligger i. I dette tilfellet vil brukeren gjerne ha et redigeringsprogram oppe samtidig. Det er muligens ikke så interessant for brukeren å bruke browseren til å se direkte på dokumentene som redigeres (selv om det naturligvis kan være tilfelle), men Amanda vi da overvåke endringene som gjøres i dokumentet og vil forsøke å finne frem til forslag til dokumenter som er interessante i forhold til de endringene som gjøres i dokumentet som redigeres. Et skjermbilde som illustrerer slik bruk finnes i figur 5.2.

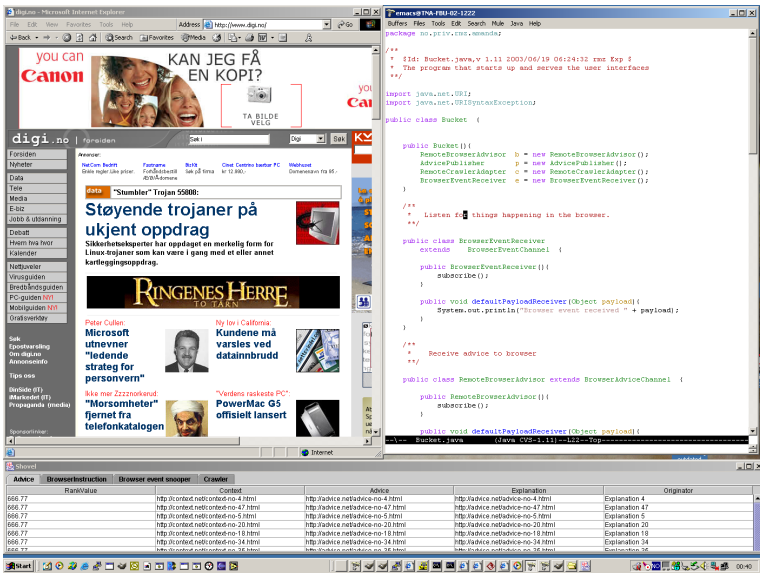
Det er her på sin plass å bemerke at selv om *implementasjonen* tar utgangspunkt i arbeid med tekstbehandler og browser, er *arkitekturen* vesentlig mer åpen med hensyn på arbeidsoppgaver og verktøy. Når brukerens handlinger behandles inne i "kjernen" av Amanda, den delen som finner frem til råd og bestemmer hvilke som skal presenteres. Her arbeides det kun om dokumenter, endringer i dokumenter, og hvilke type råd som ser ut til å ha effekt. Nøyaktig hvor dokumentene kommer fra eller hvordan råd leveres er abstraert bort på veien inn til kjernen.

Amanda er satt sammen av tre programkomponenter:

Shovel: [1] Et javaprogram med grafisk brukergrensesnitt. Lager profiler av brukerens filer, viser frem systemets råd om hvilke dokumenter som kan være interessante å se på, leser browser-logg for å finne ut hvilke dokumenter leseren faktisk ser på.

Bucket: [2] Et javaprogram uten grafisk brukergrensesnitt. Har persistent dokumentlager, ruter meldinger til Shovel og Discovery. Bucket leser internettdokumenter (det vil si dokumenter som ikke ligger på lokalt fillager), og lager profiler av disse. Alle agentene som gjennomfører søking og produserer råd til brukeren kjører inne i Bucket. Bucket har i dag ikke

[1] "skuffer inn" opplysninger fra omgivelsene.
[2] Dit ting blir lagt av shovel.



Figur 5.2: *Hele brukerens skjerm med browser som kjører "Discovery" applet (appleten er skjult) (øverst til venstre), Shovel (nederst) og et tekstredigeringsprogram (oppe til høyre) samtidig.*

noe brukergrensesnitt utover at det startes fra en kommandolinje og produserer en del test og sporingsutskrifter til konsoll, som vist i figur 5.3.

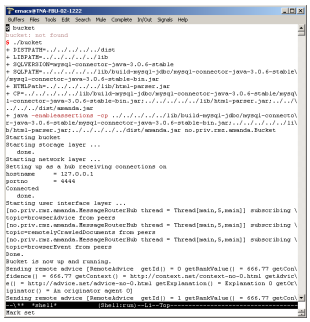
Discovery: [3] En java-applet som kjører inne i brukerens browser som instruerer denne om hvilke dokumenter som skal vises frem når brukeren velger å følge de rådene Amanda gir.

I tillegg er det nødvendig med en mekanisme for å levere appleten til browseren. Dersom browseren kjøres på samme maskin som Bucket, kan man hente appleten direkte fra lokalt fillager, ellers vil det være nødvendig med en webtjener på samme maskin som Bucket kjører som leverer fra seg appleten. Dette kommer igjen av begrensninger gitt av sikkerhetsmodellen for appleter som browseren brukere.

5.1.1 Funksjonell beskrivelse

Designet av Amanda er bygd rundt et sett av overordnede use-caser som vist i 5.6, og når alt dette kjøres sammen, ser det for brukeren ut som vist i figur 5.2. Av usecasene i figur 5.6 er det mest sentrale "Interaction regarding advice", som er detaljert ytterligere i seksjon 5.1.1. Vi har valgt å sette opp crawlerkonfigurasjon som et eget usecase, selv om denne konfigurasjon i den nåværende

[3] Hjelper til med oppdagelser.



Figur 5.3: *"Bucket" - programmet, startet i et Emacs-buffer. Er essensielt et bakgrunnsprogram uten eget brukergrensesnitt, det vi ser her er kun konsollutskrift for feilretting og sporing.*

implementasjonen er trivialisert ved at det alltid er brukerens nettleser-bokmerker ("favorites"), samt et bestemt filområde ("Amanda" under brukerens hovedfilkatalog) som crawles. Videre fører vi, under tvil, opp "ModifyTrackedDocument". Dette usecaset omfatter alle slags modifikasjoner gjort på dokumenter som skal følges av Amanda ("sporede dokumenter"). Grunnen til at oppføringen er gjort "under tvil" er at selve modifikasjonen gjøres av systemer som avgjort er utenfor Amandas systemgrense, og de gjøres dessuten mot et eksternt fillager som også er utenfor systemgrensen. Vi velger likvel å ta det med, fordi endringer gjort i sporede dokumenter er ment å oppfattes av brukeren som koblet til Amandas atferd. Vi vil likevel ikke detaljere dette usecaset videre.

UC1. Interaksjon rundt råd til brukeren

Primær aktør: *Bruker*

Øvrige aktører: *Bucket og Shovel applikasjonene samt Discovery-appleten kjørende inne i brukerens browser,*

Forbetingelser: *Alle aktører er aktive og i forbindelse med hverandre.*

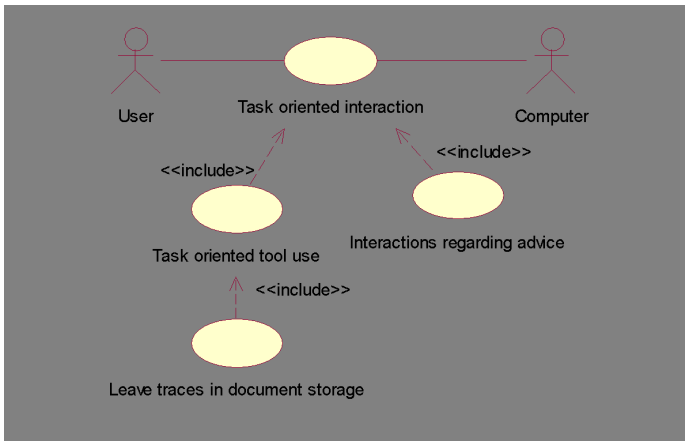
1. *Alle aktører er aktive og i forbindelse med hverandre.*

Postkondisjoner:

1. *Brukeren ferdig med å browse med assistanse.*
2. *Eksplisitt tilbakemelding fra bruker, samt og andre spor fra brukeradferd gjort tilgjengelig for Bucket.*

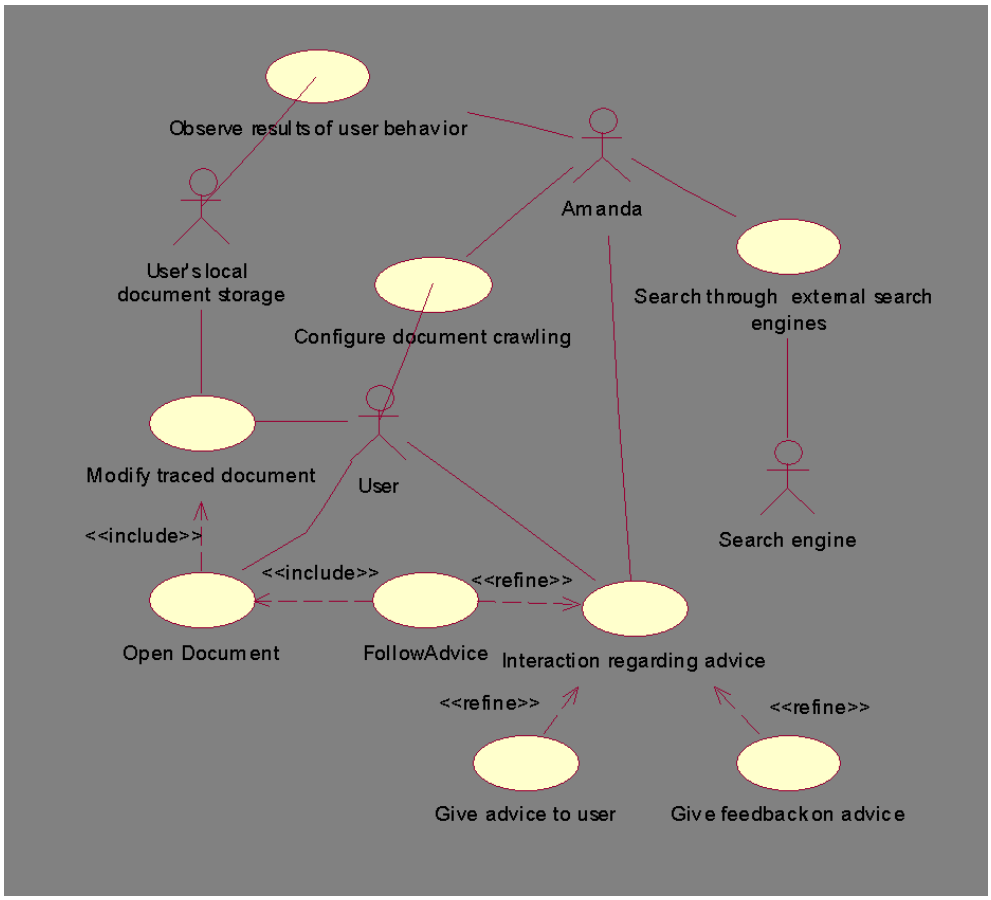
Steg:

1. *Brukeren utfører ingen eller flere av følgende handlinger:*
   (a) *Inkluder usecase 1.1 "Åpne dokument (open document)".*
   (b) *Inkluder usecaes 1.2 "Bruk råd gitt av Shovel (use advice").*
2. *Brukeren avslutter ett eller flere av programmene: Browseren, Discovery-appleten, Bucket, eller Shovel-applikasjonene.*

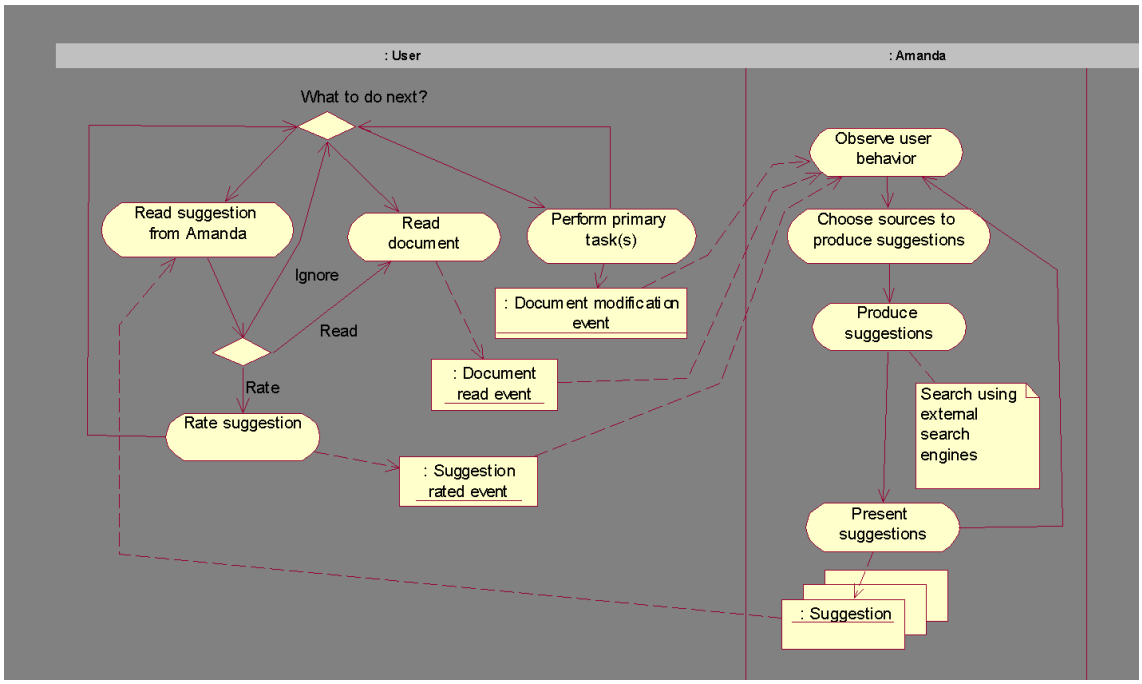


Figur 5.4: *Brukerinteraksjon fra arkitekturens perspektiv: Noe interaksjon er direkte oppgaveorientert og noe handler om hjelp og assistanse.*

Usecase 1 er detaljert videre usecase 1.1 og 1.2 under. I detaljeringene introduseres komponentene “Bucket”, “Shovel” og “Discovery”, og det beskrives meldinger som flyter mellom disse komponentene. Informasjonsflyten beskrevet i disse usecasene gir opphav til informasjonsmodellen vist i figur 5.11 side 95.

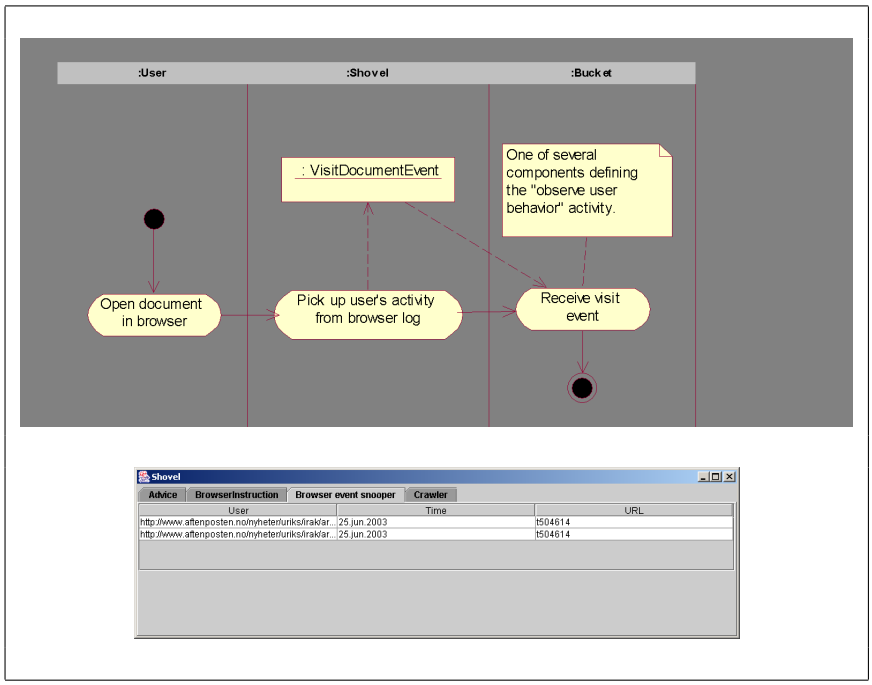


Figur 5.5: *Noe forfinet beskrivelse av brukerinteraksjon. Viser hvilke deler av denne Amanda deltar i. Alle aktører unntatt brukeren er her deler av "computer" fra figur 2.3 fra side 13.*



Figur 5.6: *Overordnede aktiviteter sett fra brukerens perspektiv, annotert med informasjonsflyt mellom bruker brukernære aktiviteter, samt Amandas kjerne.*

Usecase 1.1 Åpne dokument



Figur 5.7: Usecase 1.1: "Åpne dokument", med aktivitetsdiagram som viser informasjonsflyt, og skjermbilde fra Shovel som viser en logg av hvilke dokument-åpninger som er rapportert videre til resten av Amanda. I shovelvinduet vises det kun testdata, og videre er "user" og "URL" etikettene ved en feil byttet om.

Primær aktør: Bruker

Øvrige aktører: Browser, Bruker-assistentprogram (Shovel), Amanda Systemtjener (Bucket), Browser instruksjonsapplet (Discovery).

Forbetingelser:

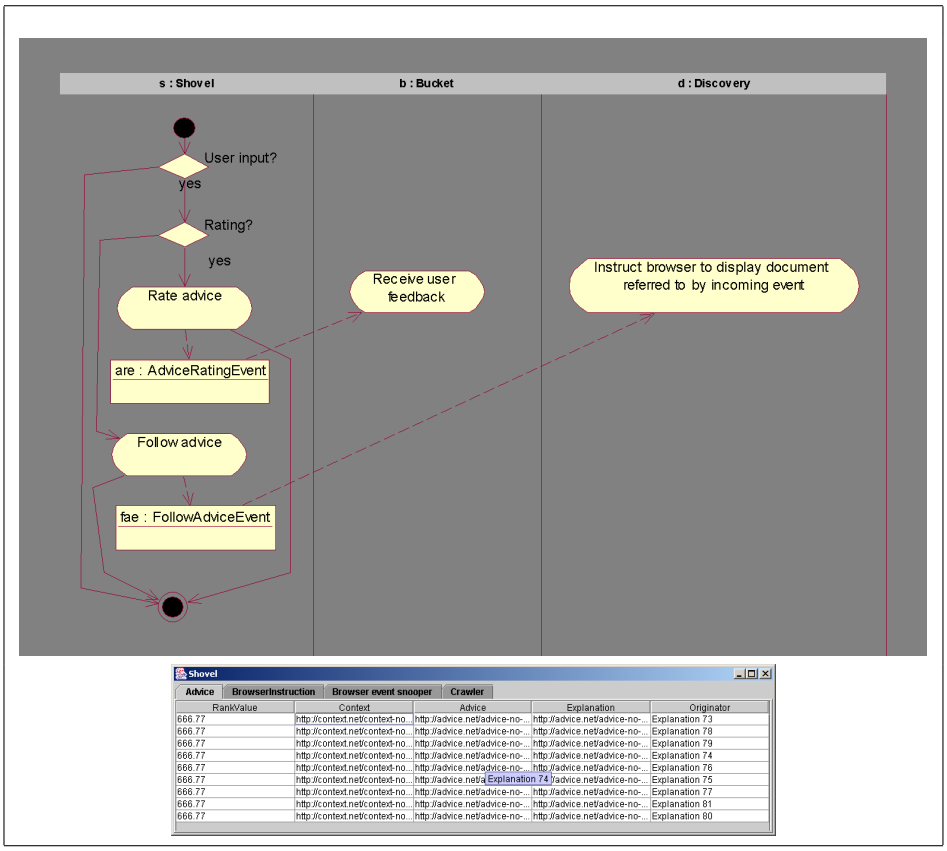
Postkondisjoner: Browser viser det dokumentet brukeren valgte å se. Bucket er informert om brukerens valg.

Steg:

1. Brukeren velger å se et dokument gjennom browserens brukergrensesnitt eller ved å følge et råd fra Shovel.
2. Browseren henter det aktuelle dokumentet og viser det frem.
3. Browseren logger hvilket dokument det har lest
4. Shovel leser browserens logg og ser at den har lest et dokument.

5. Shovel informerer Bucket om hvilket dokument brukeren har bedt om å få se.

Usecase 1.2 Bruk råd gitt i Shovel (Follow Advice)



Figur 5.8: *Usecase 1.2: Bruk råd gitt i Shovel (use advice). Med overordnet aktivitetsdiagram og detaljering av brukerens valg, samt skjermbilde fra Shovel som viser råd som vises frem for bruk. Det er også her kun testdata som vises i vinduene. Kontekst, råd og forklaringer er URLer, når man drar pekeren over en linje popper det opp en mer omfattende forklaring ("Explanation 74" i dette eksempelet). RankValue er i eksempelet konstant, men skal indikere hvilken rangering rådet får ut fra kunnskap om agentens troverdighet, og agentens egen tiltro til rådet de gir.*

Primæraktør: *Bruker*

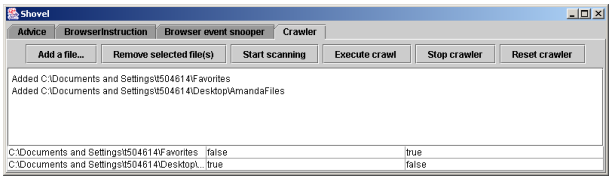
Øvrige aktører: *Bruker-assistentprogram (Shovel), Amanda Systemtjener (Bucket), Browser instruksjonsapplet (Discovery).*

Forbetingelser: *Alle aktørene tilgjengelige.*

Postkondisjoner:

Steg:
1. *Dersom Shovel mottar oppdatering og råds-status, ta hensyn til dette i hvordan råd vises frem.*
2. *Dersom bruker velger å gi en tilbakemelding, send resultatet av dem til Bucket*
3. *Dersom bruker velger å følge ett av rådene Shovel har gitt instruerer browseren om å vise frem dokumentet som rådet refererer til*
4. *Når browseren mottar instruksjonen, utføres usecase 1.1.*

UC 2. Konfigurer crawling



Figur 5.9: Skjermbilde for crawler-konfigurering fra Shovel. "t504614" er mitt brukernavn på Telenor, og derfor på maskinen jeg brukte når denne oppgaven ble skrevet.

Nr/Navn: *2.Konfigurer crawling*

Primær aktør: *Bruker*

Øvrige aktører: *Shovel og Bucket*

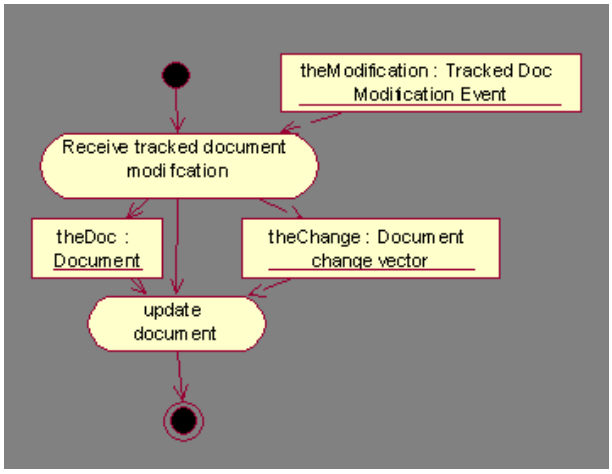
Forbetingelser: *Aktører operative og kan kommunisere.*

Postkondisjoner: *Crawlerkonfigurasjon endret i henhold til brukerens*

Steg:

1. *Brukeren velger crawler konfigurasjonsmodus i Shovel.*
2. *Brukeren redigerer konfigurasjonsparametere.*
3. *Brukeren instruerer Shovel om å gjøre endringene aktive.*
4. *Shovel sender konfigurasjonen videre til alle crawlerkomponenteter.*
   (a) *Shovel sender konfigurasjonen til sin interne crawler.*
   (b) *Shovel sender konfigurasjonen til Bucket.*

5.1.2 Usecase 3: Oppdater et sporet dokument



Figur 5.10: *Usecase 3: Oppdater et sporet dokument. Alt dette foregår inne i bucket*

Primær aktør: *Ekstern aktør.*

Forbetingelser: *1. Ekstern aktør har aksess til å endre sporet dokument*

Postkondisjoner: *1. Sporet dokument er endret i henhold til en ekstern aktørs spesifikasjon.*

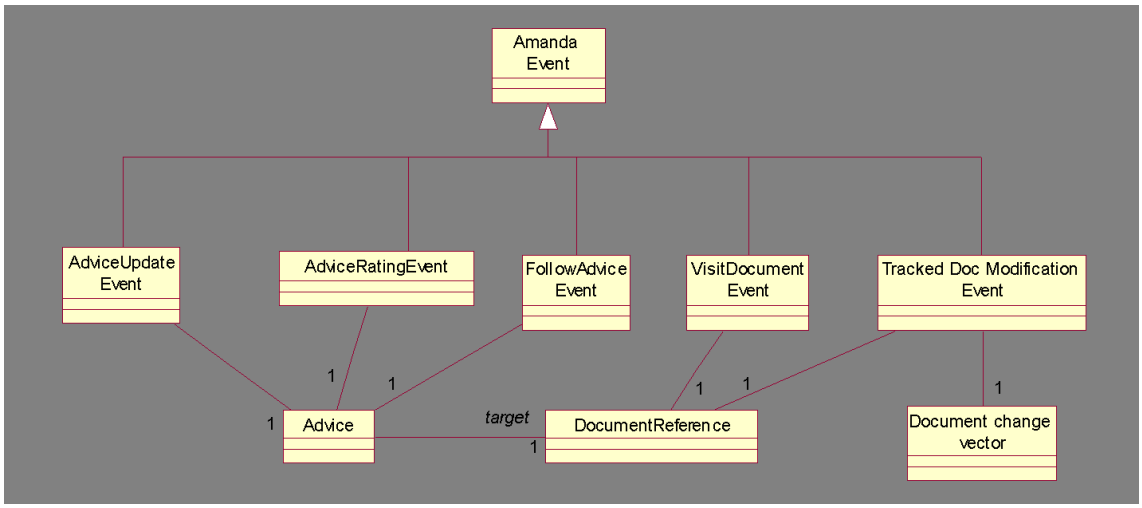
2. Endringen i dokumentet er registrert av Bucket.

Steg: *1. Ekstern aktør utfører endring og lagrer endret dokument.*

2. Shovel plukker opp spor av endring (gjennom Crawling eller på annet vis, f.eks. ved å lytte på hendelser i filsystemet.)

3. Shovel sender melding om endring til Bucket.

4. Bucket tar i mot og registrerer endringen



Figur 5.11: *Informasjonsmodell utledet av interaksjoner i usecasene vist i figur 5.6*

5.1.3 Overordnet informasjonsmodell for brukerinteraksjon

I figur 5.11 vises en overordnet informasjonsmodell som er utledet av aktivitetsdiagrammene som ble brukt for detaljere de overordnede use-casene.

Med denne modellen på plass er vi ferdige med den overordnede beskrivelsen av hva Amanda gjør. Resten av beskrivelsen vil ta utgangspunkt i aktiviteter hentet fra aktivitetsdiagrammene i figurene over, og detaljere hva som skjer internt i de forskjellige av Amandas komponenter.

Vi vil i den videre presentasjonen ikke bruke use-caser, men konsentrere oss om aktivitets og klassediagrammer, og i noen ganske få tilfeller vise frem programkode for å illustrere hvordan aktivitetene implementeres.

5.2 Transportkomponent

5.2.1 Hvorfor en egen transportkomponent?

Transportkomponenten bruker et nettverk av egne meldingsrutere til å sende informasjon mellom de forskjellige delene av Amanda. Meldinger sendes til "subjekter" som kan ha et vilkårlig antall lyttere. Transportkomponenten er et arkitekturelement som kan brukes til å spre deler av implementasjonen over flere fysiske maskiner og flere eksekveringskontekster innen samme maskin, og likevel være i stand til å fordele informasjon på en rask, enkel og konsistent måte.

Siden agentene adresserer hverandre gjennom transportlaget er det også enkelt, og for agentenes del helt transparent å rekonfigurere hvilke eksekver-

ingskontekster de skal kjøres fra. Det er f.eks. mulig å kjøre nesten alle brukernære og bakenforliggende agenter i samme eksekveringskontekst (noe som ble gjort tidlig i utviklingsprosessen), eller man kan velge å ha en eksekveringskontekst der man samler alle de bakenforliggende agenter, og så et par andre der man plasserer de brukernære (slik status er nå med oppdelingen i "Discovery", "Shovel" og "Bucket"). Om systemet blir utvidet til å håndtere flere brukere eller flere slags måter å overvåke brukernes atferd på kan man så øke antall eksekveringskontekster med brukernære kontekster men bevare den samme bakenforliggende struktur.

Alle objektene som flyttes rundt mellom de forskjellige toppnivåkomponentene i Amanda er beskrevet i klassediagrammet i figur 5.11 på side 95. Designet åpner for å kunne flytte vilkårlig mange typer meldinger. Det eneste kriteriet som må oppfylles for at et java objekt skal kunne sendes over transportkomponenten er at det implementer javas "Serializable" interface[4].

I implementasjonen er det laget et transportlag basert på TCP/IP og serialiserte javaobjekter. Fordelen med en slik implementasjon er at den er forholdsvis enkel å implementere, og ikke krever store maskinressurser, og kan kjøre på alle slags maskiner som kan kjøre J2SE (Java 2 Standard Edition) programmer. Ulempene er at det er problematisk å samarbeide med agenter som ikke er bygget over nøyaktig denne lesten, og at arbeid som er gjort rundt liknende teknologier (f.eks. (Java Message Service) [24]) ikke er direkte anvendbart. Følgelig er implementasjonen av transportlaget resultatet av en designavveining. I korthet falt vurderingen ut til fordel for egenimplementasjon siden dette bille gi maksimal frihet, noe som er svært viktig i et ekperimentelt system som Amanda er. Med dagens implementasjon er det enkelt å lage eksekveringskontekster i de aller fleste Java 2 implementasjoner, fra store Enterprise systemer basert på J2EE og ned til MIDP-2 [48] baserte systemer f.eks. i mobiltelefoner. Siden implemenasjonslaget ikke har en stor og sofistikert mengde egenskaper, er det høyst overkommelig å lage adaptere mot andre meldingssystemer (f.eks. JMS eller MQ) dersom det skulle oppstå et behov for skalering i retning av de egenskapene slike systemer har med hensyn på volum og pålitelighet. For programmer basert på transportkomponenten kan en slik endring gjennomføres ved å la ruterfabrikken vist i figur 5.13 side 100 returnere et adapter mot den eksterne rutertjenesten i stedet for en implementasjon av det eksisterende rutersystemet.

5.2.2 Hva transportkomponenten gjør på lavt nivå

Transportkomponenten er bygd opp som et nettverk av meldingsrutere. Enhver programbit som skal bruke transportkomponenten forholder seg kun til den nærmeste meldingsruteren, så vet ruterne hvordan de skal rute meldinger seg i mellom. Nettverket mellom ruterne er organisert som et "nav og eike"

[4]Noe som normalt sett gjøres ved å deklarere at "Serializable" er implementert av klassen, og så ikke gjøre noe som helst mer dersom en ikke ønsker å definere sitt eget serialiseringsformat, noe som altså ikke gjøres i denne transportkomponenten. I praksis er det altså nær trivielt å gjøre et javaobjekt serialiserbart.

nettverk (hub and spoke), hvor det finnes nøyaktig ett nav, og vilkårlig mange eiker. Informasjonsmodellen til ruteren finnes i figur 5.12.

Sending av meldinger: Meldingene som rutes sendes til "subjekter", og programkomponenter kan registrere seg som lyttere til disse subjektene. Når en melding sendes til et subjekt vil alle lyttere til dette subjektet som er tilknyttet en hvilken som helst ruter i nettet motta de samme meldingene. Rekkefølgen de mottas i behøver imidlertid ikke være den samme.

1. En melding sendes til en ruter, adressert til et subjekt. Når meldingen er sendt til ruteren returneres kallet som sendte meldingen. Ruteren har sin egne ekseveringstråder som forestår videresending av meldinger.
2. Ruteren ser på subjektet til meldingen, og sender meldingen til alle lokale lyttere for dette subjektet.
3. Dersom hub-ruteren har sagt fra at den er interessert i meldinger på dette subjektet mottar også den meldingen. Når den gjør det, vil den ta i mot meldingen, og sende den til alle de lokale lytterne på det aktuelle *unntatt* dersom det er slik at den lytteren som sendte meldingen inn til hubben også abonnerer på meldinger på de samme subjektet. Når ruting er satt opp riktig vil denne regelen være tilstrekkelig for å unngå ruting-looper.

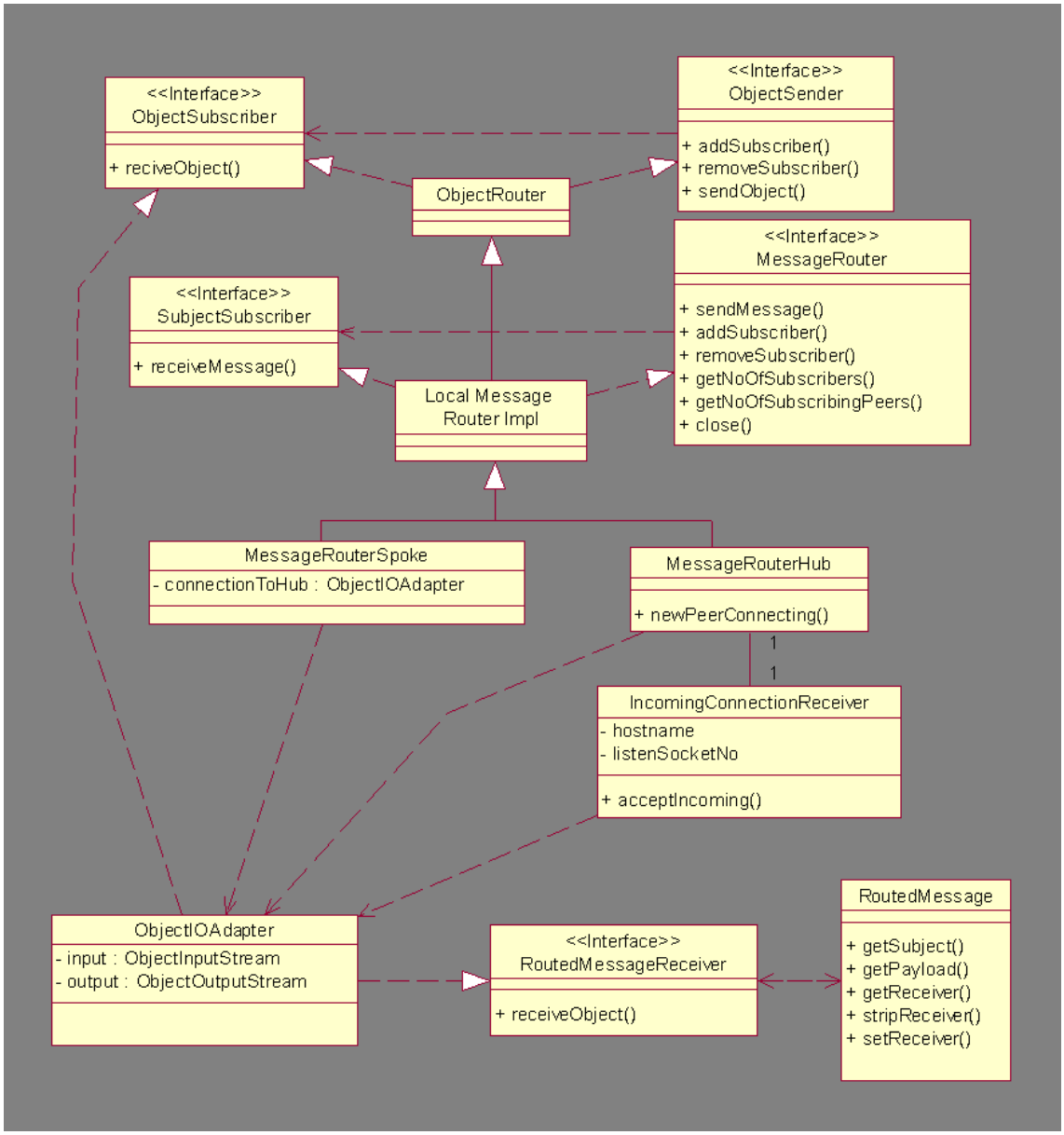
Oppkobling av rutere: Når en ny ruter kobler seg på nettverket skjer følgende:

1. Dersom ruteren er en hub, starter den en prosess som lytter etter innkomne forbindelser på en bestemt port, når det kommer nye forbindelser inn skjer følgende:
   (a) Det opprettes et nytt "ObjectIOAdapter" objekt som håndterer flytting av objekter til og fra den oppkoblende ruteren. Det antas at det i andre enden av forbindelsen også finnes et ObjectIOAdapter.
   (b) ObjectIOAdapter - objektet sender en melding på et spesielt "ADMIN" subjekt til den oppkoblende ruteren. Denne meldingen informerer ruteren på den andre siden om at den nettopp er blitt koblet opp mot en annen ruter. De to ruterne utveksler så meldinger om hvilke subjekter de abbonerer på, slik at meldinger kan rutes til riktig sted.
   (c) ObjectIOAdapteret behandles så som en hvilken som helst annen mottager av meldinger.
2. Dersom ruteren er en spoke-ruter, vil den under oppstart ha fått en parameter som forteller den hvor den kan finne hub-ruteren sin.
   (a) Spoke-ruteren oppretter derfor et ObjectIOAdapter objekt som inneholder en forbindelse mot et annet ObjectIOAdapter i hub ruteren.

   (b) De to utveksler informasjon på ADMIN subjektet om hvilke subjekter de allerede abbonerer på, slik at meldinger kan rutes riktig.
   (c) ObjectIOAdapter objektet behandles etter dette som en hvilken som helst annen mottager av meldinger den abbonerer på.

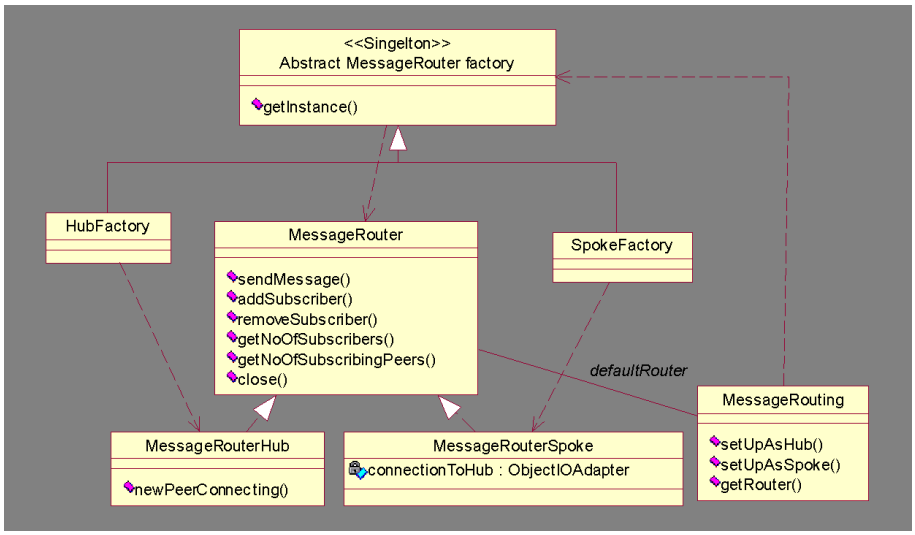
3. Når en ruter mottar en melding på et annet subjekt enn "ADMIN" gjøres dette ved at meldingen sendes til alle mottagere av subjektet. Hva disse velger å gjøre er opp til dem. Dersom dette er lokale lyttere som er registrert fra yttersiden av ruteren, vil de antakelig motta meldingene og gjøre noe med dem. Dersom de er ObjectIOAdaptere, vil de videreformidle meldingene til andre rutere, dog på en slik måte at en melding aldri sendes tilbake til den ruteren den kom fra[5]
4. Når en ruter mottar en melding fra en klient om at den skal abonnere på et subjekt, sjekkes det først om ruteren allerede kjenner til subjektet, hvis så er tilfelle legges klienten i mengden av mottagere for subjektet. Hvis subjektet ikke er kjent fra før, opprettes en ny mengde av mottagere for subjektet, og det sendes en melding over ADMIN subjektet om at det nå finnes en lytter for det aktuelle subjektet. Slike "ADMIN" meldinger sendes dog ikke rutere som allerede abboner-er på det aktuelle subjektet, dette for å unngå at det oppstår ADMIN meldinger som sendes frem og tilbake uten å bidra til at noe nytt skjer.
5. Når en ruter mottar en melding fra en klient om at den skal slutte å abonnere på et subjekt, fjernes klienten fra mengden av mottagere for subjektet. Dersom mengden etter denne fjerningen er tom, sendes en melding over "ADMIN" subjektet til alle tilkoblede rutere om at det ikke lenger er nødvendig å sende meldinger om det aktuelle subjektet til denne ruteren.
6. Når en ruter kobler seg kontrollert av, vil den først fjerne sine abonnementer for alle subjekter, deretter vil den lukke nettverksforbindelsen.
7. Når et ObjectIOAdapter objekt merker at nettverksforbindelsen den lytter på går ned, vil dette tolkes som om ruteren i den andre enden har koblet seg ned, og det vil ikke lenger prøves å sendes meldinger til den.

[5] Internt skjer dette ved at når meldinger kommer inn gjennom et ObjectIOAdapter tar meldingen med seg en referanse til dette objektet når det fraktes gjennom ruteren. Når et ObjectIOAdapter skal sende en melding videre sjekker det så om den opprinnelig kom fra seg selv, og om så er tilfelle unnlater den ganske enkelt å sende meldingen videre.



Figur 5.12: *Informasjonsmodell for transportkomponent*

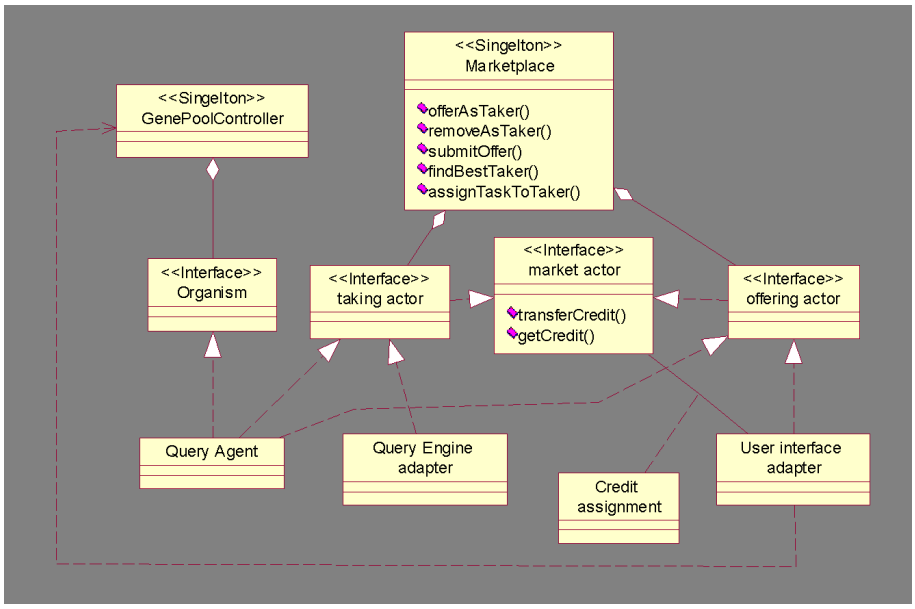
Beskrivelse av API for flytting av Amanda-meldinger



Figur 5.13: *Ruterfabrikk*

Da jeg var ferdig med å skrive selve meldingsruteren ønsket jeg ytterligere å forenkle bruken av den. Det gjorde jeg ved å pakke inn oppstart av rutere i singeltonobjekter singeltonklasse[22, s. 127] som implementerer et abstrakt fabrikk-pattern for ruterobjekter ([22, s. 87]) som automatisk starter en hub eller spoke ruter med riktige parametere (vist i figur 5.13). Videre introduserte jeg "kanaler" (se figur 5.14) som hver frakter en bestemt type objekter over et bestemt subjekt. Det finnes en type kanal for hver av meldingstypene i Amanda (AdviceUpdateEvent AdviceRatingEvent etc fra figur 5.11 på side 95 etc.). Kanalene har en "receivePayload" metode som gjennom subklassing kan spesialisers til å gjøre hva som helst med innkomne meldinger (mottak må fremdeles aktiveres gjennom et eksplisitt kall på "subscribe" metoden). Når et program nå ber MessageRouting om å få en referanse til en ruter, er det til den riktige typen ruter, satt opp på riktig vis. AbstractRemoteChannel bruker så MessageRouting for å få tak i en ruter når det trenges. I AbstractRemoteChannel bestemmes det et bestemt topic det skal abbonerer på og sendes til. For å ta i mot meldinger reimplementeres metoden defaultPayloadReceiver i en subklasse, for å sende brukes metoden "send".

Alle detaljer med oppkobling av rutere, eksplisitt adressering av subjekter er med dette innkapslet i infrastruktur-klasser, og bruken av meldingssystemet blir nå nærmest triviell, som vist i figur 5.2.3 som viser et utsnitt av koden som implementer Discovery appleten.



Figur 5.14: *Informasjonsmodell transportkanaler*

5.2.3 Discovery appleten

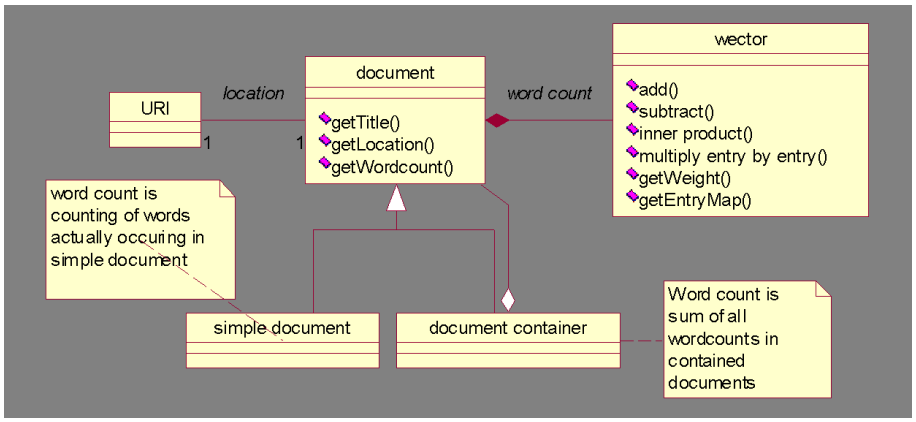
```
public class Discovery extends JApplet {
  ...
  public class AppletMessageReceiver extends UserAdviceChannel {
     public void receivePayload(Object payload){
        try {
                URL url = ((URI))uri.toURL();
                getAppletContext().showDocument(url, "AmandaAdvice");
        }
        catch (MalformedURLException e){
               System.out.println("Malformed URL" + e + " Ignoring it"); }
     }
  }
  ...
}
```

Figur 5.15: *Kjernen i appleten som viser frem brukerens valg*

Discovery-appleten er nå enkel å beskrive, og vi velger å gjøre det som et eksempel på bruk av transportkomponenten. Som vist i figur men bevare den samme bakenforliggende struktur5.2.3 lytter men bevare den samme bakenforliggende strukturappleten på `UserAdviceChannel` kanalen ved å implementere receivePayload metoden. Hver gang det kommer en melding der, plukker den ut URIen til dokumentet som refereres, og instruerer browseren appleten kjører i om at dokumentet URIen peker på skal vises frem.

5.3 Crawler

Crawleren er komponenten som henter inn dokumentprofiler, lagrer disse, og sender dem videre til agenter som måtte være interesserte i dem. Vi har allerede sett en del av crawleren, nemlig den som er beskrevet i figur 5.10 side 94.



Figur 5.16: *Klassediagram som viser hva som lagres om scannede dokumenter.*

Scan-frekvens: For å identifisere endringer i dokumenter, har Shovel-komponenten en prosess som med faste mellomrom traverserer bestemte filkataloger hos brukeren og lager profiler av dokumentene den finner der. Dersom det oppdages endringer i profilene sendes disse avgårde.

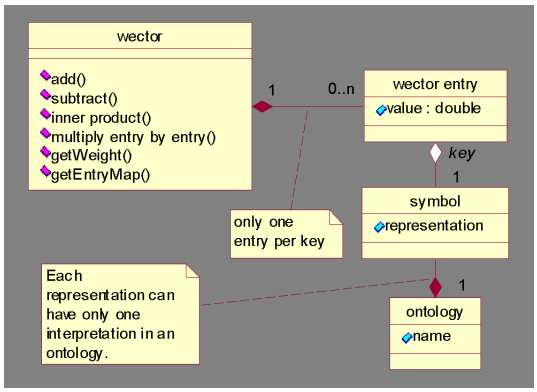
Dagens implementasjon har to kataloger som traverseres med forskjellige intervaller. Brukerens "favorites" katalog (for browseren) traverseres en gang i mellom (for tiden typisk gang om dagen), for å hente ut profiler av alle dokumentene favorites listene refererer til. I tillegg traverseres en katalog med navn "Amanda" i brukens hjemmekatalog en gang i minuttet. Det er opp til brukeren å sørge for at det ikke er flere filer i Amanda-katalogen enn at crawleren klarer å profilere dem alle sammen i løpet av et minutt.

Crawleren i Shovel sender profiler over til Bucket (som vist i figur 5.10). Datastrukturen til profilene er detaljert i figur 5.16, men det kan allerede nå nevnes at de inneholder referanser til alle eksterne dokumenter som refereres. Når en slik dokumentprofil ankommer i Bucket, vil Bucket legge referansen til det eksterne dokumentet på en kø for crawling, og før eller senere vil innholdet i det eksternt refererte dokumentet hentes inn.

Det er altså en streng arbeidsdeling: Shovel henter inn dokumenter fra lokalt fillager, og Bucket henter dokumenter fra internettet.

Dette betyr at filer med innhold som er hentet fra eksterne dokumenter, f.eks. HTML dokumenter med frames, eller ".URL"[6]-filer. Når et slikt dokument scannes av Shovel, vil kun URLene som referer til innholdet leses, selve innholdet leses ikke (siden innholdet altså ikke finnes i .URL filen, men på en webtjener ett eller annet sted). Profilen som sendes over til Bucket er derfor ganske tom for innhold som sier noe om hva som er lagret i dokumentet. Det er først når Bucket får hentet inn innholdet at en profil basert på det refererte innholdet kan bygges.

Denne oppdelingen er hensiktsmessig av to årsaker: Først at det gjør at crawlingen som skjer i Shovel kan gjøres raskt, og uavhengig av at eksterne nettresurser utover Bucket er tilgjengelig. Videre er det slik at dersom det i Bucket allerede er lagret en profil for det eksterne dokumentet, så er det ikke uten videre nødvendig å lage en ny. Til slutt er det også slik at det ikke er gitt noen garantier for når eller om eksterne dokumenter faktisk er mulige å lese, og det er derfor hensiktsmessig å samle all prøving og feiling, og administrasjon av dette til ett sted.



Figur 5.17: *Klassediagram for "wectorer", Amandas sparse-vector implementasjon*

Datastruktur for profilene som dannes: Profilene som dannes når crawleren kjøres sendes gjennom transportlaget og ender opp inne i Bucket. Bucket vil da for hver innkomne dokumentprofil opprette et dokument-objekt med struktur som indikert i figur 5.16. Vi ser at dokumentet har assosiert en unik URI (som for dokumenter scannet av Shovel alltid er en "`file://`" URI, og for dokumenter scannet av Bucket kan være hva som helst (så lenge det er en lovlig URI). Videre kan dokumenter være men bevare den samme bakenforliggende struktur"simple", eller "composite". I dagens implementasjon er alle filer "simple" dokumenter, mens filkataloger

[6]".URL" dokumenter er dokumenter som Microsoft Internet Explorer bruker for å lagre innholdet i en "favoritt". Inne i fila ligger det en liste med URLer til dokumentet som refereres, og eventuell tilleggsinformasjon som f.eks. et lite ikon som kan brukes får å representere dokumentet.

er "composite". Dette er imidlertid ingen nødvendig begrensning: Det er fullt mulig, og meningsfylt og betrakte mange slags dokumenter som composites, og dette er en rimelig utvidelse å tenke på for fremtidige versjoner av dette eller lignende programmer. Til hvert dokument er det videre assosisert et "wector" objekt som inneholder ordtellinger, og en mengde av "link" objekter, som inneholder pekere til andre dokumenter referert som det gjeldene dokumentet inneholder referanser til.

Det finnes funksjoner (ikke vist i diagrammene) som til hvert dokument kan regne ut termfrekvenser (tf), og som til hver composite kan regne ut invers dokumentfrekvens(idf) (se seksjon3.1.1 for detaljer rundt dette). Dette betyr at enhver composite kan fungere som populasjon for utregning av tf/idf vektinger. Dette blir litt problematisk i de tilfellene et dokument ikke inneholdes av populasjonen idf hentes fra (det blir en divisjon med null i idf beregningen), men dette håndteres med en ad-hoc regel som setter tf/idf til null i dette tilfellet. Fordelen med en slik funksjonalitet er at alle composites, herunder kataloger med favorites, eventuelle tråder i epostsystemer og annet, meget enkelt kan brukes som grunnlag for en agent som forsøker å lete etter dokumenter som ligner dokumentene i populasjonen sin.

Wector-objektene er Amanda-systemets implementasjon av "sparse vector"er. Det er vektorer der det finnes et stort antall indekser, men hvor hver enkelt vektor typisk har null lagret for nesten alle indekser. Et UML klassediagram som viser hvordan en Wector er bygget opp finnes i figur 5.17. Indeksene i "Wector" objektene er laget av "symboler". Hvert symbol er kodet med en tekststreng som er bundet i en "ontologi", som for våre formål kan tolkes som et navnerom, slik at hvert ord inne i en ontologi har en unik mening.

Grunnen til at en slik representasjon ble valgt var at vi ønsket å kunne kode mange slags egenskaper ved dokumenter i en og samme datastruktur: Ordtellinger av innleste ord, beregnede egenskaper ("hvor sterkt tror agent XYZ at dette dokumentet er en hjemmeside"), bestemte tolkninger av ord "i denne sammehengen betyr 'rhesus' en bestemt apeart". En introduksjon av navnerom koblet mot ontologier virket som en hensiktsmessig abstraksjon, og under arbeidet med oppgaven har jeg ikke oppdaget noe som strider mot denne antakelsen.

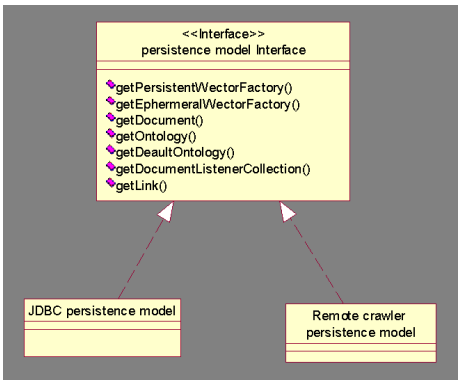
Wector objektene har noen metoder som kan utføre enkelte enkle regneoperasjoner på vektorer: Elementvis addisjon og subtraksjon og multiplikasjon samt euklidsk indreprodukt. Disse operasjonene er valgt ut fordi agentene som senere skulle gjøre beregninger basert på innholdet i wektorene trengte disse operasjonene. Dersom det i fremtiden trenges andre operasjoner er det rimelig å legge disse også inn i wectorklassen.

Propagering av endringer til omliggende "composites" : Dersom et dokument endres, og dette dokumentet er inneholdt av en composite, da vil endringen propageres til det inkluderende dokumentet. Dersom dette doku-

mentet i sin tur er inkludert av et annet dokument, da vil endringe propageres videre. Det antas at inklusjonsgrafen til compositer er asyklisk, da det ikke finnes noen sperre for propageringen. Dette er imidlertid en rimelig antagelse siden vi for tiden kun lar filkataloger være composites, og vi ikke lar linker av noe slag være del av en composite. Dersom disse forutsetningene brytes i en eventuell utvidelse må loop-deteksjon inkluderes i verdipropageringsalgoritmen.

Utløsning av videre aktivitet i agenter: Om litt skal vi se på agentarkitekturen, men allerede nå er det på sin plass å indikere at denne får betydelige mengder input fra crawleren. Når crawlerens oppdaterer en profil for et dokument, kan agenter ha meldt seg som "abonnenter" for endringer i de angjeldende dokumentene. Når en profil endres vil alle de agentene som lytter på endringer i bestemte dokumenter få en beskjed om at en endring har skjedd, og hva denne endringen innebærer. Dette skjer ved at agenten får en melding som inneholder både en referanse til dokumentet, og til endringer som har skjedd i wectoren.

5.4 Persistenskomponent



Figur 5.18: *Overordnet klassediagram for persistensmodell*

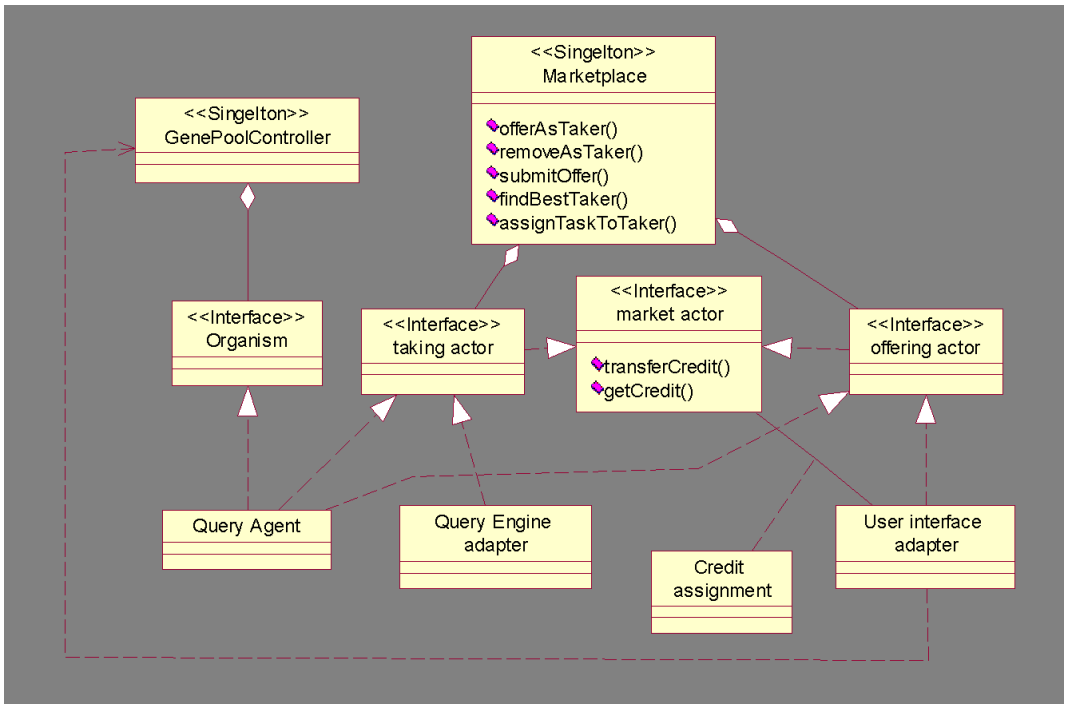
Persistenskomponentens oppgave er å sørge for at data angående dokumenter, ontologier etc. blir lagret til persistent lager. Dette gjøres ved å definere et overordnet grensesnitt for persistens, som gjengitt i figur 5.18, og så implementere dette i to klasser, en klasse som lagrer objekter i en relasjonsdatabase gjennom Javas JDBC grensesnitt, og klasse som lagrer ved å sende informasjon videre gjennom transportkomponenten i en TracedDocModificationEvent. Grunnen til at denne oppdelingen er gjort, er at crawler-komponentene i Shovel og Bucket deler mye kode, og all denne koden arbeider mot persistensgrensesnittet. Til å begynne med var det kun Bucket som hadde crawlerkoden, men etterhvert i arbeidet ble det klart at det var hensiktsmessig å dele dette arbeidet mellom Bucket og Shovel. Siden Bucket allerede arbeidet mot en abstrakt persistensmodell, var det enkelt å flytte de nødvendige delene av kodene til Shovel og så skrive en ny implementasjon av persistensmodellen som i stedet for å lagre mot en database, sender profiler over transportlaget for videre prosessering av Bucket.

5.5 En arkitektur for autonome adaptive agenter

5.5.1 Overordnet om arkitekturen

Alle komponentene vi har sett hittil, brukergrensesnitt, persistens, transport, etc. har ikke bidratt til å produsere råd til brukerne. Vi skal nå se på komponenten som produserer råd, og som bruker tilbakemeldinger gitt av brukerne til å produser stadig mer akseptable råd.

Dessverre er agent-komponenten kanskje både den viktigste delen av Amandasystemet og den delen der implementasjonen har kommet kortest. Her finnes



Figur 5.19: *Overordnet klassediagram som viser agentstrukturen*

det *kun* et design med tilhørende kjørende men bevare den samme bakenforliggende strukturalgoritmeskisser. Algoritmene er ikke integrert resten av Amanda. Det må derfor antas at agentrammeverket vil måtte endres noe under denne tilpassningsprosessen.

Figur 5.19 viser et klassediagram med overordnet struktur for agentene. Øverst i diagrammet viser en "GenePoolController" og en "marketplace". Dette er singeltonklasser som til sammen utøver kontroll over agentpopulasjoner:

- Markedsplassen "Marketplace" tilbyr et grensesnitt hvor "offering actor" agenter kan etterspørre løsninger på problemer, og hvor "taking actor" agenter kan tilby å løse bestemte klasser av problemer problemer. Alle agentene har en størrelse kalt "kreditt" som utveksles i markedsinteraksjoner: For å få utført oppgaver må agentene betale litt til de agentene som utfører oppgavene. For å få lov til å delta i markedsplassen må agenter betale "leie" i form av en "kreditt". Leien er ikke så stor, men den innebærer at agenter som over lang tid ikke produserer noe, heller ikke får anledning til å delta i markedsplassen. Dette er en ide som er hentet fra Moukas [2].

  Dersom en oppgave man har vært med på å utføre får en tilbakemel-

ding fra brukeren (positiv eller negativ), får man sin andel av "kreditt" tildelt eller fjernet. Under arbeidet med å finne frem til et råd, blir det ut fra kallgrafen mellom de forskjllige agentene (hvilke agenter som får hvilke andre til å hjelpe seg), bygget opp en graf for kredittfordeling med datastruktur som vist i figur 5.21. Når kreditt skal fordeles som følge av tilbakemeldinger, brukes denne grafen for å gi en slags rettferdig fordeling av kreditten. Markedet bidrar dermed til å fordele mest kreditt til de individene som best er i stand til å bidra til å bidra til brukerens tilfredshet, og minst til dem som ikke gjør det.

- GenePoolController kontrollerer en populasjon av agenter som optimeres gjennom genetisk programmering. Agentene må implementere interfacet "Organism" som gjør at agentene kan avgi et "genom" som kan brukes til å bygge kloner og til å persistere instanser, og som kan brukes til basis for å kombinere egenskaper fra par av individer for å lage nye individer som har egenskaper fra begge sine "foreldre". Alle organismene har dessuten en numerisk "fitness" verdi som indikerer hvor godt de "klarer seg" i den verden de er satt til å populere.

  I Amanda er "fitness" gitt ut fra hvor mye kreditt en agent som deltar i markedet har, og målet med den optimeringen som gjøres gjennom genetisk programmering derfor å sørge sørge for at agentene i populasjonen blir stadig bedre til å tilfredstille brukerens behov.

Innenfor dette rammeverket fines så tre klasser agenter:

Query Agent: Som formulerer spørringer som skal utføres av søkemaskiner. Disse agentene er enkle "screnscraper" adaptere mot søkemaskiner de gjør typisk ikke mer enn å ta i mot en liste søkeord, og levere fra seg en liste dokumenter som søkemaskinen finner frem til når den får søkeordene som parameter.

Query Engine Adapter: En agent som spør en søkemaskin, av et hvilket som helst slag innenfor eller utenfor Amanda-systemet, om et eller annet som er i stand til å besvare et spørsmål stilt av en query agent.

User interface adapter (UIA): Lytter på BrowserEventChannel for å finne ut hvilke dokumenter brukeren åpner, samt på kanal UserAdviceChannel for å finne ut hvilke tilbakemeldingen brukeren gir for de råd hun har mottatt.

- Hele tiden produseres forespørsler om å produsere relevante råd. Invarianten er at det hele tiden skal finnes viss mengde åpne forespørsler som de øvrige agentene kan arbeide med å tilfredstille.
- Når brukeren gir en tilbakemelding gjennom UserAdviceChannel, finner UIA frem til en opptegning for dette rådet som viser hvilke agenter som skal krediteres for tilbakemeldingen, og ut fra denne opptegningen gis det så tilbakemeldinger.

- Når UIA ser at brukeren ser på et dokument gjør den to ting:
  1. Plasserer et oppdrag i markedsplassen for å produsere profil til dokumentet.
  2. Når profiler mottas, brukes disse til å produsere søkeagenter spesiallaget for å finne dokumenter som er spesielt relevante i forhold til det nylig åpnede dokumentet. Disse agentene produseres ved å først syntetisere "genetiske koder" som er spesialtilpasset de nyåpnede dokumentene, og så instruere GenePool-Controlleren om å sette disse nyskapte agentene inn i populasjonen av Query agenter.

  Ved utvidelse av Amanda til å forstå flere typer brukerinput, vil UIA være et naturlig sted å plassere mange av disse utvidelsene.

Den videre fremstillingen av agentsystemet er delt i to, først en som beskriver rammeverket og generiske interaksjoner i dette, og en del som beskriver de konkrete "query", "query adapter" og "user interface" agentene.

Det er altså valgt å håndprogrammere agentene som har direkte interaksjon med brukergresesnitt og søkemaskiner. Dette er ikke fordi det er umulig å parametrisere disse og la dem være gjenstand for optimering gjennom genetisk programmering, men fordi det ikke er åpenbart at slik optimering vil bidra til å finne frem til flere relevante dokumenter enn ved å kun la søke-agentene optimeres. Dette er naturligvis kun en påstand fra min side og den kan vise seg å være feil, men selv om den er feil, vil det ikke være feil å begrense kompleksiteten i agentsystemet til å begynne med, og *det* tror jeg vil bli gjort ved å begrense de forskjellige artene av agenter som delta i "genpoolen".

Markedsplassen

Agenter som ønsker å få utført en oppgave setter inn et bud på markedsplassen (overordnet datastruktur vist i figur 5.20) . Når de får resultater tilbake kan de gi tilbakemeldinger om hvor godt fornøyde de var med resultatet. Det kan gå lang tid mellom når et råd gis og når det gis tilbakemelding, det er derfor nødvendig å huske ganske mye om hva som er presentert for brukeren og hvem som bidro til å få det presentert.

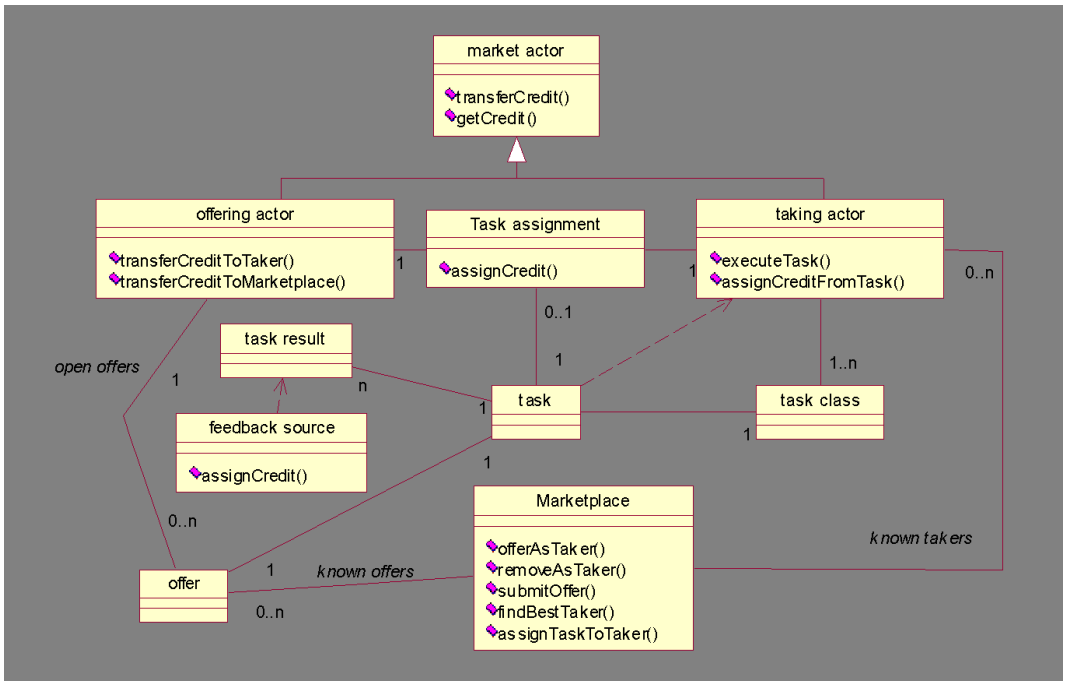
Brukergrensesnittet oppfører seg som en agent som stadig ber om å få råd, og gir tilbakemeldinger om hvor godt fornøyd det var for å få rådet.

Agenter som er i stand til å utføre oppgaver registrerer seg på markedsplassen.

Alle agenter har en mengde "kreditt" som antas å reflektere hvor gode agentene har vært til å tilfredstille omgivelsens behov.

Når en agent får tildelt et oppdrag, får den også tildelt noe "kreditt" fra den agenten som skal ha oppdraget utført. En agent uten egen kreditt kan dermed ikke få utført oppdrag av andre.

Å utføre et oppdrag kan innebære at andre deloppdrag må utføres. Dette utføres i så tilfelle også gjennom markedsplassen. Når oppdrag utføres for å



Figur 5.20: *Overordnet klassediagram for markedsplassen*

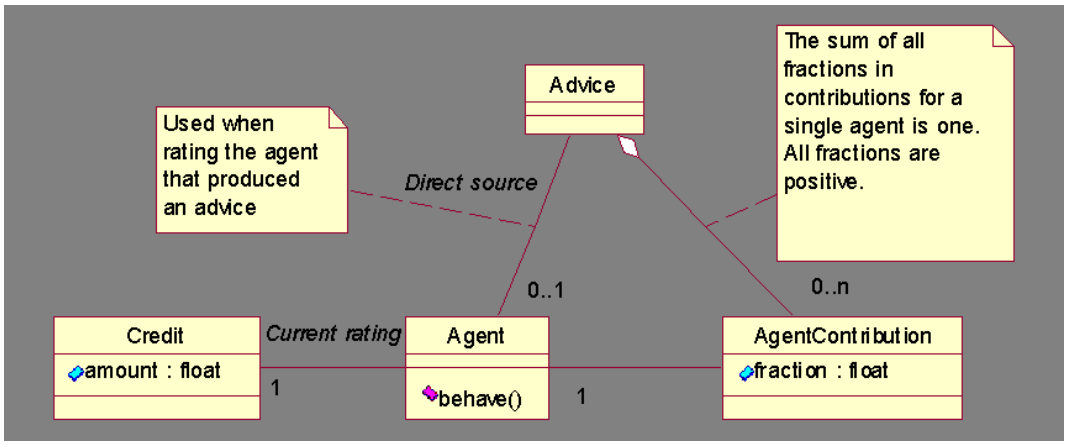
nå et mål, bygges det opp en "krediteringsgraf" (se figur 5.21 som viser hvor mye forskjellige agenter har bidratt til et resultat. Når det gis en tilbakemelding (god eller dårlig), kan positiv eller negativ kreditt fordeles på en "rettferdig" måte, eller i det minste slik at agenter som bidrar til atferd som belønnes, også ender opp med mer kreditt.

Markedsplassen bruker en algoritme for å matche tilgjengelige oppdrag med tilgjengelige agenter. Algoritmen er designet for å balansere hensynet til best mulig resultat på kort sikt mot behovet for å prøve ut nye metoder som kan gi bedre resultater over tid og flere kriterier for å matche oppdrag mot utførende agenter:

Tilfeldig: Et oppdrag tildeles en tilfeldig agent som har registrert seg som å være i stand til å utføre oppgaven

Optimerende: Ved å la agenter med mye kreditt først få vurdere i hvilken grad de "ønsker" å få et oppdrag vil agenter som tidligere har klart å levere gode resultater få anledning til å prøve igjen, og om mulig fortsette sin gode trend.

Dupliserende: Samme oppdrag kan gis til flere agenter utvalgt etter forskjellige kriterier, og resultatene som produseres kan konkurrere mot hveran-



Figur 5.21: *Graf som viser fordeling av kreditt*

dre for gode tilbakemeldinger.

Når en agent har fått tildelt et oppdrag, utfører det oppdraget, og returnerer resultatet til agenten som ba om å få oppdraget utført.

Når et resultat av et oppdrag vises frem til brukeren, kan brukeren gi tilbakemelding. Tilbakemeldingen kan være positiv eller negativ, men uansett hva den er blir den "kreditt" tilbakemeldingen gir, fordelt over alle agentene som deltok i å produsere resultatet. Dersom ingen tilbakemelding gis gis hverken positiv eller negativ belønning, men dette vil likevel innebære en viss, svak negativ belønning siden de mange av de deltagende agentene har vært nødt til å bruke opp noe av sin "godhet" for å i det hele tatt produsere resultatet.

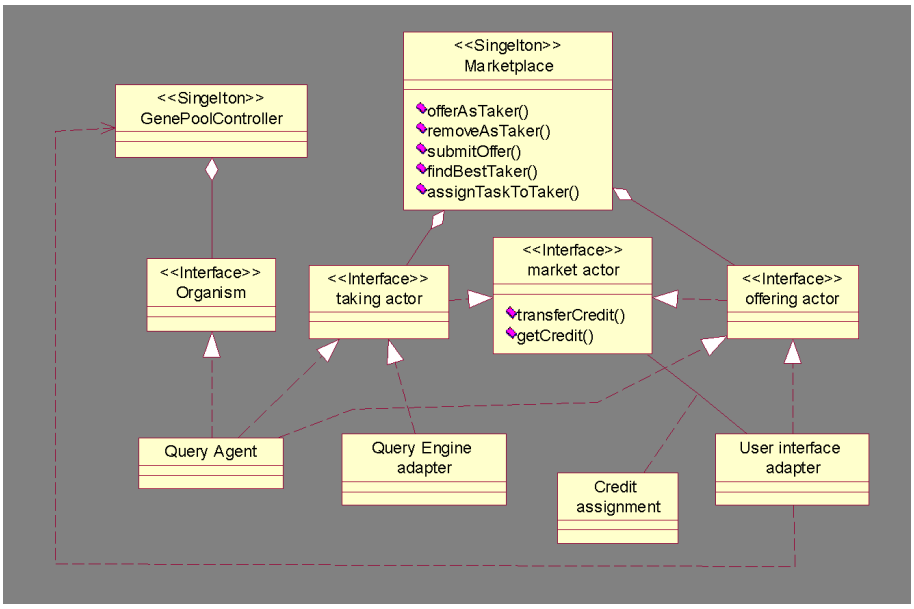
Infrastruktur for genetisk programmering

I figur 5.22 viser et klassediagram som viser infrastrukturen for genetisk programmering. Øverst til venstre er en "GenePoolController" som inneholder referanser til en populasjon av organismer og en fabrikk for nye organismer.

Populasjonen inneholder referanser til et antall organismer, og gir dessuten svar på spørsmål om enkle statistiske data om populasjonen: Hva er største og minste, snitt og median av fitness?

Organisme fabrikken produserer nye individer ut fra et genom.

- Det er meningen fabrikken skal brukes til å fylle populasjonen når genomer hentes inn fra persistent lager.
- Fabrikken skal også kunne lage nye individer ut fra en kombinasjon av to foreldres genomer, og en "mutasjons kontrollparameter" som beskriver i hvilken grad forskjellige mutasjonsmekanismer skal kunne brukes når genmateriale fra foreldrene kombinerres.



Figur 5.22: *Overordnet klassediagram for genetisk programmering i Amanda*

Genpool-kontrolleren lar organismene gå gjennom en syklus av liv og død: Når agentene har levd en periode, fjernes de dårligste av dem fra populasjonen, og nye skapes. Nye organismer skapes ved å velge ut par ("BreedingCouple" i modellen i figur 5.22) som skal parre seg. Når par velges ut kan i utgangspunktet alle organismer i populasjonen parres med alle andre organismer, og i en viss andel av parene som dannes er det nøyaktig slik rå tilfeldighet som styrer. I de fleste tilfellene er det imidlertid slik at jo større fitness en organisme har, jo større sannsynlighet er et for at den parrer seg med en annen organisme med høy fitness. De nyskapte organismene har til å begynne med fitness som er gjennomsnittet av de to foreldrenes fitness, men etterhvert vil jo denne endre seg.

Det er genpool kontrolleren som selv bestemmer hvor ofte, og på hvilken måte populasjonen skal reguleres. Systemet som helhet er nødt til å trå varsomt for å balansere mellom diversitet og lokal optimering. For stor grad av eksperimentering vil kunne få en stor andel av forslagene som presenteres for brukeren til å virke irrelevante, men dersom man fullstendig ignorerer eksperimentering er det en stor risiko for at systemet som helhet havner i et lokalt optimum og ikke gir de best mulige forslagene til bruken.

For å kode genomet til tenkes følgende struktur brukt:

```
public class AmandaSearchGenome implements Genome {
  Wector              populationTermFrequencies;
  SearchTermGenerator gen;
```

```
  // Genetic operators:
  // Combine, mutate, crossover etc. not included here
  // for brevity.
}
```

Genomet er her kodet som et par av en `Wector` og en søketerm-generator.

Kombinasjon av genomer utføres ved å gjennomføre forskjellige kombinasjons kombinasjon på likt navngitte deler i genom-recordet, så populationTermFrequencies i ett individ kan kombineres med med det samme feltet i et annet individ. Det skal finnes en parameter som styrer med hvilken sannsynlighet man gjør crossover, og hvor ofte mutasjoner skjer.

`Wector` tolkes her som en mengde av entries, sortert på leksikalsk ordning av symbolene som indekserer wektorens verdier. De to Wectorene traverseres i parallell, og fra tid til annen gjøres crossover, og andre mutasjoner, som innsetting av tilfeldig valgte symboler.

`SearchTermGenerator` er i utgangspunktet tenkt statisk, så kombinasjon av to genomer vil her bety at man for det nye individet velger generator tilfeldig fra en av foreldrene.

Det er her verd å gjøre seg to observasjoner:

1. Teknikken som er skissert over med applikasjon av kombinasjons og mutasjons - operatorer internt i de delene av genom-"recordet" som tillater det (som for Wectorer), og enkelt valg mellom de delene som ikke tillater "genetiske kombinasjoner" (som for SearchTermGenerator), er helt generisk, kan anvendes på alle record-aktige datatyper, og krever kun en traversering gjennom hvert av recordene som skal delta [7]

   ```
   public interface Genome {
       public void combine(Genome                genome,
                           MutationControlParams mcp);
   }
   ```

2. Det er også tenkelig at man kan gjøre kombinasjoner gjennom to traverseringer gjennom genomet: En traversering finner frem og klassifiserer de forskjellige delene av genetisk materiale som finnes i genomet,

[7] Siden dette er svært implemetasjonsnært velger jeg å sid et i en fotnote, men siden det er en ganske lekker teknikk velger jeg likevel å ta den med: Ved å introdusere et interface "Genome" kan man gjennomføre kombinasjoner f.eks. gjennom Javas "reflection" grensesnitt. Reflection- grensesnittet gjør at java-programmer kan "åpne opp" objekter, se hvilke atributter de inneholder, og teste hvilke typer disse atributtene har. Dersom et Genome inneholder komponenter som også implementerer "Genome", kan genetisk rekombinasjon gjennomføres rekursivt, og uten at det er nødvendig å lage eksplisitte lister av hvilke deler av objektet som skal inngå i genomet siden refleksjonen håndterer dette.

I essens blir dette en implementasjon av både funksjon og terminalsett [4, s. 110] for genetisk programmering gjennom bruk javas Reflection. Det skilles ikke mellom funsjoner og terminaler før objektene skal interpreteres, og det eneste som da kreves er at det finnes en interpret som er villig til å akseptere ethvert objekt som implementerer Genome interface.

andre traversering som gjennomfører kombinering. Med en slik variant vil mer "radikale" kombinasjoner kunne gjennomføres, så et større rom av mulig atferd vil kunne generes, men det er samtidig mulig at at konvergens mot ønsket atferd blir langsommere. Muligens kan man også kombinere de to typene kombinasjon.

Når brukeren viser interesse for et dokument produseres en mengde nye søkeagenter hvor populasjons termfrekvensen er satt ut ordfrekvensen til dokumentet.

Den originale populasjonen av agenter lages ved å scanne gjennom brukerens "favorites", og lage en mengde agenter pr. katalog i denne dokumentmengden.

Antallet agenter som lages på denne måten vil kunne bli meget stor, og jeg tror ikke det er noen ulempe å la den aktive mengden agenter være så stor som maskinressurser tillater. Mukas rapporterer at Amalthea [2] til enhver tid hadde flere hundre agenter som konkurrerte.

Til å begynne med er tre aloritmer implementert. De to første er variasjoner over termfrekvens/invers-termfrekvens (tf/idf) hentet fra Rhodes [56]: Savants første algoritme [56, s. 75], Savants "Okapi" implementasjon [56, s. 76]). Den siste algoritmen samt en egenutviklet triviell, men svært rask algoritme som ganske enkelt velger ut et antall ord (ett til tre) hvor sannsynligheten for at ordet velges er proposjonal med ordets frekvens i populasjonen.

Gjennom denne parametriseringen er det mulig å plugge inn mange andre slags algoritmer også, men uten eksperimentering er det vanskelig å se hvilken populasjon av algoritmer som vil fungere best, så har jeg valgt å starte med en liten populasjon for å få de basale mekanismene til å fungere.

Uavklarte punkter

De initielle testene som er gjort under arbeidet med algoritmeskissene indikerer at tf/idf-algoritmene kan trenge lang tid å kjøre: Det er mange og lange vektormultiplikasjoner som skal gjøres før resultatene kan frembringes. Et par mulig svar på denne type problem er å cache ekstensivt og oppdatere inkrementelt. Detr er skrevet kode for begge typer løsning, men det er foreløbig uklart hvilken kombinasjon som vil vise seg å være den beste.

Slik systemet er beskrevet nå er det tenkt brukt kun i interaktiv modus, med brukeren ved tastaturet og rådgivings-algoritmene kjørende i parallell. Dette er ikke en helt realistisk situasjon, for det vil finnes tider hvor brukeren ikke er ved tastaturet, og det er da rimelig at programmet forsøker å produsere råd som kan hentes ut raskt når brukeren faktisk er ved tastaturet.

En rimelig utvidelse er derfor å introdusere agenter som traverserer strukturen av overvåkede dokumenter, og som til hvert eneste dokument forsøker å tilordne en mengde "prefabrikerte råd" som lynraskt kan hentes ut når brukeren faktisk viser interesse for et bestemt dokument. En slik taktikk kan implementeres ved å lage en "søkemaskin" som lagrer råd assosiert med dokumenter, og så ha en bakgrunnsprosess som etter tur spør agentpopulasjonen

om å produsere råd for hvert dokument i populasjonen.

Videre er det slik at når agenter lager råd som lagres på denne måten kan det gå svært lang tid fra rådet produseres til det gis respons på det (dager, uker eller mer). Et uløst problem er derfor å finne ut hvordan mest effektivt hente inn representasjoner de gamle agentene, gi dem kreditt for arbeidet sitt, og så bruke dem til å optimere den aktive agentpopulasjonen. En mulig løsning er å aldri la agenter med aktive råd dø fullstendig, og så heller bruke mye lagerplass på gamle agenter. En annen løsning er å lagre søkeuttrykkene (altså en refleksjon av agentens atferd) sammen med rådene de produserte, og så la dette mye mer kompakte bildet av agenten leve lenge. Det er for undertegnede uvisst hva som er den mest hensiktsmessige måten å håndtere dette problemet på.

5.6 Om utviklingsmiljøet og utviklingsprosessen

Utviklingen har foregått på en Hewlett Packard Omnibook 6000 bærbar datamaskin med operativsystemet Microsoft Windows 2000, der EMACS 20.7.1 ble brukt som program-editor. J2SE (Java 2 Standard Edition) jdk 1.4.1b ble brukt som utviklingsverktøy. I tillegg til basisbibliotekene i J2SE brukte jeg HTML parser [55] Build-verktøyet Ant [3] og enhets-test verktøyet JUnit[7]. Browseren som ble brukt var Microsoft Internet Explorer v. 6. Ett viktig kriterium for utviklingsmiljøet var at det skulle være mulig å arbeide med, og teste det aller meste av systemets funksjonalitet uten å være tilkoblet internett, slik at produktivt arbeid kunne foregå når tid var tilgjengelig uavhengig av fysisk plassering og nett-tilgant. Dette designkriteriet ble oppfylt med utviklingsmiljøet som er beskrevet over.

Det kanskje mest overraskende med implementasjonen var at den tok så lang tid. I etterkant kan man jo si at dette ikke burde vært så overraskende, problemet som skal løses har betydelig kompleksitet, og komponenter både brukergrensesnitt, meldingsformidling, persistens og adaptive autonome agenter ble skrevet og spesifisert fra bunnen. For meg var det likevel litt overraskende, siden estimatene jeg i forkant gjorde for tidsbruk ble merkbart overskredet.

Persistensmodellen som arbeider mot JDBC var en forholdsvis komplisert affære å skrive. Skulle jeg gjort det om igjen ville jeg nok vurdert nøye om jeg skulle gjøre det på samme måte igjen. Jeg mistenker at jeg ville ha lett nøye etter ferdige pakker for objektpersistens (f.eks. Hibernate[28] som jeg ikke visste om når jeg startet å skrive persistenslaget), og om jeg ikke fant en god kandidat heller vurdert om det ikke heller var lurt å bruke javas objektserialiseringsmekanismer i stedet for å skrive etterhvert ganske mye kode som avbilder mellom tabeller i et RDBMS og den objektorienterte koden i Amanda.

Av utviklingsteknikker som ble brukt under veis ønsker jeg spesielt å trekke frem enhetstester med Junit rammeverket[7]. Den tryggheten man som utvikler føler for et system som kan regresjonstestes flere ganger daglig er svært motiverende. Det å kunne legge lag på lag av funksjonalitet på toppen av hveran-

dre, og ha full tillit til at lagene under virer er veldig betryggende. Spesielt under utviklingen av persistenskomponenten og transportkomponenten var dette merkbart.

Kapittel 6

Videre arbeid

Underveis i arbeidet med denne oppgaven ble det gjort mange avskjæringer: Temaer og teknikker dukket opp og pekte på interessante retninger arbeidet kunne ta, men disse retningene måtte avskjæres fordi det ikke var tid nok eller fordi fenomenet ikke på en direkte nok måte bidro til å belyse denne oppgavens problemstilling.

I dette kapitlet presenteres noen av disse "løse trådene". Temaene behandles ikke likt: Noen gis dypere behandling enn andre. Dette er tilfelle fordi jeg for endel av temaene ble utviklet som en del av oppgavens hovedtekst men senere ble vurdert som sidespor eller utenfor oppgavens rammer. Andre temaer var opplagt utenfor rammen av en hovedfagsoppgave o behandles derfor mer summarisk.

før jeg gjennom arbeidet med temaet fant ut at det ikke passet inn i hovedteksten. I disse tilfellene har jeg tillatt meg å plassere den fjernede teksten her og så omarbeidet den til å passe i et "videre arbeid" kapittel. Andre temaer var det mer opplagt ikke kunne utvikles i stor dybde og de er derfor mer summarisk behandlet her.

6.1 De første tingene som burde gjøres

Det mest frustrerende med hele denne oppgaven er at jeg ikke ferdigstilt applikasjonen til en slik tilstand at eksperimentet beskrevet i appendiks A kunne gjennomføres. Det første videre arbeidet som burde gjøres med Amanda er derfor å ferdigstille implementasjonen slik at det er mulig å gjennomføre kontrollerte eksperimenter for å måle nytteverdien av den: De første tingene som bør gjøres er derfor:

1. Ferdigstill implementasjonen til en slik tilstand at eksperimentet i appendiks A kunne gjennomføres.

2. Implementer funksjonalitet for å gjenkjenne fragmenter av dokumenter som er fremvist for brukeren i dokumenter som er produsert av brukeren.

Eksperimentet er utformet for å være ganske nøytralt med hensyn på hvilken teknologi det tester bruken av, og det vil derfor være mulig å gjenta eksperimentet på flere versjoner av programmet dersom man velger å videreutvikle det. En slik eksperimentserie vil dermed kunne gi et objektivt mål for hvilken nytte et Amanda-liknende program faktisk tilfører brukeren, og dermed også for hvilken nytte de forskjellige komponentene som kan introduseres i arkitekturen faktisk viser seg å bidra til nytte.

6.2 Teori

Utvide Baeza-Yates DQFR rammeverk (gjengitt i seksjon 3.1) til å inneholde beskrivelser av brukerens behov f.eks. parametrisert som krav til responstider koblet til oppgavene. Det ikke opplagt hvordan en slik parametrisering burde gjøres, antagelig vil et samarbeide med psykologer være hensiktsmessig for et slikt arbeide.

6.3 Arkitektur

6.4 Modellering av brukeres kontekst

I seksjon 3.2.5 beskrives en type læringsalgoritmer kalt "modellbyggere". Modelbyggere krever eksplisitte mål og mønstere som beskriver tilstand. Dersom læringsfunksjonalitet basert på modellbygging skal implementeres vil dette for Amandas del vil dette måtte bety at det vil være nødvendig å eksplisitt modellere brukerens kontekst. Det er mulig at kontekst ikke er godt nok forstått til at dette er en nyttig innfallsvinkel, men en

Et spennende prosjekt kan være å utvikle en modell for brukerkontekster som er tilstrekkelig omfattende til at modell-lærende agenter kan utvikles. Jeg har ingen som helst illusjoner om at dette skal være særlig enkelt: Det er fullt mulig at kontekst er for dårlig forstått til at dette vil være fruktbart i dag. Noe arbeid er imidlertid gjort: Lieberman og Selker og rapporterer i [37] om en begynnelse på en modell for kontekst, men av det jeg kunn forstå virket det som om dette arbeidet må omarbeides betydelig for å kunne brukes i en Amanda-liknende setting.

En mer filosofisk innfallsvinkel tas av John Seely Brown og Paul Doguid i artikkelen [33] hvor de utforsker hvordan systemers og fenomeners grenser over tid flyter ut, og det som opprinneli var kontekst flyter inn blir en stabil del av systemets bruk (derav artikkelens navn "borderline issues").

Hvordan disse innfallsvinklene, eller andre, skal kunne operasjonaliseres til å bygge lærende agenter basert på modellbygging har jeg ingen enkle svar på.

6.4.1 Bruk av epost som datagrunnlag

Få datakilder er mer "personlige" enn personlig epost. Meldinger forteller hva man drev med, når man gjorde det, hvilke andre personer som var involverte i det etc. Ved å samle "tråder" (meldinger som er svar på hverandre eller på annet vis siterer hverandre) kan man danne grafer som i sin tur kan representere enten dokumenter eller korpuser å gjøre søk i forhold til. Tilsvarende kan gjøres mot newsmeldinger.

6.4.2 Modellering av brukere og plattformer

I dag er brukere og platformer ikke eksplisitt modellerte. Med dette menes at det i Amanda slik det i dag foreligger er en underliggende antakelse om at systemet kun vil brukes av en bruker som vil bruke det, og at den eller de kanalene hun bruker til å aksessere råd alltid er tilgjengelige. Ved å slakke på disse antakelsene kan man for det første tillate at flere brukere kan dele på de bakenforliggende komponentene (f.eks. persistent lager av dokumentprofiler), men dernest også tillate at en og samme bruker *normalt sett* kobler seg opp mot Amanda gjennom et spektrum av kanaler, fra f.eks. mobiltelefon til arbeidsstasjon. Det er ingen ting prinsipielt i veien med dette, og eksempler for hvordan slike utvidelser kan realiseres er delvis beskrevet i i seksjon 4.4 om "Everywhere messaging" kapittel 4.

6.4.3 Åpnere tilgang til bakenforliggende agenter

Aksess via av web services

I dag er alle deler av systemet implementert i Java, og det er en plattform som er behagelig å utvikle i, men det er jo mange programmer i verden som ikke er skrevet i Java og som heller ikke er spesielt enkle å integrere direkte mot Java. Microsofts Office produkter er eksempler på dette. Ved å åpne opp grensesnittene mot Web Services [15] vil i praksis ethvert utvidelsesspråk for applikasjoner være i stand til å lage spor om hva brukeren gjør, og sende disse sporene direkte til en "bucket" -liknende tjeneste.

Standardisert loggformat for brukerhendelser

Dersom det åpnes opp for logging fra et stort antall applikasjoner vil det være hensiktsmessig å etablere en modell for hendelser initiert av brukere som er betydelig rikere enn den som brukes i dag med enkle endringer i dokumenter med flat eller enkelt hierarkisk struktur. I praksis vil det nok både være hensiktsmessig å etablere et lite antall standardiserte hendelser som kan rapporteres fra mange slags liknende applikasjoner (f.eks. "ord 3 i avsnitt 1.2.3 endret fra 'fisk' til 'hest"'), men også la hver enkelt applikasjon kunne produsere et så rikt sett av hendelser som dette ("Word satte akkurat inn en hyperlink til dokument XXX"). Et standardiseringsarbeide som dette vil etter alt og dømme kunne dra

stor nytte av arbeidene som er gjort for å standardisere sammenkoblinger mellom programmer både i Microsofts .NET infrastruktur, og de open source - baserte GNOME og KDE brukergrensesnittene.

6.4.4 Implementasjonsteknikker

Utvidelser av transportkomponenten

Transportlaget kan utvides til å støtte andre topologier enn hob/spoke, og til å fungere som grensesnitt mot andre meldingsrutere. Ettersom enheter som implementerer MIDP-2 [48] blir tilgjengelige vil det være interesant å lage en implementasjon som kan kjør under denne plattformens begrensninger, og dermed åpne for at Amanda - komponenter kan kjøre f.eks. i mobiltelefoner og PDAer. MIDP-2 er interessant siden dette er den første av MIDP standardene som åpner for at java applikasjoner som kjører på bærbare enheter kan lytte etter innkommende meldinger over TCP/IP. Dette gjør at det vil være mulig å lage Amanda-komponenter som "kjører" i en mobil enhet, men i virkeligheten ikke bruker strøm eller særlig mange andre maskinressurser siden de venter på innkommende meldinger.

6.4.5 Abstrakt grensnitt mot eksterne søkemaskiner

Dagens implementasjon av Amanda har et eget java "Interface" mot eksterne søkemaskiner, så innsiden av Amanda ser allerede en ren og uniform mengde av søkemaskiner. Dette bildet blir imidlertid presentert gjennom adapter-klasser[22, s. 139] som skjuler mye "grisete" screenskcraping kode. Det kunne vært veldig nyttig å lage en abstrakt søkemaskintjeneste som inneholder alle aksess og rettighetsdokumenter til søketjenestene som brukern har tilgang til å bruke, og som kunne fungere som et abstrakt adapter mot alle (eller i det minste mange) slags søkemaskiner. Et slikt abstrakt adapter kunne eksponert et Web Service grensesnitt mot brukere, men så enten brukt screenscraping, web service, eller andre integrasjonsteknikker for å snakke med søkemaskinene. Både J2EE og "Jini" er mulige implementasjonsteknikker for å lage enkelt konfigurerbare realiseringer av denne type grensesnitt.

6.4.6 Filtrering

Dagens Amanda-implementasjon bruker kun eksterne søkemaskiner for søk. En naturlig og nyttig utvidelse vil være å bruke et filter, hvor Amanda produserer filteruttrykk som oversendes et filter som kontinuerlig søker gjennom strømmer av innkomne dokumenter. Hvert dokument som matcher et filteruttrykk vil da potensielt være et råd som kan sendes til brukeren, men om og hvordan dette rådet rangeres vil det være opp til resten av Amanda å vurdere.

Mange kilder til dokumenter å filtrere mot finnes. De mer opplagte er internett news tjenester (NNTP), Epost adressert til brukeren og forskjellige Instant messaging-meldinger (Internet Relay Chat (IRC), Jabber, Microsoft In-

stant Messenger, for å nevne noen). Kanskje litt mindre opplagt er at flere operativsystemer (herunder Silicon Graphics IRIX og Linux) er i stand til å rapportere når filer legges til eller endres i filsystmer. Slike "dokument endret" hendelser kan produsere hendelser som plukkes opp av et filter. På liknende vis kan også crawler-programmer for internett søkemaskiner betraktes som kilder til hendelser som kan filtreres. Til slutt finnes det en stor mengde kommersielle nyhetstjenester (Norsk Telegrambyrå, Reuters, Clarinet, etc.) som tilbyr fulle feeds av artikler, og dessuten websider som gjennom sine RSS feeder kan gi hint om at websidenes innhold er endret, og at det kan være interessant å crawle dem. Det finnes kommersielle produkter (Virage[31]) som er i stand til å se på video (f.eks. fra CNN) og i sanntid plukke ut informasjon om hvilke ansikter som er på skjermen, hva de sier og hvilken tekst som vises på skjermen. Dette skjer naturligvis ikke uten betydlig kalibrering mot de forskjellige kanalenes visuelle formater, men det *er* praktisk gjennomførbart.

Det finnes altså mange kilder til filtere. For Amandas del er det fordelaktig å lage et fleksibelt grensesnitt mot mange typer filtere. Kanskje vil det også være nødvendig å implementere noe egen filterfunksjonalitet for lav-volum hendelser, men for de aller fleste typer hendelser vil nok eksterne filtere være både mer robuste og tåle høyere volum, så produksjon av passende søkeuttrykk (i dette tilfellet filteruttrykk), ikke søkingen (filtreringen) bør fortsatt være Amandas hovedoppgave.

For tilkobling mot kommersielle tjenester vil håndtering av betaling for bruk bli et sentralt problem. Det virker rimelig at problemstillingene rundt konfigurasjon av hvilke kilder det filtreres mot, og hvordan disse gjøres tilgjengelig både med hensyn til teknologi og kommersielle aspekter utredes grundig før man setter i gang med storstilt utvikling av filterfunksjonalitet. Dette er et stort tema både teknisk og forretningsmessig, og begge disse dimensjonene falt langt utenfor denne oppgavens skop.

6.5 Mobiltelefon som datakilde og terminal

I seksjonen over om filter ble det nevnt en lang rekke kilder til data som kan brukes av et Amanda-liknende system. Mobiltelefoner ble ikke nevnt i denne listen, men det er kun fordi jeg betrakter denne kilden som så interessant at jeg ønsker å bruke en egen seksjon på den.

Fremtidige mobiltelefoner vil som et minimum ha anledning til å ta bilder, lytte på lyd, sende multimediameldinger, og kjøre små javaprogrammer.

I seksjon 4.4.3 beskrives "Nomadic Radio" prosjektet som bruker en litt upraktisk prototype-telefon fra Nortel [16, s. 669]. Det jeg fant interessant med denne telefonen var at den brukte signalbehandling av lyd rundt den til å lage hypoteser om hvilken kontekst brukeren var i: Om hun snakket med noen, om hun var i et selskap, på et offentlig sted o.s.v. Det er i prinsippet kun to ting i veien for at en mobiltelefon skal kunne gjøre det samme: Det første er at lydkvaliteten en mobiltelefon har å lytte når den ligger nedpakket kan være dårlig, men dette er ikke nødvendigvis et relevant argument all den tid svært god lyd-

kvalitet kan være tilgjengelig gjennom et headset som er mer eller mindre permanent fremme. Det andre og mer alvorlige argumentet er at det kan kreve en god del regnekraft å gjennomføre den signalbehandlingen som er nødvendig. Dette argumentet kan imidlertid til en viss grad nøytraliseres ved å lage en arkitektur der lydsamples sendes ut av telefonen for analyse i en maskin der det er godt med regnekraft (om man ser på figur 4.9 ser man at det faktisk er dette som skjer i NR), det blir da et regnestykke å finne ut hvor mye kombinasjon av radiostråling og regnekraft man ønsker å bruke for å oppnå en viss tjenestekvalitet på kontekstsensingen.

6.6 Naturlig språkforståelse

På side 25 i teorikapitlets seksjonen om Information Retrieval påsto jeg om at bruk av teknikker for naturlig språkforståelse ikke var et fruktbart utgangspunkt for å lage en personlig agent. Dette var ikke en tilfredstillende påstand for meg å gi, og det er antagelig mulig å gjøre interessant arbeide for å bestride den.

Problemet med naturlig språkforståelse, sett fra et IR perspektiv, er at modellene som etter undertegnedes oppfatning har hatt mest fremdrift m.h.p. å fange mening (semantikk) i språklige ytringer er denotasjonell semantikk og matematisk logikk. I denotasjonell semantikk brukes en teknikk hvor setninger avbildes inn i rom av funksjoner som igjen avbildes inn i komplette partielle ordninger i lattice strukturer (Scott og Strachys metode, beskrevet i [65]). Nært beslektet, i det minste rent teknisk, er metoder der setninger parseres og ut fra sin parserte struktur avbildes inn i utsagn i logikk. Type logikk kan så hentes fra typen språk som skal modelleres. Første ordens predikatlogikk er for snevert for å fange semantikken i naturlig språk, så andre logikker f.eks. Montague gramatikk [51] har vært prøvd med større hell. Utsagnene som produseres vil så ha frie variable som refererer til en modell om verden. En slik fremgangsmåten er beskrevet f.eks. i [20] og [64].

For å produsere en tolkning kan man så ta utgangspunkt i språkteoretikerne J. Groendijk og M. Stokhof [26] sitt arbeid om "multiple worlds" tolkning av språk[73]: En setnings mening er unionen av alle meningene den kan ha i mengden av mulige modeller av verden. Dersom mengden mulige tolkninger er av rimelig størrelse, kan mulige løsninger nummereres og listes opp av en resolusjonsalgoritme (som kan tenkes på som en slags likningsløser for logiske uttrykk).

Som man ser: Det er et godt stykke fra metoden som er beskrevet over til de statistiske, vektorbaserte og indeksbaserte søkemetodene som er dominerende innen Information Retrieval, og det er derfor ikke trivielt å spenne over gapet.

For å kunne gjøre noen fremskritt ser det derfor ut som å være nødvendig å redusere skop: Enten å begrense seg til et begrenset språklig domene, f.eks. eventyr for barn, nyhetsjournalistikk på TV eller liknende. Gitt et språklig domene det er mulig å lage noen lunde troverdige modeller for, kan det være mulig å lage forskjellige slags mediatorer inn mot søkemaskiner. Andre dimensjon-

er man kan velge å begrense seg langs er størrelsen av språket (f.eks. velge å ignorere de fleste ord i setninger) eller kompleksiteten av modellene man forsøker å matche mot (langt mindre enn full bredde av mennsklig språk).

Dersom moduler som gjennomfører denne type analyse er mulig å lage, bør det også være mulig å bruke dem som kompetansemoduler innen et Amanda-system, og da i konkurranse med andre typer moduler, slik at det forhåpentligvis er mulig å finne ytterligere begrensninger som gjør at bruken av naturlig språkforståelse blir optimal.

Kapittel 7

Oppsummering

Denne oppgaven er det stilt spørsmål om hva som kan være et hensiktsmessig design av et system for å løpende gi råd som er relevante i forhold til til en brukers interaksjon med datamaskin.

Oppgaven besvarer problemstillingen gjennom en spesifikasjon og en delvis implementasjon av en agentbasert arkitektur kalt Amanda. Amanda lytter og ser etter spor om brukerens atferd fra flere kilder, transformerer disse til spørringer mot eksterne søkemaskiner og bruker resultatene fra disse spørringene for å gi brukeren løpende råd. Amandas arkitektur kan omfatte mange typer brukerinteraksjon, men den implementerte delen begrenser seg til å se på et tilfelle hvor brukeren arbeider med tekstbehandling og hvor dokumenter leses gjennom en web-browser.

Ved å problematisere det tekniske forslaget til løsning gjennom teori om Information Retrieval og adaptive autonome agenter vises det at løsningen som denne oppgaven presenterer bygger på ideer fra begge disse fagområdene, men at de kombineres på nye måter i forhold til det som er vist i teorien som er valgt ut.

Forslaget problematiseres videre gjennom å se på et knippe arbeider innen "Just In Time Information Retrieval", "Just in time messaging" og søkemaskiner. Arbeidene ble valgt ut dels fordi noen av dem representerer løsninger på problemer som ligner denne oppgavens problemstilling, og dels fordi de representerer komponenter som er nødvendige å forholde seg nært og rasjonelt til under realisering av en arkitektur som Amanda.

Problematiseringen via litteratur har vist at den foreslåtte innfallsvinkelen ser ut til å være robust med hensyn på variasjon i tilgjengelige søkemetoder, variasjon i kilder til spor om brukeratferd og brukergrenssnitt mot brukeren.

Gjennom arbeidet med litteraturen klargjøres det hvilke komponenter som er hentet fra andre steder og hvilke som er unike for denne oppgavens løsning.

Arkitekturen som beskrives har mange frihetsgrader, og for alle eksemplene på tidligere arbeid indikeres det måter denne oppgavens arbeid kan tilpasses for å helt eller delvis utføre oppgavene de tidligere arbeidene har utført. Den viktigste kilden til variasjon er her tilpasninger mot de forskjell-

ige inn og ut-kanaler som brukes av de forskjellige systemene. Den nest viktigste er antagelig designvalget om å delegere søkemaskinfunksonalitet ut av designets kjerne og gjennom dette rendyrke kompetanse rundt utforming av søk og presentasjon av søkeresultater. Amandas kjerne, bestående av adaptive autonome agenter koblet sammen av en generell transportkomponent, er i utgangspunktet tilpasningsdyktig nok til å håndtere funksjonaliteten i de beskrevne systemene så få endringer trenges der. Dette er en bekreftelse på denne oppgavens målsetning gjennom Amanda kan realiseres for mange typer brukerinteraksjon, ikke det avgrensede tekstbehandler/web-brower grensesnittet som den delvise implementasjonen forholder seg til.

Designet er også problematisert gjennom implementasjon. Implementasjonen var kun delvis, men alle de implementerte delene bekrefter at designet er implementerbart, og at de designede komponentene er i stand til å levere ønsket atferd.

I oppgavens seksjon om videre arbeide indikeres det flere ting som kan gjøres gjennom å utvide tilfanget av typer input, typer filtrering og søking, og typer arbeidsoppgaver som kan støttes. Flere av disse forslagene er inspirert av litteraturen som ble gjennomgått i forbindelse med oppgaven.

Å skrive oppgaven har vært gøy, men alt har en ende :-)

Tillegg A

Kontrollert eksperiment

For å teste hvor god søkesystemet var, bel det designet et eksperiment som skulle måle dette. Eksperimentet er en tilpasning av det eksperimentet som ble beskrevet av Rhodes i hans kapittel om evaluering [56, 87-115]. På grunn av oppgavens omfang ble aldri eksperimentet gjennomført. Å planlegge eksperimentet var imidlertid en nyttig del av systemdesignet for å designe siden det representerte et et konkret brukstilfelle (use-case) å designe og var derfor svært nyttig for å avgjøre hvilke egenskaper både arkitektur og implementasjon burde ha og ikke ha. I dette appendikset derfor inkluderes derfor forarbeidet som ble gjort for eksperimentet som aldri ble gjennomført.

Eksperimentet gikk i korthet ut på å gi en gruppe studenter i oppgave å skrive en artikkel om det samme temaet, boligsituasjonen for studenter i Oslo. Halvparten av studentene skulle kun få en tekst-editor og en vanlig Browser, og halvparten skulle i tilfelle Amanda systemet til å hjelpe seg. For å måle hvor effekten av Amanda, skulle det før og etter etter at artikkelen var skrevet fylles ut et spørreskjema, og dessuten skulle artiklene selv analyseres. Alle deltagerne skulle før eksperimentet gjennom et skriv delvis informeres om hva de skulle gjøre, og etter at eksperimentet var avsluttet skulle de få nok et skriv som beskrev hva det var de hadde blitt testet for.

Gjennomføring av dette eksperimentet og gjentagelse av Rhodes statistiske analyse vil kunne gi et godt grunnlag for sammenligning med Rhodes resultater. Innsamling av kommentarer som fritekst fra testsubjektene vil i tillegg kunne fange intrykk som ikke lar seg fange i de enkle skalaene som brukes i intervjuskjemaene.

A.1 Samtykke om deltagelse i eksperiment

Din deltagelse i det følgende eksperimentet er fullstendig frivillig. Du kan på et hvilket som helst tidspunkt, av en hvilken som helst årsak, bestemme deg for at du ikke ønsker å fortsette din deltagelse i eksperimentet. Du kan dessuten be om at alle data som er samlet inn blir ødelagt. Dersom du på noe tidspunkt føler deg ukomfortabel eller usikker på om du ønsker at resultatene dine være en del av eksperimentet, kan du avslutte din deltagelse uten noen form for problemer.

Om noen få minutter vil du bli bedt om å skrive en artikkel mens du bruker datamaskinen til å hjelpe deg. Du vil bli tilbudt ett eller flere informasjons-verktøy som du kan velge å bruke, eller avstå fra å bruke, fullstendig etter eget ønske. Du vil også bli bedt om å fylle ut ett spørreskjema før du skriver artikkelen, og ett etterpå. Det står deg fritt å avstå å svare på alle, eller noen av spørsmålene. Hele eksprimentet vil ta omtrent en time, og du vil få utbetalt Kr. **Her settes et passende beløp inn** som kompensasjon for din deltagelse.

Dersom du på noe tidspunkt føler deg ukomfortabel med det du blir bedt om å jøre, står det deg fritt å be om at eksperimentet blir avsluttet. All informasjon innsamlet under din deltagelse vil bli fjernet, og din betaling vil være i forhold til den tiden du allerede har brukt.

Dersom du har noen spørsmål anående eksperimentet, vil eksperimentledelsen svare på dem med glede og entusiasme.

Vær vennlig å les de følgende og underskriv på linjen under.

"Jeg, den undertegnede har lest og forstått forklaringen av det følgende forskningsprosjektet, og jeg gir frivillig mitt samtykke til deltagelse i det. Jeg forstår at mine responser vil forbli konfidensielle, og atjeg kan avslutte min deltagelse på ethvert tidspunkt.

Dersom fysisk skade skulle oppstå under dette eksperimentet, forstår jeg at medisinsk hjelp vil bli gjort tilgjengelig fra *noen*

Jeg forstår at jeg kan kontakte **Her skal et navn og en adresse settes inn** dersom jeg føler at jeg har blitt behandlet urimelig."

Navn: Dato:

A.2 Før-undersøkelse

1. **Kjønn:** Mann / Kvinne.
2. **Alder:**
3. **Hva er ditt morsmål?**
4. **Hvor mye vet du om boligsituasjonen for studenter i Oslo?**:

Ingen ting i det hele tatt	1	2	3	4	5	6	7	Veldig mye

5. **Hvor mye erfaring har du i å skrive nyhetsartikler eller korte essay?**

Ingen ting i det hele tatt	1	2	3	4	5	6	7	Veldig mye

6. **Hvor ofte leser du Universitas?**

Aldri	1	2	3	4	5	6	7	Hvert nummer

A.3 Oppgave-beskrivelse Nr. 1

Anta at du er en bidragsyter for studentavisen "Universitas" og du skal skrive en lederartikkel eller en nyhetsartikkel om boligsituasjonen. Artikkelen kan handle om **her skal det settes inn noen konkrete og dagsaktuelle konkrete punkter om boligsituasjonen for studenter**, eller et hvilket som helst annet tema om studentboliger og hvordan det påvirker studenlivet. Artikkelen bør være på ca. en side (rundt 600-700 ord.)

Du vil ha opp til 45 minutter på å gjøre ferdig artikkelen din. Denne grensen er ikke satt for å setresse deg, men kun for å sette en øvre grense på tiden eksperimentet skal ta. Dersom du gjør artikkelen ferdig før de 45 minuttene har gått, gi signal til eksperimentlederen som vil ta deg med til neste fase av eksperimentet. Dersom du ønsker kan du ved slutten av eksperimentet få artikkelen sendt til deg i epost og/eller den kan bli skrevet ut så du kan få en kopi. Din artikkel vil bli sammenlignet med artikler skrevet av andre deltagere i dette eksperimentet basert på mange forskjellige kriterer.

Du bør allerede ha fått instruksjoner i bruk av *universitas-søkeverktøyet*. Bruk dette verktøyet så mye eller lite du ønsker når du skriver artikkelen. Dersom du har spørsmål nå eller under eksperimentet er det fint om du spør eksperimentlederen.

A.4 Etter-undersøkelse 1

- **1. Hvor vanskelig syntes du at artikkel-skrivings oppgaven var?**

Svært enkel	1	2	3	4	5	6	7	ganske vanskelig

- **2. Hvor mye lærte du deg om emnet under eksperimentet?**

Ingen ting	1	2	3	4	5	6	7	Svært mye

- **3. Hadde du nok tid til å gjennomføre oppgaven på en skikkelig måte?**
  Ja / Nei

- **4. Hvordan vil du karakterisere din ekspertise med web-baserte søkeverktøy (sett sirkel rundt ett av svarene)?**

  1. Aldri brukt ett før i dag.
  2. Brukt dem av og til
  3. Jevnlig bruker
  4. Ekspert

- **5. Hvor distraherende synes du bruken av søkeverktøyet var?**

Ikke distraherende	1	2	3	4	5	6	7	Svært distraherende

- **6. Hvor nyttig synes du funnene søke-maskinen fant for deg var?**

Ikke nyttige	1	2	3	4	5	6	7	Svært nyttige

- **7. Dersom du skulle gejennomført en lignende oppgave, hvor mye ønsker du deg at søkemaskinen skulle vært tilgjengelig?**

Ikke i det hele tatt	1	2	3	4	5	6	7	Svært gjerne

- **8. Hvor ofte synes du resultatene gitt av søkemaskine var nyttige i seg selv, selv når du ikke klikket deg inn til de fulle artikkeltekstene?**

Aldri	1	2	3	4	5	6	7	Svært ofte

- **9. I hvor stor grad viet du søkemaskinen oppmerksomhet?**

Nesten ingen oppmerksomhet	1	2	3	4	5	6	7	Svært stor oppmerksomhet

- **10. Når du leste en artikkel foreslått fra søkemaskinen, hvor ofte var den nyttig for deg?**

Aldri	1	2	3	4	5	6	7	Svært ofte

A.5 Oppgave-beskrivelse Nr. 2

Anta at du er en bidragsyter for Universitas og du skal skrive en lederartikkel eller en nyhetsartikkel om boligsituasjonen. Artikkelen kan handle om *noen konkrete punkter om boliggreiene*, eller et hvilket som helst annet tema om studentboliger og hvordan det påvirker studenlivet. Artikkelen bør være på ca. en side (rundt 600-700 ord.)

Du vil ha opp til 45 minutter på å gjøre ferdig artikkelen din. Denne grensen er ikke satt for å setresse deg, men kun for å sette en øvre grense på tiden eksperimentet skal ta. Dersom du gjør artikkelen ferdig før de 45 minuttene har gått, gi signal til eksperimentlederen som vil ta deg med til neste fase av eksperimentet. Dersom du ønsker kan du ved slutten av eksperimentet få artikkelen sendt til deg i epost og/eller den kan bli skrevet ut så du kan få en kopi. Din artikkel vil bli sammenlignet med artikler skrevet av andre deltagere i dette eksperimentet basert på mange forskjellige kriterer.

Du bør allerede ha fått instruksjoner i bruk av *universitas-søkeverktøyet* og "Amanda"-søkeverktøet. Bruk disse verktøyene så mye eller lite du ønsker når du skriver artikkelen. Dersom du har spørsmål nå eller under eksperimentet er det fint om du spør eksperimentlederen.

A.6 Etter-undersøkelse 2

- **1. Hvor vanskelig syntes du at artikkel-skrivings oppgaven var?**

Svært enkel	1	2	3	4	5	6	7	Ganske vanskelig

- **2. Hvor mye lærte du deg om emnet under eksperimentet?**

Svært enkel	1	2	3	4	5	6	7	Ganske vanskelig

- **3. Hadde du nok tid til å gjennomføre oppgaven på en skikkelig måte?**
  Ja / Nei
- **4. Ranger hvor nyttig du synes søkeverktøyene du brukte var, bruk en skala fra 1 til 3, der 1 betyr mest nyttig:**
  1. Søkemaskin:
  2. Amanda:
- **5. Hvordan vil du karakterisere din ekspertise med web-baserte søkeverktøy (sett sirkel rundt ett av svarene)?**
  1. Aldri brukt ett før i dag.
  2. Brukt dem av og til
  3. Jevnlig bruker
  4. Ekspert
- **6. Hvordan vil du karakterisere din ekspertise med "Amanda" (sett sirkel rundt ett av svarene)?**
  1. Aldri brukt ett før i dag.
  2. Brukt dem av og til
  3. Jevnlig bruker
  4. Ekspert
- **7. Hvor distraherende synes du bruken av søkeverktøyet var?**

Ikke distraherende	1	2	3	4	5	6	7	Svært distraherende

- **8. Hvor distraherende synes du bruken av Amanda var?**

Ikke distraherende	1	2	3	4	5	6	7	Svært distraherende

- **9. Hvor nyttig synes du funnene søke-maskinen fant for deg var?**

Ikke nyttige	1	2	3	4	5	6	7	Svært nyttige

- **10. Hvor nyttig synes du funnene Amanda fant for deg var?**

Ikke nyttige	1	2	3	4	5	6	7	Svært nyttige

- **11. Dersom du skulle gejennomført en lignende oppgave, hvor mye ønsker du deg at søkemaskinen skulle vært tilgjengelig?**

Ikke i det hele tatt	1	2	3	4	5	6	7	Svært gjerne

- **12. Dersom du skulle gejennomført en lignende oppgave, hvor mye ønsker du deg at Amanda skulle vært tilgjengelig?**

Ikke i det hele tatt	1	2	3	4	5	6	7	Svært gjerne

- **13. Hvor ofte synes du resultatene gitt av søkemaskinen var nyttige i seg selv, selv når du ikke klikket deg inn til de fulle artikkeltekstene?**

Aldri	1	2	3	4	5	6	7	Svært ofte

- **14. Hvor ofte synes du resultatene gitt av Amanda var nyttige i seg selv, selv når du ikke klikket deg inn til de fulle artikkeltekstene?**

Aldri	1	2	3	4	5	6	7	Svært ofte

- **15. I hvor stor grad viet du søkemaskinen oppmerksomhet?**

Nesten ingen oppmerksomhet	1	2	3	4	5	6	7	Svært stor oppmerksomhet

- **16. I hvor stor grad viet du Amanda oppmerksomhet?**

Nesten ingen oppmerksomhet	1	2	3	4	5	6	7	Svært stor oppmerksomhet

- **17. Når du leste en artikkel foreslått fra søkemaskinen, hvor ofte var den nyttig for deg ?**

Aldri	1	2	3	4	5	6	7	Svært ofte

- **18. Når du leste en artikkel foreslått av Amanda, hvor ofte var den nyttig for deg ?**

Aldri	1	2	3	4	5	6	7	Svært ofte

A.7 Debriefing-erklæring

Eksperimentet du nettopp har deltatt i var designet for å teste "Amanda", et verktøy som automatisk foreslår dokumenter som kan være relevante for en person. Halvparten av deltagern i dette eksperimentet fikk tilbud om å bruke Amanda i tillegg til en søkemaskin når de skulle skrive artikkelen sin, den andre halvparten fikk kun tilbud om å bruke søkemaskinen.

Vi forsøker å finne ut om Amanda oppfordrer til søking etter mer infomrasjon, og om den gir mer detaljert informasjon enn standard søkemaskiner. Vi vil undersøke både hvor magne spesifikke fakta som er referert i artiklene fra begge grupper, og vi vil se på hvor mange avisartikler som ble hentet og lest med bruk av søkemaskinen og med bruk av Amanda. En logg har blitt ført av søk som er gjennomført, tidsstempet av spørringer, og bestemte nyhetsartikler som ble lest gjennom begge systemene. Denne informasjonen vil bli brukt til å avgjøre hvordan Amanda og ligendende systemer kan bli designet og brukt i fremtidige applikasjoner, og til å evaluere konseptet om implisitte spørringer.

Dersom du nå eller senere skulle oppleve noen ubehagelige effekter (mentale eller fysiske) som et resultat av din deltagelse i eksperimentet, ikke nøl med å ringe **Her settes Remseths telefonnummer inn** og spør etter Bjørn Remseth.

Institutt for Informatikk ved Universitetet i Oslo er svært glade for at du var villig til delta i dette eksperimentet. Din deltagelse vi hjelpe til med å øke forståelsen av hvordan Amanda kan designes og anvendes i forskjellige situasjoner.

Kolofolon

Manuskriptet ble i all hovedsak skrevet på en Hewlett Packard Omnibook 6000 bærbar datamaskin med operativsystemet Microsoft Windows 2000, der EMACS 20.7.1 ble brukt som tekstbehandler. Typesettingsprogrammet som ble brukt var MIKTeX 2.3.1222 (3.141592), makropakken LaTeX med BABELs "norsk" stil, men uten norske orddelingsmønstre. Det er ikke brukt noen egendefinerte LaTeX-makroer under produksjon av dette dokumentet, utallige arbeidstimer ble antagelig spart på dette viset. Manuskriptet ble satt i 10 pt. Palatino for A4 ark. MikTeX produserte en ".pdf" (Adobe Portable Document Format) fil som ble trykket på papir og bundet inn.

UML diagrammer ble produsert på Rational Rose Modeler Edition release 2002.05.20. Diagrammene ble så kopiert over til Microsoft Word 2000 (9.0.442 SR-1), hvor de ble eksportert til en webside med grafikk i som Portable Network Graphics (PNG) format, som så ble tatt inn av LaTeX som bildefiler.

Øvrige tegninger ble laget i Microsoft Powerpoint, og eksportert som screenshots til bildebehandligsprogrammet Gimp for videre import til LaTeX.

Bibliografi

[1] P. E. Agre. *The Dynamic Structure of Everyday Life*. Cambridge University Press, 1991.

[2] Moukas Alexandros. Amalthaea: Information discovery and filtering using a multiagent evolving ecosystem. In *Proceedings of the Conference on Practical Application of Intelligent Agents & Multi-Agent Technology, London, 1996*, 1996.

[3] The ant team. The apache ant project. http://ant.apache.org/.

[4] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming, an introduction: On the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers, Inc., 1998.

[5] Gouri K. Battacharyya and Richard A. Johnson. *Statistical Concepts and Methods*. John Wiley & sons, 1977.

[6] Elisabeth Bayegan. Mediators - (intelligent) middleware for information systems. Technical report, Norwegian University of Science and Technology, Department of Computer and Information Science, Trondheim, Norway, December 1998.

[7] Kent Beck and Erich Gamma. Junit, java unit test tool. http://www.junit.org/index.htm.

[8] Tim Berners-Lee. Semantic web on xml, dec 2000. http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html.

[9] Tim Berners-Lee. Xxx some article by timbl that defines the world wide web. *XXX Somewhere*, may 9999.

[10] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, pages 30–37, may 2001.

[11] B. Blumberg. Action-selection in hamsterdam: Lessons from ethology (submitted). Submitted to the third international conference on the simulation of adaptive behavior, Brighton 1994.

[12] R. Brooks. Artificial life and real robots. In *European Conference on Artificial Life*, pages 3–10, 1992. citeseer.nj.nec.com/brooks92artificial.html.

[13] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, RA-2, april 1986.

[14] Rodney A. Brooks. Intelligence without reason. In John Myopoulos and Ray Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA. citeseer.nj.nec.com/article/brooks91intelligence.html.

[15] Ethan Cerami. *Web services essentials*. O'Reilly and Associates Inc., February 2002.

[16] C.Schmandt, N. Marmasse, S. Marti, N. Sawheny, and S. Wheeler. Everywhere messaging. *IBM Systems journal*, 39(2 and 4):660–677, 2000.

[17] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The dyanamics of collective sorting: Robot-like ants and ant-like roots. In H. L. Roitblat J.A. Meyer and S.W. Wilson, editors, *From Animals to Amimats 2, Proceedings of the Second International Conferenceon Simulation of Adaptive Behavior*. MIT Press/Bradford Books, 1993.

[18] G. L. Drescher. *Made-Up Minds: A constructivist Approach to Artificial Intelligence*. MIT press, 1991.

[19] Greg Meredith Erik Christensen, Francisco Curbera and Sanjiva Weerawarana. Web services description language (wdsl) 1.1, march 2001. http://www.w3.org/TR/wsdl.

[20] Jens Fenstad. Natural language systems. Technical report, Department of Mathematics, University of Oslo, P. O. box 1053-Blindern, 0316 Oslo, March 1988.

[21] Funk and Wagnalls. *Funk and Wagnalls New Encyclopedia*. Funk and Wagnalls, 1979. 29 volumes, 25.000 encyclopedia articles.

[22] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[23] Salton Gerard and Allan James. Automatic text decomposition and structuring. *INFORMATION PROCESSING & MANAGEMENT*, 32:127–138, 1996.

[24] Paul Giotta, Scott Grant, Silvano Maffeis Michael Kovacs, K. Scott morrison, and Meeraj Moidoo Kunnumpurath Gopalan Suresh Raj. *Professional JMS Programming*. Wrox, 2000.

[25] Eric J. Glover, Steve Lawrence, Michal D. Gordon, William P. Birmingham, and C. Lee Giles. Web search – your way. improving web searching with user preferences. *Communications of the ACM*, 44(12), December 2001.

[26] J. Groendijk and M. Stokhof. *On the Semantics of Questions and the Pragmatics and Answers*. PhD thesis, Ph.D. thesis, University of Amsterdam, 1984.

[27] Murray Malone Henry S. Thompson, David Beech and Noah Mendelsohn. Extensible markup language (xml) - w3c recommendation 6 october 2000, May 2001. http://www.w3.org/TR/xmlschema-1/.

[28] The hibernate team. Object/relational mapping and transparent object persistence for java. http://www.hibernate.org/.

[29] I. Horswill. Characterizing adaption by consraint. In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems, Proceedings of the First Europeean Conference on Artificial Life*. MIT Press/Bradford Books, 1992.

[30] Google Inc. Google web apis (beta). http://www.google.com/apis/.

[31] Virage Inc. Virage technology overview. http://www.virage.com/technology/index.cfm.

[32] G. Theralaz J. L. Deneuborg and R. Beckers. Swarm-made architectures. In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems, Proceedings of the First Europeean Conference on Artificial Life*. MIT Press/Bradford Books, 1992.

[33] Paul Doguid John Seely Brown. Borderdline issues. *Human-Computer Interaction*, 3:3–36, 1994.

[34] Brian Kantor and Bill Lapsley. Network news transfer protocol a proposed standard for the stream-based transmission of news (rfc 977), februar 1986. http://www.ietf.org/rfc/rfc0977.txt?number=977.

[35] Marcin Kaszkiel and Justin Zobel. Effective ranking with arbitrary passages. *Journal of the American Society for Information Science and Technology*, 2(4), 2001.

[36] R.L. Keeny and H. Raiffa. *Decisions with Multiple Objectives*. Wiley, NY, thirteenth edition, 1976.

[37] H. Lieberman and T. Selker. Out of context: Computer systems that adapt to, and learn from, context. *IBM Systems journal*, 39(2 and 4):617–632, 2000.

[38] M. Littman. An optimization-based categorization of reinforcement learning environments. In H. L. Roitblat J.A. Meyer and S.W. Wilson, editors, *From Animals to Amimats 2, Proceedings of the Second International Conferenceon Simulation of Adaptive Behavior*. MIT Press/Bradford Books, 1993.

[39] Porter M. An algorithm for suffix stemming. *Program*, 14(3):130–137, 1980.

[40] Porter M. Improving subject retrieval in online catalogues. *British Library Research paper*, 1(24), 1987.

[41] P. Maes. *Designing autonomous agents: Theory and practice from biology to engineering and back*, chapter Situated Agents Can Have Goals. MIT Press/Bradford Books, 1990.

[42] P. Maes. Modeling adaptive autonomous agents. *Artificial Life, I*, (1&2)(9), 1994. citeseer.nj.nec.com/maes94modeling.html.

[43] P. Maes and R. A. Brooks. Learning to coordinate behaviors. In *Proceedings of AAAI-90, Boston*, 1990.

[44] P. Maes and R. Kozierok. Learning interface agents. In *Proceedings of AAAI-93, the Eleventh National Conference on Artificial Intelligence*. MIT Press, 1993.

[45] Pattie Maes. Learning behavior networks from experience. In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems, Proceedings of the First Europeean Conference on Artificial Life*. MIT Press/Bradford Books, 1992.

[46] Grant K.R Malone T.W., Fikes R.E and Howard M.T. A market-like task scheduler for distributed computing environments. In B. Huberman, editor, *in The Ecology of Computation*. North holland, 1988.

[47] Chris Welty Michael K. Smith and Deborah McGuinness. Web ontology language (owl)guide version 1.0 w3c working draft 10 february 2003, 2003. http://www.w3.org/TR/2003/WD-owl-guide-20030210/.

[48] Sun Microsystems. J2me mobile information device profile (midp). http://wireless.java.sun.com/midp/.

[49] Marvin Minsky. *The Society of Mind*. Simon and Schuster, 1986.

[50] S. Mizarro. Relevance: The whole story. *Journal of the American Society of Information Scienece*, 49(9):810–832, september 1997.

[51] R. Montague. *Formal Philosophy*. Yale University Press, 1974.

[52] Dr. Knut Omang. Samtale med Dr. Knut Omang. , 2003.

[53] B. O'onaill and D. Frohlich. Timespace in the workplace: Dealing with interruptions. In *Proceedings of CHI'95*. ACM, 1995.

[54] Sean B. Palmer. Rdf site summary (rss) 1.0). http://web.resource.org/rss/1.0/.

[55] "quiotix". Java based html parser from quiotix inc. http://www.quiotix.com/downloads/html-parser/.

[56] Bradley James Rhodes. *Just-In-Time Information Retrieval, Ph.D. Thesis.* Massachusetts Institute of Technology, June 2000.

[57] Bertier Ribeiro-Neto Ricardo Baeza-Yates. *Modern Information Retrieval*. Addison Wesley / ACM Press, 1999.

[58] J. K. Rosenblatt and D. W. Payton. A fine-grained alternative to the subsumption architecture for mobile robot control. In *Proc of the IEEE Int. Conf. on Neural Networks*, volume 2, pages 317–324, Washington, DC, 1989. IEEE Press. citeseer.nj.nec.com/rosenblatt89finegrained.html.

[59] G. Salton and C. Buckley. Term weighting approachs in automatic text retrieval. *Information processing and Management*, 24:513–523, 1988.

[60] G. et al. Salton. A vector space model for automatic indexing. *Communications of the Association for Computing Machinery*, 18:613–620, 1975.

[61] Frank Schilder. Robust discourse parsing via discourse markers, topicality and position. *Natural Language Engineering*, 8(2/3):235–255, 2002.

[62] Fast Search and Transfer. Fast data search technical highlights, 2003. http://www.fastsearch.com/article/articleview.

[63] Fast Search and Transfer. Fast search and transfer as websider, august 2003. http://www.fast.no/.

[64] Helle Frisak Sem. The representation of meaning - a comparative study of situation schemata. Technical report, Department of Mathematics, University of Oslo, P. O. box 1053-Blindern, 0316 Oslo, august 1988.

[65] Joseph E. Stoy. *Denotational semantics - the Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[66] Semantic Web Working Symposium. Web page for the semantic web working symposium. http://www.semanticweb.org/SWWS/.

[67] Andrew Layman Tim Bray, Dave Hollander. Namespaces in xml, Jan 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114/Overview.html.

[68] C. M. Sperberg-McQueen Tim Bray, Jean Paoli and Eve Maler. Extensible markup language (xml) - w3c recommendation 6 october 2000, october 2000. http://www.w3.org/TR/REC-xml.

[69] F. Toates and P. Jensen. Ethological and psychologicals of motivation - towards a synthesis. In H. L. Roitblat J.A. Meyer and S.W. Wilson, editors, *From Animals to Amimats 2, Proceedings of the Second International Conferenceon Simulation of Adaptive Behavior*. MIT Press/Bradford Books, 1993.

[70] F. Toates and S. Wilson. Environment structure and adaptive behavior from the ground up. In H. L. Roitblat J.A. Meyer and S.W. Wilson, editors, *From Animals to Amimats 2, Proceedings of the Second International Conferenceon Simulation of Adaptive Behavior*. MIT Press/Bradford Books, 1993.

[71] T. Tokunaga and M. Iwayama. Text categorization based on weighted inverse document frequency, 1994. citeseer.nj.nec.com/tokunaga94text.html.

[72] T. Tyrrell. *Computational Mechanisms for Action Selection, PhD Thesis*. PhD thesis, Centre for Cognitive Science, University of Edinburgh, 1993.

[73] Espen Vestre. Samtale med Espen Vestre. Diskusjon om hans avbrutte doktorgradsstudium i datalingvistikk, 1 2002.

[74] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *Software Engineering*, 18(2):103–117, 1992. citeseer.nj.nec.com/waldspurger91spawn.html.

[75] Webster. Merriam-websters collegiate dictionary, web version. http://www.webster.com.

[76] S.W. Wilson. Knowledge growth in an artificial animal. In Greffenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their applications*. Lawrence Erlbaum Associates, 1985.

[77] Martin Fowler with Kendall Scott. *UML Destilled - applying the standard object modelling language*. Addison Wesley, October 1997.