UNIVERSITETET I OSLO
Institutt for informatikk

Open Source og søketeknologi for virksomheter

Masteroppgave

Eivind Hasle Amundsen

April 2007

Innhold

Figurer

Forord

Søketeknologi får for lite oppmerksomhet. Jeg kan selvfølgelig bare komme med denne påstanden fra mitt eget ståsted, som er Institutt for informatikk ved Universitetet i Oslo. Jeg kan med hånden på hjertet si at studenter her på instituttet burde få lære mye mer om søk. Det kan synes som et paradoks at svært datainteresserte og flittige brukere av websøk, ikke engang vet hvordan søkemotorer fungerer innvendig.

Til gjengjeld lærer vi mye mer om databaser. På mange områder ser det ut til at søk på sikt vil supplere databaser og bli vel så viktige. På enkelte områder vil søk kanskje ta helt over for databaser. Etterhvert som stadig mer informasjon blir ustrukturert, trenger vi systemene som kan håndtere alt sammen. Teknologien bak dette er søk.

Søk er nå for alvor på full fart inn i virksomheter, noe denne oppgaven ser nærmere på. Det synes klart at de kommersielle leverandørene av søketeknologi har fått et klart forsprang i markedet. Nå er det på tide med bedre produkter innenfor åpen kildekode. Mye finnes allerede, men for å komme videre må vi lære mer om dette etterhvert nesten livsviktige faget.

Utgangspunktet for denne oppgaven var at jeg fant en ekstern veileder som hadde ideer til et produkt som kunne utvikles. Selv var jeg omtrent helt blank med hensyn på søketeknologi, men det skulle ikke mye til for å overbevise meg om at dette var et interessant felt.

Information Retrieval (IR) seilte etterhvert opp som en logisk bit i puslespillet – det var som om noen plutselig fortalte meg om hva som *egentlig* foregår når man leter gjennom store mengder av informasjon.

Denne interessen førte blant annet til at jeg jobbet for Fast Search & Transfer i Boston sommeren 2006. Her fikk jeg "skitnet til fingrene" da jeg hjalp med å tilpasse Fast ESP for store kunder. Blant annet fikk jeg erfare hvor enorme datamengder en middels stor amerikansk bedrift kan indeksere. I tillegg var jeg med på å utvikle verktøy for Fasts avdeling for kundeløsninger.

Denne masteroppgaven tok en uventet vri. Jeg brukte mye tid på å

sette meg inn i IR-fagfeltet og fri programvare, og jeg brukte litt tid på å bli kjent med produktet Apache Lucene. Da jeg siden gikk i gang med å kartlegge hvilke moduler og biblioteker som kunne sys sammen, fant jeg ut at det jeg ønsket å starte på allerede hadde vært gjort to-tre ganger. Jeg synes det er mest hensiktsmessig å finne ut noe nytt, og derfor ønsket jeg ikke å kopiere andre. I stedet tok jeg en grundig titt på hva dagens løsninger kan tilby, og hva som skal til for å bringe fri programvare og virksomhetssøk til neste nivå.

Hvis noen ønsker å ta fatt på utfordringen, vil jeg anbefale å vurdere de ulike alternativene jeg har nevnt i diskusjonen. Uansett hvilken fremgangsmåte som velges, anbefaler jeg dessuten å begynne i det små med enkle utvidelser. Husk samtidig at fri programvare ofte har størst suksess når det fokuseres på isolerte oppgaver samt optimaliseringen av disse.

Resultatet er litt annerledes enn først forespeilet. Først og fremst er jeg glad for at jeg har fått en mye større innsikt i hva søk er, både gjennom å programmere, lese og tenke. Men i tillegg har jeg fått erfare at programmeringsoppgaver bør være mer konkret definert fra starten. Det ble rett og slett for mye å gjøre frem til programmeringen kunne starte. Selvfølgelig skyldes dette hovedsakelig oppdagelsen om de eksisterende løsningene underveis, og det gjorde at jeg ikke fikk utviklet på langt nær så mye som jeg hadde håpet på.

Virksomhetssøk er utrolig interessant, og det finnes en stor mengde konkrete oppgaver som kan løses både med og uten programmering. Nysgjerrige studenter burde ikke nøle med å velge en masteroppgave innenfor dette. Første gangen du lærer om søk, vil du synes det er nesten utrolig hvordan du har klart å lære så mye annet om behandlingen av data, uten å støte borti dette fagfeltet.

Takk

Jeg vil gjerne få takke min hovedveileder, Stefan Landrø, for at han introduserte meg til et nytt og interessant fagfelt. Jeg vil også takke min interne veileder, Knut Omang, for at han fikk meg på riktig spor med hensyn til den skriftlige oppgaven.

Videre vil jeg takke alle som har lest oppgaven eller kommet med innspill. En takk går også til mine foreldre som lot meg “søke dekning” i Stavanger under den mest intense skriveperioden, og ikke minst til Silje som forstod situasjonen og støttet meg hele tiden.

Kapittel 1

Introduksjon

> I sin søken etter gull oppdaget alkymistene
> mange andre ting av større verdi.
>
> — *Arthur Schopenhauer*

Deltakersamfunnet produserer stadig større mengder av nyttige, men uheldigvis også meningsløse data som det etterhvert kan synes umulig å ha orden på. Hele tiden blir det derfor bare vanskeligere og vanskeligere å opprette nok orden og struktur i det som tilsynelatende er et eneste stort kaos. Vil oppgaven bli uoverkommelig til slutt?

Dessverre er valgmulighetene få; noen må klare å lage systemer av haugene med bits og bytes, uansett hvor endeløst det etterhvert kan virke. I tillegg krever brukerne at informasjonen skal komme frem *øyeblikkelig*. Og ikke nok med det, på toppen av det hele skal det de finner aller helst være *riktig*. Krav og forventninger er visst ikke hva de en gang var.

Denne masteroppgaven handler om en mulig løsning på slike problemer. Mer konkret handler den om tre ting: Virksomheter, søk og fri programvare.

Virksomheter. De største virksomhetene har lenge slitt med denne problemstillingen. Store, multinasjonale organisasjoner er avhengige av å kunne navigere rundt i sine gigantiske datamengder.

Et eksempel kan være et forretningsmøte hvor en kunde kommer med et totalt uventet spørsmål. Selgerne husker bare vage detaljer om noen dokumenter, og enda mindre hva de heter og hvor de finnes. Et annet scenario kan handle om forskere i et laboratorium, som uten varsel havner i en situasjon hvor de trenger nøyaktige måleresultater fra et tidligere eksperiment. Uten de livsviktige dataene går forsøket i vasken og

bedriften taper penger. Det kan tenkes ut mange flere eksempler. Felles for dem er at brukerne må kunne gjenfinne riktige data svært raskt.

Nå er datakaoset i ferd med å bli uhåndterlig for mindre bedrifter også. På grunn av utviklingen melder behovene seg lenger og lenger ned i lagene. Til slutt vil det kanskje gjøre seg gjeldende for alle, både firmaer, organisasjoner og privatpersoner. Er et nytt marked i gang med å åpne seg?

Søk. Morgendagens teknologi heter søk. Søk er lynraskt. Med tanke på de enorme mengdene av nettsider som finnes, kan det virke utrolig hvordan Google [48] klarer å gjenfinne informasjonen så raskt. Spesielt imponerende kan det virke sett i lys av at weben stort sett består av kolossale mengder informasjon som mangler tilstrekkelig struktur.

Siden søk er så raskt, og siden det finner frem på det kaotiske nettet, er det jo en besnærende tanke å overføre søketeknologi til organisasjoner. Virksomhetssøk er et voksende fagfelt. Denne oppgaven handler i stor grad om å søke etter riktig informasjon i virksomheter. I tillegg til at søk finner frem i mylderet, kan søk faktisk tilføre struktur til data som i utgangspunktet synes å mangle det.

Selv om feltet er voksende, står vanskelighetene i kø. Faktisk er det verre å få til gode søk for bedrifter og andre virksomheter, enn det er å lage søk på internett. Misforstå meg ikke, for jeg skal være den siste til å påstå at det er lett å lage en søkemotor. Jeg hevder bare at det finnes *flere* og *andre* problemer i virksomhetssøk, og at de kan være mindre forutsigbare. Faget dreier seg nemlig om å avdekke de helt individuelle behov som virksomheter har. Grunnsteinen heter søk, men det er bare starten på utfordringen.

Fri programvare. De fleste vil nok ha hørt om Linux. Lenge var det stort sett bare hackerkulturen som omfavnet åpen kildekode. Å gi bort programvare gratis var en tanke som lenge kunne synes uforenlig med markedskulturen. I dag har pendelen snudd og hackerne har vokst opp. Fri programvare ble først en "hype", men nå har utviklingen nådd enda lenger. Store programvarehus som IBM og Sun har begynt å se den direkte nytten i fenomenet [68, 119]. Det snakkes om en trend hvor man er bevisst på utviklerkultur, åpenhet og innovasjon.

Mange forbinder fri programvare med kvalitet. "Given enough eyeballs, all bugs are shallow" [90]. Søkemotorer er intet unntak. Apache Lucene [9] er navnet på den mest fremtredende av disse. Den er liten, rask og fleksibel. Skjønt søkemotor er ikke alltid søkemotor. I dette tilfellet snakkes det om en liten komponent som tar seg av det viktigste. Mesteparten rundt må man i utgangspunktet bygge selv.

Noen har dog skrevet komplette søkemotorer med Lucene som drivkraft. Problemet er bare at ingen av disse er kommet særlig langt, iallfall ikke hvis de sammenlignes med kommersielle alternativer for virksomhetssøk. Det er mulig å lage komplette løsninger ved å bruke eksisterende elementer av fri programvare, og dette gjøres allerede med relativ suksess, men som en generell og mer komplett søkeplattform kommer fri programvare til kort.

Denne oppgaven tar for seg mye av dette potensialet. Noen komponenter kan kombineres på ymse måter. Andre kan overbygges, eller alt sammen kan limes sammen og endres etter ønske. For mange er dette godt nok. Men utviklingen de siste årene har ført til at flere trenger søk. Blir virksomhetssøk ved hjelp av fri programvare for vanskelig å ta i bruk?

Er det mulig at fri programvare kan tilby noe enda bedre og enklere i fremtiden? Hvordan skal det i så fall gjøres? Vi nærmer oss noen veldig interessante spørsmål.

1.1 Problemstilling og motivasjon

Jeg lar presentasjonen i innledningen ligge litt i bakhodet og går videre til neste spørsmål. Finnes det noe som er verdt å se nærmere på? Hvilke problemstillinger reiser seg? Her er sakens kjerne:

1. Fri programvare og søk i virksomheter blir stadig viktigere.
2. Lukket og åpen programvare står på hver sin kant. De førstnevnte regjerer i det øvre markedssjiktet.
3. Allikevel er det skrevet og gjort lite for å undersøke hvordan fri programvare kan løse problemet.

Motivasjonen blir derfor å finne ut mest mulig om *hvorfor* og *hvordan* med hensyn til det siste punktet: Hvorfor har ikke fri programvare kommet lenger? Hvordan kan fri programvare møte det voksende behovet? Det er imidlertid på sin plass å detaljere disse punktene først.

Det første punktet påstår at søk i virksomheter blir stadig viktigere. Utviklingen med voksende datamengder er bare ett av tegnene på dette. Vel så viktig er det å innse at søk er i ferd med å bli allemannseie. Etterhvert som brukerne blir vant med kjappe oppslag på nettet, vil forventningene til andre miljøer, for eksempel arbeidsplassen, kunne stige. Videre er fri programvare langt på vei stuerent for bedrifter etterhvert.

Jeg er ikke alene om å hevde disse tingene. Forskningssjef Ed Thompson i analyseselskapet Gartner Group stod på talerstolen i Tønsberg i

høst. Der presenterte han hvilke ti teknologier selskapet mener er viktigst i tiden fremover. Informasjonshåndtering for store selskaper, bedriftsinterne søkemekanismer og åpen kildekode var blant disse [107]. Med hensyn til det voksende fokuset på søk fremholder Jacob Ukelson i en artikkel at:

> "There is a growing consensus that both semistructured and unstructured data sources contain business-critical information for business intelligence (BI) and operational needs." [111]

I en annen artikkel tilføyer Guy Creese at organisasjoner må innse to ting for å takle de store mengdene innhold [28]:

- Organisasjonen må forstå sine egne behov for å gjenfinne og oppdage informasjon.
- Organisasjonen må innse at de må tilpasse søkeløsninger for sine behov.

Dette er tre utsagn om at fri programvare og virksomhetssøk stadig blir viktigere. En direkte observasjon er at antallet frie søkeløsninger, spesielt for nettsider, men også for virksomheter, øker. Wikia Search [120] er ett av de aller ferskeste tegnene.

Det andre punktet forteller hvordan det kan finnes et gap mellom kommersielle aktører innenfor virksomhetssøk, og løsninger som fri programvare kan skilte med. Det er ikke dermed sikkert at leverandører av lukket programvare har så mye å frykte, siden de sannsynligvis er godt rustet til å satse på lavere markedssegmenter etterhvert. Men fra motsatt side er det rom for at fri programvare kan sikte høyere. Kanskje har virksomhetssøk enda et godt stykke å gå. La meg begynne med de proprietære.

Autonomy [16], Fast [38] og Endeca [37] er de største aktørene i virksomhetssøk for øyeblikket [24]. Firmaene har selvfølgelig siktet mot de største kundene. Der er behovene størst og markedet mest lukrativt, men etterhvert kommer stadig flere kunder fra lavere segmenter til. Disse aktørene har lang erfaring med og spesialisering innenfor søk i virksomheter. De er eksperter. Samtidig kan de holde øye med markedet, og har muligheten til å nærme seg lavere markedssegmenter ovenfra.

Apache Lucene [9] står i spissen på den andre siden, som et teknisk godt utviklet og åpent bibliotek. Programvaren som bygger på Lucene er først og fremst integrerte søk eller vanlige nettsøk, som eksempelvis Apache Nutch [10]. Imidlertid begynner det å dukke opp enkel programvare for søk i virksomheter. Mest relevant for denne oppgaven er Apache Solr [11]. Bevegelsen betrakter neppe markedet på den samme måten som de kommersielle gjør. Realiseringen av prosjektene springer først og fremst ut fra behov i utviklernes verden.

Det tredje og siste punktet sier at det er undersøkt lite om kombinasjonen av virksomhetssøk og fri programvare. Ett moment er at det har vært vanskelig å finne relevante artikler om emnet. Det jeg finner handler som regel enten om tekniske problemer i virksomhetssøk, eller om nettsøk med fri programvare. Først og fremst er nok likevel denne observasjonen viktigst:

- Det finnes ingen komplett, åpen plattform for virksomhetssøk.

Med tanke på alle delene som finnes, kan det synes litt underlig at det ikke eksisterer noen åpen programvare som tilsvarer en hypotetisk "Fast Light". Dette ønsker jeg å undersøke nærmere. Hvorfor finnes ikke dette produktet? Er kunnskapen om virksomhetssøk for liten blant hackerne? Eller er troen på åpen kildekode for svak blant eksperter på virksomhetssøk?

- Er det mulig å lage det?

Oppsummert er det nokså tydelig at søk i virksomheter kan utvikle seg mer. Creese påpeker i sin artikkel at feltet ikke er modent – "set it and forget it" er ennå ikke her [28]. En slags hypotese for hele oppgaven blir derfor at det må være mulig å skape et godt og mer helhetlig produkt for virksomhetssøk ved hjelp av eksisterende fri programvare. Dette kan også gjøres i form av et fritt prosjekt.

Det neste avsnittet tar en titt på fremgangsmåten som er fulgt, og presenterer samtidig de viktigste observasjonene og funnene som er gjort.

1.2 Observasjoner og funn

Arbeidet med denne masteroppgaven startet med en grad av optimisme og spenning. Jeg fokuserte på de tekniske løsningene og så det som absolutt mulig at en bedre og større plattform for søk i virksomheter kunne utvikles relativt lett – eller i hvert fall at det var overkommelig. Etterhvert ble jeg klokere og handlingsplanen min måtte revideres flere ganger. Utgangspunktet så noenlunde slik ut:

- Kartlegge dagens løsninger. Hva mangler de?
- Finne ut hva en plattform skal inneholde og hvordan den eventuelt kan se ut.
- Finne komponenter som kan brukes.
- Lære om andre forutsetninger.

- Utvikle ny løsning, eventuelt delta i forbedringen av eksisterende programvare.

Rekkefølgen var ikke fast. Arbeidet skjedde i flere iterasjoner og punkter fikk konsekvenser for hverandre etterhvert som jeg lærte mer. Jeg var aktiv på e-postlister, analyserte kode og prøvde å utvide prosjekter ved å programmere. Men motstanden jeg møtte var av et helt annet kaliber enn jeg var forberedt på.

De viktigste lærdommene ble:

- Teknisk sett holder fri programvare mål, og flere fremgangsmåter er mulig. Imidlertid er det ennå for langt igjen til å kunne utfordre de kommersielle aktørene. Fri programvare egner seg godt til nødvendig integrering og utvikling, men å lage en generell plattform som både fungerer ut av boksen og er tilstrekkelig fleksibel er for vanskelig å gjøre alene.

- Planen søkes utført i et domene som ennå har for lite til felles med problemdomenet. Prosjekter innen fri programvare må "leve". Hackingen springer ut fra virkelighetens behov, og virksomhetssøk er ennå ikke en stor nok del av hverdagen for utviklerne.

- Virksomhetssøk handler om å skreddersy systemer, mer enn det handler om teknologi. Det er for vanskelig å finne gode nok fellesnevnere. Dagens åpne løsninger passer derfor utmerket for konsulentselskaper som ønsker å tilpasse fri programvare til et konkret formål, men det svarer seg altså neppe å lage en mer generell plattform med det aller første.

Hva skjer fremover? Jeg tør å spekulere i at situasjonen kan være ganske annerledes allerede om få år. Hvis mye av utviklingen fortsetter i samme retning, er det langt ifra umulig at søkeløsninger blir mye mer utbredt i forskjellige virksomheter. Derfor kan fundamentet for åpne utviklingsprosjekter endre seg. Om noen år har dessuten dagens eksisterende prosjekter utviklet seg noe.

Selv om dette ikke skulle skje, og man allikevel ønsker å starte på en så stor oppgave, kan løsningen være å begynne i det små.

1.3 Oppgavens struktur

Denne masteroppgaven gir et innblikk i to store fagfelt. Det legges mye vekt på bakgrunnsstoff og teori, ettersom dette kan være ukjent for mange lesere. En grundig kartlegging og diskusjon av løsningsmuligheter er nødvendig for å kunne angripe problemet fra riktig kant.

Kapittel 1 starter med å presentere en kort bakgrunn, før problemstilling og motivasjon blir nærmere diskutert. Det gis et kort overblikk over viktige funn. Oppbygning, omfang og begrensninger nevnes også.

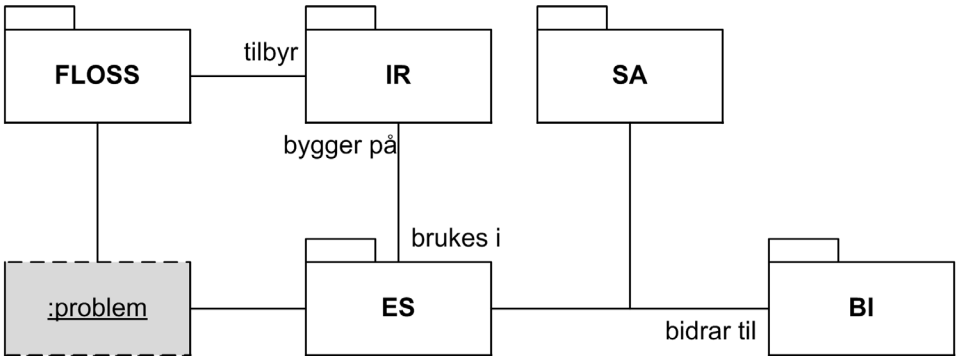
Kapittel 2 handler om søk i virksomheter, men starter med en kort innføring i søk generelt. Du søker kanskje på nettet hver dag? Enterprise Search (ES), som på norsk kan kalles virksomhetssøk, er en ganske annen sport. I tillegg presenteres forholdet til databaser. Mange ganger utfyller de hverandre, skjønt av og til vil den ene erstatte den andre. Videre ser kapitlet på alternativene for virksomhetssøk i dag, før det til sist gis en mer utfyllende bakgrunn for hvorfor dette fagfeltet sannsynligvis vil fortsette å vokse.

Kapittel 3 ser nærmere på utfordringer innenfor ES. Det blir forklart hvordan rask informasjonsgjenfinning bare er en liten del av problemet, siden virksomhetssøk handler om spesialtilpassede systemer til bestemte formål. Kapitlet gir en god innsikt i hva som er "state of the art". Mot slutten presenteres en rekke problemer som foreløpig har mangelfulle eller for dårlige svar.

Kapittel 4 går nærmere inn på informasjonsgjenfinning (IR). Fagfeltet er eldre og mer matematisk. Virksomhetssøk og informasjonsgjenfinning er ofte tett sammenvevet. Allikevel står sistnevnte på egne ben som motoren i de fleste typer moderne søk. Kapitlet tar for seg de viktige datastrukturene, og hva som skjer når data skrives inn (indeksering) og leses ut (spørring). Lingvistikk og språk, relevans, erindring og presisjon, samt skalérbarhet er avgjørende for gode søk.

Kapittel 5 setter fokuset over på fri programvare, ofte kalt Free/Libre and Open Source Software (FLOSS). Noen tror kanskje man må være altruist for å begi seg ut på slike prosjekter, men tvert imot er FLOSS en trend som fører med seg uventede fordeler. Nytenking kan oppstå som en følge av åpenhet, bare friheten brukes riktig. Hvorfor har det blitt slik, og hvordan kan dette tilføre noe til virksomhetssøk?

Kapittel 6 diskuterer de nødvendige delene i fri programvare for virksomhetssøk. Svært mange viktige fragmenter finnes allerede som åpen kildekode, og her belyses de fleste alternativene. Hva skal til for å sy disse sammen? Resultatet behøver ikke være like komplett som kommersielle konkurrenter, siden målet er å tilføre virksomheter en merverdi innenfor informasjonsgjenfinning ved å



Figur 1.1: Problemområde for oppgaven

bruke fri programvare. Potensialet er stort, men utfordringene for integrering er fremdeles litt for store.

Kapittel 7 oppsummerer og konkluderer. Et av de aller største hindrene er at virksomhetssøk fokuserer på å løse forretningsproblemer. FLOSS-samfunnet er på sin side relativt teknologifokusert. Dette gjenspeiles i at det er mange mindre aktører i søkemarkedet i dag som bruker fri programvare i sine lukkede produkter. De tekniske løsningene innen fri programvare holder sannsynligvis mål, men måten FLOSS-programvare best utvikles på (horisontale prosjektnett) faller litt utenfor det domenet som problemene søkes løst i. Det finnes tegn på at domenene nærmer seg hverandre, så kanskje vil større prosjekter eksistere om noen år.

1.4 Problemområde og omfang

I prosessen med å skrive denne oppgaven har det vært nødvendig å tilegne seg mye kunnskap. Temaet som tas opp har relevans til disse områdene:

- Enterprise Search (ES)
- Information Retrieval (IR)
- Free/Libre Open Source Software (FLOSS)
- Business Intelligence (BI)
- Systems Architecting (SA)

Det uformelle UML-diagrammet i figur 1.1 er et noe enkelt forsøk på å vise hvordan feltene henger sammen samt hvilket problem oppgaven handler om.

Virksomhetssøk (ES) sammen med godt gjennomtenkt systemarkitektur (SA) bidrar til å løse forretningsproblemer, og dette kan settes i sammenheng med Business Intelligence (BI). ES har på sin side sammenheng med mange disipliner, hvor informasjonsgjenfinning (IR) nesten alltid utgjør en viktig byggestein. Fri programvare (FLOSS) tilbyr kodebiblioteker innenfor IR og en rekke andre områder. Problemet er å finne ut hvordan FLOSS og ES kan kombineres for å gi noe som har verdi.

1.5 Avgrensninger

Oppgaven diskuterer kun i et begrenset omfang de kommersielle løsningene for virksomhetssøk. Først og fremst er Fast ESP hentet frem som et eksempel på en forretningsmessig moden løsning, og påvirkningen herfra merkes nok. En del av terminologien er imidlertid forsøkt vridd i retning av mer generelle termer.

En vel så tydelig avgrensning går mot fagfeltene forretningslogikk og beslutningsstøtte (BI), systemarkitektur (SA) og prosessforbedring. Det er helt klart at virksomhetssøk overlapper både mot forretningsproblematikk (i et informasjonsstrategisk perspektiv), og mot integrering samt overordnet arkitektur (i et teknisk perspektiv), og kanskje får det positiv innvirkning på prosesser innenfor systemutvikling og forretning.

Mange tekniske detaljer rundt informasjonsgjenfinning (IR) er utelatt, siden det tekniske fokuset rundt søk først og fremst skal være på virksomhetssøk (ES). Nødvendig bakgrunn innenfor IR er imidlertid inkludert, med spesiell vekt på de delene som har mye til felles med ES.

Med hensyn til fri programvare (FLOSS) gir oppgaven en relativt god dekning. Det er viktig å skjønne hvilke mekanismer som kjennetegner denne bevegelsen i dag. For å forstå hvilken retning den går i er det hensiktsmessig med litt historisk bakgrunn. Imidlertid konsentrerer dekningen seg om aspekter som har mer eller mindre betydning for diskusjonen og konklusjonen.

En siste avgrensning går på at oppgaven kun tar høyde for programmeringsspråket Java. Det finnes svært mange FLOSS-prosjekter i andre språk som med fordel kunne ha vært brukt i et søkeprodukt. En viktig motivasjon for denne avgrensningen går på å holde en søkeplattform så enkel som mulig, og den viktigste delen – Lucene – er skrevet i Java.[1]

[1] Dette er en sannhet med modifikasjoner. Se side 116.

Kapittel 2

Søk i virksomheter

> Jeg leter ikke. Jeg finner.
>
> — *Pablo Picasso*

Alle har et forhold til søk, og de fleste søker jevnlig på internett. Søkemotorer som Google [48] og Sesam [98] lar deg taste inn én eller flere *termer*, og som svar får du lenker og sammendrag av en rekke nettsteder. Allerede nå reiser det seg noen spørsmål. Hvordan vet søkemotoren hvilke sider du ønsker å lese? Hvorfor kommer svarene opp så raskt, og hva skjer bak kulissene?

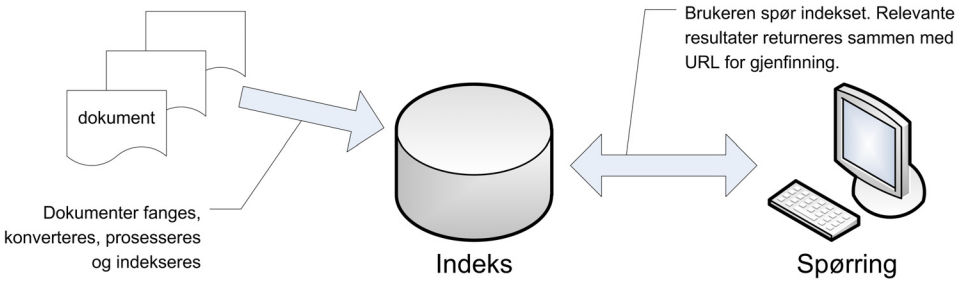
Det virker intuitivt fornuftig at søkemotoren ikke besøker hver eneste side på nettet hver gang du søker. Mange tror derfor at nettsider lagres i vanlige databaser. Jeg traff en IT-medarbeider i New York i fjor sommer, en med lang erfaring og god lønn. Da jeg meddelte ham at jeg jobbet for Fast, hevdet han hardnakket at Google er så raskt fordi det bygger på MySQL [79]. Etter gjentatte forsøk på å forklare ham sannheten ga jeg opp. Han stod på sitt.

Dette kapitlet avslører hva søk faktisk er. Google og andre søkemotorer bygger på sammenfallende fundamenter. De samme byggeklossene kan brukes i andre forbindelser også, for eksempel til å lete etter informasjon i små og store virksomheter. Senere i kapitlet presenteres fagfeltet Enterprise Search (ES), som jeg har valgt å kalle virksomhetssøk, og etter veis ende vil du nok forstå hvorfor søk er så viktig.

2.1 Hva er søk?

I grove trekk består et søkesystem av tre deler, som vist i figur 2.1 på neste side. Fra venstre mot høyre:

1. Innsamling og prosessering – først samles det som skal gjøres søkbart inn og bearbeides, slik at det passer i indekset.



Figur 2.1: Et typisk søkesystem sett fra "ti tusen meters høyde"

2. Oppbevaring i et indeks – her lagres dataene samt en varierende mengde metadata i en spesiell datastruktur.

3. Spørring og resultater – den tredje biten tolker brukerens spørringer og sender tilbake preparerte resultater.

Dette må utdypes litt.

Data som fôres inn i systemet kan stamme fra ulike kilder. For vanlige nettsøk er det crawlere, ofte kalt nettroboter, som ustanselig traverserer nettsider og fanger innhold automatisk. Mesteparten av dette innholdet er HTML-filer, men mye er PDF-filer og lignende. I prinsippet kan innholdet være hva som helst. Det neste steget er vanligvis å trekke ut fritekst og metadata som skal indekseres. I forskjellige steg blir informasjonen behandlet før den avbildes til predefinerte felter i et indeksskjema.

I midten (figur 2.1) lagres alle dataene. I stedet for å bruke en relasjonsdatabase, indekseres dokumentene i et invertert filsystem. Dette kan være distribuert over mange maskiner. Prinsippet er enkelt. Dokumentene "vris om" ved at de enkelte søkeordene peker på en liste av de posisjonene de befinner seg i. Søkeordene er i seg selv sortert. Dette er mye av magien bak lynkjappe søk i ufattelige datamengder.

På den andre siden av systemet sitter brukeren og søker, og tilbake kommer en rangert liste over dokumenter som inneholder søkeordene. Brukeren snakker ikke med systemet direkte, men benytter seg av et dataprogram. For vanlige nettsøk er dette en webapplikasjon. Den kommuniserer med det underliggende systemet, som i sin tur består av en spørreprosessor og en resultatprosessor. Førstnevnte tolker henvendelsen fra brukeren, og oversetter den som regel til et internt spørrespråk. Sistnevnte rangerer dokumentene som gjenfinnes i indekset. Det er derfor lurt at spørreprosessor og resultatprosessor samarbeider, slik at rangeringen kan foregå i henhold til brukerens søk. Applikasjonen mottar resultatene og fremviser dem for brukeren, gjerne i form av dokumentsammendrag.

Dette er hovedtrekkene i søk – indeksering og prosessering, datastrukturer, og til slutt spørringer og rangering. Denne grovmodellen be-

nyttes ikke bare av nettsøk, men stort sett av all rask gjenfinning. Til og med mange av dagens PC-programmer bruker dette prinsippet for interne søk.

Nå som det er gitt et overblikk over søk, er det på tide å forklare hvordan søk kan hjelpe for virksomheter, og ikke bare på weben. Det neste avsnittet angir derfor hva som egentlig menes med nettopp virksomheter i denne oppgaven, eller rettere sagt hva som karakteriserer dem.

2.2 Hva er en virksomhet?

Å rette spørsmålet i overskriften mot mannen i gata resulterer muligens i et svar à la "for eksempel bedrifter og organisasjoner" eller "grupper med kontrakter om økonomi, mennesker, regelverk og formål". Virksomheter kan saktens være både firmaer og organisasjoner, små og store. Virksomheter kan også være en bestemt målgruppe eller markedsgruppe. Virksomheter kan være nettverk, ja til og med systemer. Definisjonen må altså utvides noe i denne oppgaven.

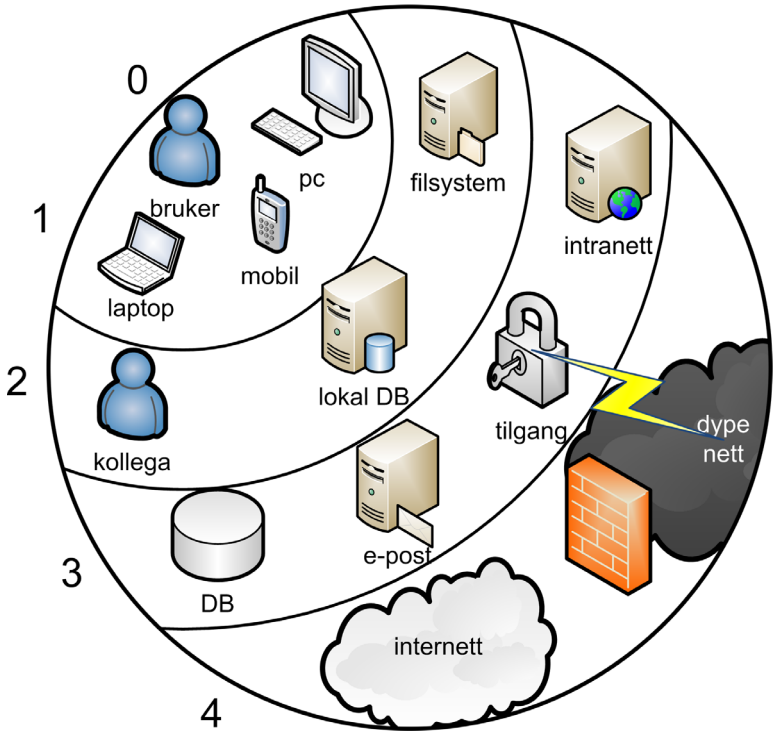
Noen typiske trekk kan være et bra sted å starte for en karakteristikk. Disse kan kanskje være [78]:

- Mangfoldighet i datakilder og -formater
- Ulike sikkerhetsnivåer og adgangskontroll
- Strukturert og ustrukturert informasjon
- Individuelle oppfatninger om relevans
- Behov for å binde sammen resultater
- Mennesker, rollemønstre og oppførsel

I tillegg har virksomheter ofte egne prosesser for generering og publisering av informasjon, som målinger, publiseringsregler og policy. Det er lett å skjønne at virksomheter kan være *svært ulike*.

Samspillet mellom virksomheter er et siste punkt. For eksempel kan et firma ha en bestemt kundegruppe, og begge kan betraktes som virksomheter ut fra det som til nå er sagt. Virksomheter kan defineres og skapes av andre virksomheter. Det engelske ordet *enterprise* gir kanskje et hint om denne kompleksiteten.

Denne diskusjonen danner et nødvendig grunnlag for å presentere det neste temaet. Grunnprinsippene for søk og den komplekse oppfatningen av virksomheter kan kombineres. Dette gir rom for nye muligheter, og det følgende avsnittet handler om å skreddersy søk etter bestemte målgrupper, kildegrunnlag, presentasjonsbehov og ønsker om integrasjon. Dette er kjernen i virksomhetssøk.



Figur 2.2: Ulike soner av informasjon sett fra en brukers ståsted

2.3 Enterprise Search (ES)

Kort fortalt går virksomhetssøk ut på å søke etter informasjon i virksomheter. Derfor kan problemstillingene være alt fra enkle, små oppgaver, som å skaffe seg noenlunde oversikt i et nettverk, til enormt komplekse systemer og plattformer. For å få en bedre forståelse er det ikke dumt å ta utgangspunkt i *brukeren* av et virksomhetssøk.

Figur 2.2 viser hvordan virkeligheten kan være.[1] Tilgjengelig informasjon er delt inn i ulike soner eller lag. Brukeren øverst til venstre i figuren forholder seg til alle disse sonene fra sitt eget ståsted.

I et tenkt scenario skal brukeren løse en bestemt arbeidsoppgave, og må følgelig lete frem informasjon om et bestemt emne. Hvordan blir fremgangsmåten?

Sone 0 representerer selve brukeren som konsulterer sin personlige kunnskap og bruker sitt referansegrunnlag for å løse problemet.

Sone 1 er brukerens personlige datamaskiner, mobiltelefon, notater

[1] Denne ideen er hentet fra problemet om virksomhetsportaler i David Hawkings "Challenges in Enterprise Search" [51], men er her videreutviklet til å gjelde en generell problemstilling innen virksomhetssøk.

eller andre hjelpemidler i umiddelbar nærhet.

Sone 2 inneholder alle informasjonsressurser i nærmiljøet eller avdelingen, for eksempel datamaskiner, måleapparater eller kollegaer.

Sone 3 kan være hele firmaet eller organisasjonen, og kan gjerne inneholde intranett, e-postarkiver eller store databaser.

Sone 4 er informasjon utenfor virksomheten. Bedrifter har gjerne tilgang til lukkede tjenester, som tilsammen danner såkalte dype nett. Svaret kan også finnes på internett.

Punktene angir ikke nødvendigvis rekkefølgen i et søk.

Figuren tar utgangspunkt i at brukeren er en typisk ansatt i en litt større bedrift, men prinsippet kan også overføres til andre virksomheter. For eksempel kan brukeren være en kunde i et bestemt markedssegment, som utgjør en slags virksomhet. Uansett handler virksomhetssøk om å løse informasjonsbehov fra brukerens betraktningsvinkel.

Virksomhetssøk dreier seg som regel om punkt 2 og 3, altså indeksering av filsystemer, databaser, egne systemer og intranett. Punkt 1 kan også være en del av løsningen ved at for eksempel skrivebordssøk integreres. ES-systemer kan dessuten sy inn kommunikasjon med eksterne innholdsleverandører. Hovedoppfatningen er at informasjon skal letes frem fra flere kildesystemer samtidig.

IR – et sentralt fagfelt

Information Retrieval (IR), på norsk informasjonsgjenfinning, utgjør et viktig fagfelt innenfor virksomhetssøk. ES og IR er altså ikke det samme, selv om mange av problemstillingene kan ligne på hverandre eller flettes inn i hverandre. Virksomhetssøk bygger ofte store og komplekse systemer rundt søk og en mengde andre systemer. IR tas opp som et eget emne i kapittel 4 på side 69. Resten av dette kapitlet på sin side for seg trekk ved virksomhetssøk spesielt. Utfordringene er ikke bare av teknisk art.

Viktigheten av datastrukturer

Kunne ikke de samme problemene ha vært løst ved hjelp av tradisjonelle databaser eller annen form for lagring? På mange måter er dette sant, men integrasjon mellom mange systemer og datakilder medfører til slutt at kompleksiteten blir høy og datamengdene uoverkommelige. Det er derfor fornuftig å bygge søk på toppen av smarte datastrukturer.

> Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious. (...) Representation *is* the essence of programming. [23]

Frederick P. Brooks, Jr. uttalte dette i *The Mythical Man-Month* allerede i 1975, basert på erfaringer fra prosjektledelse i IBM. Den gangen var informasjonsgjenfinning (IR) i sin barndom, og behovet for virksomhetssøk var mindre fremtredende. Datamaskiner ble først og fremst brukt til spesielle oppgaver og automatisering.

Eric Raymond fremholdt allikevel følgende et kvart århundre senere, basert på egne erfaringer med åpen kildekode:

> 9. Smart data structures and dumb code works a lot better than the other way around. [90]

Virksomhetssøk er i høy grad avhengig av godt fungerende datastrukturer på flere nivåer:

- Ustrukturerte data skal automatisk kunne struktureres i tilstrekkelig grad. Relasjonsdatabaser tilbyr høy grad av struktur, men også en del finesser som ikke trengs innenfor problemområdet.
- Datastrukturen må tillate rask gjenfinning. Moderne IR-teknikker søker svært raskt i fritekst, men i utgangspunktet bevares ikke nok av strukturen.

Dette representerer to ytterpunkter. Hverken tradisjonelle relasjonsdatabaser eller moderne IR-teknikker kan brukes direkte mot organisasjoner uten modifikasjoner. Virksomhetssøk krever mest mulig automatisk generert struktur, og på samme tid ekstremt høy ytelse.

De to sitatene ovenfor hevder at strukturering av data er viktigere enn effektive algoritmer. Dette er på mange måter sant også for virksomhetssøk i 2007.

Informasjonsgjenfinning ved hjelp av inverterte filstrukturer (IR) brukes i virksomhetssøk fordi skalérbarhet og ytelse står mer sentralt enn relasjonen mellom dokumenter, selv om sistnevnte ofte også er ønskelig. Manglende funksjonalitet og struktur må derfor bygges inn. Virksomhetssøk handler om nettopp det: å bygge et rammeverk rundt IR som tilbyr bedre struktur, automatisering, tilpassing og dataekstrahering, samt å presentere funn på en smart måte.

2.3.1 ES i forhold til websøk

Innledningsvis i dette kapitlet ble det påstått at de fleste tenker på Google og lignende søkemotorer når noen spør dem hva søk er. For bedre å forklare hva virksomhetssøk er, kan det være lurt å peke på noen av de

største forskjellene mellom vanlige nettsøk og typiske virksomhetssøk – eller forskjellen mellom Google og Fast, om du vil. Ulikhetene er imidlertid svært mange og omfattende. Det fokuseres her bare på tre sentrale forskjeller. Det ene er konseptet om "best kontra riktig". Det andre er at nettsøk forholder seg til en mer homogen samling av data, og det siste går på hvordan søkene implementeres.

1. Best kontra riktig.

   Den første og kanskje viktigste observasjonen inntreffer ved å studere typiske brukere. Personer som søker på nettet leter i enorme mengder med nettsider, og de har kanskje ingen anelse om hva som dukker opp. Svært ofte søkes det på helt ukjente temaer eller søkeord. De rangerte treffene undersøkes, og brukeren blir forhåpentligvis fornøyd med det dokumentet som er *best*.

   For virksomhetssøk, derimot, er det vanlig at brukeren er kjent med dokumentmengden på forhånd. Kanskje har en ansatt lest et dokument, og ønsker å gjenfinne dette bestemte dokumentet. Alle andre treff er fullstendig irrelevante. Brukeren ønsker kun å finne det dokumentet som er *riktig*.

2. Homogenitet kontra heterogenitet.

   Vanlige nettsøk forholder seg i prinsippet kun til én kilde, nemlig internett. Nettet er visstnok stort og sammensatt, men alt som kan finnes på det åpne nettet kan gjenfinnes ved hjelp av de største søkemotorene. Som et resultat kan innsamlingsteknikker og rangering raffineres i høy grad. Til en viss grad kan det muligens sies at internett er relativt forutsigbart og *homogent*.

   Virksomhetssøk indekserer mange ulike kilder, som allerede diskutert. Resultatet er at en rekke nye problemer oppstår. Det er vanskelig å sammenligne og rangere de ulike datakildene. En mengde forskjellige innsamlingsteknikker må brukes, og filformatene kan være rike i tallet. Det er heller ikke intuitivt hvordan dokumenter refererer til hverandre. Virksomhetssøk er knotete og *heterogent*.

3. Anvendelse kontra konfigurasjon.

   En viktig observasjon med nettsøk er at alle bruker den samme søkemotoren. Google er ekstremt avansert, men den er implementert én gang for alle. Det finnes få innstillinger, og den som søker kan ikke justere rangeringsalgoritmer, endre på grensesnitt eller besteme hvor ofte bestemte nettsider skal traverseres. Nettsøk satser på *anvendelse*.

> Søkemotorer for virksomheter, derimot, er beregnet på separate implementasjoner. Antallet brukere er som regel lavere, og de har ofte en uttalt kontekst og gitte forventninger. Virksomhetssøk skal passe inn i et eksisterende miljø. Det er vanlig at rangeringen justeres i hver enkelt implementasjon. Videre kan bruksområdene variere stort. Virksomhetssøk satser på *konfigurasjon*.

Denne vesle oversikten får bare frem tre av de viktigste forskjellene mellom nettsøk og ES-systemer. Resultatet blir ofte at sistnevnte står overfor større utfordringer, og disse er diskutert nærmere i et eget kapittel som begynner på side 45. Det kan sies at mange av utfordringene er de samme, bare at ES-systemer har en rekke tilleggsproblemer som må løses. I mange tilfeller må virksomhetssøk løse noen av de samme oppgavene som databaser gjør også, og derfor er det hensiktsmessig å foreta en sammenligning mellom virksomhetssøk og relasjonsdatabaser.

2.3.2 ES i forhold til databaser

Problemene som søkes løst ved hjelp av virksomhetssøk, omfatter til dels enorme mengder med data. Disse dataene kan være av både strukturert og ustrukturert art. En tradisjonell tilnærming går da ut på å kartlegge eksisterende datamateriale og -flyt, for deretter å utvikle en relasjonsdatabase over problemet. Dette krever at dataene kan struktureres etter bestemte og definerte regler. Moderne søkesystemer stiller ikke fullt så strenge krav, og kan dermed automatisere den nødvendige omformingen eller tilpassingen av dokumenter samt deres innhold i langt større grad. Denne og flere andre ulikheter mellom databaser og søkesystemer belyser at førstnevnte kanskje ikke er å egnet til å løse alle typer problemer. Allikevel vil det ofte være smart å bruke databaser og søkesystemer i en kombinasjon, på ett eller flere nivåer.

To forskjellige verdener

Siden både databaser og søkeløsninger brukes til å administrere data, blir begrepene av og til forvekslet. Bruksområdene kan være dels overlappende, men teknologisk sett er det snakk om to forskjellige verdener.

I en vanlig relasjonsdatabase innordnes data etter regler spesifisert i et skjema. Skjemaet inneholder tabeller. Disse tabellene har relasjoner til hverandre ved hjelp av blant annet fremmednøkler, og de inneholder strukturerte data. Relasjonsdatabasens styrke ligger i at biter av informasjon relateres til hverandre, altså har slektskap, og at alt dette alltid oppfyller gitte kriterier.

Søkesystemer skiller seg fra denne modellen. De baserer seg også på et forhåndsdefinert skjema, men den viktigste byggesteinen er ikke

tabeller; snarere *dokumenter*.[2] Søkesystemer er basert på en spesiell datastruktur – et invertert filsystem. Dette forklares bedre litt senere, men poenget er at styrken ligger i at data kan gjenfinnes svært raskt. I tillegg behøver ikke dataene være strukturerte, selv om de defineres inn i felter på en måte som kan minne om tradisjonelle databaser.

Hovedforskjellen mellom de to kan derfor oppsummeres slik: I relasjonsdatabaser tilpasses data en streng struktur som innbyrdes relateres til hverandre matematisk. I et søkesystem kartlegges dataene etter gitte felter og lagres i en invert filstruktur for rask gjenfinning. Herav kommer uttrykket *informasjonsgjenfinning*. Dermed blir bruken også ulik. Databaser brukes mest til lagring av data, mens søkesystemer ikke nødvendigvis lagrer alle dataene. Først og fremst inneholder de pekere til informasjon. En viktig opplysning blir derfor alltid *hvor* denne informasjonen befinner seg.

Fritekstsøk er vanskelig i databaser

Informasjon som lagres i en vanlig database kan gjenfinnes ved hjelp av et strukturert spørrespråk, for eksempel SQL. Dette er ideelt for mange bruksområder. Men behovene har endret seg over tid, og i dag ønsker mange mindre kyndige brukere av datamaskiner å gjenfinne informasjon. Det mest naturlige for disse personene er å foreta en spørring ved hjelp av naturlig språk, gjerne formulert som et direkte stilt spørsmål [95]. Her faller relasjonsdatabasene gjennom. De støtter ikke kraftfulle finesser som eksempelvis uklare (fuzzy) søk, se side 70.

Riktignok er det mulig å indeksere enkelte felter slik at spørring mot disse går raskt. Likevel ønsker stadig flere brukere å søke på tvers av enorme mengder data for å finne informasjon som er *relevant* i forhold til et naturlig stilt spørsmål. I et slikt fritekstsøk er brukeren som regel heller ikke interessert først og fremst i hvordan ulike entiteter relaterer til hverandre, men altså hvilken relevans den gjenfunnede informasjonen har til spørringen. Dette er én av hovedårsakene til at databaser ikke egner seg for generelle fritekstsøk; de finner informasjon definert etter konsise regler om blant annet projeksjon og seleksjon. Informasjonsgjenfinning finner derimot ut hvor mye eller hvor lite for eksempel et dokument er ønsket.

Det er selvfølgelig mulig å kjøre analyser over store mengder med data lagret i en relasjonsdatabase. Ytelsesmessig gir det imidlertid begredelige resultater, siden en rekke aktuelle bruksområder setter svært høye krav til svartid. Informasjonsgjenfinning er den riktige teknologien for fritekstsøk.

[2] Nøyaktig hva et dokument består av er nærmere forklart i seksjon 2.5.3 på side 39.

Søk og databaser kan kombineres

Informasjonsgjenfinning (IR) har ofte den begrensningen at kun alminnelig tekst indekseres. Det er ikke fullt så vanlig å large binærfiler her som i databaser. Ved å være varsom i oppsettet av et søkesystem, er det selvsagt mulig å ta vare på all informasjon slik at de originale filene eller dokumentene kan gjenskapes. I de fleste tilfeller er dette imidlertid ikke ønsket, og det er heller ikke helt optimalt.

Det forklares senere hvordan informasjon består av henholdsvis dokumenter og innhold. Det er sistnevnte som er interessant for fritekstsøk. Ved å lagre alt innholdet, i tillegg til informasjon om hvor det originale dokumentet befinner seg, utfyller søkesystemer og databaser hverandre. Dokumentets lokasjon, også kalt URL, vil i dette tilfellet være en SQL-spørring som angir de nøyaktige tuplene i relasjonen som er gjenfunnet.

Jo mer komplekst databaseskjemaet er, og jo mer fritekst de enkelte feltene inneholder, dess større er gevinsten ved å supplere databasen med et verktøy for informasjonsgjenfinning som er helt eller delvis frittstående.

Det kan ofte svare seg å kombinere tradisjonelle databaser og IR-systemer. Ulike tilnærminger kan være:

- Bruk av databaser internt i et IR-system. Spesielt store systemer for virksomhetssøk kan med fordel bygge inn databaser for å holde orden på konfigurasjon, logger eller andre interne ressurser.
- Bruk av IR-systemer som teppe over databaser. Formålet her er å utstyre eksisterende databaser med fulltekstsøk. DBSight [30] er et eksempel på programvare som kan indeksere én eller flere databaser fra ulike leverandører og tilby friteksøk på tvers av all informasjonen.
- IR-systemer kan brukes som indeksaksessmetode for databasesystemers interne oppslag. Eksempelvis har Lie og Clausen vist i sine hovedoppgaver [26, 69] hvordan den proprietære søkeplattformen Fast FDS [38] (nå ESP) kan integreres i FLOSS-databasen PostgreSQL [87].

Uansett hvilken tilnærming som velges, bør en tett integrasjon mellom databaser og søkesystemer bevare noen viktige trekk fra begge verdener [31]:

1. Støtte for lagring, indeksering, gjenfinning og endring av dokumenter.

2. Transaksjonssemantikk (ACID) slik at alle operasjoner blir atomiske, konsistente, isolerte og bestandige.

3. Samtidig opprettelse, endring og gjenfinning av dokumenter.

4. Utvidelser for databasespråket for å støtte rangering under gjenfinning.

5. Skalérbar ytelse under indeksering og søk.

Et vanlig bruksområde for IR er dessuten å avlaste databaser med stor spørretrafikk (database offloading).

Hva med databaser i fremtiden?

De neste avsnittene gir en viktig belysning av begrepene data og informasjon. I informasjonsalderen får søkeløsninger basert på informasjonsgjenfinning (IR) stadig større innpass. Teknologier rundt effektivisering av IR hos virksomheter er under sterk utvikling. Er det utenkelig at stadig større krav til ytelse i enorme datahauger på sikt vil gjøre tradisjonelle databaser overflødige?

Selv om enkelte i den kommersielle søkebransjen uttaler at søk vil ta helt over [114], er det nok ennå et godt stykke igjen til virksomhetssøk er et like modent fagfelt som relasjonsdatabaser. IR har definitivt gått i bresjen for søk på internett, men for virksomheter er behovene mye mer individualiserte og komplekse. I avsnitt 3.4 på side 56 diskuteres noen viktige oppgaver som virksomhetssøk med fordel kan finne bedre løsninger på.

Det kan også spekuleres i om relasjonsdatabaser vil overleve uansett. De representerer et litt eldre, strukturert dataparadigme som fremdeles er aktuelt, og gir éntydige og binære (boolske) svar på strikte spørringer. Dette, kombinert med databasenes enorme styrke i å relatere biter av strukturert informasjon til hverandre, tilsier nok at innføringen av IR-systemer neppe betyr slutten for de tradisjonelle databasene riktig ennå.

2.4 Dagens status

Hvordan ser markedet for virksomhetssøk ut i dag? Forrester [24] deler markedet grovt sett inn i to segmenter, nemlig plattformer og løsninger. Stadig flere henger seg på i kampen, men de konkurrerer altså innenfor litt ulike segmenter. I tillegg kan fri programvare regnes for å være et slags tredje segment. Rapporten fra Forrester nevner ikke fri programvare med ett ord – et sikkert tegn på at markedsandelen er forsvinnende liten. Men hva er det egentlig som kjennetegner de ulike segmentene?

Plattformer er et segment med relativt få kommersielle aktører, og disse tilbyr stort sett egen teknologi. Fokuset ligger på å kunne tilby alle tenkelige løsninger, eller rett og slett å løse så mange problemer som mulig ved hjelp av søk. Aktørene her tilbyr et felles søkesystem med spørreprosessor, indeks og bindeledd, i tillegg til alle mulige slags tilleggsverktøy. Tradisjonelt har konkurransen dreiet seg om tre ting: å kunne koble seg til størst mulig antall kildesystemer, god skalérbarhet og tilleggsfunksjonalitet som eksempelvis rapporter, sikkerhet og klassifiseringsverktøy.

Søkeløsninger, på den andre siden, er et segment hvor aktørene vektlegger bestemte problemtyper. Forrester-rapporten deler disse inn i fem kategorier [24]:

1. *Intrasøk* – omfatter blant annet intranett og portaler, ECM (innholdsstyring), samarbeidssystemer, opplæring og virksomhetssystemer.

2. *Intelligente systemer* – kan være markedsanalyse, overvåkning, sikkerhet, kvalitetsstyring og lignende.

3. *Avlasting av databaser* – for eksempel datavarehus eller mellomlagring av data.

4. *Handelssystemer* – omfatter eksempelvis e-handel, kundesystemer og brukerstøtte.

5. *Mediasystemer* – disse kan være sammenslåing av nyheter, multimediesøk, biblioteker og annet.

Videre nevner rapporten at punkt 1, 4 og 5 utgjør modne områder med mye konkurranse, og at punkt 2 og 3 er på vei opp. Spesialisering er altså nøkkelen.

Fri programvare er et siste segment, og da mener jeg fri programvare som fenomen og bevegelse, fremfor teknologi og produkter. En rekke mindre og større selskaper selger løsninger som omfatter fri programvare, men disse havner ikke i denne kategorien. Fri programvare som fenomen medfører at brukere får kjennskap til produktene, tar dem i bruk og deltar ved å utvikle dem videre. Det kan muligens kalles en plattform, men ikke på den samme måten som over. Kapittel 5 forklarer mer.

I årene som kommer vil markedet kunne vokse. Dersom det skjer, er det også sannsynlig at det kommer et press på priser hos de kommersielle. Forrester-rapporten [24] gir dessuten tegn på at dette er i ferd med å skje gradvis etterhvert som store, tradisjonelle selskaper kaster seg på karusellen. De to neste avsnittene forteller kort om de ulike aktørene i markedet og hva de kan tilby.

Kommersielle aktører

Autonomy [16], Fast [38] og Endeca [37] er markedsledere innenfor virksomhetssøk. Autonomy kjøpte i desember 2005 en av sine argeste konkurrenter, Verity. Sistnevnte var en pionér innen feltet på midten av 90-tallet, og Fast Search & Transfer vokste ut av søkemiljøet i Trondheim omtrent på samme tid. Fast er siden blitt et norsk teknologieventyr med kontorer i alle verdensdeler og store, viktige kunder. Flaggskipet heter ESP – Enterprise Search Platform. Endeca har også et sterkt produkt, men står klart i skyggen av de to store.

Større, tradisjonelle aktører har begynt å melde seg på i kampen. IBM, Google og Microsoft tilbyr alle løsninger; førstnevnte tilbyr sågar en bred plattform. Riktignok kan ikke IBM OmniFind hamle opp med Fast ESPs markedsposisjon ennå, men IBM har et svært spennende produkt: IBM OmniFind "Yahoo! Edition" er gratis virksomhetssøk basert på åpen kildekode, nemlig Apache Lucene [9]. IBM tilbyr noe fra det aller minste og oppover til tunge plattformer for multinasjonale bedrifter, og angriper altså markedet totalt. Google er mer fokusert på løsninger. Google Search Appliance er et kjent produkt som svarer til punkt 1 i forrige avsnitt, altså *intrasøk*. Den er så lukket at den kommer som et fysisk produkt i form av en server.

Entopia, Microsoft og Convera er andre aktører, men Forrester-rapporten mener det skal mye til for at disse skal strekke seg opp til toppen. I tillegg finnes det som sagt en rekke små og løsningsorienterte selskaper som bygger på fri programvare – typisk Lucene.

Fri programvare

Så hva er det som er så spesielt med Lucene? Saken er først og fremst at Lucene står så å si helt alene om å tilby et effektivt IR-bibliotek innenfor fri programvare. Andre forsøk har stort sett feilet, kanskje med unntak av Egothor [36]. Det er imidlertid Lucene som har all oppmerksomheten nå, og siden biblioteket har en fleksibel lisens kan det brukes til hva som helst – både av åpne og av lukkede prosjekter. Dermed har Lucene kanskje klart å oppnå større utbredelse enn dersom det hadde vært solgt.

Holder Lucene mål teknologisk sett? Ifølge en rekke artikler og presentasjoner på hjemmesiden [9] er ytelsen svært god. Antallet prosjekter som er basert på Lucene bærer kanskje et tydeligere vitnesbyrd om at kvaliteten holder mål. I spissen finnes søkemotoren Nutch [10] som i dag implementeres i en åpen Google-konkurrent kalt Wikia Search [120], så den skalérer nok bra. I tillegg finnes det et svært lovende produkt kalt Solr [11], som tar sikte på å bli et fullstendig produkt for virksomhetssøk.

Andre halvdel av kapitlet om fri programvare (fra side 114) tar for seg flere av disse prosjektene, og hvordan de er bygget opp av eksisterende komponenter av fri programvare. Det mest interessante for denne oppgavens del er hvordan disse kan brukes videre for å skape en mer allmenn ES-plattform på den samme måten som Autonomy, Fast, Endeca eller IBM kan tilby det. Lucene, og for den saks skyld også Nutch og Solr, er svært grunnleggende – men gode – prosjekter som potensielt kan få meget stor utbredelse hvis markedet for virksomhetssøk fortsetter å vokse.

"Fast versus Lucene"

Det er fristende å foreta en sammenligning à la "Fast ESP versus Apache Lucene". Problemstillingen er imidlertid ikke så enkel. Det forklares kanskje med en analogi: Prøv å gjøre sammenligningen "Windows 2003 Enterprise Server SP2 versus Linux 2.6.9-42"! Det blir litt meningsløst. Da er det heller bedre å forklare forskjellene mellom proprietære og åpne søkeløsninger.

Forskjellene er konseptuelle. De lukkede løsningene, som Fast ESP, har kommet relativt langt som plattformer. Det tilbys en samlet pakke som kan brukes til mangt, på samme måte som Microsoft Windows kan brukes til en rekke ting. Lucene, på sin side, tilbyr bare kjernen i et søk, og resten må bygges selv. I operativsystemverdenen kalles dette enten GNU eller en distribusjon, alt etter hvordan man velger å se på det.

Dermed blir det slik at de kommersielle tilbyr en lang rekke støtteverktøy, programmer, administrering og lignende, mens fri programvare satser på enkelhet i produktene og fokus på et lite sett med funksjonalitet. Lucene er et IR-bibliotek, et veldig bra et, også, men ikke noe mer. Senere i denne oppgaven blir det diskutert hvordan Lucene og avledede produkter kan brukes til å lage noe som matcher de kommersielle, i hvert fall med hensyn på verdien slike produkter kan ha for virksomheter – både som plattform og som løsninger på enkle scenarier.

"Fast versus Lucene" er en sammenligning som blir belyst mer eller mindre mellom linjene, i hele denne oppgaven.

2.5 Data og informasjon

I denne oppgaven gjøres det et lite skille mellom begrepene *data* og *informasjon*. I problemstillinger rundt søk forekommer begge deler, og virksomhetssøk er intet unntak. Data finnes som regel i større mengder enn informasjon, og det kan argumenteres for at data har lavere verdi. Informasjon er på sin side verdifull av forskjellige årsaker. Allikevel er ikke ordet informasjon éntydig og må derfor presiseres. Forskjellen

mellom informasjon og data eksisterer dessuten kun for mennesker og brukere, og ikke for datamaskiner. Siden virksomhetssøk handler om å gjenfinne verdifull informasjon for brukere i en gitt kontekst, kan det være nyttig å differensiere mellom de to begrepene.

▷ Data måles *ofte* kvantitativt.

Data kan naivt defineres som en endelig mengde vilkårlige tegn, instruksjoner eller lignende. En god karakteristikk av data er derfor at de kan være helt tilfeldige eller synes vilkårlige, og at de heller ikke er bearbeidet på noen måte. Et eksempel er rikholdige målinger foretatt i en partikkelakselerator. Et annet kjennemerke ved data er at de kan være *informasjonsbærere*, uten at det dermed er et krav. Et eksempel på dette er en JPEG-fil som i seg selv utgjør en mengde tilsynelatende vilkårlige data dersom filen inspiseres byte for byte, men som samlet inneholder et bilde, eller kanskje bare støy. Et siste kjennetegn ved data er derfor at de angivelig forekommer i relativt store kvanta. Altså er det ikke mulig å fastslå verdien av noen gitte data ved å se på dem som data alene.

▷ Informasjon måles *ofte* kvalitativt.

Informasjon er litt mer abstrakt. Verdifulle data for en gitt kontekst kan utgjøre informasjon. Et godt eksempel på denne beskrivelsen er dataene *seks milliarder*, som i konteksten "antall mennesker på jorda" representerer informasjon. Analyser av data kan hente ut mer informasjon, og sammen med referansebakgrunn gir fortolkninger ny informasjon. Kunnskapen om at jordas befolkning var tre milliarder for n antall år siden tilfører, i tillegg til den gitte informasjonen, avledet informasjon om at befolkningen har fordoblet seg i løpet av det samme tidsrommet. Foredling av data kan på denne måten medføre økt informasjon, slik en JPEG-fil kan vise et meningsfullt bilde gjennom en forhåndsbestemt fortolkning (algoritme). Informasjon har altså en viss *verdi* og *semantikk* og fremstår som *foredlet*, fordi noen mennesker har tatt seg bryet ved å samle, oppdage eller fremstille dataene de representeres av. Det kan virke klokt å strukturere informasjon fremfor data.

Data og informasjon er altså ikke det samme. Datamaskiner behandler i utgangspunktet ikke disse forskjellig, men mennesker gjør. Det er nyttig å samle data hvis de kan bearbeides til informasjon ved et senere tidspunkt. Siden informasjon er kontekstavhengig (data for noen kan være informasjon for andre, og omvendt), og siden virksomhetssøk ofte er svært kontekstspesifikt, har begrepet *informasjon* ekstra stor betydning for virksomhetssøk.

Data kan danne grunnlag for forretningsmessige beslutninger, men først etter en menneskelig fortolkning. Informasjon har relativt langt

høyere verdi enn data, men må vurderes i kontekst. En bruker av virksomhetssøk ønsker å tilegne seg informasjon, og ikke data. I det neste avsnittet gjøres det et forsøk på å tilbakevise eller begrense myten om at informasjonsmengden er sterkt økende.

2.5.1 Datalagring – en flaskehals

Moores lov sier at antallet transistorer som får plass på et gitt areal fordobles hver 18. måned.[3] Denne "loven" omhandler egentlig kun transistorer, selv om det kan synes å være en generell oppfatning i IT-bransjen at kapasiteten på datamaskiner generelt øker i samme hastighet. Imidlertid tyder utsagn fra ingeniører i lagringsbransjen på at kapasiteten på datalagring øker raskere enn for halvledere [52, 118, 2]. Samtidig er hastigheten for datalagring avhengig av nettopp halvledere. Vil dette etterhvert bli et problem? Hvilke implikasjoner får det i så fall for fremtidig datalagring?

Den legendariske informatikeren John Backus påpekte allerede i 1977 forløperen til dette problemet, som han kalte *von Neumanns flaskehals* ("the von Neumann bottleneck") [17]. Arkitekturen i dagens maskiner bygger på von Neumann-modellen og kan således vanskelig unngå problemet med mye utveksling av data når algoritmer eksekveres. I dag har problemet fått eskalere i 30 år siden Backus' forelesning.[4]

Moderne datamaskiner inneholder deler som hver for seg lagrer og behandler data i ulike mengder og med ulik hastighet. De viktigste delene kan rangeres fra de raskeste med minst kapasitet, til de treigeste med størst kapasitet. Rekkefølgen blir da seende omtrent slik ut:

$$\text{CPU} \longleftrightarrow \text{RAM} \longleftrightarrow \text{HDD} \tag{2.1}$$

Doblingen av antallet transistorer hver 18. måned i minnebrikker har ført til at det tar stadig lenger tid å aksessere hele hurtigminnet. En tilsvarende relativ distanseøkning er tilfellet for forholdet mellom hurtigminne og harddisk. Dette har medført en rekke tiltak. Spesielt nevnes den stadige innføringen av mellomlagring, eller cache, på ulike nivåer. Moderne prosessorer har flere nivåer av cache for å fylle det voksende gapet. På den samme måten har harddisker fått innført cache, og meget snart vil disse utstyres med enda et nivå av mellomlagring ved hjelp av flash-minne i statiske disker og hybriddisker. For å oppsummere med

[3] Moores lov burde være kjent for de fleste. Det finnes mange noe feilaktige omformuleringer av dette utsagnet. Den mest vanlige går ut på at regnekraften til en datamaskin fordobles hver 18. måned.

[4] Den nylig avdøde [72] Backus ble i 1977 tildelt Turing-prisen for sitt arbeid med høynivåspråk, BNF og funksjonell programmering. Artikkelen [17] henviser til forelesningen han holdt i forbindelse med denne prisen.

utgangspunkt i rangeringen ovenfor, kan situasjonen i skrivende stund se omtrent slik ut:

$$\text{CPU} \leftrightarrow \text{cache } 1 \ldots \text{cache } n \leftrightarrow \text{RAM} \leftrightarrow \text{cache} \leftrightarrow \text{flash} \leftrightarrow \text{HDD} \qquad (2.2)$$

Denne oppstillingen er ingen fasit, men gir et generelt bilde av alle mellomstegene som stadig opprettes. Lengst til venstre er CPU-ens register med svært liten lagringsplass, men ekstremt rask tilgang. Lengst til høyre står harddisken med enorm lagringskapsitet, men tilsvarende langsom tilgang.

I en artikkel i Scientific American i august 2005 uttalte Mark Kryder[5] at økningen i kapasitet på harddisker vokser i et høyere tempo enn økningen i prosessorkapasitet, altså beregningshastighet [118]. Senest i fjor beviste hackernettstedet *Tom's Hardware* at dette stemmer. I en artikkel publisert den 27. november 2006 ble blant annet følgende resultater presentert [97]:

1991: En moderne harddisk hadde en tetthet på 26 MB per plate. Lesehastigheten var 0.7 MB/s, og søketiden var 27 ms. Det tok 37 sekunder å skrive en hel plate med data.

2006: En moderne harddisk hadde en tetthet på 200 GB per plate. Lesehastigheten var 64 MB/s, og søketiden var 13 ms. Det tok 52 minutter å skrive en hel plate med data.

I løpet av 15 år har altså kapasiteten på harddisker blitt opp mot 10000 ganger så stor, mens lese-/skrivehastigheten har blitt nesten 100 ganger så høy. Testen fremholder dermed at det tar nærmere 100 ganger så lang tid å lese en hel plate med data. Det kan synes som at opplevd hastighet bare er én prosent av det den var for 15 år siden. I virkeligheten er nok ikke dette helt sant, ettersom størrelsen på datafiler og programmer ikke har økt i like stor takt. Men dagens harddisker oppfattes allikevel som vesentlig treigere enn gårsdagens.

Problemene må løses. Det er allerede nevnt at innføringen av flere mellomlagre, i neste rekke flash-disker, vil bremse problemet. Bedre skalérbarhet i maskinvarearkitektur er også nødvendig, som påpekt av John Hennessy [52]. En tredje tilnærming er å løse utfordringene ved å skrive bedre programvare, slik Backus foreslo [17].

Denne oppgaven dreier seg om det sistnevnte. Programvare, i dette tilfellet søkeverktøy, kan imidlertid ikke løse hele problemet. For eksempel vil ikke sanntidsredigering av video kunne dra særlig nytte av dette. Det gjøres derfor ikke forsøk på å komme med revolusjonerende løsninger. Men det er påvist at antallet filer i datasystemer, at datamengden øker, og at det stadig tar lenger tid å bla igjennom alt. Håpet er at

[5] Mark Kryder er CTO (Chief Technology Officer) ved Seagate Technology, en av verdens ledende innovatører innen lagring og produsenter innen harddisker (2007).

investeringer i søketeknologi kan bidra til reell effektivisering for organisasjoner og privatpersoner.

2.5.2 Strukturerte og ustrukturerte data

Det ligger i informatikkens natur å håndtere data på en strukturert måte. Datamaskiner "tenker" strukturert og ordnet. Utvikling av algoritmer går i korte trekk ut på å definere systemer for behandling av data, og regne seg frem til den beste måten å håndtere store mengder av disse dataene på i henhold til systemet som ble definert. Derav kommer det at det er enklest å håndtere data når de i all hovedsak er like eller ligner på hverandre.

Søkesystemer skiller seg ut på den måten ved at de forsøker å gjenfinne ustrukturerte data. Også her ordnes data etter en struktur ved bruk av algoritmer, men behovene for ytelse er som regel større ettersom datamengdene kan være enorme.

Strukturerte data kan defineres som allerede ordnet innhold. Svært verdifull informasjon kan for eksempel struktureres i henhold til relasjoner (tabeller) med innbyrdes referanser. Denne kunsten kommer neppe til å forsvinne med det første. Blumberg og Atre referer imidlertid til analyseselskapet Merrill Lynchs tall, som fremholder at over 85 prosent av all forretningsinformasjon foreligger i ustrukturert form [19].

I forrige avsnitt ble det påpekt at datamengdene øker med tiden, for eksempel for en gitt bedrift, og at det tar stadig lenger tid å gjenfinne verdifull informasjon blant data som ikke er strukturert. Altfor ofte er det en meget kostbar jobb å strukturere slike data og legge dem i en database, til tross for at de kan være vitale for en virksomhet. Totalt ustrukturerte data må også kunne systemeres – eller iallfall søkes i – med akseptabel ytelse.

Mye data kan regnes for å være en mellomting, nemlig semistrukturert. Dette kan bety data som i noen tilfeller er strukturert, og i andre tilfeller ikke. Det må være mulig å skrive et analyseprogram eller en algoritme, som ved hjelp av definerte filtre strukturerer visse kategorier og finner hovedtrekk og emner i innholdet, samtidig som mesteparten av teksten betraktes som ustrukturert. Det eksisterer dog et litt mer definerbart syn på hva semistrukturerte data er. Ustrukturerte filer med konsistente metadata har en ustrukturert og en strukturert del, og betraktes derfor som semistrukturert. I denne oppgaven er sistnevnte definisjon gjeldende.

Konklusjonen blir at det er ikke lett å skille mellom disse definisjonene. Det neste avsnittet gir derfor noen eksempler innenfor hver kategori.

Eksempler

Det går ikke noe skarpt skille mellom strukturerte og ustrukturerte data, og det finnes derfor ingen fasitsvar på hva som hører til hvor. Figur 2.3 viser noen eksempler innenfor kildefiler, innhold og datahåndtering, og kan bidra til å gi et inntrykk av typiske tilfeller.

Når det gjelder kildefiler, er XML et godt eksempel på en bærer av strukturert innhold, mens budskapet i en Word-fil typisk er ustrukturert. Allikevel kan tilfellet være det motsatte. Dokumentformatet er ikke i utgangspunktet avgjørende for innholdets art.

Litteratur og presentasjoner er gode eksempler på ustrukturert innhold. Samtidig kan resultater fra markedsundersøkelser inneholde flere ordnede felter, og det betraktes derfor som strukturert.

Den klassiske formen for lagring av strukturerte data er relasjonsdatabaser (RDBMS). XML er en populær bærer av strukturerte data, spesielt innenfor kommunikasjon/datautveksling på applikasjonsnivå. Informasjonsgjenfinning (IR) er en svært god måte å lagre og indeksere ustrukturerte eller semistrukturete data på.

Problemet blir til sist situasjonsbetinget. Om utviklere velger å betrakte innhold som det ene eller det andre, kommer i størst grad an på hva man har tenkt å gjøre med dataene etterpå. Skal sammenhenger og referanser være absolutte, passer en strukturert tilnærming. Skal dataene gjenfinnes raskt, men ikke brukes som en database? I så fall kan dataene med fordel betraktes som ustrukturert materiale.

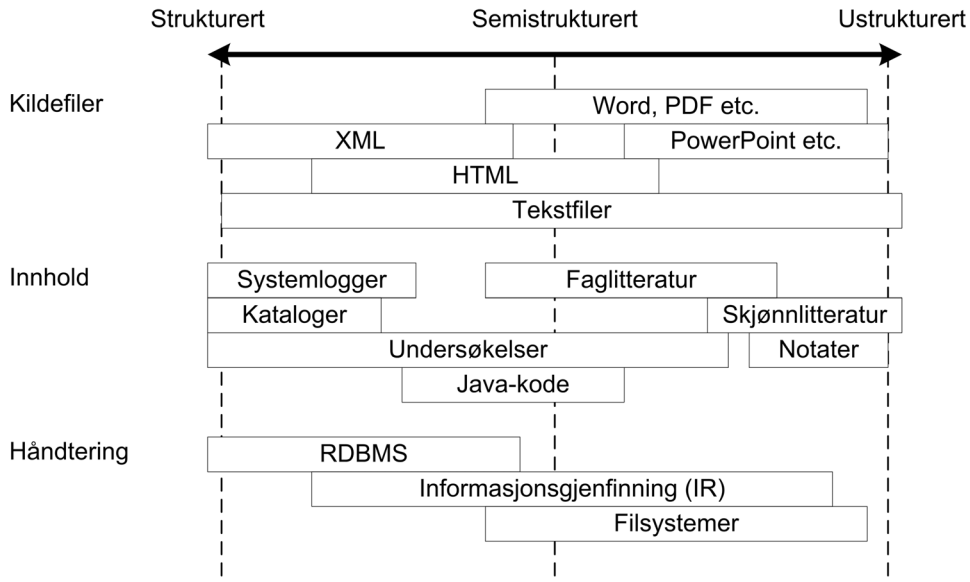
I seksjon 2.5.4 på side 41 blir ulike måter å ordne informasjon på tatt opp.

2.5.3 Dokumenter og innhold

Innenfor virksomhetssøk er ikke dokumenter og innhold det samme, og det er velbegrunnet. Dokumenter er biter av ubehandlede og opphavsmessige data som samles inn under innmatingsprosessen.[6] Det er oftest dokumentenes innhold som blir indeksert. Av denne årsaken kan det i noen tilfeller være vanskelig å gjenskape et kildedokument, men det er ikke nødvendigvis negativt. Det er viktig å være klar over forskjellen på kildedokumenter og indekserte dokumenter.

Et kildedokument kan eksempelvis være en fil i et åpent eller lukket format som PDF, Word-dokument, presentasjon, regneark, HTML, Javakode og så videre. Kildedokumenter er ikke nødvendigvis filer allikevel. Her er noen eksempler på andre ting et kildedokument kan være:

[6] Innmatingen består av innsamling, analysering og dokumentvask, samt indeksering. Dette blir forklart i avsnitt 4.3 på side 72. I ren IR-sammenheng er dokumenter en indeksert forekomst.



Figur 2.3: Eksempel på klassifisering av strukturerte og ustrukturerte data

- En del av en fil. Dette er opp til bruksområdet for virksomhetssøk. Hvis det er ønskelig å skille mellom ulike deler av en fil og presentere dem i et resultat hvor de listes opp hver for seg, blir dette riktig.
- Et databasetuppel fra en relasjon. Selve relasjonen kan stamme fra én eller flere tabeller. Det vesentlige er at én bestemt spørring har blitt indeksert, og at hvert av tuplene i resultatrelasjonen har blitt ansett som og dermed blir lagret som separate dokumenter.
- En måling fra et apparat. Det kan være opp til et ekstern dataprogram å dele en datastrøm opp i mindre biter. Hver av disse bitene kan inneholde en del ulike data og metadata.

Hvor et kildedokument stammer fra og hvilken form det har, eller om det faktisk er blitt sammensatt i et komplisert steg på veien, er helt uvesentlig for definisjonen. Et indeksert dokument er sammensatt av alt innholdet som er lagret i IR-systemet om kildedokumentet.

Innhold er på den andre siden definert som potensielt informasjonsbærende data som trekkes ut fra kildedokumenter eller indekserte dokumenter. Eksempler på innhold kan være:

- Teksten på en nettside.
- Nøkkelord (metadata) fra et bilde.

- Tallene i et regneark.
- Den samlede semantikken i et Word-dokument.
- Brødteksten i et Word-dokument.
- Sted, temperatur, luftfuktighet og tidspunkt fra en værmåling.

Det er på ingen måte gitt at innhold er interessant for enhver bruker, og dermed gir noen informasjon. Alt innhold er allikevel data som hypotetisk sett kan være interessant i en gitt kontekst. Av denne grunnen er innhold informasjonsbærende.

Mens et kildedokument er en kombinasjon av syntaks og semantikk, eller form og mening om man vil, så er informasjon den semantikken som kan trekkes ut fra dokumenter. Av dette følger det at kildedokumenter i utgangspunktet bare delvis kan gjenskapes fra et indeks, ettersom det er informasjonen som lagres og ikke dokumentet. Hva som betraktes som semantikk og dermed hva som utgjør innholdet i et dokument, er selvfølgelig avhengig av sammenhengen. I noen tilfeller er det ønskelig å kunne rekonstruere originaldokumenter, men for de fleste applikasjoner er det det trivielle og altså faktiske innholdet som er viktigst.

I de fleste tilfeller er innholdet rett og slett teksten samt metadataene i et dokument. Seksjon 4.5 på side 86 gir forøvrig noen mer formelle definisjoner på kvalitet.

2.5.4 Ordning av informasjon

Ordning av informasjon kan logisk sett skje på ulike måter. Her nevnes tre tilnærminger som med fordel kan benyttes innenfor virksomhetssøk. Den første måten, hierarki eller taksonomi, er nyttig i for eksempel nettbutikker og biblioteksystemer. Relasjonsmessig ordning er mer vanlig i tradisjonelle databaseapplikasjoner. Assosiative systemer er en tredje måte som er blitt mer vanlig med fremveksten av “Web 2.0”.

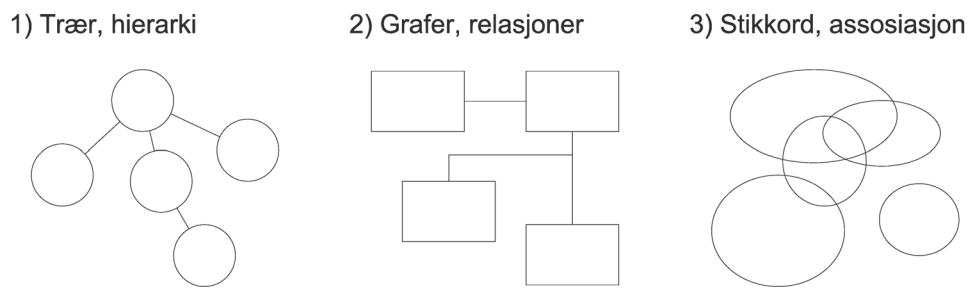
Figur 2.4 forsøker å vise de tre forskjellige filosofiene. Merk spesielt delfigur 3 som viser hvordan stikkord danner kompliserte delmengder over dokumenter med assosierte tilknytninger.

Taksonomi (trær)

Taksonomi går ut på å gi entiteter en hierarkisk tilhørighet. Dette brukes blant annet i den biologiske systematikken:

$$klasse \rightarrow orden \rightarrow slekt \rightarrow \cdots$$

Ved å kategorisere entiteter på denne måten bygges trær. I informatikken benytter katalogstrukturer seg av dette. Mest kjent er LDAP-protokollen.



Figur 2.4: Tre tilnærminger for ordning av informasjon

Trær er også grunnleggende innenfor algoritmeteori. Når det gjelder virksomhetssøk kan taksonomisk inndeling foregå manuelt eller automatisk.

Den forhåndsdefinerte kategoriseringen tildeles manuelt og kalles ofte statisk taksonomi. Dette er attraktivt for virksomhetssøk i mange sammenhenger. Spesielt dersom oversikten over det indekserte materialet er god, eller hvis dataene er tilstrekkelig strukturerte, kan forhåndsdefinerte kategorier defineres. Et godt eksempel er nettbutikker som har store mengder produkter, og som ønsker at disse automatisk skal klassifiseres etter kategorier, underkategorier og lignende.

Denne kategoriseringen skjer som en del av indekseringen. Innkommende data analyseres i forhold til gitte kriterier – for eksempel ved hjelp av regulære uttrykk – og det opprettes tilleggsfelter i indekset som plasserer dokumentet taksonomisk. Den kan selvfølgelig også foregå manuelt ved at hvert enkelt dokument undersøkes av en domeneekspert.

Automatisk taksonomi beregnes ved spørretidspunktet, altså er den dynamisk. Denne formen for kategorisering passer bedre hvis dataene er mindre oversiktlige og homogene. Algoritmer forsøker å analysere dataene i indekset og skape taksonomi i sanntid. Dette er ikke alltid like nyttig og er mindre brukt.

Felles for all type taksonomi er at de forenkler kategoribasert navigering i grensesnittet. Derfor kalles det som returneres ofte for *navigatorer*. For strukturerte applikasjoner, gjerne en nettbutikk, er det praktisk for brukeren å kunne navigere i et systematisk tre. Et annet uttrykk for dette er aspektorientert leting.[7]

Tilordning av taksonomi kan være en dyr operasjon. I tillegg krever det menneskelige ressurser å vedlikeholde gode kategorier. Siden den forhåndsdefinerte typen foregår ved indekseringen, vil den ikke forårsake lavere ytelse for spørringer.

[7]På engelsk omtales dette gjerne som “faceted browsing”, “navigators” eller “drill-down”. Les mer om dette i et senere avsnitt.

Relasjoner (grafer)

Et tre er egentlig en spesiell graf. Brytes reglene om hierarki kan kanter tegnes fritt mellom alle noder. En slik kobling minner om klassediagrammer i UML eller ER-diagrammer i databasemodellering. Denne typen ordning er mindre brukt innenfor virksomhetssøk, ettersom sammenhengen mellom relasjoner er viktigere enn fritekstsøk.

Det er imidlertid ikke noe prinsipielt i veien for å bygge slik funksjonalitet inn i IR-biblioteker eller IR-rammeverk. Faktisk kan et skjema som defineres for indekseringsprofiler minne om databaseskjemaer (tabeller og relasjoner). Forskjellen går i stor grad ut på at koblingen er matematisk absolutt i en relasjonsdatabase, mens IR-systemene legger vekt på det enkelte dokumentets innhold og felter.

Assosiativ ordning (dynamiske grafer)

I takt med fremveksten av deltakersamfunnet og "Web 2.0" har såkalt sosial bokmerking blitt et fenomen. Dette går ut på at websider merkes ved hjelp av stikkord.[8] Nettstedet del.icio.us [32] er en kjent pionér. Stikkordene velges fritt, uten hensyn til taksonomi eller relasjoner. Hensikten er å tilby større dynamikk. Virkeligheten endrer seg ofte raskere enn den kan systematiseres. Stikkordsmerking gir en fleksibel og assosiativ inndeling av informasjonen, og når datamaterialet er stort nok kan det automatisk tegnes grafer og lignende over informasjonen.

Prinsippet har fellestrekk med tradisjonelle relasjoner, men gir langt større frihet. Koblingene er løse, og det benyttes ikke et skjema eller noen bestemt oppskrift. Ulempen er at mindre datamengder umuliggjør ønskede statistiske analyser, samtidig som skrivefeil slår inn.

Selvfølgelig kan det generaliseres til å brukes på dokumenter, ikke bare websider. Det har allerede vært lenge i bruk innen arkivsystemer for spesifikke områder, som for eksempel bildekatalogisering. Virksomhetssøk kan dra svært stor nytte av slik dynamisk klassifisering. Tilordningen kan enten skje manuelt ved dokumentanalyse før indeksering, eller ved spørretid.

2.5.5 Ekstrahering og utvinning

Information Extraction (IE) utgjør et eget fagfelt innenfor informasjonsgjenfinning. Målet er å trekke ut strukturert informasjon til bruk i gitte sammenhenger. Dette kan for eksempel omformes til XML RDF eller andre semantiske strukturer. Et eksempel er at naturlig språk omformes til

[8] På engelsk kalles dette "tagging" eller "labelling". Se også figur 3.3 på side 65.

meningsfulle strukturer:

$kilde$: "... at X kjøper opp Y nå på mandag, ..."

⇓

$resultat$: `Oppkjøp(X,Y,2007-01-29)`

Dette gjøres ved hjelp av regulære uttrykk og avanserte algoritmer. Vanlige bruksområder er gjenkjenning av navngitte entiteter (NE), utvinning av terminologi, samt oppdagelse av referanser. Eksempel:

$kilde$: "... dere sa at dere skulle reise til Kina."

⇓

$resultat$: `Utsagn(x,Reise(x,Kina)), x = dere`

Poenget er altså forstå at "dere" er det samme som "dere", og å utnytte denne semantikken for senere bruk.

Neste avsnitt ser nærmere på NE-ekstrahering, som brukes mye innenfor virksomhetssøk. Her er ikke hensikten hovedsakelig å forstå semantikk, men å gjenkjenne datatyper.

NE-ekstrahering

Et beslektet emne for automatisk ordning er såkalt NE-ekstrahering. NE står for *navngitt entitet*. Merking og kommentering av indekserte dokumenter gjennomføres ved at tekst analyseres. En algoritme gjenkjenner navngitte termer (søkeord), for eksempel substantiver, numeriske uttrykk, stedsnavn eller dato.

Eksempel:

> "Onsdag den 13. november 1991 kjøpte Peder Aas to tomter i Stavanger til en verdi av to millioner kroner."

NE-ekstrahering vil her måtte gjenkjenne termer som ukedag, dato, måned, årstall, navn, kjøpsobjekt, by, beløp og valuta. Videre må dette oversettes til fornuftige datatyper, samt plasseres ontologisk. Slike analyseverktøy har kommet relativt langt i sin utvikling, men det er fremdeles utfordrende å vedlikeholde relevant taksonomi. I eksemplet over er dette nødvendig blant annet for å kunne fastslå at Stavanger er en del av Rogaland, som igjen er en del av Norge – i hvert fall for å muliggjøre kraftfulle søk.

Sureka har skrevet en kort og god artikkel om emnet [106].

Kapittel 3

Utfordringer i virksomhetssøk

> I alt kaos finnes kosmos,
> i all uorden en hemmelig orden.
>
> — *Carl Jung*

Tidligere i oppgaven kom det hint om at søk i virksomheter kan representere en bredere utfordring enn søk på internett. Begge deler utgjør avanserte fagfelt, men virksomhetssøk skiller seg først og fremst ut ved at det implementeres hos hver virksomhet. Derfor støter det potensielt borti flere relaterte problemstillinger, og i dette kapitlet blir noen av de vanligste utdypet.

Den største utfordringen er allerede nevnt, nemlig at implementasjonen vanligvis må tilpasses grundig i hvert enkelt tilfelle. Denne tilpassingen går ofte mot spesielle kildesystemer og egendefinerte filformater. For at en plattform for virksomhetssøk skal være brukervennlig og fleksibel, er det altså en fordel med gode verktøy for administrering, samt god modularisering med smarte API-er. Videre blir både grensesnitt og sikkerhet tatt opp, siden dette ofte kan skille seg fra andre typer søk.

En rekke problemer står også uløste. Den siste seksjonen i kapitlet går derfor igjennom noen punkter som kan være interessante for videre arbeid.

3.1 Tilpassing og integrering

Noe av essensen i produkter for virksomhetssøk er evnen til fleksibel tilpassing og integrering. Google, Kvasir og andre websøkemotorer tilgjengelig på internett tilbyr egentlig bare én og samme søkeinstallasjon til alle sine brukere. I motsetning må virksomhetssøk tilpasses for hver installasjon. To eksempler er internsøk i bedrifter og innholdsleveranser

gjennom internettportaler, og dette stiller igjen krav det aktuelle grunnsystemet.

I dette avsnittet presenteres noen enkle krav. Modularisering er en fordel av mange grunner, men ikke minst for å øke fleksibiliteten. Gode API-er for tilpassing mot eksisterende systemer er dessuten en selvfølge. Videre bør søkesystemet fungere på en mengde ulike plattformer. Til sist er det et pluss om brukervennligheten er såpass god at nytteverdien "ut av boksen" er forsvarlig høy.

3.1.1 Eksisterende systemer

Som allerede nevnt skiller virksomhetssøk seg fra vanlige nettsøk, blant annet fordi flere ulike kilder skal leses. Sannheten er at selv for de tilsynelatende trivielle kildene, som web, er utfordringene større av flere årsaker. Videre er tilpassing til avanserte databasesystemer utfordrende, og i tillegg finnes det en stor klasse av øvrige bindeledd. Hvordan disse kildene forbindes til selve søkemotoren forklares på side 74 (her er fokuset på mer generelle utfordringer med kildesystemer).

Vanlige søkemotorer på nettet har crawlere som fanger nettsider og annet innhold over HTTP-protokollen. Dette kan gjøres i ES-systemer også, men det kan være vanskeligere. For eksempel vil traversering av intranett måtte integreres mot sikkerhet og tilgang. Dessuten kan det samme innholdet ofte fanges både over HTTP eller filsystemer. Hvordan skal dette eksponeres? Det må sannsynligvis stilles inn i de enkelte tilfellene. I tillegg lenker ikke intranettsider til hverandre så ofte, og de er færre i tallet enn offentlige nettsider, noe som gir utfordringer for lenkeanalyse og dermed fastsettelse av relevans. Til sist er innhold som regel langt mindre kaotisk på intranett enn på internett. Dette gir en anledning til bedre tilpassing, men da må følgelig tilpassingen faktisk også gjøres.

Det andre poenget går på integrering med databasesystemer. I sin enkleste form vil ES-systemet betrakte hvert tuppel i hver tabell som et dokument og tilordne disse en unik adresse. Tenk for eksempel på en typisk tabell kalt `person`. Et utdrag fra tabellen kan se slik ut:

```
id        etternavn     fornavn       tlf
...       ...           ...           ...
31336     Nordmann      Kari          27182818
31337     Nordmann      Ola           31415927
```

Ola Nordmann kan i dette tilfellet bli indeksert som et dokument med en SQL-spørring som adresse:

```
SELECT * FROM person WHERE id = 31337;
```

Men ofte består databasesystemer av svært mange tabeller, avledede tabeller (views), beskrankninger, fremmednøkler, funksjoner og andre kompliserende elementer. Derfor må som regel databasesystemer som skal gjøres søkbare på en hensiktsmessig måte gjennomgå en lang prosess for tilpassing.

Et siste og viktig punkt for integrering går mot andre typer systemer. Her stiller de kommersielle aktørene som regel sterkt. Virksomheter ønsker kanskje å indeksere innhold gjennom webtjenester og tjenesteorientert arkitektur, datavarehus, emnekart, semantiske nettverk, målinger fra apparater, egne applikasjoner eller annet.

Ta en titt tilbake på figur 2.2 på side 24. Et hav av ulike kildesystemer, gjerne delt inn i soner – lokalt, sentralt og eksternt – danner et komplisert bilde av situasjonen. Selvfølgelig kan virksomhetssøk også være enkelt. For eksempel kan en bedrift ønske seg en løsning som indekserer alle filsystemene på lokale PC-er. Den største utfordringen ligger derfor i utviklerne av søkeplattformen, som må ta høyde for alle de potensielle situasjonene som her er blitt nevnt.

3.1.2 Modularisering

De fleste programsystemer kan og bør modulariseres. Det er utenfor oppgavens omfang å beskrive tiltak som gjøres under selve designfasen og programmeringen.[1] Allikevel tas modularisering opp spesielt. Det finnes gode grunner til dette:

- Kompleksiteten er høy
- Abstraksjon i overordnet design er ønskelig
- Gjenbruk er ønskelig

I tillegg er det rimelig å anta at alle deler i et system for virksomhetssøk neppe kan utvikles fra bunnen.

Hva er kjennetegnene på høy kompleksitet? I boka *The Art of Systems Architecting* forklarer forfatterne allerede i første kapittel at et system ikke behøver å være stort eller dyrt for å være komplekst [74]. Når antallet deler og forbindelsene mellom dem øker, stiger kompleksiteten. Hvor sofistikerte forbindelsene er, gir også utslag. Som det fremgår av dette kapitlet er søkesystemer for virksomheter derfor komplekse. Modularisering vil bidra til å kontrollere eller redusere kompleksiteten.

Videre er det ikke forventet eller i det hele tatt nødvendig at alle utviklere skal vite alt om et søkesystem. Modularisering fører til en naturlig oppsplitting slik at programmerere kan konsentrere seg om de bitene

[1] Grensesnitt og objektorientering bidrar til å modularisere og abstrahere under selve programmeringen.

som betyr noe. Abstraksjonen dette gir på høyere nivå gjør at arkitekter og systemintegratører kan distansere seg fra de tekniske detaljene, for eksempel når det gjelder søkealgoritmer eller språkdeteksjon.

I et sterkt modulært system kan enkeltkomponenter i større grad byttes ut. Dermed gis det rom for gjenbruk av komponenter. Det er nesten utenkelig at alle komponentene i et komplekst system skal utvikles fra bunnen av. Hvis arkitekturen er godt gjennomtenkt, vil det være en smal sak å benytte seg av godt gjennomtestede delprodukter. Dette er fornuftig både i en proprietær kontekst og med hensyn på fri programvare.

Modularisering er altså svært fornuftig, men hvordan skal det gjennomføres? Moduler på flere nivåer og metanivåer er mulig. De neste avsnittene ser litt nærmere på spesielle forhold innenfor virksomhetssøk som har betydning for en modulbasert arkitektur.

Intermodulær kommunikasjon

Et søkesystem består av flere deler, for eksempel bindeledd mot datakilder, søkebiblioteker og språkbiblioteker. Hvordan skal disse bitene kommunisere? Det er klart at interne API-er må brukes, eller om nødvendig lages på ny i såkalte innpakninger. Det er en stor fordel om de fleste modulene er skrevet i samme språk, for eksempel Java.

Et alternativ er å benytte protokoller på litt høyere nivå. I en distribuert arkitektur er det vanlig å bruke XML. Ikke minst har dette tatt av med hensyn på integrering av ulike systemer som tradisjonelt passer litt dårligere sammen. Imidlertid er det nyttigst å finne “minste felles multiplum”, fordi ytelse er viktig. Med det menes en protokoll som passer til alle modulene på et direkte nivå, og med så lite oversetting og behandling som mulig.

Det er en god idé å bygge ett eller flere rammeverk. Ferdige rammeverk kan også tilpasses og benyttes. Hensikten er først og fremst å sørge for enda bedre abstraksjon og muligheter for modularisering. I tillegg nyter noen rammeverk godt av gjenkjennelse hos utviklere, samt at det legges føringer for utviklingsmønstre (patterns).

I tillegg til ytelse spiller sikkerhet en viktig rolle. Internt på én datamaskin er det ikke ofte at sikre forbindelser har spesielt stor hensikt. Straks en søkeløsning distribueres over flere servere – gjerne globalt – oppstår det likevel et behov for å sikre kommunikasjon. Innbrudd utenfra kan selvfølgelig forhindres i andre lag, mest naturlig på nettverks- og OS-nivå. Dette vil dog ikke forhindre at sensitiv informasjon kan lekke internt. For store søkeinstallasjoner kan derfor *sikker intern kommunikasjon* være nødvendig.

Ekstern modultilgang og API-er

Arkitekturen må skille klart mellom intern kommunikasjon og ekstern tilgang. For størst mulig fleksibilitet er det selvsagt ønskelig at store deler av systemet kan konfigureres og kommuniseres med eksternt. Allikevel bør ikke hele søkesystemet automatisk eksponeres, av hensyn til sikkerhet, brukervennlighet og ytelse. Hensikten med tilgangspunkter for programmerere er følgelig at søkeproduktet i størst mulig grad kan tilpasses andre systemer, og det er ikke uvanlig å tilby støtte for flere protokoller eller språk samtidig.

Virksomheter som tilpasser søkeplattformen ønsker kanskje allikevel å få tilgang til de ulike modulene gjennom API-er. Fordelen med dette er at deler av systemet kan brukes, og at komponenter enklere kan byttes ut. Her kan det tenkes at det kommersielle aktørene tjener mer på å holde kortene tettere til brystet, slik at kunder kan holdes på én og samme plattform ("vendor lock-in"). Sagt på en annen måte vil dette kunne være en gyllen mulighet for en ES-plattform basert på fri programvare å skille seg ut.

En slik modularisering med elegant eksponering av delsystemer, kan for eksempel gjøres gjennom et rammeverk. Da er det en klar fordel hvis hele søkeplattformen implementeres i samme programmeringsspråk. I tillegg til at kompleksiteten i systemet kan minimaliseres, vil virksomheter som ønsker å se subsystemer få det mer levelig.

Fysiske noder

I kapitlet om informasjonsgjenfinning (fra side 69) forklares det hvordan ulike oppgaver kjøres på forskjellige fysiske noder. For virksomhetssøk er det fornuftig med enda flere maskiner. Først og fremst vil dette være webtjenere og applikasjonstjenere spesielt for det tilpassede søkeproduktet.

Modularisering i design er altså nærmest en forutsetning for effektiv distribuering og skalérbarhet.

3.1.3 Verktøy for styring og brukervennlighet

Tilpassing og integrering foretas i stor grad av eksperter og programmerere, og det er derfor hensiktsmessig at mye programmeres og konfigureres i spesielle formater og API-er. Samtidig bør et produkt være brukervennlig for litt mindre teknisk kyndig personale. Ofte kan terskelen for å komme igang med et system være for stor selv for programmerere, og jo tidligere et søkesystem kan demonstrere sine styrker – jo lettere det er å komme igang med det – dess større er kanskje sjansen for at det blir valgt.

Verktøy for styring, såkalte admin-verktøy, er svært viktig her. En stor og kompleks søkeplattform kan lett bli uoversiktlig og vanskelig å håndtere. Flere gode søkeplattformer tilbyr grensesnitt via web. Et slikt grensesnitt kan tilby enkel konfigurering, oversikt over dataflyt, samt overvåking av prosesser og moduler.

Automatisert installasjon av søkeplattformen vil også bedre brukervennligheten. Eksperter som tilpasser et stykke programvare vil kanskje installere alt manuelt uansett, men dette er mye lettere å gjøre dersom kjennskapen til programvaren er større. Og kjennskapen oppnås best dersom man først har prøvd og fått installert programvaren...

Mer generelt kan verktøy for styring og installasjon inngå i et konsept kalt *innpakking*. Andre elementer som eksempelvis kan ligge her er hjelpefunksjoner, brukerfora eller konsulenttjenester. En programvareplattform vil fungere bedre hvis den har et stort apparat rundt seg.

Innpakking og brukervennlighet er et av de områdene hvor de kommersielle aktørene skiller seg mest fra fri programvare, iallfall innenfor virksomhetssøk, men kanskje også innenfor programvare som helhet. Et eksempel er Fast ESP [38] som kommer med brukerstøtte, installasjon, styring via web, visualisering av dataflyt og indeksering, grafisk overvåking av noder og lignende. Det åpne alternativet Apache Solr [11] tilbyr riktignok et verktøy for styring via web (se figur 5.3 på side 120), men dette tåler på ingen måte sammenligning med Fasts variant.

3.1.4 Dokumentformater

Virksomheter søker som regel å løse bestemte problemer, og da er det ofte gitt hvilke systemer og formater det skal kommuniseres med. I noen tilfeller kan virksomheten selv skrive programmer som omformer dataene til et egnet format, for eksempel XML-RDF. Men svært ofte ønsker også virksomheter en mer eller mindre ferdig løsning på problemet sitt. Da vil det være en fordel for en ES-plattform om den støtter mange filformater som standard. Det er ikke tilfeldig at de kommersielle aktørene, som FAST, har støtte for flere hundre filformater.

I en god del tilfeller vil allikevel ikke filformatet som ønskes fortolket støttes fra før, for eksempel hvis virksomheten ønsker å indeksere dokumenter laget i et gammelt og spesielt system som de ikke ønsker å avvikle. Av denne grunnen er det fornuftig at ES-plattformen lett kan bygges ut med støtte for nye filformater. En mulighet er å tilby et eget grensesnitt (API) for å programmere dette etter behov.

Et nærliggende problem handler om dokumentgranularitet. Dokumenter trenger ikke nødvendigvis være skrevet av mennesker, men de kan være målinger, databasetupler, tall, og så videre. Det kan være hensiktsmessig å indeksere mange bittesmå dokumenter, eller å slå sammen

og gruppere data til større dokumenter, alt etter behov. Dette taler også for at et API, som nevnt ovenfor, er en fordel.

3.2 Presentasjon og grensesnitt

Søk gir seg ikke alltid ut for å være søk. En konsekvens av dette er at mange brukere søker langt oftere enn de tror. Dette forutsetter en definisjon av ordet "søk" hvor det kjøres en spørring inn mot et IR-system. Dette har sin årsak i at brukergrensesnittene kan være radikalt forskjellige. Kanskje minner de overhodet ikke om søk.

	Eksplisitt	Implisitt
Visualisert	Tradisjonelle søk	Navigérbare søk
Skjult	-	Integrerte søk

Tabell 3.1: Ulike typer grensesnitt for søk

Tabell 3.1 viser et forsøk på å identifisere tre hovedtyper av brukergrensesnitt for søk. Felles for dem alle er at det som sendes inn til IR-systemet er formulert i et bestemt spørrespråk. Under kulissene foregår stort sett det samme. Spørringer forklares nærmere i seksjon 4.4 på side 79.

De neste avsnittene forklarer og gir eksempler på de ulike brukergrensesnittene.

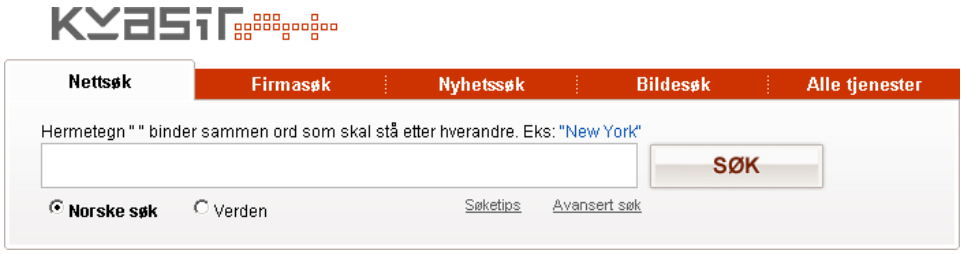
3.2.1 Tradisjonelle grensesnitt

Figur 3.1(a) på neste side viser et typisk tradisjonelt søkegrensesnitt fra den norske søkemotoren Kvasir [66]. Brukeren ser en boks hvor han skal taste inn ord eller fraser. Ofte kan disse termene supplementeres med metainformasjon i form av kommandoer, operatorer og lignende (mer om dette siden). Her er et typisk Google-søk:

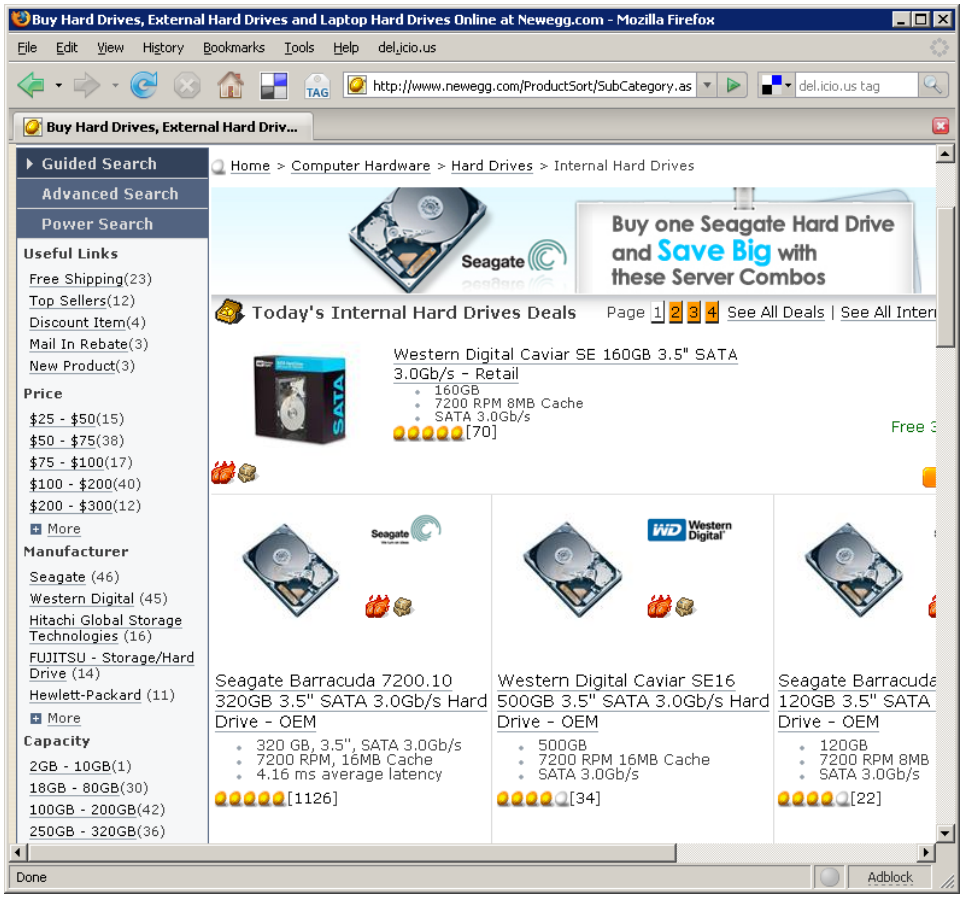
```
+windows +vista site:.no
```

Om søkemotoren skal returnere informasjon om vinduer, utsikt eller et operativsystemer er et IR-relatert spørsmål. I denne sammenhengen ønsker brukeren dokumenter – eller hovedsakelig websider, for Googles vedkommende – som inneholder både ordet "windows" og ordet "vista", og som finnes på nettsider med norske domener (`*.no`).

Denne metoden er visualisert fordi den har et synlig søkegrensesnitt, og eksplisitt fordi det er tydelig at det foretas et reelt søk.



(a) Fritekstsøk



(b) Navigatorer

Figur 3.1: Tradisjonelt grensesnitt for søk (øverst) og aspektorientert leting (nederst). Disse ulike konseptene kan selvfølgelig også brukes samtidig.

3.2.2 Navigérbare grensesnitt

Figur 3.1(b) på forrige side viser et alternativt søkegrensesnitt fra den amerikanske nettbutikken Newegg.com [80]. I venstre side av skjermbildet sees typiske navigatorer som innsnevrer utvalget av produkter. Hver gang brukeren klikker på en av disse lenkene, kjører det underliggende systemet en spørring med en gitt beskrankning. Den faktiske spørringen skjules for brukeren og kan være mer presis enn ved direkte fritekstsøk.

Også dette grensesnittet er visualisert, men i motsetning til fritekstsøk representerer ikke navigatorer søk eksplisitt. Det er ikke tydelig for brukeren at det foretas et søk i et IR-system. For den saks skyld kan det også være at det underliggende systemet er en vanlig, gammeldags relasjonsdatabase.

Eksemplet med Newegg.com er valgt fordi det ligner svært mye på et IR-system med navigatorer, men det er ikke blitt undersøkt om det faktisk er et søkesystem som ligger under panseret. Dette er tilsiktet, og viser egentlig bare at grensesnittet er visualisert, mens søket – eller hva et nå er – er implisitt.

3.2.3 Integrerte grensesnitt

Ofte bygges informasjonsgjenfinning inn i systemer eller programmer. Søking kan da foregå helt uten brukerens direkte interaksjon eller viten. Spørringene som genereres behøver heller ikke være et resultat av at en bruker gjør noe.

Et nærliggende eksempel er utviklingsplattformen Eclipse [35]. Her benyttes Apache Lucene [9] til å indeksere hjelpefunksjonen. Riktignok kan det foretas søk som initieres av brukeren, så grensesnittet er ikke helt skjult.

Sanntidssystemer, våpensystemer, produksjonssystemer og måleprogrammer er kanskje bedre eksempler. Det kan være nyttig å finne frem i store datamengder svært raskt, og i mange tilfeller tilbyr ikke tradisjonelle relasjonsdatabaser den nødvendige hastigheten.

Filtreringssystemer (se side 85) benyttes ofte innen sanntidssystemer som for eksempel gir børsinformasjon. Et spesielt kjennetegn er at slike systemer ofte bruker filtrering i stedet for vanlige spørringer. Intuitivt kan det regnes for å være det “motsatte” av vanlige søk. Prinsippet tilsier at den samme spørringen kjøres hele tiden, og at det heller er datamengden som endrer seg.

Integrerte søkeoperasjoner foregår dermed som regel sømløst. Poenget er at brukeren ikke aktivt søker (eller begrenser søket), men bruker et system eller program uten egentlig å vite at IR blir brukt som intern datastruktur.

3.3 Sikkerhet

Det norske ordet sikkerhet kan deles inn i sikring (security) og forsikring (safety). Her fokuseres det på den første typen sikkerhet. Jeg tør påstå at sikkerhet er spesielt viktig innenfor virksomhetssøk. Det kan sidestilles med kravene til sikkerhet innenfor relasjonsdatabaser. I tillegg brukes kanskje søketeknologi i en litt annen skala og på en litt annen måte enn databaser, og dette gir andre utfordringer.

Hvorfor er sikkerhet så viktig? I svært mange anvendelser av søketeknologi oppbevares det mye informasjon som senere kan misbrukes. Dette anerkjennes av fagfolk i søkemiljøet, og professor Dag Johansen har uttalt:

> "Selv om Google har 'Do no evil' som slagord, og mener det, er det all grunn til å være skeptisk. Ingen vet hva som kommer til å skje med denne informasjonen i fremtiden under helt andre politiske forhold. Bare se på hvordan Google har tilpasset søketjenesten så den tilfredsstiller kinesiske politikere." [113]

Problematikken er kanskje enda mer uttalt i store, sosiale nettverk. I enkelte tilfeller registrerer brukere store mengder sensitiv informasjon som seksuell legning, religion, forhold til venner og lignende. Det sier seg selv at slik informasjon ikke må komme på avveie.

De neste delavsnittene ser litt nærmere på sikkerhet og virksomhetssøk. Det første handler om hvordan teknologien må tilpasses all mulig annen teknologi for å fungere optimalt. Etterpå tas noen av utfordringene under indeksering og spørring frem. Til sist diskutereres transaksjoner, noe som stadig blir mer etterspurt innenfor virksomhetssøk (ettersom søketeknologi ofte skal erstatte eller supplere relasjonsdatabaser).

3.3.1 Integrering og kompatibilitet

Kravene til sikkerhet medfører ofte økt kompleksitet, spesielt som en følge av integrering og kompatibilitet med eksisterende systemer. Dette har først og fremst tre perspektiver: Interaksjon med brukersystemer, forvaltning av rettigheter og tilgangsrettigheter i selve søkesystemet.

Virksomhetssøk er spesielt i det henseende at det tilpasses eksterne systemer i stor grad, og ofte er det ønskelig å overføre rettigheter fra brukersystemer. Disse kan være:

- Katalogsystemer og protokoller som brukes til dette, for eksempel NIS, Active Directory (AD) og LDAP
- Databasesystemer (Oracle med flere)

- Store systemleverandører (for eksempel SAP)

I tillegg kan det være at tilfeldige eller spesielle brukersystemer finnes. Universitetet i Oslo bruker eksempelvis egenutviklet programvare for håndtering av brukere.

Alle disse brukerne skal så integreres med korrekt forvaltning av rettigheter på dokumenter. Dette inkluderer filrettigheter på Unix/Linux og Windows, databaserettigheter til spesifikke tabeller og tupler og rettigheter på intranett. En god rettighetsforvaltning kan dessuten støtte manuell manipulering av rettigheter på et detaljert nivå. I slike tilfeller reiser det seg også spørsmål rundt om dette skal kunne gjøres med eller uten synkronisering mellom kildesystem og indeks, og om det skal inngå i en transaksjon.

Tilgangsrettigheter i selve søkesystemet skal så gjenspeile alt dette. Dette får konsekvenser for både opprettelse, endring, sletting og søking i dokumenter. Det er naturligvis også en forutsetning at eventuelle brukere for et styringsverktøy på web ikke får større rettigheter enn de skal ha. En forenkling er selvfølgelig mulig dersom virksomheten kan tillate at IT-personell skal kunne se all informasjon, men dette er i mange tilfeller uholdbart.

Sikkerhet representerer derfor en signifikant utfordring. Alle ledd må ta hensyn til sikkerhetsmodellen. I praksis betyr dette at både indeksering/innmating, selve indekset, samt spørre- og resultatprosessorene bør støtte en svært fin granularitet av sikkerhet.

3.3.2 Transaksjoner

Noen ganger er det hensiktsmessig at handlinger utføres i transaksjoner. Dette betyr i all hovedsak at en rekke handlinger grupperes sammen. Transaksjoner er vanlige innenfor relasjonsdatabaser, og mange applikasjoner avhenger av dem. Transaksjonsegenskapene i et databasesystem omtales gjerne som ACID-egenskaper:[2]

Atomitet garanterer at alle handlingene i transaksjonen blir gjennomført, eller at ingen av dem blir det. Det er vanlig å sammenligne dette med uttak i en minibank; enten får man pengene og beløpet trekkes fra saldoen, eller så skjer ingen av delene.

Konsistens betyr at databasen (eller systemet) er i en lovlig tilstand ved begynnelsen og slutten av en transaksjon, gitt av beskrankningene. Et eksempel er forbud mot negativ saldo; alle forsøk på transaksjoner som vil medføre brudd på regelen stanses.

[2] På engelsk står ACID for Atomicity, Consistency, Isolation og Durability.

Isolasjon betyr at alle handlinger i transaksjonen blir usynlige. Hvis man eksempelvis overfører penger mellom to konti, og det tar lang tid, vil man allikevel aldri kunne se at pengene har gått ut fra den ene kontoen, men ikke kommet inn på den andre kontoen.

Bestandighet medfører at transaksjonen er garantert enten gjennomført eller ikke, uansett om systemet feiler. For eksempel vil man allikevel ikke bli trukket fra konto hvis en minibank skulle miste strømmen under en transaksjon.

For å holde orden på transaksjoner bruker databasesystemer logger hvor transaksjoner skrives inn. Dette kan gjøres på flere måter. Mest vanlig er *undo*-logg og *redo*-logg, som forteller henholdsvis hvilke transaksjoner som kan trekkes tilbake eller fullføres ved systemgjenoppretting. Isolasjon i transaksjoner reiser også en mengde spørsmål omkring samtidighet. Hvis flere transaksjoner utføres samtidig, kan den ene risikere å lese data som er endret av den andre, før den første er fullført – ofte kalt *dirty read*. Databasesystemer har ulike tilnærminger til disse problemene, men de vil ikke bli diskutert nærmere her. Se gjerne Ullman et al. [42] for mer om dette.

Siden IR-systemer ofte skal løse de samme problemene som relasjonsdatadatabaser gjør, er det i mange tilfeller ønskelig med ACID-overholdelse i søkemotorer. Behovet for ACID er altså mye høyere innenfor virksomhetssøk enn websøk, ikke minst fordi databaser og informasjonsgjenfinning kan integreres [31]. Denne oppgaven kommer ikke til å gå nærmere inn på den tekniske gjennomføringen, annet enn å nevne at ACID kan overholdes mer eller mindre på flere nivåer.

3.3.3 Datasikkerhet

Den andre typen sikkerhet handler om forsikring, noe som innebærer forsvarlig forvaltning av verdifulle data slik at risiko minimeres. Mye av dette er et driftsmessig problem og derfor ikke relevant for oppgaven, bortsett fra det arkitekturmessige perspektivet. Først og fremst går dette ut på distribuering og ytelse. Enhver større søkeløsning må støtte distribuering over flere noder. Dette er et IR-relatert problem som berøres mer i seksjon 4.6 på side 89.

3.4 Uløste problemer

Hvor går veien videre for virksomhetssøk? Hvilke utfordringer står åpne i dag?

Søk og informasjonsgjenfinning er et relativt modent fagfelt i seg selv, og mange nyttige algoritmer, prinsipper og teknologier er utviklet.

Enkelte bruksområder har dessuten kommet nokså langt, som for eksempel søk på internett. Virksomhetssøk tar i bruk mye av kunnskapen fra informasjonsgjenfinning, men er i seg selv et mye yngre felt. En rekke sentrale problemer står enten uløst, eller har suboptimale løsninger i dag. Hva kan det komme av?

Det mest nærliggende er kanskje å innse at virksomhetssøk er et ungt fagfelt. Med det følger det at mange av problemene også er ganske nye. Virksomhetssøk handler om å løse problemer i organisasjoner. Behovene for spesielle løsninger som fagfeltet kan tilby oppstod i første rekke i de aller største organisasjonene. Etterhvert som datamengdene vokser oppstår også mange av de samme problemene i mindre virksomheter. Derfor er det nok mange som ikke har innsett mange av de fremtidige behovene ennå. Dette henger også sammen med at virksomhetssøk har færre fellesnevnere enn for eksempel søk på internett, som de fleste vanlige brukere forbinder med ordet søk. Betraktes internett som en homogen virksomhet eller masse, er det lettere å skjønne at bedrifter som Google og Yahoo har brukt mange år på å utvikle og optimalisere sine løsninger for dette segmentet. Virksomheter, derimot, har vidt forskjellige behov. Dermed må søkeløsningene tilpasses i hvert enkelt tilfelle. Denne mangelen på fellesnevnere innenfor virksomhetssøk er kanskje en medvirkende årsak til mengden av uløste problemer innenfor fagfeltet.

Kan det tenkes at mange utviklere er relativt lite kjent med informasjonsgjenfinning? Her er det vanskelig å konkludere uten å gjøre nærmere undersøkelser. En rask titt på for eksempel utdanningstilbudet ved Universitetet i Oslo avslører imidlertid at søk ikke er et sentralt emne for studentene ved denne instutisjonen, på tross av at fagfeltet er modent, utviklet og tilsynelatende nokså grunnleggende. Det gjenstår å finne ut om dette har konsensus blant utviklere på mer generell basis. Dersom dette virkelig er tilfellet – altså at søk er et fagfelt som de fleste utviklere kan lite om – er det i så fall ikke rart at virksomhetssøk har mange uløste utfordringer. Utviklerne kan være oppmerksomme på forretningsproblemene som finnes, men dersom de ikke kjenner til virksomhetssøk tyr de til andre metoder. En undersøkelse gjennomført av Freund og Toms i fjor [40] kan synes å tale mot denne hypotesen. Et viktig funn her var at utviklere i gjennomsnitt bruker mer sammensatte og kompliserte spørringer ved bruk av virksomhetssøk. Allikevel er ikke dette entydig med at de kjenner godt til IR-fagfeltet. Kanskje er de rett og slett gode databrukere som er vant med søkemotorer på internett, og som overfører vanene sine fra internettdomenet til virksomhetsdomenet.

En siste mulig årsak kan rett og slett være kompleksitet. Få kan vel fornekte at virksomhetssøk er et stort og komplekst område. For det første bygger det på et stort fagfelt, og all kunnskapen herfra må tas med. I tillegg kommer andre fagfelt inn i bildet som dette skal integreres

mot, eksempelvis relasjonsdatabaser, datavarehus, transaksjonshåndtering eller applikasjonsutvikling. Dette varierer selvfølgelig for de enkelte tilfellene. Dessuten skal virksomhetssøk legge seg tett opp mot forretningsproblematikk. Dette fordrer utviklere med god kjennskap til det aktuelle domenet, direkte organisasjonstilhørighet eller kompetanse innenfor forretningsanalyse og -utvikling; eller aller helst alt på én gang.

Disse hypotesene er kan i beste fall karakteriseres som kvalifisert synsing, men de kan og bør undersøkes nærmere. Det ser ut til å være en mangel på informasjon som kan støtte opp under årsakene til at fagfeltet ikke har utviklet seg mer enn det har. De neste avsnittene tar for seg noen av de uløste utfordringene innenfor virksomhetssøk. Mange av disse er hentet fra en undersøkelse gjort av David Hawking [51]. Sannsynligheten er neppe liten for at en del av disse problemene vil få et større søkelys rettet mot seg i årene som kommer.

3.4.1 Kunnskapsformidling

▷ Er kjennskapen til ES for dårlig?

Denne oppgaven har allerede satt søkelyset på at virksomhetssøk og informasjonsgjenfinning er relativt ukjent for mange. Dette gjenspeiles blant annet i at det ikke finnes noen moden ES-plattform som kan konkurrere med kommersielle løsninger, iallfall ikke i samme marked. Problemet har kanskje ikke direkte sammenheng med tekniske utfordringer innenfor fagfeltet. Det nevnes allikevel, siden mangelen på allmenn fokusering på ES muligvis kan medføre en langsommere utvikling.

Bedre kunnskapsformidling og kjennskap omkring feltet vil nok kunne være sunt. Om dette vil tilta etterhvert som behov for løsninger øker, er ikke godt å si. Det er også uvisst om økt fokus vil bidra til at flere av de åpne problemene, inkludert denne begrensede kunnskapsformidlingen, vil bli løst i et raskere tempo.

3.4.2 Utvelgelse av gode testdata

▷ Hvordan tar man et tverrsnitt av en virksomhet?

Vanlige søk på internett leter som nevnt gjennom et relativt homogent domene. Det aller meste av innhold er HTML-filer, selv om det finnes en del PDF-dokumenter, Word-dokumenter, multimediefiler og annet. Under utviklingen av nettsøk tas det utgangspunkt i at internett (HTTP i dette tilfellet) er åpent for alle, og at brukere i utgangspunkt har tilgang til det meste.[3] Slike internettsøk kan altså betraktes som én enkelt ins-

[3]Det tas selvfølgelig hensyn til kontekst i vanlige, avanserte nettsøk, men det ødelegger ikke det faktum at vanlige nettsider generelt ligner svært mye på hverandre og at behovene er homogene.

tallasjon eller implementering av informasjonsgjenfinning.

For virksomhetssøk er det til gjengjeld svært vanskelig å forutsi hvilke data som skal samles inn, og hvordan brukerne benytter seg av systemet. Datamengdene er langt mindre éntydige enn vanlige nettsider er. I tillegg kan forventningene til erindring og presisjon være ulike.[4] Utviklerne trenger en mengde testdata som de kan benytte seg av. Innstilling av rangering må gjøres i henhold til spørringer som kjøres i det fremtidige systemet.

Problemet er spesielt stort hvis systemet skal gjøre noe helt nytt, og altså ikke erstatte et eksisterende system for virksomhetssøk. Internett har jo alle tilgang til, så der er det fullt mulig å få tilgang til store mengder med relevante testdata, samtidig som det er gjort mange grundige undersøkelser for hvordan brukerne her oppfører seg. Skal utviklerne måtte basere seg på antakelser når det gjelder en gitt virksomhet? Det tar tid å lage gode testdata.

Et mulig bøtemiddel, som etter alt å dømme brukes mye i dag, er å la erfarne konsulenter ta seg av søkedelen av prosjektet. Dette gjøres for eksempel i det kommersielle selskapet Fast, som har sin egen prosjekt- og konsulentavdeling.[5] Andre konsulentselskaper kan også spesialisere seg på søketeknologi. Dermed opparbeides det heurestikk, og det blir lettere å utarbeide løsninger på bakgrunn av relatert domenekunnskap. Problemet er bare at all denne kunnskapen forblir lukket og intern.

Et annet viktig punkt er å analysere logger fra eksisterende systemer i forkant. Dersom en gitt virksomhet har søkeløsninger på plass allerede, bør spørringer herfra analyseres. Det kan være vanskelig å få tilgang til slike data i forkant, og derfor kan det oppstå en tidsmessig utfordring i prosjekter.

Til sist kan det være et forslag at prosjekter gis en lang innkjøringsfase etter implementasjonen, slik at søkeløsningene kan optimaliseres på bakgrunn av nye data. Problemet med dette er bare at det kan tenkes at brukerne vil forkaste systemet. Søkeresultatene bør altså være som forventet allerede fra starten. Ellers vil det kunne oppstå en mistro til virksomhetssøket.

Likevel er alle disse forslagene om tiltak fånyttige, sammenlignet med den gevinsten det ville ha vært hvis problemet hadde en mer generell løsning. Hvis markedet for virksomhetssøk får en oppsving, vil dette igjen kunne gi rom for flere og bedre undersøkelser. Slik forskning er en god start for å komme i mål her.

[4] Erindring og presisjon sier noe om kvalitet i søkesystemer, og hører inn under IR-fagfeltet. Begrepene forklares i avsnitt 4.5.1 på side 86.

[5] Fast Solution and Customer Services (SCS) arbeider med å tilpasse firmaets søkeløsninger for kunder.

3.4.3 Rangering av heterogene resultater

▷ Hvordan kan et regneark rangeres mot et bilde?

På grunn av at virksomhetssøk karakteriseres ved heterogene data, er det vanskelig å finne en god og allmenn metode for rangering av dokumenter. Det virker intuitivt fornuftig at HTML-sider som har en høyere frekvens av gitte søketermer rangeres høyere for et søk etter disse termene. Så enkelt kan ikke for eksempel et regnskap vektes mot en e-post, eller en kontrakt mot et tilfeldig notat.

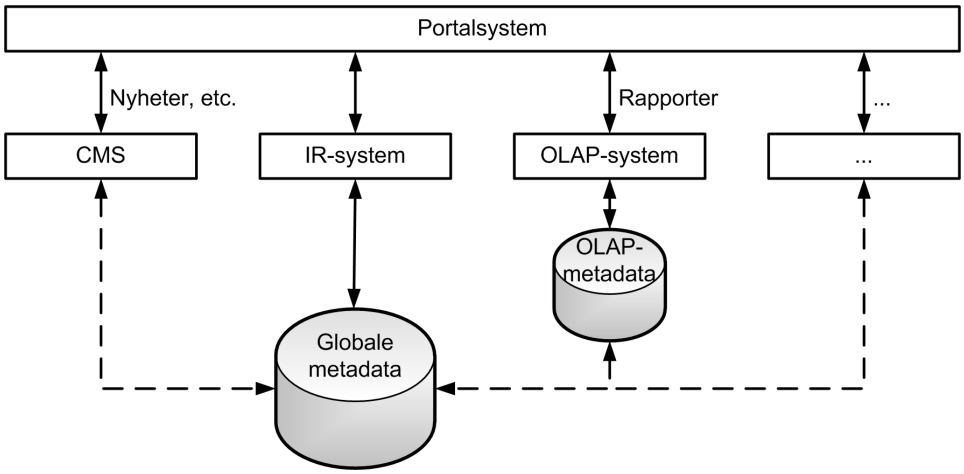
Selv om problemet fremstår som uløst, kan en tilnærming være å dele de heterogene dataene opp i flere ansamlinger (collections). Hver av disse kan søkes i separat. Inndelingen eller kategoriseringen kan skje etter for eksempel hvilket kildesystem dataene samler fra, hvilken egenart de har, hvilken filtype de har, eller andre relevante metadata. I en videreutvikling av denne ideen kan ansamlingene kombineres i grupper, eller siloer om man vil, som hver for seg kan søkes som en helhet. Vektingen mellom de ulike ansamlingene innbyrdes i hver silo kan tilpasses. Dette vil også muliggjøre bedre kontekstuelle søk, og tilnærmningen benyttes av kommersielle aktører. Ideen om siloer er sågar hentet fra Fast og deres Unity-programvare.

3.4.4 Utvikling av gode, personlige portaler

▷ Hvordan kan “min bedrift” synliggjøres?

Visjonen med brukertilpassede portaler er å skape en hensiktsmessig start- eller søkeside for ansatte i en bedrift. Dette problemet ligger innenfor distribuert IR. Det er kanskje enklest å forklare dette med et eksempel.

> Peder Aas arbeider i OilWells norske R&D-avdeling. Han utvikler borekroner, og derfor er ressurser rundt boring mest interessant for ham, men også nærliggende fagfelt som geologi og materialteknologi er relevante. Siden OilWell er et multinasjonalt firma, riktignok med røtter i Norge, finnes det en global intraweb som han kan logge inn på. Mye av det som ligger her er interessant for Peder Aas, men det meste er langt utenfor hans arbeidsområde. Videre har avdelingen hans lokale databaser og en felles filtjener, samtidig som Peder Aas lagrer mange av resultatene sine både på arbeidsstasjonen og den bærbare PC-en. På internett finnes det også mengder av interessante ressurser; noen av dem er abonnementstjenester som jobben hans betaler i dyre dommer for.



Figur 3.2: Priebe og Pernuls forslag til metasøk i portaler [88]. Metadata samles i et sentralt hvelv, men dokumentene forblir i sine respektive systemer.

I eksemplet over er det behov for å samkjøre en rekke underliggende søkesystemer (ta gjerne et blikk tilbake på figur 2.2 på side 24):

- Lokale søketjenester på arbeidsstasjonen og den bærbare PC-en
- Nettverks- og databasesøk i avdelingens ressurser
- Kontekstsensitivt søk på intranettet
- Søk på internett, helst kontekstsensitivt
- Søk i eksterne kilder som firmaet betaler for

I tillegg til at alt skal søkes over en distribuert plattform, skal grensesnittet tilpasses brukerens situasjon. Dette omfatter ikke bare tilgangsrettigheter og arbeidsområder. En portal for ansatte skal bygge opp under den enkelte brukers synsvinkel for at systemet skal virke så transparent og nyttig som mulig.

En mulig løsning her kan være å bruke metasøk, som beskrevet av Priebe og Pernul [88]. Ved å bruke informasjonsgjenfinning i metadata fremfor fritekst, kan semantikk brukes bedre i globale søk. Figur 3.2 viser hvordan forfatterne foreslår at metadata samles fra ulike kildesystemer i et globalt hvelv. Dette kan så indekseres av globale IR-systemer. En utfordring ved denne løsningen er blant annet ulike standarder for metadata, og artikkelforfatterne foreslår å opprette en global ontologi for oversettelse mellom disse standardene.

3.4.5 Effektive søk i e-postarkiver

▷ Hvordan kan potensialet i e-post utnyttes bedre?

For å illustrere dette punktet viser David Hawking til en fiktiv korrespondanse over e-post [51]. For rent tekstsøk er det stort sett få problemer med store mengder e-post. For virksomhetssøk er problemet annerledes, spesielt fordi brukeren gjerne forventer å finne et spesielt dokument eller en korrespondanse som vedkommende kjenner til fra før. Her er noen av utfordringene:

Tråder. Diskusjoner kan forme store hierarkier eller trær, greit nok, men samtidig kan mottakere settes av og på underveis. Hva skjer da med tilgangsreguleringen? Skal brukeren akseptere mangler i korrespondansen dersom enkelte meldinger er skjult for ham? Hva da hvis noen av disse meldingene er sitert i senere meldinger hvor han har tilgang? Dessuten er det vanskelig å analysere semantikk ettersom e-post ofte inneholder et innslag av personlige eller uformelle meldinger. Det er ikke enkelt å forstå hvem som svarer på hva, for et kontekstbasert søkesystem.

Arkivering. Lover og regler setter strenge føringer for arkivering av e-post. I en fersk artikkel beskriver Guy Creese hvordan myndigheter ofte forventer at virksomheter skal ha informasjon tilgjengelig nærmest øyeblikkelig ved enkelte forespørsler [28]. Samtidig har forskjellige land sine egne regler for hvor lenge e-post kan lagres.

Vedlegg. Vedlegg i ymse filformater skaper hodebry. Hvis eksempelvis en kontrakt i PDF-format legges ved en e-post, faller den under andre lover og regler. Derfor kan det argumenteres for at den skal behandles uavhengig av selve meldingen den var bundet til. Samtidig kan viktig informasjon ligge i meldingen, ikke minst om tidspunkt, avsender og mottaker, som må tas vare på. Virksomhetssøket må kunne identifisere viktigheten til, og kanskje også lødigheten av, slike dokumenter automatisk og behandle dem ulikt. Hodebryet blir komplett når lover og regler legger ulike føringer på de ulike delene av informasjonen.

Struktur. Struktur innen hver melding må tolkes. Her ligger det heldigvis stor hjelp i at e-post følger gitte standarder. Uansett må skjemaene for IR-systemer tilpasses e-post. Dette er ikke en umulig oppgave i dag. Verre er det med struktur i selve meldingene.

3.4.6 Relevans uten lenkeanalyse

▷ Hvordan kan "løse dokumenter" vurderes og rangeres?

Estimering av dokumenters viktighet (på generelt kontekstgrunnlag) og relevans (på spesielt spørringsgrunnlag) utgjør også et uløst problem. Vanlige søkemotorer på nettet har et kolossalt fortrinn i hypertekstmediets natur. Dokumenter lenker til hverandre stort sett på samme måte ved hjelp av ankere. Dermed faller det kanskje naturlig at et dokument er mer nyttig *i den allmenne internettbrukerens kontekst* hvis flest mulig andre dokumenter lenker til det. Nettsider peker på hverandre, og dette avslører mye om deres popularitet.[6]

Vanlige dokumenter i virksomheter, derimot, peker ikke så ofte på hverandre. Ikke minst skyldes dette den sedvanlige mangelen på unike identifikatorer. Imidlertid er det vanlig at det refereres til andre dokumenter, men uheldigvis ved hjelp av naturlig eller uformelt språk.

Problemet gjøres lettere av at mange organisasjoner bruker CMS-er med automatiske lenker, vedlegg og lignende.[7] Allikevel er det også her en hindring av at terskelen synes å være høyere for å opprette sider på intranett enn på internett, av sosialpsykologiske årsaker.

Dmitriev et al. har undersøkt hvordan relevans i intranett kan bedres ved hjelp av kommentarer til hvert dokument [33]. Disse gis enten eksplisitt av brukeren, eller de utvinnes ved å analysere spørringslogger.

3.4.7 Søkekontekst innenfor virksomheter

▷ Hvordan kan systemet forstå hva brukeren vil ha?

Videre fremholder Hawking at utnyttelse av søkekontekst i virksomheter ikke har kommet langt nok [51]. Her spørs det egentlig hvor strenge krav som stilles. Mye av konteksten kan faktisk legges ved i spørringen uten brukerens viten. Aktuell informasjon er geografi i ulik skala og sammenheng, informasjon om brukeren som alder, kjønn, språk, interesser, stilling og arbeidsoppgaver, søkehistorikk, samt art og formål for det problemet som ønskes løst ved hjelp av søk.

Kanskje tenker Hawking først og fremst på spørreytelsen. Hvis kontekst kan tilpasses delvis ved indekseringstidspunktet, vil det være lettere å oppnå høyere ytelse. Det kan derfor tenkes at utfordringene er større for store, multinasjonale organisasjoner. Større spredning i brukermassen gjør tilpassingen vanskeligere.

[6] Dette utnyttes også til å luke ut duplikater; se side 78. Rangering og relevans forklares også nærmere på side 83.

[7] CMS står for Content Management System og brukes for det meste til å håndtere publiseringen av nettsider på intranett og internett.

3.4.8 Fremtidige, kontinuerlige data

▷ Hvordan kan et system vurdere lydklipp mot hverandre?

Lyd, video og annet innhold som strekker seg over tid, øker i volum hos en del organisasjoner. Nye formater og nye standarder som ennå ikke er fastsatt, bør ikke stikke kjepper i hjulene når de ankommer. På internett er noen av de mest suksessfulle nettstedene distributører for medier i kontekst. YouTube [124] er et godt eksempel. Et fellestrekk er at disse bygger opp nettsamfunn samtidig som benytter seg av kommentarer. Er dette noe å bringe videre inn i virksomhetssøk?

Freyne et al. [41] utreder hvordan nye brukergrensesnitt som sosiale søk og sosial navigering kan kombineres for effektiv informasjonsgjenfinning, men stiller ikke diskusjonene opp mot virksomheter spesielt. Et annet forsøk med sosial bokmerking i virksomheter har imidlertid vist at informasjonsgjenfinning kan forbedres også her [77].

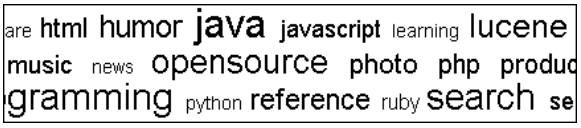
3.4.9 Crawlere og standardisering

▷ Hvordan kan all ønskelig informasjon eksponeres?

Crawlere og spidere er i prinsippet enkle programmer som traverserer internett, intranett eller andre systemer. Dagens crawlere har kommet et godt stykke videre og er relativt avanserte. Allikevel har de et godt stykke igjen å gå før de blir virkelig smarte. Problemet er nemlig at selv om mange crawlere i dag er svært konfigurerbare, og kan tilpasses ulike intranett relativt godt, så kunne de ha tatt enda mer hensyn til fremveksten av eksterne systemer hos virksomheter. ES-systemer skal helst være i stand til å kommunisere effektivt med eksterne systemer, og da oppstår enda et problem, nemlig at mange eksisterende systemer ikke er designet for å ta i bruk ekstern søketeknologi. Kanskje har de egen søketeknologi bygget inn i seg, samtidig som de har mangelfull arkitektur for implementasjon av alternativer. Det finnes et par måter å bedre denne situasjonen på.

For det første er det ikke ubetydelig at utviklere følger standarder for søk. Hvilke standarder som eksisterer og om de er gode nok er ikke poenget her; applikasjoner som bygges for fremtiden bør i best mulig grad enes.

For det andre er det en fordel hvis applikasjoner og systemer blir best mulig til å lagre data. Med andre ord bør programmene eksponere mest mulig og produsere gode metadata. På denne måten kan andre systemer, inkludert IR-systemer, ha større muligheter til å utnytte semantikk. Som over er det prinsippet som teller. Om standarden heter XML RDF eller noe annet, spiller mindre rolle.



Figur 3.3: Stikkord og sosial bokmerking kan forbedre dokumenters relevans, også i virksomheter. Kan de sosiale mekanismene etterlignes ved hjelp av kunstig intelligens?

Håpet er at crawlere gjennom dette kan kunne nå nye høyder. Det vil kunne oppstå bedre muligheter for utløserbasert crawling (trigger-based), samtidig som den kan bli mer målrettet med hensyn på domene, kontekst og semantikk.

Fremtidige crawlere ser ut til å bevege seg i denne retningen, dersom problemene nevnt her bekjempes [78]. Forhåpentligvis vil også dette bidra til at unyttige data i form av støy reduseres. Som nevnt i avsnitt 3.4.11 på neste side er nettopp slik støy en stor utfordring.

3.4.10 Etteraping av sosiale mekanismer

▷ Hvordan kan ES-systemene lære å tenke?

Sosial bokmerking, tilordning av stikkord eller kommentarer, er som allerede nevnt et nyttig konsept som relativt nylig har fått stor popularitet på internett. Fenomenet medfører engasjement og deltakelse i letingen etter virkelig nyttig informasjon. Likevel er det mye som ikke blir eller kan bli kommentert eller markert på denne måten. Hva er grunnen, og kan noe gjøres?

En mulig årsak kan være at det ikke er tid til å skrive stikkord i alle sammenhenger. I enkelte situasjoner er folk så opptatt med å finne frem til stoff at de ikke orker merarbeidet det er å skrive inn noen stikkord – selv om de måtte finne svært matnyttig informasjon for andre problemer enn det de nødvendigvis ønsket å løse denne gangen... Det kan jo også tenkes at mange rett og slett ikke gidder å bruke stikkord.

Ellers er det interessant å observere at taggingens styrke ligger i at ønsket informasjon stiger i popularitet. Med andre ord settes det menneskelige mål på kvalitet. Dette medfører at informasjon som noen faktisk har undersøkt og funnet frem til, nærmest stiger i gradene. Hva så med nyttig informasjon som aldri gjenfinnes? Det kan hende at et optimalt dokument blir rangert som nummer 50, men at brukeren fant noe helt greit som nummer 15. Dette kan i verste fall bli en negativ sirkel hvor det blir vanskelig for enkelte dokumenter å "debutere" i ES-system, spesielt hvis rangeringen i utgangspunktet er suboptimal.

Løsningen kan bli at fremtidige ES-systemer drar nytte av adaptive teknologier for læring, med andre ord kunstig intelligens, med det for-

mål å etterligne sosiale nettverk. Systemer har et langt større potensiale til å traversere gjennom sine egne datasamlinger. Hvis de bare klarer å forstå hvordan brukere i gitte domener eller kontekster markerer nyttige dokumenter, kan oppførselen etterapes. Dette kan være nok til at et dokument får bedre relevans og dermed etterhvert blir markert videre med stikkord av mennesker.[8]

3.4.11 Eliminering av støy

▷ Hvordan bli kvitt alt rotet?

Datamassene innenfor virksomheter inneholder en god del støy, altså dokumenter som har lav eller ingen verdi på den formen de foreligger. En medvirkende årsak til at mengden av støy tiltar kan være den generelle økningen i datamengde. Samtidig bidrar stadig forbedrede eliminasjonsteknikker til bekjempelse av støymengdene. Det er ikke godt å si om summen er at "netto støy" øker innenfor virksomheter, men veksten er uansett lavere enn på for eksempel internett.

Store mengder støy kan få konsekvenser for kvaliteten i et informasjonssøk. Ytelse er ett av problemene, men kanskje verre er potensialet for redusert pålitelighet og feil rangering. Spesielt er det to varianter av støy som skaper problemer for virksomhetssøk [78], og metodene for bekjempelse har et stort forbedringspotensiale.

Feil metadata er en type støy som direkte påvirker påliteligheten til virksomhetssøk. Ta for eksempel en artikkel; hvis navnet på forfatteren er feilstavet på en finurlig måte, vil IR-systemer ikke gjenfinne artikkelen i det hele tatt. Selv om mange moderne datasøk ved hjelp av språkanalyse klarer å luke ut vanlige, menneskelige stavefeil, vil en feil av mer teknisk art ikke bli fanget av disse mekanismene. Minst like ille er det hvis et helt annet forfatternavn er oppgitt. Et menneske vil raskt kunne fortelle at Henrik Ibsen ikke har skrevet *Modern Information Retrieval* [18], men hvordan skal søkesystemene finne ut av dette?

Den andre vanlige støytypen har med redundans å gjøre. Den enkleste formen for redundans går ut på at to dokumenter inneholder den samme informasjonen. Selv dette kan være en større utfordring innenfor virksomhetssøk enn for websider, på grunn av linkproblematikken. Langt verre blir problemet imidlertid hvis det kun er deler av informasjonen som er redundant. Ulike utgaver av det samme innholdet kan finnes på forskjellige steder, i ulike kildesystemer, være av vidt ulike filformater og lignende. Det nytter heller ikke alltid å se på publiseringsdato.

[8] Kanskje er det også en mulighet å gjøre det motsatte, altså at dokumenter som eksponeres ofte men aldri får tilordnet stikkord, blir markert som mindre nyttige? For alt *jeg* vet kan det være at dette allerede gjøres i dag. En enorm mengde ukjente faktorer spiller for eksempel inn på Googles rangering, og selv om det firmaet først og fremst produserer nettsøk er analogien illustrerende nok.

Et gammelt tekstdokument kan ved en feiltakelse være åpnet og lagret igjen, med en liten kommaendring. Den offisielle PDF-versjonen kan da få lavere relevans, selv om det er det "riktige" dokumentet. Igjen – et kyndig menneske vil kunne oppdage feilene ut fra kontekst og andre referanser, men for en stakkars medarbeider som ønsker å finne ut noe nytt kan følgene bli heller sørgelige.

Hvordan elimineres støy? Automatiske rettelser ved hjelp av smartere programvare er svært ønskelig, men dette vil kreve at systemene blir langt flinkere til semantisk dokumentbehandling enn de er i dag. I mellomtiden er botemiddelet bedre menneskelig kontroll, rutiner og opplæring, samt smidigere regelverk.

Den største utfordringen blir kanskje å ordne problemet med så få ressurser som mulig. Menneskelig og manuell kontroll er dyrt både i tid, penger og opplæring. Etterhvert som bedre systemer blir utviklet, er det derfor opp til hver enkelt organisasjon å avgjøre i hvor stor grad menneskelig kontroll er nødvendig. Hva koster mest – eliminering av støy eller å leve med støy?

Kapittel 4

Informasjonsgjenfinning

> Evnen til å forenkle innebærer å fjerne det unødige
> slik at det nødvendige får tale.
>
> — *Hans Hofmann*

I de foregående kapitlene har mye dreiet seg om hvordan virksomhetssøk ikke bare er søk, og at utfordringene er relatert til større sammenhenger og systemer. Dette kapitlet ser nærmere på selve søkingen. Hittil har bare overflaten blitt skummet, så derfor rettes nå fokuset mot det tekniske i IR-biblioteker, som datastrukturer, indeksering, søk, kvalitet og ytelse.

Et mer presist ord for søk i denne sammenhengen er *gjenfinning*, fordi man søker etter et dokument som fra før av er indeksert. Hvordan gjenfinnes et dokument? Det finnes flere måter å velge ut hvilke dokumenter som passer for et gitt søk. Måten disse finnes frem på, og hvordan de rangeres, danner ulike modeller. Som tidligere nevnt handler virksomhetssøk ofte om å finne det *riktige* dokumentet, ikke bare det *beste*. I avsnittet etterpå blir det forklart hvordan gjenfinningen går så raskt, og hvordan en invertert fil ser ut; modeller og datastrukturer, altså.

Indeksering og søk er neste tema. Det første handler om hvordan data mates inn i systemet, altså skrives til datastrukturen. Dette skjer gjennom mange steg. Det andre omhandler leseoperasjoner og spørringer. En del tekniske detaljer rundt selve fremletingen av data er sløyfet, og vekten er heller lagt på ulike typer spørringer og hvordan disse transformeres til mer konsise språk.

Til sist tar kapitlet opp mål på kvalitet og ytelse. Kvalitet handler om hvor "gode" søkene er, altså hvor relevante de synes å være. En vanlig formell definisjon på dette omfatter noe som kalles erindring og presisjon. Etterpå forklares det hvordan ytelse og skalérbarhet går hånd i

hånd. Dette er vesentlig siden alle typer søk, også virksomhetssøk, kan omfatte millioner av dokumenter og høy trafikk.

Selv om dette kapitlet går mer i dybden på det tekniske, forsøkes det å sette søk i sammenheng med resten av oppgaven. Som allerede nevnt er det mange utfordringer innenfor IR og ES som glir over i hverandre. Under diskusjonen mot slutten blir flere søkeprodukter evaluert, og mye av grunnlaget for dette ligger i emnene som dette kapitlet tar opp.

4.1 Modeller

Noe av det mest fundamentale i søk er å kunne avgjøre om dokumenter er relevante i forhold til brukerens ønsker. Dette henger nøye sammen med hvordan dokumenter gjenfinnes. For å gjenfinne og avgjøre relevans finnes det flere modeller og algoritmer. De tre kanskje mest klassiske modellene er henholdsvis den boolske modellen, vektormodellen og sannsynlighetsmodellen. Det finnes i tillegg flere avarter av hver av disse.

Den boolske modellen, også kalt den binære modellen, tar utgangspunkt i at indekstermer enten forekommer eller ikke forekommer i et dokument. En indeksterm k_i assosiert med et dokument d_j får derfor en binær vekt som er $w_{i,j} \in \{0,1\}$. Denne modellen gjør rangering vanskelig, og det finnes derfor et par mengdeteoretiske videreutviklinger som ikke skal gjennomgås her. Disse kalles hendholdsvis uklare (fuzzy) modeller og den utvidede boolske modellen, hvorav varianter av den førstnevnte er utbredt.

I vektormodellen blir dokumenter og spørringer representert som vektorer i et t-dimensjonalt rom. Hensikten er å regne ut graden av likhet mellom søketermer (spørring) og et dokument i indekset. Metoden bruker frekvensen av en term k_i i et dokument d_j, samt vanligvis en logaritmisk formel, for å kunne regne ut en mer nøyaktig vekt $0 \leq w_{i,j} \leq 1$. Den største fordelen med vektormodellen er økt søkekvalitet (se side 86). I tillegg er metoden enkel og rask, og er derfor utbredt. Avledede algebraiske modeller inkluderer generalisert vektormodell, latent semantisk indeks og nevrale nettverk, uten at disse blir forklart her.

Den siste klassen av modeller baserer seg på sannsynlighetsregning, og har flere avanserte varianter. Hovedprinsippet går ut på at brukeren gir tilbakemelding om dokumenters reelle relevans, og denne heuristikken blir brukt i senere utregninger og anslag. Detaljene er ikke vesentlige i denne sammenhengen.

Ifølge Baeza-Yates og Ribeiro-Neto [18] regnes de mengdeteoretiske modellene, inklusiv den boolske, for å være den svakeste gruppen, samtidig som det er uenigheter om hvilken av de to siste klassene av modeller som gir best resultater. For denne oppgavens sammenheng holder det

å forstå prinsippene om at dokumenter kan gjenfinnes på flere måter. Viktig er det dessuten at rangering og relevans henger nøye sammen med hvilken modell som velges. Flere IR-biblioteker tillater justeringer på disse områdene.

4.2 Datastrukturer

Søk er utrolig raskt. Den store hemmeligheten ligger i all hovedsak i effektive datastrukturer. I indekset er dokumentene organisert for raske oppslag. Tenk gjerne på indekset som et stikkordsregister i en bok. Stikkordene er sortert alfabetisk, og under hvert av dem står sidetallene hvor ordet kan gjenfinnes. Den mest vanlige datastrukturen i informasjonsgjenfinning er basert på det samme prinsippet. Mer formelt kan det snakkes om et invertert indeks, inverterte filer og posisjonsindeks.

Tenk på en mengde dokumenter. Dette kan karakteriseres som "dokumenter med ord". Hvert dokument inneholder en rekke ord. Invertert blir dette "ord med dokumenter". En mengde med ord peker altså på dokumentene hvor de forekommer. Dette blir et invertert indeks. Formelt er dette en liste med termer hvor hver term t peker på dokumentmengden D de befinner seg i; $t_i \rightarrow D$.

En invertert fil følger det samme prinsippet. Dokumentet er en mengde av termer, hvor hver av dem har en posisjon. I teksten "se her" vil det første ordet ha posisjonen 1 og det andre ordet ha posisjonen 4, hvis man teller tegn. Hver term t peker da på en mengde av posisjoner P hvor termene befinner seg; $t_i \rightarrow P$.

I dag brukes stort sett posisjonsindeks, siden man har mange dokumenter, mange termer og mange posisjoner. Dette er en kombinasjon av de to prinsippene over. Hver term t peker dermed på en mengde av tupler T som er på formen (d, p) hvor d er et dokument hvor termen forekommer, og p er posisjonen i det aktuelle dokumentet for den gitte forekomsten. Som eksempel man forestille seg to dokumenter d_1 og d_2 med følgende innhold:

d_1 = "Gode eksempler er bil og båt. Teksten er et godt eksempel."
d_2 = "Dyre biler er gode."

Mange av termene her finnes flere ganger både i samme dokument, og i det andre dokumentet. Indekset kan derfor bli seende slik ut:

$$\begin{aligned} \text{"bil"} &\rightarrow \{(d_1, 19), (d_2, 6)\} \\ \text{"båt"} &\rightarrow \{(d_1, 26)\} \\ \text{"dyr"} &\rightarrow \{(d_2, 1)\} \\ \text{"eksempel"} &\rightarrow \{(d_1, 6), (d_1, 50)\} \end{aligned}$$

$$\begin{aligned} \text{“god”} &\rightarrow \{(d_1, 1), (d_2, 45), (d_2, 15)\} \\ \text{“tekst”} &\rightarrow \{(d_1, 31)\} \end{aligned}$$

Den observante leser legger sikkert merke til at noen ord mangler, at tegn og mellomrom er borte, og at noen ord har endret seg. Dette er et resultat av dokumentvask og forklares litt senere i det neste avsnittet. Akkurat nå holder det å konstatere at posisjonsindeks er en datastruktur hvor alle unike termer sorteres alfabetisk. De kan derfor gjenfinnes svært raskt. Disse peker så videre på hvilke dokumenter de befinner seg i, samt posisjonen i disse dokumentene.

En konsekvens av inverterte filstrukturer er at termene tar betraktelig mindre plass enn den originale teksten, typisk 40 til 60 prosent av plassen [18]. Derimot tar *adressene*, det vil si mengden av dokumenter og posisjoner, en del plass. Derfor er det vanlig å bruke blokkadressering og tilsvarende triks for at adressene skal bli kortere. Dette medfører at hele innholdet av det opprinnelige dokumentet må eksistere for at termene skal kunne gjenfinnes nøyaktig. Dette er akseptabelt, siden det allikevel er vanlig å ta vare på hele dokumentet i tillegg til en indeksert utgave.

Datastrukturer innenfor søk er selvfølgelig en ganske mye større vitenskap enn dette. I tillegg til andre indeksstrukturer som signaturfiler, finnes det mange teknikker rundt konstruksjon av datastrukturene, partisjonering for store indekser, fordeling mellom minne og eksternt lager, trestrukturer og så videre. For denne oppgavens del holder det å være klar over det viktigste prinsippet, nemlig invertering.

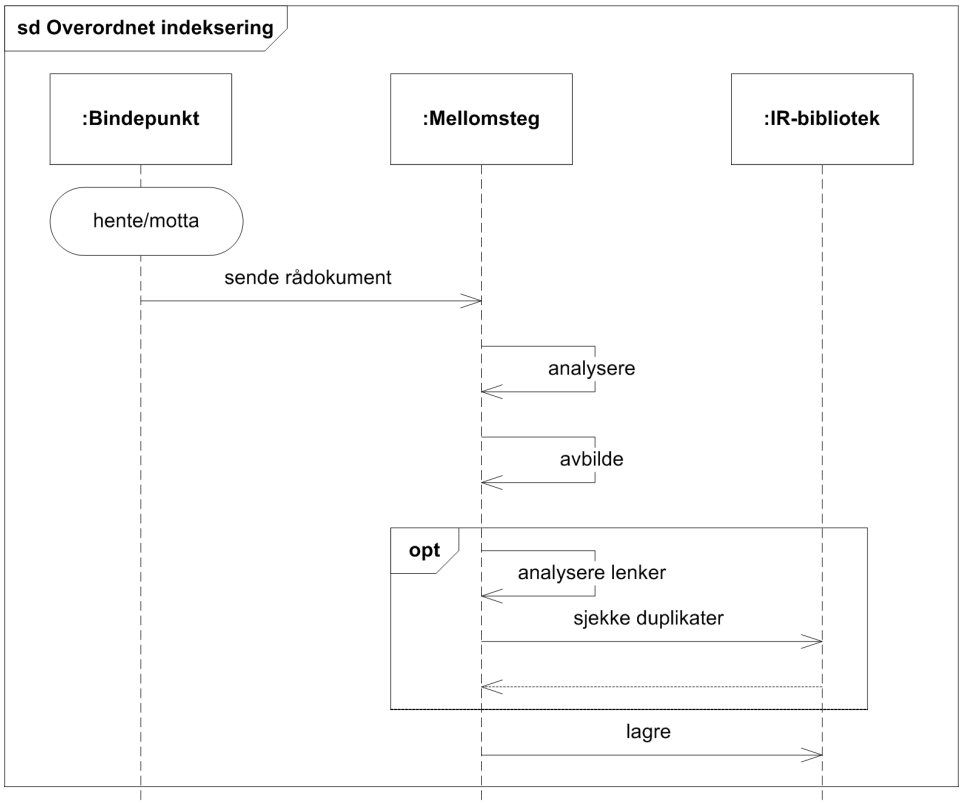
4.3 Indeksering

Sekvensdiagrammet i figur 4.1 på neste side illustrerer hva som skjer når et dokument skal lagres i indeksstrukturen. Tre overordnede ledd kan kalles bindeledd, mellomsteg og IR-bibliotek. Det går an å si at omverdenen ligger til venstre for bindeleddet, og at indekset ligger til høyre for IR-biblioteket.[1]

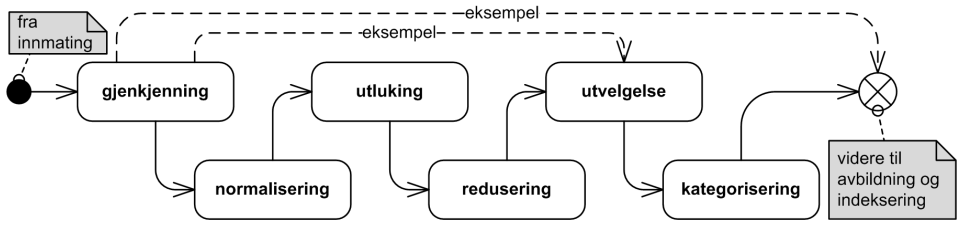
Mulige skriveoperasjoner er opprettelse av nye dokumenter, endring av eksisterende dokumenter og sletting. Beskrivelsen i de neste avsnittene er således gjeldende for alle disse operasjonene. Typiske aktiviteter inkluderer innsamling av data, analysering og dokumentvask, avbildning i henhold til skjema, lenkeanalyse og indeksering.

Merk at ikke alle stegene er nødvendige. Rekkefølgen på noen av dem kan dessuten legges om. Det blir ikke gått nærmere inn på diskusjoner

[1] Det kan argumenteres for at både mellomledd og indeks er en del av IR-biblioteket. I denne sammenhengen er det uvesentlig, og *IR-bibliotek* kan betraktes som et tenkt kommunikasjonspunkt.



Figur 4.1: Overordnet sekvensdiagram for typiske skriveoperasjoner



Figur 4.2: Dokumenter som er matet inn går gjennom analysering og dokumentvask. De stiplede pilene indikerer eksempler på at ikke alle steg er obligatoriske.

om hvilke punkter som tas med, og hvilken rekkefølge de kan omrokkeres til. Her blir de vanligste veiene lagt frem.

4.3.1 Bindeledd

For at dokumenter skal kunne mates inn må de hentes fra kildesystemer. Dette gjøres via et bindeledd. Crawler er et eksempel på et bindeledd som traverserer nettsider og sender dem og andre dokumenter videre til IR-systemet. Et annet eksempel er filtraverserer som kan hente dokumenter fra lokale stasjoner eller nettverk. Bindeledd for databaser er en tredje, vanlig type. Noen eksterne systemer krever at det skrives et eget og tilpasset bindeledd.

Et avgjørende designspørsmål er om bindeleddene dytter eller drar dokumenter.[2] Crawlere og filtraverserere er eksempler på sistnevnte, siden de aktivt oppsøker dokumenter og drar dem inn i IR-systemet. Dytting kan være aktuelt i eksempelvis publiseringssystemer. Bindeleddet kan da fungere slik at det passivt lytter på at systemet skal gi beskjed om endringer.

Bindeledd er viktige innenfor virksomhetssøk, siden kravene om tilpassingsdyktighet er høye. Derfor er det avgjørende at ES-systemer har en fleksibel arkitektur som gjør det enklest mulig å utvikle nye bindeledd.

4.3.2 Analysering og dokumentvask

Etter at et dokument er blitt fanget inn går det gjennom preprosessering, også kalt dokumentvask eller raffinering. Denne delen består av en rekke steg. Figur 4.2 presenterer noen av de vanligste operasjonene, som kan være:

- *Gjenkjenning* av datatyper, struktur og språk

[2] På engelsk kalles dette henholdsvis *push* og *pull*.

- *Normalisering* av aksenter, mellomrom og lignende
- *Utluking* av de vanligste ordene
- *Redusering* av ord til sin enkleste form
- *Utvelgelse* av termer for indeksering
- *Kategorisering* ved hjelp av synonymer og lignende

Alle disse punktene er naturligvis ikke alltid med. For eksempel er det ikke alle som behøver å gjenkjenne språk eller legge til lister av synonymer. Dessuten må leddene tilpasses den bruken de er tiltenkt. Eksempelvis er det mange virksomheter som opererer med egne lister over stoppord, alt etter dokumenters opphav eller art.

Det er også fordelaktig å kunne lage sine helt egne steg i preprosesseringen. Fast ESP er et eksempel; her kan kunder skrive sine egne steg i programmeringsspråket Python.

Med referanse til figur 4.1 på side 73 foregår alt sammen i mellomsteget, nærmere bestemt i aktiviteten som er kalt “analysere”. Etterpå avbildes de ulike datakategoriene i henhold til et indekseringsskjema før de indekseres. Disse prosessene er omtalt litt senere. Aller først må de ulike punktene ovenfor utdypes.

Gjenkjenning

Dette første steget består av tre mindre deler, nemlig gjenkjenning av filtype, gjenkjenning av struktur og gjenkjenning av språk. Filtype og struktur er ofte bundet tett sammen. For konfigurerbare søk i virksomheter er det vanlig å lage egne skjemaer for indekset. Da er det lurt å la gjenkjenning av struktur være tilpasset både filtype og skjema. Til identifisering av språk brukes n-gram-algoritmen mye [75].

Normalisering

Under normaliseringen foregår det en leksikalsk analyse av teksten. Hovedformålet er å konvertere en rekke av bokstaver og tegn til en rekke av kandidater for ord, som siden kan bli til indekstermer. Den første oppgaven blir derfor å kunne kjenne igjen bokstaver, tall og tegn som hører naturlig sammen.

Her gjøres det stor bruk av regulære uttrykk. For eksempel kan en bindestrek mellom to grupper av bokstaver indikere at gruppa utgjør én term, mens bindestrek mellom to tall kan bety at det er snakk om et omfang. Komma, punktum og andre tegn sier mye om innholdet, og dette bør utnyttes, og det må tas hensyn til forkortelser.

Etter at den leksikalske analysen er ferdig, skjer normaliseringen. Som begrepet indikerer går dette ut på å omgjøre ordene til en mer sammenfallende form. Aksenter og spesialtegn lukes ut, og alle store bokstaver gjøres om til små bokstaver. Resultatet er en mengde med termer som har langt mindre innbyrdes variasjon, og derfor kan de lettere gjenfinnes i et indeks.

Her er et eksempel:

```
Nyhet: Première på ICAs barne-
mat m/gelé på toppen!
```

Etter leksikalsk analyse og normalisering kan resultatet kanskje bli en mengde som ligner litt på dette:

$$dokument = \{\text{nyhet}, \text{premiere}, \text{på}, \text{icas}, \text{barnemat}, \text{gele}, \text{toppen}\}$$

Dokumentet går nå over i en form hvor unødvendige mellomrom og annet er borte. Det som sendes av gårde er en mengde med termer som skal analyseres videre. Resultatet vil selvfølgelig også inneholde informasjon om alle posisjonene for termene i dokumentet, og sikkert en del annet også. Her er bildet altså forenklet litt.

Utluking av stoppord

Nesten alle vanlige tekster inneholder en mengde hyppig brukte ord. Her er et eksempel (Keplers første lov):

```
Planetene går i ellipsebaner
med sola i det ene brennpunktet.
```

Ordene *i*, *med* og *det* er to preposisjoner og en artikkel som forekommer i svært mange norske tekster. Det kan derfor synes meningsløst å indeksere dem.

Slike ord kalles stoppord. Hvis et ord forekommer i 80 % av dokumentene i en ansamling, er det helt verdiløst for informasjonsgjenfinning [18]. Vanlige stoppord er som regel artikler og preposisjoner (som i eksemplet over), samt konjunksjoner. En del vanlige verb, adverb og adjektiver kan også regnes som verdiløse, for eksempel verbet *går* og adjektivet *ene* i setningen over.

Stoppord fjernes derfor ofte. En heldig konsekvens er at størrelsen på de inverterte filene blir redusert betraktelig. Ikke sjelden er gevinsten på over 40 % [18].

En ulempe er derimot at kvaliteten kan lide i form av at *erindringen* (se side 86) blir dårlig. Se for eksempel på denne spørringen:

```
for is og the kan kombineres
```

(Bær over med den gammeldagse stavemåten for ordet *te*.) Eliminering av stoppord gjør det nærmest umulig å gjenkjenne frasen. Problemet blir ikke lettere av at det brukes en kombinasjon av både norske og engelske stoppord. I verste fall vil bare ordet *kombineres* brukes i søket.

Totalt sett vil altså relevans kunne bli bedre, og indeksstørrelsen kan reduseres. Derimot medfører elimineringen av stoppord at frasesøk blir vanskeligere – et godt argument for at teksten skal fullindekseres i tillegg. Et lignende steg i språkbehandlingen, antifrasering, skjer på spørre- og resultatsiden for å hente ut essensielle søkeord.

Stemming og lemmatisering

Stemming og lemmatisering går ut på å redusere ord og termer til sin enkleste form. Bakgrunnen er at det skal være mulig å gjenfinne ord ved omskrivinger. Her er noen eksempler:

$$\begin{aligned} \text{gås, gåsa, gåsen, gjess, gjessene} &\Rightarrow \text{gås} \\ \text{gi, gir, ga, gav, gitt} &\Rightarrow \text{gi/gir(?)} \\ \text{fisker} &\Rightarrow \text{fisk/fiske/fisker(?)} \end{aligned}$$

Det er altså ikke bare å redusere substantiver og verb til sin enkleste form – algoritmene må også tilpasses alternative skrivemåter, vanlige endelser, ord som naturlig hører sammen, samt eventuelle tvetydigheter.

Redusering er ressurskrevende fordi det må tilpasses hvert enkelt språk. For engelsk er Porter-stemming [86] utbredt, og det finnes også tilsvarende fremgangsmåter for nordiske språk, germanske språk og en rekke andre. Mange av disse er fritt tilgjengelig. Imidlertid har nok kommersielle søkeaktører med lang fartstid neppe ønsker om å gi bort sine dyrebare resultater til konkurrenter.

Utvelgelse av termer for indeksering

Ofte velges ord ut på grunnlag av hvor mye informasjon de bærer. For allmenne tekster er det som regel substantiver, etterfulgt av verb, som i størst grad sier noe om dokumentenes innhold og skiller dem godt fra hverandre. På engelsk særskrives sammenhengende substantiver, så her er det nyttig å identifisere grupper av ord som kan indekseres sammen, for eksempel “graduate student”. Men også på norsk kan det være nyttig å definere en terskel for maksimal avstand mellom substantiver og regne disse frasene som enkelttermer.

Synonymordlister

I noen tilfeller er det fornuftig å ha et steg hvor synonymer blir lagt til. Et søk etter “platelager” vil dermed kunne gi treff på dokumenter som

inneholder "harddisk". I sammenheng med virksomhetssøk er det også nyttig å lage egne lister over assosierte ord, for eksempel i elektronisk handel.

4.3.3 Avbildning

Hensikten med avbildningen (engelsk: *mapping*) er å få den bearbeidede informasjonen over i indekset på riktig måte. I eksemplene over er det rent intuitivt tatt utgangspunkt i at dokumenter består av ren tekst hvor alt behandles likt, og at indeksene er enkle, uten datatyper eller annen form for felter. I virkeligheten inneholder selvfølgelig dokumenter både struktur og metadata, og indekser inneholder flere felter som kan defineres i et skjema.

Avbildning foregår ved at dokumentet filtreres gjennom et sett med regler som bestemmer hvilke biter av struktur, og eventuelt hvilke metadata, som skal inn i hvilke felter i indekset. Vanlige eksempler på felter er "tittel" og "brødtekst". Dersom kildedokumentet er en nettside, vil tittel vanligvis være det som er trukket ut fra `<title>`-feltet, og brødteksten vil være teksten som er ekstrahert fra selve nettsiden.

Det er viktig å være klar over at denne informasjonen, altså eksempelvis tittelen og brødteksten, er sendt over fra tidligere ledd i prosessen, spesielt leddet for gjenkjenning. Avbildningen har derfor normalt ikke ansvaret for å ekstrahere data og metadata, men å sørge for at data og metadata som allerede er ekstrahert kommer til bestemte felter i indekset. Det er derfor lurt at avbildningen samarbeider med gjenkjenningen, eller i det minste at regelsettet stemmer overens.

Under tilpassingen i virksomhetssøk står både avbildning og gjenkjenning sentralt, og regelsettet må defineres (eventuelt programmeres) ut fra indeksskjemaet. Feltene her vil også kunne danne grunnlag for tilpasset rangering av resultater. For eksempel er det vanlig å ønske at termer gjenfunnet i "tittel" skal havne høyere enn termer gjenfunnet i "brødtekst".

4.3.4 Lenkeanalyse og duplikater

Et vanlig problem innenfor automatisk innsamling av data, er forekomsten av duplikater. Dette er dokumenter som i det store og det hele inneholder det samme – altså at informasjonen er lik, men ikke nødvendigvis at dataene er identiske. Hvis to eller flere dokumenter er så like at de ikke supplerer hverandre med ny informasjon, medfører dette redundans. Dette får konsekvenser for både plass og ytelse. Løsningen på dette problemet er å analysere dokumenter for å avsløre duplikater.

Å finne duplikater er en dyr prosess. I store ansamlinger av dokumenter vil det sette ned indekseringsytelsen betraktelig, men gevinstene

kan likevel være verdt investeringen. Siden det er dokumenters innhold som skal sammenlignes, er det hensiktsmessig å ha dette leddet etter at dokumentene har blitt konvertert til sin interne indeksform.

Siden dokumentene først skal sammenlignes, er det fornuftig å implementere såkalt lenkeanalyse i tillegg. Lenkeanalyse går ut på å finne ut hvordan dokumenter refererer til hverandre. Dette er spesielt nyttig for nettsider, som i svært høy grad har denne karakteristikken. I tillegg til at lenkeanalyse hjelper til å finne ut ulike adresser til identiske eller nær identiske dokumenter, kan opprettelsen av lenkegrafer, telling, sammenligning og andre tilnærminger bidra til å avgjøre dokumenters viktighet. Spesielt på internett er det vanlig at dokumenter det ofte linkes til er mer etterspurte.

For virksomhetssøk kan dette steget være en utfordring. Nettsider kan lenke til hverandre, men hva med referanser mellom for eksempel PDF-dokumenter? Det er heller ikke bare å bruke vanlig lenkeanalyse på innhold fra intranett, siden intranett ofte ikke har tilstrekkelig uavhengige eller komplekse lenkegrafer. Duplikater og lenkeanalyse er dermed viktig også for virksomhetssøk, men som med nesten alt annet innen ES er det vesentlig å analysere hvert enkelt brukstilfelle og være klar over de begrensningene som foreligger.

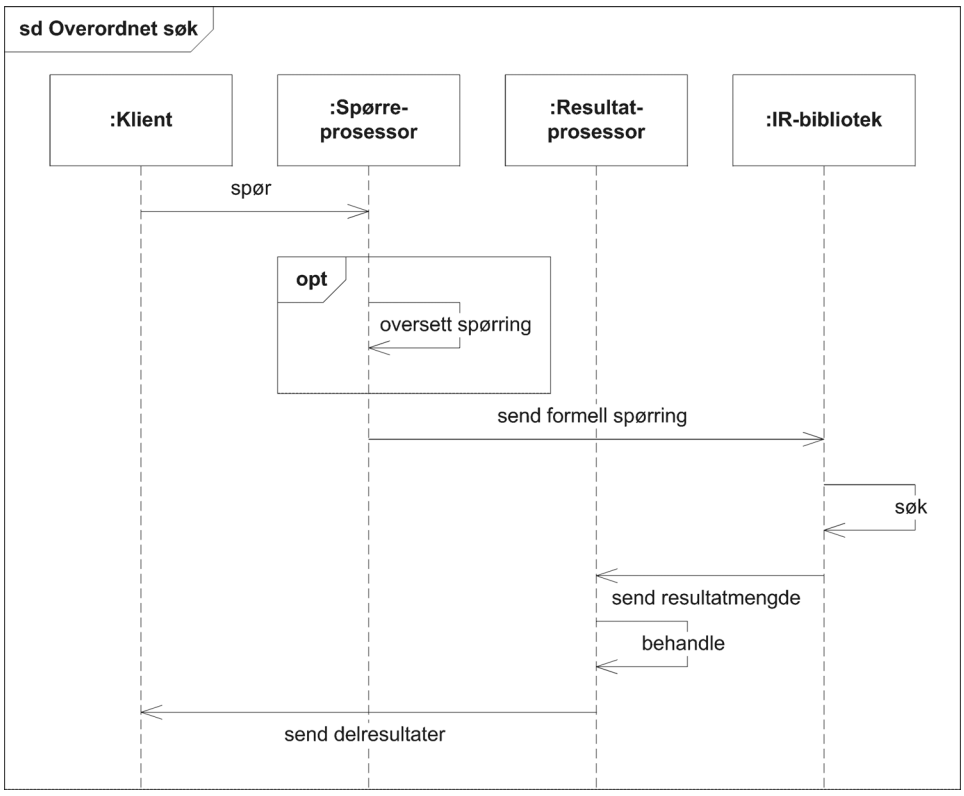
Analysering av lenker har dessuten en uheldig konsekvens. På internett er trafikk ofte ensbetydende med økonomisk gevinst, og derfor spekuleres det i å utnytte søkemotorers lenkeanalysering til å komme høyt opp på rangeringen. Mest utbredt er såkalte "lenkefarmer", men etterhvert som Google og andre aktører blir flinkere til å avsløre det de nok selv ser på som juks, blir følgelig metodene mer utspekulerte eller sofistikerte. For virksomhetssøk som ikke bruker lenkeanalyse på eksterne data er dette problemet selvsagt langt mindre, om ikke neglisjérbart.

4.4 Søk

For en sluttbruker som benytter seg av *vanlig søking* (se side 85) betyr spørringer som regel følgende:

- ▷ Et spørsmål stilles, i naturlig eller strukturert språk.
- ▷ IR-systemet finner dokumenter som passer til kriteriene.
- ▷ Brukeren får se resultatene, som er vektet etter relevans.

I virkeligheten er bildet naturligvis langt mer komplisert. UML-diagrammet i figur 4.3 på neste side viser neste detaljeringsnivå. Her er fire overordnede moduler blitt innført.



Figur 4.3: Overordnet sekvensdiagram for et typisk søk (spørring)

Med klienten menes det programmet som foretar selve spørringen. Dette kan være en webapplikasjon, en PC-applikasjon, eller kanskje et underliggende system som er enda mindre synlig for brukeren. I neste steg finner spørreprosessoren ut om spørringen er formulert i naturlig språk eller IR-systemets interne språk. I første tilfelle oversettes spørringen til sistnevnte språktype. IR-biblioteket søker så frem til relevante dokumenter og returnerer en vektet resultatmengde. Hvert resultat inneholder én eller flere attributter, hvor dokumentets adresse eller nøkkel står sentralt. Resultatprosessoren mellomlagrer resultatmengden og gir tilbakemeldinger til klienten, alt etter hva den blir bedt om.

Spørreprosessoren og resultatprosessoren kan med fordel kombineres i én og samme modul. Grunnen til dette kan være at spørsmålet fra klienten inneholder informasjon som har betydning for delresultatene som returneres. For eksempel kan spørringen angi at bare et visst antall dokumenter skal vises.

Spørringer utgjør et sentralt kapittel innenfor informasjonsgjenfinning. Hovedutfordringen består i i størst mulig omfang å forstå hva sluttbrukeren er ute etter, og samtidig i hvilken grad indekserte dokumenter er relatert til dette ønsket. Det finnes metoder for å fastslå kvaliteten i en søkemotor etter kriterier relatert til denne problemstillingen; de viktigste av disse er gjennomgått i seksjon 4.5 på side 86.

Valget av hoveddelene springer ut fra et ønske om en viss funksjonalitet. Hensikten er i grove trekk å illustrere hvordan meldinger sendes, og at det finnes ekstra ledd mellom bruker og søkemotor, samt at meldingene fra begge opphav bearbeides.

De neste delavsnittene tar for seg noen av de viktigste utfordringene for et IR-system.

4.4.1 Spørringer, spørrespråk og transformasjoner

Spørringer er det mange brukere kaller selve "søket". Spørringer er den informasjonen som IR-systemet benytter for å finne frem relevante dokumenter. En spørring er formulert i et språk. Dette språket kan være muntlig, halvstrukturert eller strengt strukturert. I mange tilfeller vil det derfor foregå en *transformasjon* av spørringer internt. Her er tre eksempler på ulike spørrespråk:

1. `sjostakovitsj symfonier strykekvartetter`
2. `Hvor mange symfonier eller strykekvartetter skrev Sjostakovitsj?`
3. `sjostakovitsj AND (symfonier OR strykekvartetter)`

Disse spørrespråkene representerer ulike paradigmer.

Den første spørringen er kanskje den flest brukere av websøk er vant med. Her søker man enkelt og greit etter dokumenter som inneholder flest mulig av disse termene. Slike spørringer blir vanligvis transformert til et internt og mer strukturert spørrespråk. Hos Google [48] er det mulig å spesifisere små operatorer som eksempelvis tegnet '+' foran termer (i dette tilfellet vil termen være obligatorisk). Uten slik spesifisering vil søkemotoren prioritere dokumenter som inneholder alle termene. I tillegg har rekkefølgen av termene betydning, samt nærheten de har til hverandre i dokumentene (proximity).

Den andre spørringen representerer en formulering i naturlig språk. Å spørre systemer som om de var mennesker på denne måten, kalles *Question Answering* (QA). En del av dette paradigmet går også ut på direkte og presise svar. I eksemplet over kunne svaret kort og godt ha vært "15 av hver". Med hensyn til transformasjoner er QA langt mer avansert enn vanlige, halvstrukturerte spørringer, og det omfatter i større grad avansert språk og til dels psykologi. Se gjerne Kim Sands masteroppgave [95].

Den tredje og siste spørringen er et enkelt eksempel på en strukturert spørring. Dette kan, men må ikke nødvendigvis, samsvare med søkemotorens interne spørrespråk. Slike spørrespråk baserer seg på mengdelære og binære operasjoner med utstrakt bruk av operasjoner, prioriteringer av operasjoner, parenteser, og så videre. For vanlige brukere kan dette være vanskelig å bruke, men enkelte setter pris på nøyaktigheten som dette gir.

Generelt kan spørrespråk rangeres etter hvor nært de ligger naturlige språk, og dermed hvor lette de blir å bruke for nybegynnere. Som regel kan denne faktoren synes å være omvendt proporsjonal med hvor nøyaktig og kraftfult spørrespråket er, iallfall i skrivende stund. Derfor er det fremdeles behov for å kunne tilby ulike spørrespråk for både ferske og erfarne søkere.

Virksomhetssøk og spørringer

For virksomhetssøk er spørringer og spørrespråk litt annerledes enn for søk på nettet. En undersøkelse gjennomført av Freund og Toms viser at systemutviklere bruker flere termer og mer avanserte spørringer enn gjennomstnittlige brukere av intranett [40]. I undersøkelsen ble det tatt utgangspunkt i vanlige spørringer (tilsvarende punkt 1 ovenfor). Gjennomsnittlig spørrelengde var da 4.38 termer for systemutviklere, mens det for resten kanskje ligger så lavt som 1.4 termer [104]. Dette betyr kort og godt at de som utvikler søkesystemet må ta hensyn til hvem som skal bruke det.

Virksomhetssøk åpner også for større bruk av *spørrekontekst*. Et eksempel på dette kan være hvilken avdeling den ansatte som søker, job-

ber i, men mange andre typer metadata kan tenkes brukt. Slik tilleggsinformasjon kan være essensiell for å bedre rangering og relevans på intranett. Dette er viktig både fordi rangering ofte er vanskeligere på intranett, og fordi typiske brukere av intranett i størregrad leter etter et dokument de kjenner til fra før.

For aspektorientert leting (navigérbare søk, se side 53) kan systemutviklerne benytte seg av strukturerte spørringer. Det som i virkeligheten skjer i aspektorientert leting er faktisk at beskrankninger legges til eller trekkes fra det opprinnelige søket, for eksempel ved hjelp av ulike metadata.

4.4.2 Rangering og relevans

Rangering av dokumenter er et svært viktig og detaljert område innenfor søk. Denne oppgaven vil ikke gå nøye inn på dette detaljerte feltet, men kun forklare det som er relevant i sammenhengen.

Rangering (såkalt *ranking*) skjer som regel på to måter:

Statisk. Statisk rangering settes under innmatingen, før dokumentet havner i indekset. Rangeringen er ikke relatert til spørringer, og opphavet er ofte eksternt. For HTML-dokumenter, spesielt for innhold fra internett, er lenkeanalysen avgjørende for individuelle dokumenters poengsum (scoring).

Dynamisk. Dynamisk rangering skjer ved spørretidspunktet, og regnes derfor ut på grunnlag av selve spørringen. Termenes frekvens i dokumentet, posisjon i dokumentet og nærhet mellom termene i dokumentet er avgjørende. I tillegg brukes attributter som autoritet (opphav) og alder for å avgjøre relevans dynamisk.

Det er vanlig å bruke en kombinasjon av statisk og dynamisk rangering.

Ah Chung Tsoi forklarer hvordan relevans avhenger av to hovedfaktorer: Lenkestruktur og kontekst [110]. Videre beskriver han utførlig en rekke algoritmer som brukes i dag. En god, men tidlig kilde på Google PageRank er [21]. Senere har Tsoi et al. vist hvordan den kan optimaliseres [109].

4.4.3 Dokumentsammendrag (teasere)

Figur 4.4 på neste side viser de tre første resultatene fra spørringen “kaffe latte”, foretatt på søkemotoren Sesam [98]. De to linjene mellom overskriften og lenken i hvert resultat er et dokumentsammendrag. Det engelske ordet *teaser* er egentlig mer dekkende, siden brukeren skal lokkes til å klikke på lenken.

Figur 4.4: Eksempel på dokumentsammendrag

Formålet med dokumentsammendrag er å hjelpe brukeren i å avgjøre de enkelte dokumenters relevans. Som regel inneholder de alle ordene fra spørringen i uthevet form som på figuren. De inverte tekstdokumentene er noe av bakgrunnen for dokumentsammendragene. Forretningsmessig kan dokumentsammendrag på internett være svært viktige.

Stort sett kan sammendragene genereres på to ulike måter, statisk eller dynamisk. Begge metodene har sine fordeler og ulemper.

Statisk generering

Den statiske varianten blir generert før et dokument indekseres. Hensikten er å gi et sammendrag av dokumentets innhold, uavhengig av spørringer. Så lenge dokumentet er uforandret i indekset vil dokumentsammendraget ikke endre seg.

Fordelen med denne tilnærmingen er at genereringen ikke stjeler verdifull tid fra spørringer, og det kan derfor legges mer vekt på semantisk analyse og NE-ekstrahering (se 2.5.5 på side 43). Ulempen er at sammendraget vanskeligere kan knyttes til termene i spørringen, og deres nærhet til hverandre.

Dynamisk generering

De dynamiske sammendragene blir generert i spørretidspunktet, på bakgrunn av spørringen og dokumentets innhold. Disse vil altså endre seg for ulike spørringer med det samme dokumentet.

Fordelen med dynamiske sammendrag er at ord før og etter termene kan tas med, og dermed får brukeren en mulighet til å se i hvilken sam-

menheng selve søkeordene er nevnt. I mange tilfeller er dette derimot ikke nok til at brukeren får en forståelse av dokumenters innhold.

I figur 4.4 på forrige side er sammendragene tydeligvis dynamisk generert. Se på det første treffet. Her forekommer søkeordene *minst* tre ganger (i tillegg til tittel og lenke), og for hver av disse tre forekomstene står det noen ord foran og bak.

Utheving av søketermer

Det er opp til teaser-generatoren å utheve søketermer, og dette bør ikke gjøres i en utenforliggende applikasjon. Hovedårsaken til det er at IR-systemet inneholder mye mer informasjon enn selve dokumentsammendraget, og termer kan derfor bedre utheves på bakgrunn av hyppighet, nærhet, eventuelle metadata eller andre faktorer relatert til selve den inverterte teksten.

4.4.4 Vanlig søking kontra filtrering

Helt til nå er det tatt høyde for at søk foregår på vanlig måte, også kalt *ad hoc*. Det finnes måter å betrakte spørringer på som kan synes konseptuelt annerledes. Hva er forskjellen på ad hoc-spørringer og filtrering? Medfører disse ulike synene store konsekvenser for informasjonsgjenfinning generelt?

Ad hoc-spørringer går ut på at en bruker søker etter termer (eller angir mer avanserte spørringer) i en fast dokumentmengde. Dokumentmengden endrer seg selvfølgelig over tid, men for enkelhets skyld kan den her betraktes som statisk. Det er brukerens spørring som endrer seg, og på grunnlag av dette presenteres ulike dokumenter.

Filtrering, derimot, medfører at spørringen til brukeren ikke endrer seg, men nye dokumenter kommer kontinuerlig inn. Altså er brukeren ute etter stort sett det samme hele tiden, mens indekset er dynamisk. Eksempler på bruksområder kan være militær overvåking, systemer for kunstig intelligens, børsanalyser eller sanntidsmålinger. Rangering av dokumentene er mindre viktig [18], men kan brukes til å sette terskler ved store dokumentmengder.

Utfordringen innenfor filtrering er hovedsakelig å lage gode spørringer, gjerne kalt spørreprofiler. I utgangspunktet kan disse være like enkle som ad hoc-spørringer. Det beste er likevel å la profilen oppdateres og finjusteres kontinuerlig ved at brukeren rangerer dokumentene. Med mindre interesser endrer seg drastisk vil profilen over tid stabilisere seg.

Spørsmålet om ad hoc kontra filtrering er relatert til brukergrensesnitt (eller systemgrensesnitt), og endrer ikke på fundamentene bak IR i det hele tatt. Derfor er ikke emnet viet spesielt stor oppmerksomhet i

denne oppgaven, selv om det er viktig og være klar over de ulike bruksområdene. Det er heller ikke utenkelig at ad hoc-spørringer og filtrering kan kombineres for å forbedre relevans i resultater fra klassiske søk.

4.5 Kvalitetsmål

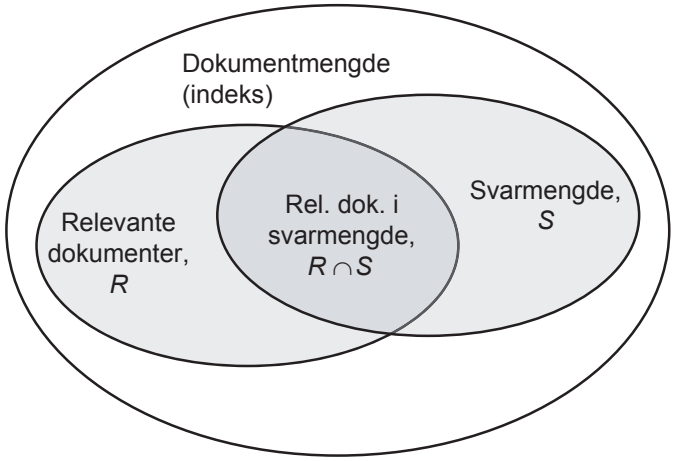
Hvor *god* er en søkemotor? For å måle ytelsen til et søkesystem kan flere tilnærminger benyttes. Hensikten med et slikt system er som regel å finne relevante resultater. Dette kan beskrives som kvalitativ ytelse, og tradisjonelt har balansen mellom *erindring* på den ene siden, og *presisjon* på den andre siden, vært satt som målestokk. Det finnes dessuten mer moderne beskrivelser av søkekvalitet som fokuserer på sluttbrukerens situasjon i større grad. Denne beskrivelsen passer kanskje bedre for virksomhetssøk, fordi man ofte er ute etter en bestemt mengde dokumenter som er kjent fra før – eller kanskje til og med ett eneste, kjent dokument.

I tillegg kan hastighet og skalérbarhet være mål for ytelsen, men dette tas først opp i neste avsnitt. Det interessante i første omgang er altså å måle ytelsen ved å evaluere den mengden av informasjon som returneres når en spørring foretas. For å kunne sammenligne kvalitet mellom ulike søkemotorer er det publisert en rekke kjente ansamlinger av dokumenter for test. Noen vanlige ansamlinger er *Cranfield*, *TREC* (med sin årlige IR-konferanse), *GOV2*, *NTCIR*, *CLEF* og *20 Newsgroups* [75].

4.5.1 Erindring og presisjon

Når en bruker sender en spørring q til et søkesystem, returneres en mengde dokumenter. Disse presenteres så for brukeren. Hvordan søkesystemet kommer frem til det gitte resultatet kan imidlertid virke som et mysterium. For å kunne besvare dette spørsmålet må et par definisjoner først på plass.

La den samlede mengden av indekserte dokumenter i et søkesystem, dokumentmengden, være I. Indekset inneholder dermed $|I|$ dokumenter. La videre mengden av relevante dokumenter i indekset i henhold til spørringen q være definert som mengden R. Kriteriene for hva som er relevant eller ikke for den gitte spørringen, er hypotetisk definert ut fra en "perfekt ekspert" sine formeninger for hver individuelle spørring. Det finnes dermed $|R|$ dokumenter som er relevante for q i I. La svarmengden være S. Snittet $R \cup S$ vil da utgjøre mengden av alle relevante dokumenter i svarmengden, som illustrert i figur 4.5 på neste side.



Figur 4.5: Sammenhengen mellom relevante dokumenter og svarmengde

Erindring er definert som andelen av relevante dokumenter som er blitt returnert, det vil si

$$erindring \equiv \frac{|R \cup S|}{|R|} \tag{4.1}$$

Presisjon er definert som andelen av de returnerte dokumentene som er relevante, det vil si

$$presisjon \equiv \frac{|R \cup S|}{|S|} \tag{4.2}$$

Disse betraktningene fører til at kvalitet innenfor informasjonsgjenfinning har to innfallsvinkler. For det første må R og S være så like som mulig. Dette er dessverre svært vanskelig. Derfor må erindring og presisjon balanseres riktig. Høy erindring vil medføre at flere relevante dokumenter returneres, men da vil samtidig flere irrelevante dokumenter returneres. Omvendt vil høy presisjon medføre at en større andel av svarmengden er relevant, men da vil svarmengden være mindre og flere relevante dokumenter må utelates.

Et litt mer subtilt problem finnes når selve fremvisningen av resultatene tas med i bildet, siden resonnementet ovenfor bare er gyldig hvis sluttbrukeren betrakter hele svarmengden. I et vanlig brukergrensesnitt vil en begrenset mengde dokumenter presenteres samtidig, og derfor er det viktig at IR-systemet har algoritmer for å tilordne grad av relevans og at søkeresultatene sorteres deretter. I kapittel 3 av boka *Modern Information Retrieval* gjøres det et poeng ut av at gjenfinningsalgoritmer kan sammenlignes ved å utforme diagrammer over gjennomsnittlig presisjon ved ulike nivåer av erindring. Et stort antall standardspørringer

brukes, slik at

$$\overline{P}(r) = \sum_{i=1}^{N_q} \frac{P_i(r)}{N_q} \tag{4.3}$$

hvor $\overline{P}(r)$ er den gjennomsnittlige presisjonen ved erindringsnivå r, N_q er antallet spørringer og $P_i(r)$ er presisjonen ved erindringsnivå r for den i-te spørringen [18].[3]

Disse metodene skal ikke vurderes nærmere her. Hensikten er bare å poengtere at det finnes vitenskapelige tilnærminger for helt konkret å måle og sammenligne kvaliteten av ulike gjenfinningsalgoritmer.

4.5.2 Alternative kvalitetsmål

Rucardi Baeza-Yates og Berthier Ribeiro-Neto nevner også alternative mål på IR-kvalitet, siden erindring og presisjon ikke alltid har vært mest hensiktsmessig å bruke [18]. En mulig tilnærming har da vært å skape en harmonisk middelverdi F mellom mellom 0 og 1 som gjengir kvalitet, hvor 0 betyr at ingen relevante dokumenter ble funnet, og 1 betyr at alle rangerte dokumenter er relevante. Dette kan skrives som

$$F(j) = \frac{2}{\frac{1}{r(j)} + \frac{1}{P(j)}} \tag{4.4}$$

hvor $r(j)$ og $P(j)$ henholdsvis er erindringsnivået og presisjonen for det j-te dokumentet i rangeringen, og $F(j)$ er den harmoniske middelverdien av $r(j)$ og $P(j)$. Denne formelen kan naturligvis omskrives for å muliggjøre at erindring og presisjon tildeles ulik vekt.

Såkalte *brukerorienterte mål* kan være av større interesse. Disse målene tar høyde for at sluttbrukere har ulike preferanser og meninger om hvilke dokumenter som er relevante. Denne problemstillingen er spesielt aktuell for virksomhetssøk. En typisk bruk av virksomhetssøk kan være å gjenfinne en bestemt mengde dokumenter B som brukeren kjenner til fra før.

Dekningsgrad er definert som andelen av relevante dokumenter *som brukeren også kjenner til* som er blitt returnert, det vil si

$$dekningsgrad \equiv \frac{|R \cup S|}{|B|} \tag{4.5}$$

med utgangspunkt i Venn-diagrammet i figur 4.5 på forrige side. Tilsvarende vil det eksistere et mål på i hvilken grad IR-systemet returnerer

[3] I boka [18] nevnes det også at interpolering må brukes for å kunne oppnå predefinerte erindringsnivåer, fordi disse nivåene kan være ulike fra de predefinerte nivåene ved gitte spørringer.

relevante dokumenter som sluttbrukeren ikke kjente til fra før, kalt *avdekningsgrad*.

Oppsummert kan følgende begreper brukes som målestokk for kvaliteten av funnene returnert fra et virksomhetssøk:

- Hvor stor andel av relevante resultater som gjenfinnes; *erindring*.
- I hvilken grad funnene er relevante; *presisjon*.
- Andelen av relevante dokumenter som brukeren kjenner til, som gjenfinnes; *dekningsgrad*.
- I hvilken grad funnene lærer brukeren nye, relevante dokumenter å kjenne; *avdekningsgrad*.

Disse kvalitetskriteriene har nokså liten betydning hvis ikke hastigheten på selve søkingen er god nok. Det virker derfor intuitivt fornuftig at hastighet må kunne si noe mer om kvaliteten til et IR-system, og i neste seksjon vises det hvordan denne typen ytelse, samt evnen til å skalere, egentlig er to sider av samme sak.

4.5.3 Systemfaktorer

I tillegg til de formelle målene for kvalitet i gjenfinning, spiller en rekke andre faktorer i det generelle IR-systemet inn for hvordan en bruker kan oppleve kvaliteten. Manning et al. argumenterer for at en del praktisk måling kan si mye om et søkesystem [75]:

- Hastighet på indekseringen (dokumenter per sekund i innmatingsprosessen)
- Hastighet på søkingen (tidsforsinkelse som en funksjon av størrelse på indekset)
- Uttrykkskraften i spørrespråket
- Hastighet ved komplekse spørringer

Antallet dokumenter i ansamlingen samt hvorvidt indekset er distribuert har også betydning for å vurdere kvaliteten til en søkemotor.

4.6 Ytelse og skalérbarhet

Høy hastighet ved håndtering av enorme datamengder har tradisjonelt vært et av de største salgsargumentene for kommersielle aktører i IR-markedet. Velkjente Google regnes for å ha verdens største indeks over

websider, mens aktører i virksomhetssegmentet, som Fast, har kunder med enda større datamengder [114]. Det synes klart at tradisjonell organisering i databaser og lignende tilnærminger vil avdekke store utfordringer med hensyn til hastighet. IR baserer seg på teknologi som i disse sammenhengene er overlegen. En naturlig forlengelse av hastighetsspørsmålet blir å forstå at skalérbarhet henger tett sammen med ytelse. Til sist nevnes det at ytelse også har betydning under indeksering, og ikke bare under selve gjenfinningen.

IR-systemer har et stort fortrinn foran alternative teknologier. Dette utsagnet baserer seg på at IR-systemer er bygget opp rundt inverterte filstrukturer, som forklart i avsnitt 4.2 på side 71. Faktisk er hastighet og ytelse den største suksessfaktoren for disse datastrukturene. Tradisjonell organisering av data, for eksempel i relasjonsdatabaser, fordrer systematiske og strukturerte spørringer. Selv om disse kan være konsist og nøyaktig formulert, vil et fritekstsøk som ikke setter spesielle beskrankninger med hensyn til omfang, kunne ta utilfredsstillende lang tid å utføre. Dette problemet er velkjent for utviklere av databasesystemer, og har faktisk resultert i at IR-teknologi i liten og begrenset skala står sentralt i slike systemer.[4] Det spesielle med IR-systemer er imidlertid at *alt* bygges opp rundt prinsippet om indeksering.

Moderne datamaskiner er relativt raske, og indeksering av tusenvis eller kanskje millioner av dokumenter presenterer derfor overkommelige problemer. For mange aktuelle bruksområder er likevel problemstillingen en helt annen. Husk at et dokument i IR-sammenheng ikke nødvendigvis representerer et dokument i virkeligheten (for eksempel en PDF-fil), men gjerne en forekomst i en database, eller en måling i et laboratorium. Det er ikke vanskelig å tenke seg situasjoner hvor det kan registreres flere millioner dokumenter hvert døgn. Videre kan det bygges systemer oppå IR-systemene som kontinuerlig analyserer og foretar nye spørringer på bakgrunn av beregninger og funn. Hyppig indeksering, voksende ansamlinger av dokumenter og store mengder spørringer medfører at én datamaskin ikke er tilstrekkelig. En helt avgjørende kvalitet ved IR-systemer er derfor at de skalerer godt. Ytelsen må kunne være konsistent når utfordringene stadig øker. Skalérbarhet kan derfor betegnes som egenskapen til å opprettholde ytelsen når antallet tjenestenoder øker i et distribuert oppsett.

Hvordan søkesystemer brukes har stor betydning for hvilke faktorer som må vektlegges med hensyn til ytelse og skalérbarhet. Dette illustreres kanskje best ved et par eksempler. Et tenkt firma besitter et stort og verdifullt arkiv som er tilgjengelig gjennom søk for ansatte. Her kan datamengdene være store, og kanskje oppdateres dokumentene ofte. Antallet spørringer er typisk beskjedent. Derfor har *indekseringsytel-*

[4] Indeksering benyttes ofte for viktige felter i databasesystemer.

se størst betydning, og dette belyses mer i neste avsnitt. I et annet tenkt eksempel er dokumentmengden relativt stabil, men et stort antall brukere, for eksempel gjennom internett, fører til at skaleringen må være god med hensyn til antallet spørringer.

De neste avsnittene forklarer litt mer konkret om hvilke hensyn som spiller inn under ytelse og hvordan ytelse kan måles.

4.6.1 Ytelse under indeksering

Selve arbeidet med å skrive til indeksstrukturen tas hånd om av det interne systemet. Flaskehalsen ligger som regel et annet sted. Det er tidligere vist hvordan indeksering omfatter en rekke trinn. Ytelsen vil derfor avhenge av hvilke steg som tas med, dokumentenes art og kompleksitet, samt graden av analyse og bearbeiding.

Et komplett søkesystem inneholder mekanismer for å hente inn, bearbeide og indeksere data. For å unngå problemer bør disse oppgavene samkjøres. For eksempel vil det være lite hensiktsmessig å hente inn dokumenter i et større tempo enn resten av prosessen klarer å ta unna. Dette kan synes trivielt, og løsningen behøver heller ikke være komplisert, men det er viktig å være klar over at indekseringsytelsen aldri vil være bedre enn det svakeste ledd. Derfor gjelder det å utvikle en fleksibel arkitektur, muligens ved bruk av skedulering og mellomlagring. I tillegg er det ikke uvanlig å la flere indekseringsprosesser kjøre parallelt ved bruk av mange datamaskiner.

Ytelse under indeksering måles som regel i antall dokumenter per sekund. Et typisk tall kan være alt fra langt under 1 til over 20 per dedikerte prosessorkjerne (2007).

Indeksering foregår både ved tilføying og endring av dokumenter, slik at indekseringsytelsen vil være sentral i systemer med stor grad av forandringer og oppdateringer.

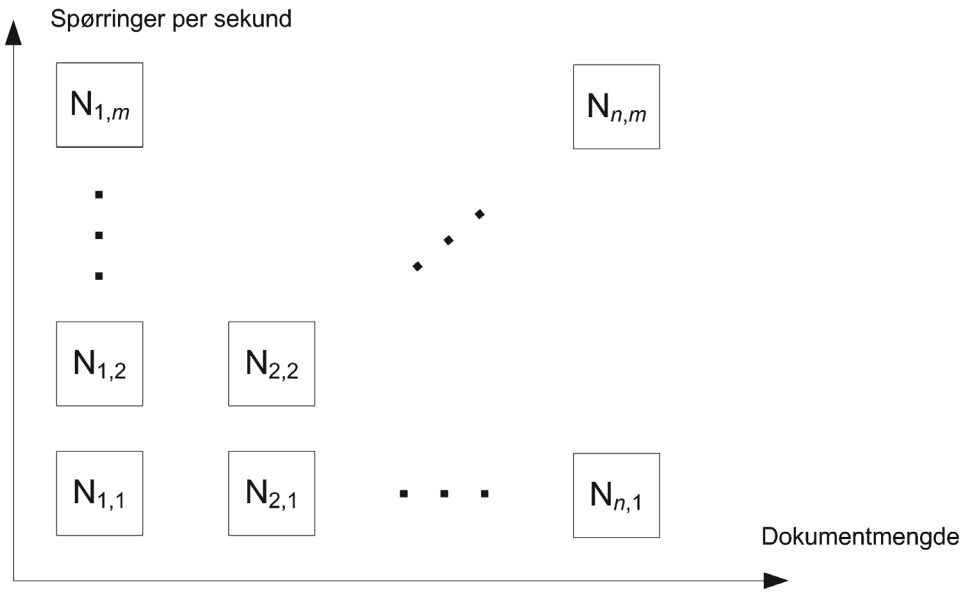
4.6.2 2-dimensjonal ytelse ved søk

Dersom ikke hele dokumentmengden I får plass på én maskin av kapasitetsårsaker, må indekset fordeles over en mengde noder N. La antallet dokumenter være $|I|$. Det antas at hver node har kapasitet til å holde C_d dokumenter. Antallet noder n som trengs er dermed gitt ved:

$$n \geq \left\lceil \frac{|D|}{C_d} \right\rceil \tag{4.6}$$

Dette kan kalles horisontal skalering.

Tilsvarende har hver maskin en maksimal grense for antallet spørringer den kan håndtere per tidsenhet, som regel angitt som spørringer



Figur 4.6: Skalering langs to akser tilpasser ytelsen til henholdsvis indeksering og søking.

per sekund. Dersom QPS (queries per second) overskrider datamaskinens spørrekapasitet C_q, må antallet noder *med samme dokumentinnhold* økes. På tilsvarende måte som ved dokumentmengde, blir antall noder m som trengs gitt ved:

$$m \geq \left\lceil \frac{\text{QPS}}{C_q} \right\rceil \tag{4.7}$$

Dette kan kalles vertikal skalering.

Følgelig blir antallet noder $|N|$ som trengs totalt gitt ved

$$|N| = n \times m + a \tag{4.8}$$

hvor a er eventuelle andre noder som er nødvendige for indeksering og øvrige oppgaver. En søkemotor må kunne skalere godt i begge disse dimensjonene. Figur 4.6 oppsummerer problematikken.[5]

En positiv bieffekt ved å legge til flere noder langs den vertikale aksen er redundans. Dersom en node $N_{j,i}$ skulle forsvinne, vil de samme dokumentene fremdeles være søkbare i nodene $N_{x \neq j,i}$ – med andre ord de resterende datamaskinene i samme "søyle" (jfr. figur 4.6).

4.6.3 Et mål på ytelse

Det er vanskelig å forbedre noe som ikke kan måles.

[5] I tillegg behøves det maskiner for indeksering og prosessering av innhold.

Oppgavene som skal løses ved hjelp av moderne IR er ofte svært omfattende med hensyn på datamengde og spørrefrekvens, og det er vist hvordan det er nødvendig å bruke flere noder i en konfigurasjon siden ingen datamaskin har tilstrekkelig kapasitet til å løse alle oppgaver alene. Et naturlig oppfølgingsspørsmål er da hvordan et gitt system oppfører seg under ulike konfigurasjoner. Dette kalles strukturert skalérbarhet. Forholdet mellom ulike konfigurasjoner kan dermed gis en faktor hvor verdien 1.0 betegner "perfekt" skalérbarhet. Men dette idealet vil neppe kunne nås siden en rekke forhold spiller inn i virkeligheten.

Bondi betegner *skalérbarhet under last* som et systems evne til grasiøs degradering av gjennomstrømning ved økt last [20]. Det vil si at systemet ikke plutselig stopper opp om antall forespørsler øker stort. Det er likevel *strukturert skalérbarhet* som er mest interessant. Et system er strukturert skalérbart dersom administrasjonskostnader ikke blir uforholdsmessig store ved økning av antallet noder. Ytelsen avhenger altså blant annet av algoritmer og datastrukturer, samt mekanismer for synkronisering. Disse blir ikke tatt opp nærmere her.

For å gi ytelsen et konkret mål har Jogalekar og Woodside definert en faktor Ψ_P for forholdet mellom to ulike konfigurasjoner for et gitt system [63]. La λ være gjennomstrømning, T være gjennomsnittlig svartid, og C være kostnad. Da er forholdet gitt ved

$$\Psi_P = \frac{\lambda_2 / C_2 T_2}{\lambda_1 / C_1 T_1} \tag{4.9}$$

slik at perfekt skalérbarhet representeres ved en faktor $\Psi_P = 1.0$. Sagt på en annen måte betyr formelen over at dersom konfigurasjon 2 gir k ganger gjennomstrømning for k ganger kostnad og en tilsvarende svartid, er skaleringen perfekt. Forfatterne viser også til ideelle forutsetninger for skalérbarhet hvor $\Psi \rightarrow 1.0$. Dette kalles ofte 100 % skalérbarhet.

Moderne søkemotorer skalerer godt, selv om en dobling av maskinkapasitet ikke gir dobling av gjennomstrømning. Tilleggskostnader oppstår fordi nodene må kommunisere med hverandre. Under utviklingen av algoritmer og datastrukturer for søkeverktøy og -motorer, er det derfor viktig at disse problemene vies mye oppmerksomhet.

Strukturert skalérbarhet i forbindelse med distribuert søk kan altså måles, og idealet er å nå en faktor på 1.0. Det synes kanskje ikke umiddelbart klart, men selv en faktor på 0.9 vil være ugunstig ved et relativt moderat antall maskiner, fordi administrasjonskostnadene blir altfor store. Fortjenesten ved å legge til flere servere vil da forsvinne helt.

Kapittel 5

Hvorfor fri programvare?

> If you want to accomplish something in the world,
> idealism is not enough – you need to choose a method
> that works to achieve the goal.
>
> — *Richard Stallman*

Det er ikke tilfeldig at dette kapitlet begynner med et sitat av Richard Stallman. Han er en av de mest ivrige forkjempere for fri programvare i historien. Enkelte vil gjerne kalle ham fanatisk, og det er nok ikke helt fjernt å betegne ham som radikal. Jeg synes allikevel det er langt mer interessant å se forbi overflaten og trekke frem nytteverdien i det han sier, altså være litt pragmatisk. Sitatet reflekterer denne vinklingen. Fri programvare kan ofte oppfattes som idealisme, men jeg ønsker heller å legge vekt *hva* fri programvare er – og ikke minst *hvordan* det kan brukes.

Jeg tør våge å påstå at fri programvare har endret seg mye i årenes løp, skjønt idealismen og grunntanken er mye av den samme. Det som har endret seg er oppfatningen og bruken, men kanskje særlig bevisstheten omkring fenomenet. I starten ble fri programvare praktisert implisitt, og det var først da programvare som industri ble mer lukrativt at bevisstheten om åpen kildekode ble mer uttalt. Videre er fri programvare etterhvert blitt stuerent. Med det mener jeg at bedrifter og organisasjoner i stadig mindre grad betrakter fenomenet som fiendtlig, og i større grad prøver å finne nytteverdien. Nå blir jo til og med selve programmeringsspråket Java fri programvare [105].

Motivasjonen for å ta med fri programvare i denne oppgaven kan virke åpenbar, siden oppgaven handler om hvordan søk i virksomheter kan løses med nettopp fri programvare. Det vesentlige blir derfor måten temaet tas opp på, og hva som får størst fokus. Jeg har valgt å legge vekt på forutsetninger. Spesielt viktig er det å vite hvordan fenomenet

fri programvare kan utnyttes best mulig. Det er ikke bare å åpne opp kildekoden. Fri programvare er også en *metode*.

Kapitlet starter derfor med en gjennomgang av karakteristikkene for fri programvare. Her nevnes opphavsrett, utviklingsmønstre og ressursforvaltning. I tillegg gis det en kort gjennomgang av historien, i håp om å tilby perspektiv. For å kunne bruke fri programvare som et verktøy, er to emner valgt ut. *Motivasjon* handler om å kunne se fordelene i fri programvare, både for enkeltpersoner og for bedrifter. *Innovasjon* handler om å utnytte disse fordelene.

I den andre halvdelen av kapitlet tar jeg en titt på de ulike søkeløsningene som finnes innen fri programvare i dag. Apache Lucene står helt sentralt, og mange prosjekter har bygd videre på dette.

Lukket eller fri programvare? Alt handler om prosjekter. Det er ikke først og fremst produktene som skiller de to formene for utvikling. Her er faget systemutvikling gjeldende uansett hva som blir valgt. Nei, det er prosessene som i første rekke er ulike. Tradisjonelle prosessrammeverk og modenhetsmodeller behøver derimot ikke avskrives.

5.1 FLOSS – tre grunnsteiner

Hva er fri programvare? For å få et overblikk kan det grovt tegnes tre overordnede prinsipper. Det første av dem handler om hvordan såkalt intellektuell eiendom håndteres, altså hvordan opphavsrett ivaretas. Det finnes mange ulike lisenser beregnet på åndsverk innenfor fri programvare. Felles for dem er at de muliggjør innsyn, og fremmer modifikasjon og åpenhet. Videre er systemutvikling preget av aktive bruker- og utviklerflater samt hyppige sykluser. Som et tredje element kjennetegnes ressursforvaltningen ved reelle behov, utveksling og flat lederstruktur. De neste avsnittene forklarer disse prinsippene mer inngående.

Lederen for Free Software Foundation (FSF) [39], Richard Stallman, har en alternativ forklaring. Han er spesielt opptatt av frihet i programvare og fremholder at det finnes fire essensielle friheter for brukeren av et program [44, 122]:

Frihet 0 er friheten til å kjøre programmet slik du selv ønsker, for ethvert formål.

Frihet 1 er friheten til å undersøke hvordan programmet virker og tilpasse det etter dine egne behov.

Frihet 2 er friheten til å kopiere programmet og distribuere det til andre når du selv ønsker.

Frihet 3 er friheten til å endre på programmet og kunne publisere det, slik at hele samfunnet kan dra fordeler av forbedringene dine.

Videre påpeker han at hvis én av disse frihetene mangler vesentlig eller ikke er tilgjengelig nok, så er programvaren proprietær. Dermed er den distribuert i et "uetisk system" og skal ikke brukes eller utvikles i det hele tatt, etter hans definisjoner [103].

Definisjoner på fri programvare kan med andre ord være mer eller mindre subjektive. I denne oppgaven blir det ikke tatt noe standpunkt i den politiske debatten. De neste avsnittene forklarer de ulike sidene ved fri programvare, men andre fremstillinger er definitivt mulig.

5.1.1 Opphavsrett og åndsverk

Håndteringen av intellektuell eiendom er sentralt for fri programvare. Hovedprinsippet er at programvare (og annen intellektuell eiendom) som utvikles alltid skal være lisensiert under en bestemt, åpen lisens. Dette avsnittet tar for seg ulike virkemidler som kan benyttes innenfor fri programvare. Det fokuseres på tre elementer:

- Åpen kildekode – all programkode skal være fritt tilgjengelig for innsyn og endring.
- Public domain – utviklerne frasier seg alle rettighetene til programvaren.
- Copyleft – utviklerne bruker egne rettigheter til å sikre fremtidig frihet for programvaren.

Nedenfor forklares disse nærmere.

Åpen kildekode er den viktigste forutsetningen. Det vanlige synet er at programkode alltid skal være fri, nemlig:

> (...) "free" as in "free speech", not as in "free beer" [46]

Dette berømte utsagnet medfører at programvare ikke nødvendigvis skal være gratis. Faktisk er det ikke så interessant om programvare koster penger. Friheten som fremholdes gir også hvem som helst anledningen til å selge programvaren videre. Dette representerer vanlig praksis, ikke minst for de ulike GNU/Linux-distribusjonene fra Red Hat [92] og Novell [81], med flere. Ifølge denne filosofien skal imidlertid all programkode alltid være åpent tilgjengelig. Dette gir hvem som helst muligheten til å kjøre programvaren, endre koden og distribuere den videre. I avsnitt 5.1.3 på side 100 forklares det hvordan dette får implikasjoner for ressursforvaltningen i åpne programvareprosjekter.

En annen filosofi er å si fra seg alle rettigheter til programvaren. I praksis betyr dette at programvaren blir "fritt vilt". Utvikleren kan aldri påberope seg noen rettigheter etter at han har sluppet koden (eller programmet, eller begge deler) til såkalt public domain. Dette muliggjør at

prosjekter som skaper lukket eller proprietære åndsverk kan inkludere hele eller deler av åndsverket, uten å måtte kreditere opphavspersonen på noen som helst måte. Public domain stiller ingen betingelser for at programvaren er åpen. For eksempel er det vanlig å gi ut ferdig kompilert programvare på denne måten. Freeware og shareware er beslektet med public domain, men kjennetegner som regel ikke fri programvare.

Et tredje prinsipp kalles Copyleft, og det bygger på å bruke lover og regler om åndsverk til å sørge for at programvare forblir åpen for alltid, og aldri kan brukes i proprietære prosjekter [47]. GNU GPL [45] er et godt eksempel på en lisens som kombinerer dette med prinsippet om frihet. I motsetning til public domain sier utvikleren ikke fra seg rettighetene sine. Linux-kjernen benytter en slik lisensiering. Hele eller deler av Linux-kjernen kan aldri benyttes i et lukket prosjekt som for eksempel Microsoft Windows. Free Software Foundation pleier å vinkle dette ved å si at de benytter de samme lover og regler om opphavsrett og åndsverk som proprietære prosjekter benytter, men til å oppnå det motsatte resultatet.

Hvilke av disse reglene som følges, og i hvilken grad, bestemmes av den valgte lisensen for åndsverket. Free Software Foundation er kjent for sin radikale holdning overfor selskaper som Microsoft. Men at man utvikler fri programvare behøver ikke bety at man har noen politisk agenda, eller ønsker å gå til krig mot den proprietære motparten. FLOSS-prosjekter kan være lønnsomme også for organisasjoner som ønsker å tjene penger. Det er nemlig flere attributter enn frihet som er fordelaktige for programvarekvaliteten. De tre prinsippene om åpen kildekode, fraskrivelse av rettigheter og beskyttelse mot utnyttelse kan kombineres på mange måter i ulike lisenser.

Trusselbildet endres

Selv om dette kapitlet handler om økt frihet, og ikke frarøving av frihet, er det verdt å nevne hva som lurer i den motsatte enden av skalaen.

Helt siden Bill Gates begynte å motarbeide hackerkulturen i 1976 [43], har programvareselskapet Microsoft representert fienden for fri programvare, især for de utadvente forkjemperne og evangelistene. Selskapet har tilnærmet monopol i flere markedssegmenter med sine proprietære og lukkede løsninger. Dette fiendebildet er i ferd med å endre seg og bli mer nyansert. Det er ikke lenger så selvfølgelig at Microsoft står alene som den store, stygge ulven, selv om de ubestridelig fortsetter å kjempe for sine egne, kommersielle interesser.

Rettighetshavere og innholdsleverandører opplever i dag at deres intellektuelle eiendom spres nærmest på lik linje med public domain. Derfor har de utviklet et nytt våpen - digital rettighetsforvaltning (DRM). Selv om mange bedrifter, som for eksempel Sun, etterhvert omfavner fri

programvare og dermed bidrar til en trend, skjer det en utvikling i den andre retningen også. DRM handler om å innskrenke forbrukeres rettigheter, og står dermed i skarp kontrast til prinsipper i fri programvare.

Det er utenfor oppgavens omfang å diskutere DRM, som forøvrig går langt utover gammeldags kopibeskyttelse. Allikevel er spørsmålet brennhett. Debatten er i sin spede begynnelse og det gjenstår å se hva som skjer på sikt. Det er ikke utenkelig at ringvirkningene for fri programvare vil bli kolossale, uansett hvilken vei det bærer i den politiske og kommersielle krigføringen.

5.1.2 Utviklingsmønstre

Når programvare er åpen, fri eller gratis, får dette konsekvenser for hvordan den kan produseres. I et tradisjonelt FLOSS-prosjekt skjer mye av utviklingen på frivillig basis i store, aktive nettverk. Dessuten utkommer nye versjoner av programvare hyppig i mange tilfeller. Modularisering og gjenbruk er også en viktig fordel for slike prosjekter. I det neste blir disse karakteristika forklart, selv om de ikke er betegnende for absolutt alle prosjekter. Hvor forrige avsnitt skisserte forutsetninger for fri programvare, er utviklingsmønstrene som betegnes her snarere typiske trekk – ikke krav.

Kanskje det mest karakteristiske utviklingsmønsteret for FLOSS-prosjekter, er bruken av store, aktive nettverk av menneskelige ressurser. Disse sitter som regel svært geografisk spredt, og består ofte av både brukere og utviklere. I mange tilfeller er overgangen mellom disse to grupperingene glidende. Åpenheten rundt selve produktet smitter som regel over på selve prosjekthåndteringen, slik at verktøy for systemutviklingen er tilgjengelig for hvem som helst. For eksempel er det vanlig at feilrapportering (bugs), prosjektutkast og design er tilgjengelig via åpne websider, og at diskusjon foregår åpenlyst gjennom e-postlister eller lignende. Derfor er det lett for sluttbrukere å bidra i utviklingen. Dette er dessuten av stor betydning for innovasjonen, noe som i senere tid har lokket mange selskaper over på FLOSS-prosjekter.

Høy deltakelse og stort antall utviklere medfører i mange tilfeller en rask utgivelsessyklus. Det er naturlig at bidragsytere verden over ønsker å se sin programkode i et fungerende produkt så raskt som mulig. Nøyaktig hvor lang tid det går mellom hver versjon i et prosjekt, er selvsagt svært varierende. Det kan handle om alt fra daglige utgivelser, til årlige eller enda sjeldnere slipp. Imidlertid er utviklerkode nesten alltid tilgjengelig, slik at en bruker kan teste ut de aller siste kodesnuttene. Store prosjekter opererer med forgreininger av versjoner, hvor et prosjekt kan være delt inn etter stabilitet og modenhet. Slik versjonshåndtering er ofte et essensielt styringsverktøy for FLOSS-prosjekter.

Programvare som er fri kan fritt benyttes av andre. Gjenbruk av kode blir mer vanlig etterhvert som stadig flere FLOSS-prosjekter startes. Mange prosjekter bygger på eksisterende komponenter og videreutvikler dem. Gitt at lisensene tillater det, er dette en attraktiv tilnærming fordi den som regel er tids- og arbeidsbesparende. Funksjonalitet som ønskes i programvare forefinnes ofte i form av åpne biblioteker. Dersom de ikke har all funksjonaliteten tilgjengelig, er det i mange tilfeller mulig å utvide funksjonaliteten selv. Slik modulbasert utvikling er ikke noe som kun foregår innenfor fri programvare, men på grunn av de store mengdene med biblioteker og moduler som er tilgjengelig, spesielt med tanke på at moduler under GNU GPL og lignende lisenser kan brukes, blir dette attraktivt og utbredt for FLOSS-prosjekter. Arbeids- og ansvarsfordelingen i prosjektnettverket passer i tillegg sammen med modularisering av kode.

I tillegg til disse generelle trekkene finnes det en del trender som gjør seg bemerket innen moderne FLOSS-utvikling. Kommersielle selskaper satser stadig oftere på fri programvare, og mange av dem støtter og bygger opp under disse trendene. Ikke minst har bruken av spesielle verktøy og moderne programmeringsspråk skutt fart. Dette er et resultat av at fri programvare og bedrifters utviklingsmetoder til en viss grad nærmer seg hverandre.

5.1.3 Ressursforvaltning

Hvordan forvaltes ressurser innenfor fri programvare? Det kommer selvfølgelig an på prosjektets art. Drives prosjektet av et firma, eller skjer utviklingen kun på frivillig basis? Hvor mange utviklere er med? Dette avsnittet setter fingeren på det som er felles, nærmest uavhengig av disse faktorene. Fellestrekkene kan være:

1. At gode ideer oppstår gjennom reelle behov.
2. At utviklerne forplikter å gi, å motta og å utveksle.
3. At visjoner, engasjement og kode forsterkes gjennom en flat lederstruktur.

Deltakersamfunnet er voksende.[1] Internettets demokratisering og fremveksten av lette systemutviklingsmetoder forsterker disse trendene for ressursforvaltning.

Punkt 1 handler om praktisk problemløsning og motivasjon. I et tradisjonelt prosjekt kan forretningsbehov eller andre forhold avgjøre hva

[1] Deltakersamfunnet henger sammen med "Web 2.0"-paradigmet. Stadig flere mennesker deltar i informasjonssamfunnets utvikling mot deltakersamfunnet. Brukerne går fra å være passive til å være aktive. Et godt eksempel på dette er Wikipedia [121].

som skal utvikles og produseres. For eksempel kan ledelsen i et firma vedta at et program skal utvikles, og ikke minst *hvordan* det utvikles. Innen fri programvare oppstår derimot ideer som en følge av reelle behov og praktisk problemløsning. Man "klør i fingrene" etter å løse et problem.

Eric Raymond meisler dette ut i essayet *The Cathedral and the Bazaar*, først innledningsvis og senere også når han snakker om fri programvare i sosial kontekst.

> "1. Every good work of software starts by scratching a developer's personal itch." [90]

> "18. To solve an interesting problem, start by finding a problem that is interesting to you." [90]

Dette punktet har også konsekvenser for innovasjon; se side 111.

Videre handler punkt 2 om at fri programvare muliggjør distribusjon og deltakelse i en skala som er vanskelig innenfor lukket utvikling. Derfor er utviklerne nærmest pliktet til å benytte seg av potensialet ved å gi fra seg kode og kunnskap, ta imot og studere eksisterende kode, og å utveksle aktivt. På denne måten vil nettverket stadig styrkes og utvides – det er iallfall hensikten.

Til sist handler punkt 3 om demokratiske styringsprosesser. Så å si alle FLOSS-prosjekter kjennetegnes ved en flat lederstruktur. Detaljer i implementasjonen, og delkrav i designet, kan med fordel bestemmes av den enkelte utvikleren. Dermed oppnår prosjektet en fleksibilitet som igjen kan få det til å virke mer meningsfylt. For at et prosjekt ikke skal sprike i mange retninger, legges det vekt på formidling av visjoner. Håpet er at felles, langsiktige mål skal bidra til at prosjektet utvikler seg i optimal retning for det reelle behovet. I tillegg vil dette engasjere utviklerne. Økt eierskap medfører som kjent forhøyet effektivitet. Fokuset er på kode – den faktiske løsningen er viktigere enn flotte spesifikasjoner og teoretiske modeller.

Det er ikke tilfeldig at smidige metoder som *Extreme Programming* (XP) og andre lette utviklingsmetoder er i vinden. Ressursforvaltningen passer godt sammen med dette. Kanskje kan alt sammenfattes i ett ord: *pragmatisk*. FLOSS-prosjekter er ofte fleksible og har et potensiale til å være effektive. Men bedrifter står også overfor klare utfordringer. Mest nærliggende er det faktum at en slik ressursforvaltning som presentert ovenfor, krever at bedrifter har en lederstruktur og arbeidskultur som passer. Hvis avgjørelsene skal ligge hos utviklerne må organisasjonene være klar over konsekvensene dette kan få. I tillegg kan det diskuteres hvorvidt enhetlig og målrettet utvikling ofres på demokratiets alter.

5.2 Kort historie

Historien om fri programvare og åpen kildekode kan fortelles på mange måter og med flere utgangspunkt. Motivasjonen for å inkludere dette i oppgaven er å bringe klarhet i hva FLOSS egentlig er, og da er det følgelig nyttig å kjenne til opphavet. Først og fremst springer FLOSS ut fra *hackerkulturen*. Den er godt beskrevet av blant andre Eric Raymond i *A Brief History of Hackerdom* [89]. Med bakgrunnen i denne kulturen oppstod Free Software Foundation (FSF) [39] med Richard Stallman i spissen. Operativsystemet GNU/Linux og programvarelisensen GPL er to høydepunkter her. Senere er åpen kildekode bragt ut til massene ved hjelp av Open Source Initiative (OSI) [82], og det er blitt mer vanlig å utvikle lukket programvare på bakgrunn av åpen kildekode. FSF og OSI havnet dermed i en konflikt.

Hackerbevegelsen

Å være en hacker har i denne sammenhengen ingenting med datakriminalitet å gjøre.[2] Ordet betegner egentlig en person som har stor interesse for, og mye kunnskap om, datamaskiner og programmering. Videre er hackere tilknyttet et bestemt miljø av fellesskap, utvikling og til dels også frihet. Til sammen dannet (og danner stadig) hackerne en *hackerkultur*, og det er denne som beskrives her.

De aller første hackerne begynte sitt virke i datamaskinens barndom ved ulike forsknings- og utdanningsinstitusjoner i USA. Mest fremtredende av disse var *Massachusetts Institute of Technology* (MIT), og mange vil si at hackerkulturen startet her ved anskaffelsen av den første PDP-1. Dette var en relativt billig datamaskin sammenlignet med IBMs stormaskiner, men fremdeles var den så dyr at datakraften måtte deles. Dette ble først gjort ved at programmer eksekverte satsvis. For å effektivisere bruken av regnekraft ble systemer for *time-sharing* utviklet. Disse systemene ble skrevet i maskinkode og kunne ikke flyttes til nye maskiner som etterhvert ble utviklet.

Ti år senere skrev Ken Thompson operativsystemet Unix (Uniplexed Information and Computing Service, Unics). Inspirert av det mislykkede Multics klarte han å lage et portabelt operativsystem, og i 1973 ble dette implementert i Dennis Ritchies nye programmeringsspråk C. Endelig var det mulig å utvikle programvare for ulike maskinarkitekturer. Bruken av datamaskiner tok av ved universiteter i USA og Europa. Meldingsnettet Usenet gjorde det mulig for hackere å dele programmer seg imellom.

[2] Bruken av ordet *hacker* med hensyn på datainnbrudd og ulovligheter beror på mediers tidlige utbredte misoppfatninger omkring terminologien. Feilen er nå så vanlig at den moderne betydningen av ordet har endret seg, vil noen si. Andre vil insistere på å bruke ordet *cracker* om en som bryter seg inn i datasystemer.

Fremdeles var det ingen som tenkte på dette som "fri programvare", selv om det var det. Men eierskapet rundt Unix – systemet var utviklet ved AT&T som eide alle rettigheter – skulle komme til å skape store problemer for dette miljøet få år senere.

Den tredje bølgen av hackere blir ofte referert til som BASIC-hackerne. Jeg føler en personlig tilknytning til denne gruppen, som begynte sitt virke med de første populære, personlige datamaskinene. Commodore 64 tilbød en BASIC-fortolker umiddelbart idet maskinen ble skrudd på, men mange andre maskiner som IBM-kloner, Amiga og Macintosh ble raskt populære. Jeg har gode minner om hvordan jeg eksperimenterte med BASIC-kode og ikke minst leste programkoden til min storebror og hans kamerater, og lærte av det. Firmaet Micro-Soft (senere Microsoft) var ekstremt tidlig ute med proprietær programvare for dette såkalte hobbymarkedet, noe de skulle tjene eventyrlig på etterhvert.

Hensikten med å forklare hackerkulturen er å gi en ide om hvordan hackerne utviklet programmer, utvekslet kode, leste andres programmer, forbedret ting de ikke likte, og så videre. For de første to-tre generasjonene av hackere var "fri programvare" en del av en livsstil. I den tredje generasjonen begynte det også å bli vanlig med såkalt freeware eller shareware. Dette er programmer som fritt kan kopieres, men som ikke er åpen kildekode. I retrospekt kan alt dette tas som tegn på mangel om bevissthet. Men verden var virkelig annerledes den gangen, og markedet for kommersiell programvare var svært mye mindre enn det er i dag. Enkelte var imidlertid i stand til å skjønne farene som lurte, ikke minst Richard Stallman.

Polarisering og frihetskamp

Free Software Foundation (FSF) ble stiftet for å bekjempe de kommersielle kreftene og forhindre utvikling av lukket programvare. Innsatsen ble for det meste konsentrert om forkynne frihet og utvikle fri programvare. De fire frihetene som ble nevnt på side 96 oppsummerer FSFs filosofi.

I kjølvannet av Unix lurte faren om at AT&T eide rettighetene til operativsystemet. Etter at firmaet havnet i rettslige konflikter ble det tvunget til oppsplitting. For å gjøre en lang historie kort; firmaet fikk nå muligheten til å selge Unix, og utover 80-tallet steg prisen til over hundre tusen amerikanske dollar [44]. Et initiativ ved *University of California, Berkeley* (UC Berkeley) om å utvikle frie programmoduler for Unix (BSD) havnet dermed i trøbbel. Unix' fremtid så mørk ut, iallfall sett med hackernes øyne.

FSF startet initiativet med å utvikle GNU (GNU's Not Unix), et operativsystem som skulle etterligne all Unix' funksjonalitet, men ikke kopiere noen programvare. Richard Stallman stod i spissen for dette samt programlisensen GNU General Public License (GPL). Den var designet for å

ivareta alle FSFs fire friheter. Til slutt manglet GNU stort sett bare kjernen, og den finske hackeren Linus Torvalds videreutviklet en kjerne som fikk navnet Linux, lisensiert under GPL. Operativsystemet GNU/Linux ble født. For første gang på mange år, eller faktisk for første gang i det hele tatt, fikk de "sanne" hackerne et helt fritt operativsystem.

Åpen kildekode når massene

Open Source Initiative (OSI) ble dannet i 1998 som et svar på kritikken mot FSFs radikale holdninger. Hensikten var først og fremst å gjøre åpen kildekode mer utbredt. Det ble utviklet en strategi for markedsføring. I spissen stod mange profilerte IT-personligheter, men Richard Stallman var ikke invitert. Strategien bar umiddelbart frukter ved at noen av industriens største aktører kastet seg ut i dansen. Oracle og IBM var blant disse, og sistnevnte støttet Apache-prosjektet.

Det betegnende for OSI var at fokus var på åpen kildekode fremfor fri programvare. I stedet for å motarbeide de proprietære aktørene, forsøker OSI å gjøre åpen kildekode tilgjengelig for alle og enhver. Apache, for eksempel, bruker en lisens som er langt mindre restriktiv sett med industriens øyne.

Free Software Foundation anså dette som et rent svik. Å bruke fri programvare i proprietære prosjekter er langt fra det som Stallman ønsker. Fri programvare garanterer nemlig åpen kildekode, men ikke omvendt. Allikevel skal OSI ha mye av æren for at åpen kildekode har blitt mye mer populært de siste årene, og ikke minst at det er blitt stuerent for de store selskapene.

FLOSS (Free/Libre Open Source Software) er et begrep som uttrykker kompromiss i konflikten mellom FSF og OSI. Programvaren skal være både fri og åpen, men ikke nødvendigvis gratis. I denne oppgaven brukes FLOSS, åpen kildekode og fri programvare litt om hverandre. Dette er bevisst. Jeg ser det ikke som hensiktsmessig å være helt presis, siden svært mange kilder og forfattere blander disse begrepene uansett. I de tilfellene hvor det er behov for utdyping, blir dette selvfølgelig gitt.

Historien om FLOSS er interessant, ikke minst fordi den gir et innblikk i spredningen av oppfatninger blant utviklere. Dessuten synes jeg det er viktig for denne oppgaven å legge et grunnlag for en moderne oppfatning av begrepet. Med det mener jeg at FLOSS kan tilnærmes pragmatisk, både av individer og av virksomheter. FLOSS har sine fordeler og sine ulemper, og ikke alle av dem er alltid gjeldende, heller. Forhåpentligvis har denne oppsummeringen hjulpet med å gi en innsikt rundt dette.

5.3 Motivasjon i fri programvare

Dette avsnittet tar en nærmere titt på motivasjonen bak fri programvare i en moderne kontekst. At den finnes, vet vi jo. Problemet er bare å forstå eksistensgrunnlaget. Hvilke motiver har personer for å gi bort programvare? Hvilke kan organisasjoner ha? Her blir noen vanlige påstander om motivasjon og fri programvare presentert, og så blir de sammenlignet med kulturell og økonomisk teori, før påstandene til sist diskuteres i lys av klare tallfakta.

Mytene rundt fri programvare og motivasjon kan til tider virke mange. I enkelte miljøer er det kanskje enighet om at "Linux er best"; i så fall bør en liten alarmklokke begynne å ringe. Jeg påstår ikke at Windows "er bedre enn" GNU/Linux som operativsystem. Men her blandes mange typer maling. Konturene av sosiale og kulturelle aspekter kan skimtes gjennom debatten. Økonomi og teknologi veves også tett sammen med dette. Ting må skilles fra hverandre. Noen av oppfatningene er mer riktige enn andre.

Etterpå gjøres det et forsøk på å identifisere hva som er kultur, og hva som er økonomi. Gjennom dette kan muligens noen av mytene revurderes. Blant annet argumenteres det for at innsats må gi en form for avkastning. Dermed kan fri programvare betraktes som rasjonelle investeringer, enten det dreier seg om bedrifter eller enkeltpersoner.

På slutten presenteres i tillegg noen klare tall. Hvem står bak fri programvare? Hva oppgir de som sine grunner til at de bidrar? Hvilke fordeler føler de selv at de oppnår? Det viser seg faktisk at tallene langt på vei gir sin støtte til diskusjonen om at fri programvare ikke er svaret på alle verdensproblemer. Allikevel kan FLOSS ha stor nytte og holde høy kvalitet i ulike sammenhenger, samtidig som det er utfordrende, kreativt og gøy.

De neste delavsnittene går altså nærmere inn på myter, diskuterer dem og kommer med klare tall. Hva betyr alt dette? Hvor er fri programvare på vei? Det er svært farlig å spekulere i dette, men det kan kanskje være interessant å se på hvorvidt våre *oppfatninger* omkring fri programvare kan endre seg over tid. Dette vil i så fall kunne endre på mye, men om utviklingen vil være positiv eller negativ er umulig å svare på. Sjansen er faktisk tilstede for at du vil få ulike svar basert på individuelle personers politiske og kulturelle syn.

Vedtatte sannheter?

Åpen kildekode kan mange ganger synes å være et myteomspunnet felt. Av og til støter jeg på "vedtatte sannheter" som det kan være vanskelig å tilbakevise på stående fot. Mange betegner åpen kildekode som en bevegelse fri for markedskrefter og hevet over vedtatte økonomiske mekanis-

mer, noen andre kaller det en underskog av talentfulle programmerere, og atter andre ser på fenomenet som en grasrotbevegelse med et uttalt budskap om frihet. Enkelte vil muligens hevde at FLOSS-bevegelsen har en politisk agenda. Litt mer kategorisert kan påstandene være at fri programvare

1. alltid og nærmest automatisk gir bedre programvarekvalitet,
2. ikke kan forklares ut fra tradisjonelle økonomiske modeller, og at
3. fri programvare er moralsk og etisk mer forsvarlig.

Det finnes elementer av sannhet i alle disse påstandene, men å ta dem for god fisk uten å kjenne nyansene er neppe så lurt. Kanskje er det til og med nødvendig å stille spørsmål ved agendaen til enkelte tilhengere av fri programvare.

Kvalitet i programvare er det første punktet. Enkelte tror at fri programvare er en automatisk garanti for kvalitet. I en over hundre sider lang og relativt kjent artikkel utbroderer Wheeler [119] om påståtte fordeler. Han konkluderer med at FLOSS har store markedsandeler, ofte er mest pålitelig, i mange tilfeller har best ytelse og ofte gir best sikkerhet. FLOSS skalerer også best i problem- og prosjektstørrelse, fremholder han. Andre fordeler som ikke kan måles går på frihet fra å bli kontrollert av ett firma, frihet fra håndtering av lisenser, at FLOSS kan implementeres gradvis i bedrifter og så videre. Wheeler står i spissen for en nærmest påfallende anstrengelse i å "bevise" noe ut fra en personlig overbevisning. Han støtter seg på mye tallmateriale og har mange gyldige poeng, men undergraver samtidig sin egen diskusjon ved å ignorere viktige nyanser. Det burde være langt mer interessant å finne ut *hvordan* fri programvare kan brukes til å øke programkvalitet.

Vesentlig mer allment akseptert er kanskje Eric Raymond. Han var svært tidlig ute med å skrive om fri programvare. Mange av ideene hans er fruktbare, og opplysningene hans kan i mange tilfeller ikke bestrides generelt (som for eksempel "A brief history of hackerdom" [89]). Likevel er det ikke til å stikke under en stol at Eric Raymond har sitt utspring fra den samme bevegelsen som han beskriver [89, 90]. Målgruppen kan derfor sies å være hackerne selv [67]. Påstandene hans går hovedsakelig ut på at FLOSS-bevegelsen rokker ved tradisjonelle økonomiske modeller. Fenomenene som inntreffer og mekanismene som gjør seg gjeldende, må heller forklares ut fra kulturelle årsakssammenhenger [91, 90]. Dette blir imidlertid en litt for stor utfordring all den tid han ikke klarer å redegjøre for kulturfenomenets eksistensgrunnlag [67, 68]. Har Eric Raymond rett? Ut fra hans eget ståsted er det liten tvil om at åpne prosjekter ofte starter ved behovet for å løse konkrete, dagligdagse problemer ("scratching an itch"). Utvilsomt har han god innsikt og blir av mange

ansett som en guru innen åpen kildekode, men det spørs om det alltid er like fruktbart å nøre opp under gammeldagse oppfatninger om fri programvare som fenomen.

Den siste kategorien av vedtatte sannheter som jeg ønsker å presentere, er ideen om at fri programvare uten videre er moralsk og etisk ønskelig. Dugnadsånd synes å være en god ting. Tanken om å lage programvare på dugnad er besnærende. Fri programvare kan være med på å skape en sosial tilhørighet, og å løse et problem sammen burde jo motvirke den onde kapitalismen, ikke sant? Det er nok ikke tilfeldig at Microsoft var, og fremdeles sikkert er, den definitive, uttalte fienden for Richard Stallman og Free Software Foundation. Således kan fri programvare sies å motvirke kapitalisme, eller i hvert fall kan utviklerne ha det som påtatt ideal. Selvfølgelig påstår jeg ikke at Stallman ikke tror på det han gjør. Han blir av mange beskrevet som en "sann troende" [122]. Men har alle tilhengerne hans samme motiver som han selv? Og motvirkes kapitalismen *egentlig* gjennom å utvikle åpen kildekode? I et litt større perspektiv kan FLOSS-bevegelsen kalles politisk; kanskje er den liberal, kanskje heller den mot venstre, kanskje dyrker den frihet. En annen ting går på det "gode". Ved å utvikle fri programvare gjør man en innsats for fattigere land og mindre utviklede økonomier, vil enkelte muligens hevde. Det spørs vel egentlig om dette holder stikk, ettersom svært mange ressurssterke personer såvel som Fortune 500-selskaper er blant de som tjener *aller mest* på fri programvare. Alt sammen koker ned til spørsmålet om altruisme. Finnes altruisme?

Programvarekvalitet, kulturelle og økonomiske fenomener, altruisme som motiv; disse tre punktene danner altså relativt vanlige oppfatninger om motivasjonen bak fri programvare. Skal vi velge fri programvare fordi GNU/Linux er best? Skal hackerne få ha et florerende, kulturelt miljø utenfor gjeldende markedskrefter? Skal vi støtte en god sak? Som de står er spørsmålene altfor overfladiske. Noen punkter er det selvfølgelig hold i, men det spørs om ikke det finnes helt andre – og langt bedre – motiver for å velge fri programvare. Dette blir klarere i løpet av de neste par avsnittene.

Ideen om altruisme slår sprekker

Aller først er det nokså fristende å tilbakevise ideen om altruisme. Om altruisme i det hele tatt finnes skal selvfølgelig ikke diskuteres her, men det gjøres forsøk på å tilbakevise at altruisme skal finnes spesielt innenfor programvareindustrien. Det enkleste er å stille seg spørsmålet helt naivt: Hvorfor skulle programvare være mer preget av altruisme enn andre industrier og fagfelt? Kulturelle og økonomiske teorier rundt fri programvare er kanskje mer konvensjonelle enn enkelte tror. Aller først blir det gjort et økonomisk poeng ut av investering og avkastning, og

videre blir det sett på kortsiktige og langsiktige økonomiske motiver for å utvikle fri programvare.

Lerner og Tirole [68] er blant de mest ansette kritikerne av Eric Raymonds forsøk på å avfeie tradisjonelle økonomiske modeller [67]. De introduserer et troverdig begrep som de kaller *netto avkastning* ("net benefit"). Kort fortalt er dette en formel som sier at netto avkastning er lik summen av kortsiktige og langsiktige fordeler, fratrukket summen av kortsiktige og langsiktige ulemper. Dermed kan FLOSS-prosjekter betraktes som investeringer hvis det er mulighet for netto avkastning. På kort sikt er kompetanseheving og personlig tilfredsstillelse typiske fordeler, mens ulempene kan være utsettelse av umiddelbare lukrative prosjekter, samt mangel på lønn.[3] På lang sikt er de såkalte signaliserende motivene viktige, det vil si motiver med hensyn på karrière (for eksempel jobbtilbud) og ego (typisk anerkjennelse i miljøer). Lerner og Tirole mener de signaliserende motivene styrkes hvis synligheten for relevante iakttakere er god, hvis arbeidsinnsatsen får direkte innvirkning på resultatet, og hvis resultatet sier mye om utviklerens talent [68]. Med andre ord kan fri programvare være en god, langsiktig investering for personer eller organisasjoner, dersom gitte forutseninger er tilstede.

På kort sikt har fri programvare en klar fordel i at lisenskostnadene kan holdes på et minimum. Ta GNU/Linux som et eksempel. Hvis skoler og universiteter tilbyr GNU/Linux av kostnadsårsaker, vil fremtidige utviklere være bedre kjent med denne plattformen når de siden skal utvikle programvare for den. Dermed blir kostnadene for utvikling under GNU/Linux lavere. Et annet moment er at kostnadene ved tilpassing og retting av feil i programvaren kan synke som et resultat av en aktiv brukerskare.

På lang sikt er motivene for å velge fri programvare litt andre. Siden kildekoden er åpen, kan alle se hvor bra jobb utviklerne gjør. Arbeidet spores tilbake til enkeltpersoner. Å synliggjøre arbeid på denne måten kan ha en skjerpende effekt på den enkelte. En annen langsiktig fordel er at økt eierskap vil medføre større personlig engasjement. Derimot vil det å utvikle sluttprodukter for vanlige brukere svekke den signaliserende effekten med hensyn på fri programvare – utviklere ønsker gjerne å høste respekt av *andre utviklere*, og ikke den allmenne bruker. Det er kanskje ikke tilfeldig at svært mange av FLOSS-prosjektene som finnes er beregnet på systemadministrasjon og avanserte verktøy.

De allment aksepterte økonomiske modellene står dermed fast; David Lancashire presenterer denne hypotesen [67] om at de danner et fundament på hvilket hackerkulturen kan utspille seg. Eric Raymond har sannsynligvis rett i at mange av fenomenene kan forklares ut fra kultur.

[3] I denne sammenhengen er det utvikleren som investerer, men de samme økonomiske reglene må også gjøre seg gjeldene hvis en organisasjon er "investor".

Men det finnes altså en litt større sammenheng. Det gjenstår å se, som Lancashire selv sier, om fremtidig litteratur vil støtte hypotesen, men han er nok definitivt inne på noe. Det ser ikke ut til å finnes noen åpenbare grunner til at blåøyd altruisme skal eksistere innenfor programvare.

Harde fakta

Det har vært gjennomført en rekke undersøkelser omkring motivasjon innenfor fri programvare. Det er lettere å presentere disse nå; med de forrige avsnittene i bakhodet burde sjansene være mindre for å misforstå tallene. Det meste av materialet som presenteres her er hentet fra BCG (Boston Consulting Group), som har gjort en grundig kartlegging basert på strukturerte intervjuer med utviklere innenfor fri programvare. Materialet er tilgjengelig som en presentasjon [112]. Lancashire presenterer imidlertid noen litt motstridende tall [67]. Selv om grunnlaget hans er litt dårligere enn BCGs, trekker han frem noen interessante poeng som kan belyse motivasjonen bak fri programvare generelt.

Hvem? Hackere har eksistert lenge, og som tidligere nevnt danner de en slags bevegelse eller et miljø hvor de kan drive med sitt. De første 20-30 årene ble spørsmålet lite diskutert. Hovedsakelig er hackerne jevnt fordelt mellom Amerika og Europa, med omtrent 10 % i resten av verden, og de er som regel profesjonelle IT-folk med erfaring. Lancashire prøver å gjøre et poeng ut av at fri programvare stimuleres av arbeidsledighet og lav lønn, men at det motarbeides av piratkopiering, og derfor beveger seg østover og sørover i verden, etterhvert som økonomien vokser i den samme retningen. Isolert sett er hackerne typisk *generasjon X*.[4]

Hvorfor? BCG-undersøkelsen ba deltakere i FLOSS-prosjekter angi sine tre viktigste grunner for at de deltok i de aktuelle prosjektene. De viktigste svarene ble som følger [112]:

1. Intellektuelt stimulerende (44.9 %)
2. Forbedrer programmeringsferdigheter (41.3 %)
3. Skaper funksjonalitet som trengs i jobbsammenheng (33.8 %)
4. Kildekode bør være åpen (33.1 %)

[4] Generasjon X betegnes i USA som de som ble født på 60- og 70-tallet, mens generasjon Y er født et tiår senere. BCG-undersøkelsen gir ingen svar på om FLOSS vil fortsette å være utbredt blant nye generasjoner. Det er også uklart om undersøkelsen virkelig antyder at generasjon X vil fortsette å være de typiske utviklerne, eller om det snarere var alderen til generasjon X som gjorde utslaget i tidspunktet for undersøkelsen.

5. Skaper funksjonalitet som trengs på fritiden (29.7 %)

6. Føler plikt til å bidra innen fri programvare (28.5 %)

Disse tallene peker i retning av at det først og fremst ligger egoistiske motiver til grunn, og dette er i tråd med ideen om at altruismen rakner. Så mye som en tredjedel svarer imidlertid at kildekode bør være åpen. Betyr dette at ideene om altruisme må revurderes, eller kan det stilles spørsmål ved motivene bak selve svaret? Et mulig motiv kan være ønsket om å utvikle FLOSS-bevegelsen generelt, slik at sjansene for å synliggjøre seg selv øker. Dette vil i så fall falle i kategorien "signaliserende motiver", som diskutert i forrige avsnitt.

Resultat. Hva er så fordelene ved å velge fri programvare, ifølge disse undersøkelsene? Det meste kommer indirekte frem gjennom motivene som utviklere og organisasjoner vitterlig har. Økt kunnskap, kreativitet og innovasjon ser ut til å være nøkkelgevinster. Dette samsvarer godt med diskusjonene om avkastning på ulike nivåer.

Avslutningsvis er det godt å kunne konstatere at undersøkelsen fra Boston Consulting Group er god overensstemmelse med diskusjonene omkring egoistiske motiver og netto avkastning. Imidlertid kan ikke slike tall brukes i videre bevisførsel om kulturell og økonomisk teori. Det gjøres derfor ingen forsøk på dette her, og disse spørsmålene vil derfor stå ubesvart ennå. Det vil dessuten være svært spennende å følge med på fremtidige undersøkelser som kan kaste lys over demografien blant utviklerne om noen år. Vil for eksempel fri programvare trekke nedover i de neste generasjonene? Vil fri programvare spre seg østover og sørover i verden, eller vil ekstrem piratkopiering i de nye økonomiene medføre at proprietær programvare stjeler ufortjent store andeler her?[5]

Avsluttende tanker om motivasjon

Denne seksjonen har vært et forsøk på å belyse motivasjon fra tre perspektiver. Først ble en rekke tradisjonelle oppfatninger presentert. Det er selvfølgelig ingen påstand at alle tror på alt, og oppfatninger blant folk er ulike. Men mange mer eller mindre "vedtatte sannheter" ble uansett belyst og delvis nyansert eller tilbakevist. Videre ble det vist hvordan teoretikere forsøker å trekke eksistensgrunnlaget for fri programvare inn på trygg grunn. En hypotese om at de samme økonomiske reglene som ellers er gjeldende i samfunnet ligger i bunnen, ble utbrodert. Til sist ble det vist til noen tall fra en undersøkelse om motivasjon innen fri programvare, og kort diskutert hvordan disse nummerne stiller seg i

[5]Med "nye økonomier" tenker jeg på land og markeder som er på vei oppover, i skrivende stund for eksempel Kina og India.

forhold til avsnittene foran. Men hva er bakgrunnen for å ta opp motivasjonsspørsmålet i denne oppgaven?

En viktig årsak er kunnskap om suksessfaktorer i et FLOSS-prosjekt. I denne oppgaven kommer det frem at fri programvare har et godt stykke å gå i forhold til virksomhetssøk. Hvis det skal være håp om å utvikle en komplett søkeplattform, er det interessant å kjenne til hva som motiverer utviklere. For eksempel kan det være verdifullt å tenke på at signaliseringen kan styrkes hvis programvare utvikles for utviklere. I denne sammenhengen dukker det også opp spørsmål om det rett og slett finnes nok utviklere med kunnskap om virksomhetssøk, til at signaliserende motiver får sin kraft.

Vel så nyttig er det å vite *hvem* som driver med fri programvare – at det stort sett dreier seg om erfarne programmerere med fast jobb. Dette viser at FLOSS passer godt i en kommersiell "setting". Samtidig understreker det at realisme er nødvendig. Utviklere skal ha svært sterk alternativ motivasjon dersom de ikke får mat på bordet for jobben.

Neste seksjon tar for seg innovasjon i fri programvare. Dette springer på mange måter ut av motivasjonen som diskutert her. Motivasjon og innovasjon innenfor FLOSS vil til sammen kunne legge et godt grunnlag for diskusjonen senere i oppgaven.

5.4 Innovasjon

Hensikten med å ta med innovasjon i denne oppgaven, er å vise at fri programvare kan bidra på andre, nye og positive måter til et tenkt FLOSS-prosjekt innenfor virksomhetssøk.

Faget informatikk avhenger av innovasjon. Men ikke minst er innovasjon viktig for *IT-bransjen* som må holde seg konkurransedyktig. En suksessrik parring av fri programvare og søketeknologi kan vise seg å avhenge nettopp av innovasjon. Nytenkningen her foregår på en annen måte enn ved tradisjonell, lukket utvikling. Denne erkjennelsen kan få konsekvenser for metodevalg. Motivasjonen bak å ta opp innovasjon er derfor tredelt.

Først må det muligens erkjennes at Enterprise Search (ES), i den grad det skal utvikles en komplett og fri plattform, synes å være upløyd mark for typiske hackere. Poenget er at produktet som tenkes utviklet, representerer et relativt nytt eller ukjent problem innen fri programvare.

Innovasjon foregår ikke på samme måte i lukket, tradisjonell utvikling som innenfor fri programvare. Tradisjonelt er det ønskelig å bevare og beskytte innovasjon gjennom hemmelighold og patenter. Prinsippet for fri programvare er helt motsatt, og dette er mulig ved hjelp av såkalte horisontale innovasjonsnett.

Samtidig vil konsekvensene av innovasjonsformen være store, slik at

systemutviklingen må utføres med hensyn på dettte. Metoden i denne oppgaven vil således påvirkes av denne erkjennelsen.

Hvilken innovasjonsform er best? Det gjøres ikke forsøk på å gå nærmere inn på å sammenligne innovasjon som sådan, samt kvaliteten innenfor fri programvare kontra lukket utvikling. Det er heller ikke en aktuell problemstilling nå. Målet i denne omgang bør være å identifisere flest mulig forhold for å legge et brukbart teoretisk fundament.

Åpen innovasjon

Eric von Hippel trekker frem to hovedpoeng bak motivasjon i fri programvare [116]. Det første er behovet for å kunne skape noe. Det andre er å være villig til å dele det man skaper med andre, vederlagsfritt. På grunnlag av dette fremholder han fem elementer som kjennetegner innovasjon i fri programvare. (De følgende fem punktene tar utgangspunkt i dette, men er videreutviklet på egenhånd.)

1. Utbredelse og bruk stimulerer til innovasjon. Når programvaren er åpen og fri, blir det enkelt å ta den i bruk for andre. Distribusjon er lett, og den potensielle diffusjonen blant brukere – dersom den slår til – medfører personlig tilfredsstillelse i at man skaper en nytte og ser resultater i form av volum.

2. Lett tilgang på “sticky” informasjon; innovasjoner og ideer har en tendens til å forbli på det stedet de oppstod, eller iallfall være mer tilgjengelig der. Dette kan forklares ved at en utvikler eksempelvis skifter jobb mellom to firmaer som utvikler lukket programvare. I det nye selskapet kan vedkommende i så fall ikke bygge på tidligere resultater. Men dette er ikke tilfellet for fri programvare. Her kan innovasjon bygge på tidligere resultater som ikke nødvendigvis kommer innenfra en tenkt organisasjon som utvikler fri programvare.

3. Selv om utvikleren gir ut kildekoden, beholder vedkommende kanskje helt essensielle biter av kunnskap selv. Dermed kan utvikleren også oppnå en slags kultstatus. Dette kan medføre en form for økonomisk beskyttelse, og slår litt hull i påstanden om at all fri programvare egentlig er helt utsatt for alles øyne. Åpen kildekode er nemlig verdiløs uten kunnskapsgrunnlaget for å utnytte den.

4. Fabrikanter og produsenter mangler oversikt. I fri programvare er det ofte avanserte brukere eller utviklere som gir den mest verdifulle tilbakemeldingen til prosjekter. Disse “elitebrukerne” som står på kanten mellom prosjektet og markedet, kan vise seg å være verdifulle motorer for å fremme innovasjonen.

5. En motivasjon for å røpe innovasjoner, spesielt for ansatte i vanlige bedrifter, er at koden gjerne ville ha blitt gjort ulovlig tilgjengelig av andre uansett. Dette synes kanskje som et noe søkt argument, men det er ikke sjelden at utro tjenere på innsiden av murene torpederer innovasjon. Hippel [116] mener at det nærmest er umulig å holde et produkt hemmelig. Det er jeg ikke helt enig i. Uansett har han et godt argument i at denne risikoen kan tilintetgjøres ved at programkoden gjøres fritt tilgjengelig først som sist.

Mange av disse punktene henger sammen med motivasjon. Det er ganske naturlig at motivasjon for individuelle utviklere, og for firmaer, ofte kan fremme innovasjon. På den andre siden kan innovasjon sies å være en viktig motivasjonsfaktor.

Horisontale nett og Brooks' lov

I *The Mythical Man-Month* [22] skriver Frederick P. Brooks blant annet om antallet utviklere på et prosjekt, og tiden det tar å utføre prosjektet. Hvis et prosjekt kun har én utvikler, brukes ingen tid på kommunikasjon. For hver person som som legges til prosjektet øker tiden som trengs til kommunikasjon, og dette kan gjengis i en formel som jeg ikke skal presentere her.

Fri programvare kjennetegnes av såkalte horisontale prosjektnett. Ofte er tallet på bidragsytere og aktive utviklere svært høyt. Flere hundre personer står nevnt i Linux' kildekode som sentrale utviklere [44]. Åpenbart ser allikevel denne modellen ut til å fungere relativt godt. Kommer den ikke i konflikt med Brooks' betraktninger?

En viktig erkjennelse her er at alle ikke nødvendigvis trenger å vite alt. Det er svakheten ved Brooks' modell. I fri programvare er dette kjørt ut til det ekstreme ved at svært mange utviklere overhodet ikke aner hva de fleste andre driver på med. Dette er mulig ved hjelp av oppsplitting og modularisering, sentral styring, bruken av utviklingsverktøy og lignende. Dessuten er ikke antallet utviklere forbundet med kostnader.

Brooks' lov sier videre: "Adding manpower to a late software project makes it later" [22]. Jeg skal ikke bestride riktigheten i dette utsagnet generelt, men mye av faren forsvinner i åpne prosjektnett. For et gitt FLOSS-prosjekt vil det altså virke fordelaktig å opprette et godt prosjektnett, men det kan være vanskelig å ta styring over denne prosessen.

Å utvikle noe i åpen kildekode er ingen garanti for et sunt FLOSS-prosjekt. Som nevnt tidligere er FLOSS mer enn åpen kildekode.

Tradisjonell innovasjon

I forhold til lukket programvare har fri programvare en del ulemper. Relatert til innovasjon kan dette dreie seg om et par ting. Konfidensialitet kan være fordelaktig fordi det stimulerer til økonomisk gevinst. Det samme gjør patenter. Mye av hensikten med patentsystemet er jo ganske enkelt å bedre innovasjon ved å stimulere motivasjon.

Lønn kan også nevnes som en motivasjonsfaktor. Dette er ikke direkte relatert til forskjellene mellom fri og lukket programvare, siden det ikke er noe i veien for å lønne utviklere som arbeider med FLOSS. Det samme gjelder styring og kontroll av prosjekter. Selv om det kan være en utfordring å forhindre forgreininger av prosjekter, er det i utgangspunktet ikke noe i veien for å ha en organisert form for utvikling. Imidlertid bør dette skje på premisser om at det karakteristiske ved FLOSS blir utnyttet. Horisontale prosjektnett er en del av denne særegenheten.

5.5 Apache Lucene

Utviklingen av IR-biblioteket Lucene [9] begynte i 1997 ved at Doug Cutting skulle lære språket Java. Som så mange andre FLOSS-prosjekter [90], oppstod Lucene ved at utvikleren ville løse et konkret og personlig problem [49]. Tre år senere la han Lucene ut på SourceForge, før Apache innlemmet det i sin portefølje et års tid senere. I dag vedlikeholdes Lucene av et aktivt miljø.

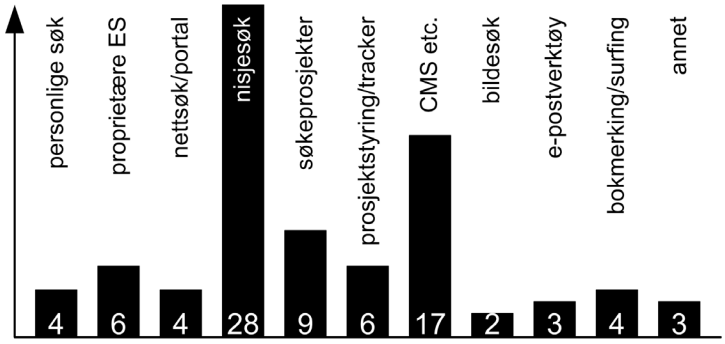
En av kjepphestene til Lucene er at det skal være enkelt, og fokuset ligger derfor på å tilby et API på lavt nivå, og så får det bli opp til utviklere av andre prosjekter å lage noe mer. Lucene rendyrker en forsiktig og gjennomtenkt avdekning av finessene i IR-biblioteket, og dette er kanskje en medvirkende årsak til utbredelsen Lucene har fått i dag.

Som utviklingsprosjekt er Lucene aktivt og sunt. Skaperen Doug Cutting har for lengst overlatt programmeringen til ivrige etterkommere. Nye versjoner kommer med jevne mellomrom. Aktiviteten på e-postlistene er stor, med hundrevis av meldinger hver uke. Alt i alt ser det horisontale prosjektnettet ut til å fungere som det bør. Lucene er helt klart et moderne FLOSS-prosjekt, og utvikles i varierende grad med enhetstesting og automatisk bygging.

5.5.1 Utbredelse og bruk

Lucene er i dag svært populært, relativt sett. Ingen andre IR-biblioteker basert på åpen kildekode er i nærheten av Lucene.[6] Biblioteket brukes

[6] Jeg er nødt til å ta et visst forbehold her, siden jeg ikke har lykkes i å få tak i noen vitenskapelig eller pålitelig undersøkelse med hensyn til Lucenes utbredelse.



Figur 5.1: Innrapporterte prosjekter basert på Apache Lucene

i en god mengde frie prosjekter, men ikke minst er det populært som motor i proprietær eller lukket programvare.

Ifølge skaperen Doug Cutting [49] brukes Lucene av en rekke *Fortune 100*-selskaper og er utbredt i kommersielle rapporteringssystemer. Til og med Microsoft bruker Lucene til søking i e-post, ifølge Cutting. Andre kjente brukere av Lucene er IDE-miljøet Eclipse [35] (som bruker Lucene til å søke i hjelpefunksjonen), prosjektverktøyet JIRA [62], nettleksikonet Wikipedia [121] og selveste SourceForge [101]. Er dette typiske prosjekter?

Det har ikke vært mulig å få tak i noe pålitelig materiale over bruken av Lucene, men en titt på nettsiden til Lucene kan gi noen hint. Her finnes det en liste over innrapporterte prosjekter som bruker biblioteket. Diagrammet i figur 5.1 er basert på tall hentet den 3. mars 2007, løst fordelt etter kategorier. I tillegg er 105 nettsider listet opp. Innrapporterte tall er selvfølgelig upålitelige, og grunnlaget representerer ikke noe godt utvalg av brukermassen. Først og fremst kan det spekuleres i at underrapporteringen antakeligvis er kolossal. Antallet prosjekter som bruker Lucene er nok mange ganger høyere enn denne oversikten gir inntrykk av.

Den største gruppa med unntak av nettsider, “nisjesøk”, representerer spesialiserte søkemotorer på nettet. Disse fokuserer enten på innhold eller målgruppe, eller begge deler. Noen av dem legger vekt på utradisjonelle grensesnitt, mens andre igjen gir tilgang til søk i dype nett, altså deler av nettet som vanligvis er lukket for anonyme brukere. Kategorien “CMS etc.” står for blogger, publikasjonsverktøy, diskusjonsfora, ECM (Enterprise Content Management) og lignende. “Søkeprosjekter” er søkemotorer som andre kan laste ned og implementere, mens “nettsøk/portal” betyr implementasjoner av generelle nettsøk. En interessant observasjon er at Lucene brukes i prosjekter som er små i stør-

Dog peker alle tegn i retningen av at Lucene er i en klasse for seg selv, og det virker rimelig å uttrykke dette så sterkt som det gjøres.

relse og krever høy ytelse, som eksempelvis nettlesertillegg.

Konklusjonen er at det er vanskelig å si mer konkret av Lucenes utbredelse. IR-biblioteket benyttes åpenbart av mange, til både store og små datamengder, med mye og lite trafikk, og i tunge og typisk lette prosjekter. Kan det være at det er denne allsidigheten som gjør Lucene så populært? Den fleksible Apache-lisensen begrenser i hvert fall ikke bruken.

5.5.2 Varianter

Lucene Java [8] er egentlig et underprosjekt av Lucene [9]. Sistnevnte er et generelt prosjekt som spesifiserer en søkemotor. Av denne grunnen har det vært mulig å implementere Lucene i en rekke andre språk enn Java.

Den viktigste av disse er Lucy, en implementasjon i språket C med bindinger for Ruby og Perl. Ellers finnes det implementasjoner i C++, .NET, Objective-C, Python, Zend (PHP), Perl, Delphy, Ruby og flere. Alle variantene kan i utgangspunktet lese og skrive til hverandres indekser, så lenge implementasjonen er av samme versjon.

I denne oppgaven brukes begrepene Lucene Java og Lucene om hverandre.

5.5.3 Betraktninger

Hva mangler for at Lucene skal bli en komplett søkemotor? Spørsmålet er feil. Lucene er ingen søkemotor. Det er bare et IR-bibliotek som tar seg av skriving og lesing fra et indeks, og en søkemotor må derfor bygges rundt biblioteket. Lucene inneholder også en del analyseklasser og språkverktøy. Men siden fokuset er på enkelhet, ytelse og rendyrking av konsept, tas slike finesser kun med i biblioteket dersom de er av allmenn interesse. Så hvor godt gjør egentlig Lucene jobben sin?

Lucene indekserer og søker ren tekst, og ifølge hjemmesiden er ytelsen på dette svært god [73] (skjønt dette er ikke uavhengige, vitenskapelige målinger). Indekseringsytelsen skal visstnok være målt til 20 MB/s på en middelmådig PC. Kravene til RAM er visstnok små, og kompresjonen av indeksert tekst er god. Men hva betyr egentlig dette? Det er viktigere å se på hva Lucene brukes til. Den brukes både av små verktøy internt, og til indeksering av hele weben, så dette bærer vitne om beskjedne krav og god skalering.

Hva er negativt? Lucene indekserer bare ren tekst, i motsetning til for eksempel Fast ESP. En annen svakhet er at selv om Lucene støtter spørringer av intervaller, noe som kan brukes i aspektorientert leting, så

er dette visstnok en del treigere enn hos Fast.[7] Lucene støtter heller ikke søking på alle indekserte felter samtidig.

En signifikant ulempe med Lucene er at det ikke overholdet ACID, slik at alle transaksjoner må programmeres utenfor biblioteket – med alle ulempene det kan medføre. Videre støtter Lucene samtidig indeksering og søking. Dette kan føre til inkonsistente resultater og "skitten lesing" (se side 55). Videre har Lucene fått kritikk for lav ytelse på oppdatering av dokumenter som allerede ligger i indekset.

Oppsummert er det klart at Lucene er et knøttlite og fleksibelt bibliotek, implementert 100 % i ren Java. Funksjonaliteten er mager. Dette er både en styrke og en svakhet. Ytelsen blir god, men om man ønsker støtte for mer funksjonalitet må dette legges rundt. Lucene er et IR-bibliotek på svært lavt API-nivå. Mange vil kunne finne det lite hensiktsmessig å arbeide på så lavt nivå under utvikling av applikasjoner. Dette vises også igjen på antallet prosjekter som pakker inn Lucene i litt "mykere papir", som for eksempel Solr [11].

5.6 Andre søkeverktøy

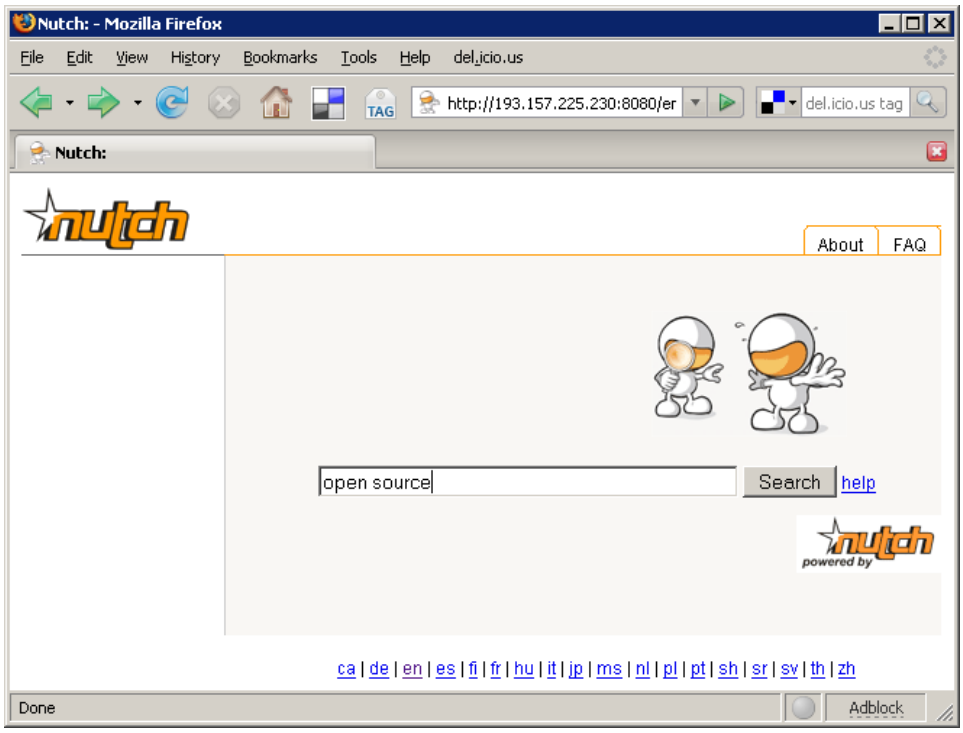
Det aller meste som finnes av søkeverktøy er basert på Apache Lucene. De viktigste av disse ser ut til å være henholdsvis Nutch [10] og Solr [11]. LIUS [70] og regain [94] er to interessante, men relativt ukjente prosjekter. Ellers så Red Piranha [93], LARM [5], karakoram [65] og Zilverline [125] ut til å være lovende, men av ulike grunner er de mindre aktuelle. Enkelte komponenter av dem kan imidlertid brukes, og det er verdt å studere prosjektene for å få ideer og inspirasjon til arkitektur og sammensetning.

5.6.1 Apache Nutch – søkemotor for web

Nutch [10] er en komplett søkemotor for nettet. Den bygger på Lucene og er følgelig skrevet helt i Java. Allikevel er ikke Nutch en applikasjon, selv om mye funksjonalitet er bygget inn og en rekke operasjoner kan gjøres fra kommandolinjen. Nutch krever en omsluttende applikasjon på samme måte som Lucene gjør det, men bygger altså ett steg videre. Kort sagt er Nutch det samme som Lucene, med følgende tillegg:

- Crawler
- Lenkeanalyse/-database (se side 78)
- Identifisering av filformat

[7] Jeg har dessverre ingen offentlig tilgjengelige kilder med hensyn på ytelsessammenligning av Fast ESP og Lucene.



Figur 5.2: Apache Nutch: Standard søkegrensesnitt

- Analysering av HTML (flere filformater via tillegg)
- Identifisering av språk og tegnsett, samt noe prosessering

I tillegg tilbyr Nutch en utvidelse av Lucenes funksjonalitet for indeksering og søking.

Et delmål med Nutch (sammen med Lucene) er å gjøre nettsøk mer transparent. Initiativtakerne synes at søketeknologi er altfor viktig til at ett eller flere store selskaper skal sitte på hemmelige algoritmer. Utover dette har Nutch lite sentral styring, og det ligner derfor på en del andre FLOSS-prosjekter (se seksjon 5.1.3 på side 100).

Nutch er et offisielt underprosjekt av Lucene, og har eksistert en god stund. Mange nettsider har implementert søkemotoren. E-postlistene er aktive og mange er med i utviklingen. Uten at det er foretatt noen konkret sammenligning, ser Nutch ut til å være det mest aktive delprosjektet av Lucene.

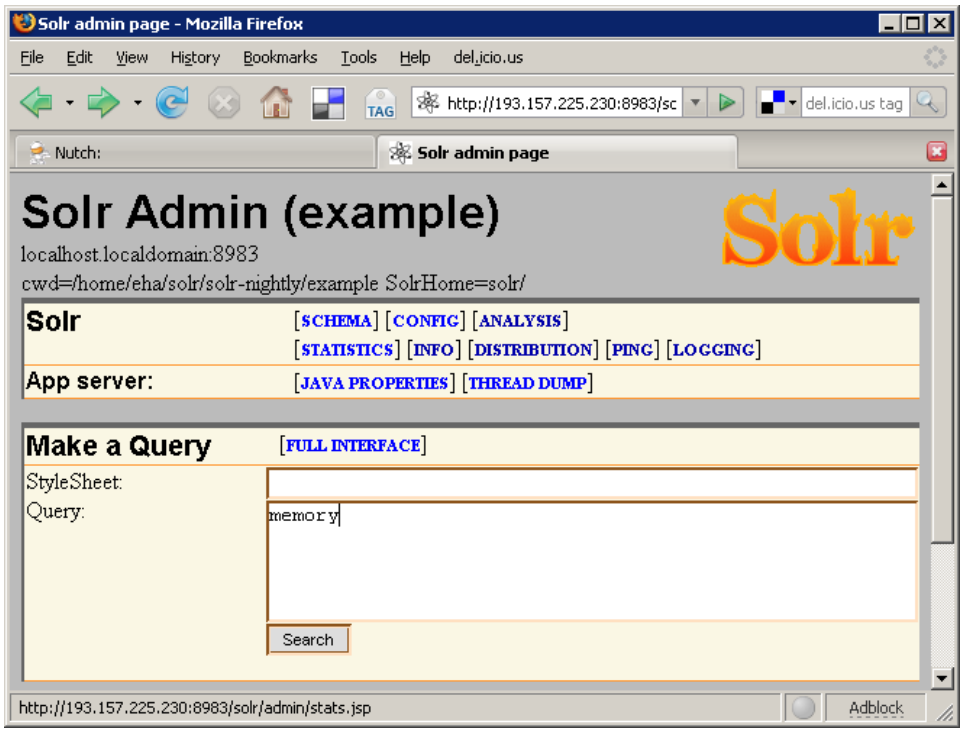
Nutch har en plugin-arkitektur, noe som gjør det enklere å lage tillegg for søkemotoren. Dette har resultert i blant annet

- støtte for mange filformater,
- delansamlinger (sub collections),
- alternative scoring-algoritmer (OPIC, som foretar en online rangering, i motsetning til vanlig lenkeanalyse),
- en rekke ulike URL-filtre (prefix, suffix, regexp, automatisering),
- gjenkjennelse av språk (n-gram-algoritmen med flere), og
- stemming (Porter-stemming [86] for engelsk, samt Snowball-implementasjoner [99] for andre språk).

Imidlertid er en del av Nutch' design at skjemaet er relativt fast. Dette, og en del andre designvalg, gjør at Nutch kanskje ikke er så fleksibelt som mange skulle ha ønsket.

Apache Hadoop

For å distribuere mellom hundrevis eller tusenvis av servere, fant Google opp en algoritme som kalles *map/reduce*. Selve algoritmen ligger utenfor omfanget av denne oppgaven, men det er nyttig å vite at Nutch har sin egen implementasjon av denne. Prosjektet Apache Hadoop (tidligere en del av selve Nutch-prosjektet) gjør at Nutch enkelt kan brukes i samme skala som de aller største søkemotorene. I tillegg inneholder rammeverket Hadoop et distribuert filsystem.



Figur 5.3: Apache Solr: Administratorverktøy

5.6.2 Apache Solr – søk i strukturerte data

Apache Solr [11] er muligens det produktet som fra før ligger nærmest en løsning for problemstillingen i denne oppgaven. Solr er basert på Lucene og tar ifølge prosjekthjemmesiden sikte på å bli en komplett programvareplattform for virksomhetssøk. Selv om Solr er et svært lovende prosjekt, er dette en sannhet med store modifikasjoner. Solr begynte som en løsning internt i et firma, og i januar 2006 ble all koden for dette prosjektet donert til Apache Software Foundation. Med andre ord er Solr så nytt at det ble offentliggjort etter at denne masteroppgaven ble planlagt.

Først og fremst er Solr en innpakking av Lucene. For å bruke sistnevnte må man gjøre kall i et programmeringsspråk (vanligvis Java). Med Solr er dette endret til at alle kall foregår via HTTP i XML, selv om det fremdeles er mulig å gjøre kall via Java. Derfor har Solr hovedsakelig to grensesnitt mot verden:

- Skrive (opprette, endre, slette)
- Lese (søke)

I tillegg foregår konfigurasjonen av Solr i XML-filer. Mest nevneverdig er kanskje at Solr tilbyr å konfigurere indeksskjemaet på denne måten. Dette burde i utgangspunktet danne et bedre utgangspunkt for bruk i virksomheter, siden tilpassing av skjema er mer eller mindre nødvendig her. I tillegg tilbyr Solr et enkelt grensesnitt for administrering via web.

Solr utvider i tillegg en del nyttige finesser, ikke minst aspektorienterte søk. Dette er særlig aktuelt for virksomhetssøk (se seksjon 3.2 på side 51). Videre tilbyr Solr bedre støtte for mellomlagring (caching), samt replikering av indekser.

Selv om det er lett å sette opp Solr, er det på ingen måte en komplett plattform for virksomhetssøk. Iallfall kan det ikke fungere rett ut av boksen på linje med kommersielle produkter. For eksempel støtter det ikke ulike filformater, lenkeanalyse eller avansert behandling av dokumentene under innmatingen. Dette kommer av at Solr tar sikte på å gjøre alt i XML, og det er dermed opp til brukeren eller virksomheten å bearbeide sine data slik de ønsker. Solr har derfor heller ingen bindepunkter mot aktuelle kildesystemer som filsystemer, FTP, HTTP og så videre. Det ligger kanskje en liten erkjennelse i at dette er noe av naturen i fri programvare. Prosjekter tar ofte sikte på å gjøre én ting svært godt, i stedet for å tilby alle mulige slags finesser. Hvilke konsekvenser dette har blir tatt nærmere opp i diskusjonskapitlet.

Bruken av XML for både indeksering og søk åpner for spørsmål omkring ytelse. I uformelle tester ser det ut til at Solr klarer seg bra på indekseringssiden, men jeg har ikke klart å finne noen pålitelige kilder.

Det later også til å være et ubesvart spørsmål i hvilken grad ytelse er et problem på spørresiden. Etter å ha fulgt med på e-postlister i mange måneder, virker det ikke som dette er et stort problem, men det er selvfølgelig ikke lett å vite disse tingene uten å undersøke dem nærmere. Dessverre faller det litt utenfor problemstillingen for oppgaven.

Jeg nevnte såvidt e-postlister, og Solr er et svært aktivt prosjekt med en stor bruker-/utviklerbase (husk at skillet mellom brukere og utviklere ofte ikke er så tydelig innenfor fri programvare). I løpet av kort tid har det klart å oppnå status som et offisielt delprosjekt av Lucene (januar 2007). Solr ser ut til å bli mye brukt innenfor distribuerte løsninger med mange ulike kildesystemer, blant annet ved hjelp av tjenesteorientert arkitektur (Service Oriented Architecture – SOA).

Det at Solr bygger på Lucene gjør det relativt enkelt å dra nytte av funksjonalitet herfra. De konfigurerbare skjemaene er et eksempel. I egentlig forstand er dette kun en forlenget arm fra funksjonalitet som er innebygget i Lucene. Indeksene til Lucene støtter nemlig dynamisk opprettelse av felter under kjøring. Poenget er at mens Nutch i utgangspunktet setter krav til hva pakken kan brukes til, bygger Solr videre på prinsippene om at Lucene skal være fleksibelt, og det rendyrker på mange måter disse prinsippene.

Solr har allikevel et godt stykke igjen å gå før det rendyrker alt, også med hensyn til brukervennlighet (se seksjon 3.1.3 på side 49). Som Nutch kjører Solr på en hvilken som helst applikasjonsserver, for eksempel Tomcat [13].

5.6.3 Interessante søkeprosjekter

Red Piranha

Dette produktet virker i utgangspunktet lovende. Det tar sikte på å være en komplett søkeplattform for virksomhetssøk [93]. I tillegg bruker det XML-RDF for å oppnå det utviklerne hevder er evnen til å lære. Her må klinten skilles fra hveten. XML-RDF er først og fremst en spesifikasjon fra W3C [117] som hjelper å strukturere XML-dokumenter på en standardisert måte. Sentralt i RDF står et "subjekt-predikat-objekt"-tuppel som tilfører semantikk. Ifølge Ma et al. [76] brukes RDF mye i store organisasjoner, men med suboptimal ytelse. Funnene deres viser at fulltekstindeksering av RDF-tripler gir vesentlig økt ytelse for store volum, men det synes uklart hvordan Red Piranha tar i bruk RDF.

Bakgrunnen er at Red Piranha viser seg å være et proprietært prosjekt. Det finnes riktignok en "community edition" hvor kildekoden er fritt tilgjengelig på SourceForge. Denne virker imidlertid gammel og utdatert, og på hjemmesiden for det kommersielle produktet fremstår den

som en prøveutgave. Utviklerne har tydeligvis fått øynene opp for markedspotensialet i produktet.

I tillegg er mengden av offentlig dokumentasjon begrenset, og derfor er det vanskelig å basere noe som helst på prosjektet. Red Piranha er forøvrig, som mange andre søkeløsninger, basert på Apache Lucene. Det er bygget ved hjelp av rammeverket Spring [102].

LARM

Jakarta LARM Project [5] er nok et produkt basert på Apache Lucene, og et godt eksempel på at det kan være vanskelig å ta seg frem i jungelen av FLOSS-programvare.[8] Ifølge hjemmesiden er LARM en 100 % Java-basert søkeløsning for sluttbrukere. Videre sies det at den inneholder metoder for indeksering av filer, databasetabeller og en crawler for å indeksere nettsider.

I virkeligheten er LARM kun et påbegynt prosjekt. Prosjektsiden på SourceForge har ikke vært oppdatert siden sommeren 2003. Selv om det ifølge e-postlistene er blitt gjort en begrenset innsats etter denne tiden, later det til at prosjektet har stagnert fullstendig. En av administratorene for prosjektet er Otis Gospodnetic, en sentral skikkelse i Lucene-miljøet. Otis har uttalt at han ikke har tid til LARM-prosjektet, og det kan virke som om Solr og Lucene får mye av oppmerksomheten i stedet.

Prosjektet LARM er allikevel interessant. Selve utgangspunktet var å bli en ren innsamler av data, altså programvare med funksjoner for crawling, databaseutvinning, filtraverserer, Web Services (SOAP) og annet. Hadde prosjektet vært fullført, kunne det ha dannet et godt utgangspunkt for "venstresiden" i et prosjekt for virksomhetssøk (se figur 2.1 på side 22).

Kildekoden røper at svært lite er implementert. Ddokumentasjonen er sparsom, dog informativ. På innsamlingssiden er det kun crawleren som er implementert. Denne velger ut data på bakgrunn av nyttige felter for nettsider. Mer konkret dreier det seg om:

- Initiell URL (adressen hvor forespørselen startet)
- Endelig URL (ved omdirigering)
- Metode for forespørsel
- MD5-kontrollsum
- HTTP-status (for eksempel `200 OK`)
- Koding (*encoding*)

[8] LARM står for *Lucene Advanced Retrieval Machine*.

- Sist endret
- Topptekst (*headers*)

I tillegg kommer selvfølgelig selve innholdet. Bakgrunnen for å dokumentere disse feltene er å vise hva som kan være aktuelt for en crawler å hente ut fra en HTML-side. På den andre siden er det et tankekors at feltene er fastlåst i kode. Kanskje vel så interessant er det at LARM tar i bruk prinsippet med *processing pipeline*, noe som kan minne om Fast ESP. I tillegg legger arkitekturen opp til en styringsnode som tar seg av de ulike "rørledningene" og deres datakilder, samtidig som crawleren i seg selv er designet for god asynkron operasjon.

Hva kan LARM brukes til? Dagens Nutch er langt mer avansert enn LARM, for ikke å snakke om at sistnevnte er utdatert. Det kan allikevel være fornuftig å ta med seg noen av ideene fra prosjektet, for eksempel kodestrukturen og arkitekturen. Når det er sagt er mye videreført i form av at utviklerne nå arbeider på andre prosjekter.

LIUS

LIUS står for Lucene Index Update and Search [70]. Det er et overbygg på Apache Lucene, og har mange likhetstrekk med Apache Solr – ikke minst er det open source. LIUS kan tolke mange ulike filformater, og det er forholdsvis enkelt å legge til nye pakker. På mange måter ser det ut til å ligge litt bak Solr, kanskje mest fordi Solr er et offisielt støttet prosjekt med en større brukerbase. Og selv om man kommer relativt raskt i gang med LIUS "ut av boksen", skal det noe til å konfigurere det riktig. Det krever minst like mye påpakking som Solr, og har en relativt rotete arkitektur til sammenligning. Mye av koden er dokumentert på fransk, og i det store og det hele virker prosjektet litt mindre planlagt og styrt.

LIUS støtter oppsett av skjema i XML-filer på en måte ikke helt ulikt Solr. Konfigurasjonen er enkel, og kanskje er den litt *for* enkel. En del av mulighetene som Lucene tilbyr er nemlig ikke videreført.

Litt overraskende er det kanskje at LIUS tilbyr transaksjonshåndtering. Det er vanskelig å teste hvordan dette fungerer i praksis, og dokumentasjon finnes nærmest ikke. Implementasjon av transaksjonsstøtte er interessant i en virksomhetskontekst (se avsnitt 3.3.2 på side 55). Allikevel er LIUS på alle relevante punkter bak andre løsninger, ikke minst Solr, og ikke minst er dokumentasjonen elendig. Derfor er nok LIUS mindre aktuelt enn sine konkurrenter.

5.6.4 Andre søkeprosjekter

karakoram

karakoram [65] er et verktøy bygget rundt Lucene, og tilbyr en innebygget crawler, webapplikasjon for søking og en enkel XML API. Det er uvisst om karakoram er fri programvare siden det kun ligger en .war-fil på hjemmesiden (WAR-filformatet er JAR-filer, Java-arkiver, for webapplikasjoner skrevet i Java), men programmet er iallfall gratis og benytter seg av en rekke frie biblioteker inkludert Struts [12], Lucene Java [8], Hibernate [53], log4j og dom4j [34].

Ifølge utvikleren er programmet inspirert av LARM [5]. Det har ikke vært mulig å få tak i utvikleren bak karakoram til tross for gjentatte forsøk via e-post. Lite er dokumentert og programvaren virker utdatert – den bruker Lucene versjon 1.3.

Hva kan karakoram brukes til? Håpet var å få tilbakemelding fra utvikleren for å få et innsyn i oppbygning og kildekode.

sharehound

Dette FLOSS-prosjektet indekserer innhold i filsystemer over nettverk (for tiden kun SMB – Windows-basert fildeling – og FTP). Forøvrig indekserer det ikke innholdet i filene, bare katalog, filnavn og lignende. Det tilbyr en enkel webapplikasjon for overvåking av søking og innmating, i tillegg til en RSS-tjeneste.

Hva kan sharehound brukes til? Det er mulig å se på hvordan indekseringen kan utvides til å omfatte innhold av filene, og webapplikasjonen kan gi ideer.

spindle

spindle er et enkelt og lite verktøy i åpen kildekode som bygger på Lucene, og tilbyr crawler, indeksering og søk. Det kan først og fremst brukes som eksempel på hvordan Lucene kan brukes. Kanskje kan crawleren benyttes.

Zilverline

Zilverline [125] er en gratis søkeapplikasjon basert på Apache Lucene. Den støtter indeksering av filsystemer og nettverk, og kan lese en rekke ulike filformater. På mange måter fungerer Zilverline som et skrivebordssøk (desktop search) basert på Lucene. Det er lett å installere og har et godt grensesnitt for administrering via web. Dessverre er ikke Zilverline fri programvare.

Compass

Rammeverk som Hibernate [53] og iBATIS [55] muliggjør såkalt ORM (Object to Relational Mapping), det vil si avbildning mellom objekter og relasjonsdatabaser. Dette kan være spesielt smart å bruke i Java-prosjekter hvor det benyttes en separat datamodell. I rammeverket Spring [102] implementeres dette ved hjelp av såkalte DAO-objekter, objekter for datatilgang. ORM-rammeverket skal på sin side sørge for automatisering av persistens for data i applikasjoner. Avbildningen mellom database og objekter skjer typisk ved å konfigurere XML-filer eller lignende.

Compass [27] er på mange måter et tilsvarende rammeverk, men i stedet for å benytte seg av relasjonsdatabaser er det søkemotor som gjelder. Dette kan kalles OSEM – *Object to Search Engine Mapping*. Ideen er å kunne bruke Lucene deklarativt i programmer basert på Java. Integrasjon med ulike rammeverk er god, spesielt mot Hibernate og Spring, og datakilder kan dessuten synkroniseres automatisk.

Et OSEM-rammeverk kan fint benyttes i prosjekter for virksomhetssøk. Compass tilfører en del nyttige finesser til Lucene. Ikke minst forenkler det integreringen med Lucene ved at det tilbyr API-er på høyere lag. I tillegg tilbyr Compass støtte for transaksjoner. Mer konkret muliggjør det *two phase commit*, en egenskap godt kjent fra databaseverdenen.

Carrot^2

Carrot^2 [25] er et rammeverk for å lage klynger av søkemotorer. Det kan ta i bruk resultater av alle mulige slags søkemotorer, både åpne og proprietære. Integrasjonen er nok allikevel best med søkemotorer skrevet i Java, for eksempel basert på Lucene, siden Carrot^2 da kan kobles mer direkte. For lukkede søkemotorer kan SOAP, XML-RPC eller HTTP POST brukes (dette vil følgelig kunne gå ut over opplevd ytelse).

Søkemotoren Nutch kommer sågar med et tillegg for å lage Carrot^2-baserte klynger. På hjemmesiden (`http://www.carrot2.org/`) finnes det dessuten en demonstrasjon av rammeverket.

Annet

Java-Source.net [83] er en god kilde for å finne fri programvare skrevet i Java. Her nevnes søkemotorer som Egothor, Oxys, BDDBot og YaCy. Ingen av disse er aktuelle for denne oppgaven. Et relativt godt kjent alternativ for C++ er søkemotoren `ht://Dig` [54].

Kapittel 6

Diskusjon

> I love sports. I love animals. I love kids. I want to save the world. So how do I combine all those things? I don't know.
>
> — *Joan Jett*

Hittil har oppgaven grovt sett handlet om klare fakta. Utfordringer og særtegn ved i de forskjellige fagfeltene har blitt belyst. I forrige kapittel kom det dessuten frem hvilke produkter som finnes innen fri programvare og virksomhetssøk.

Dette kapitlet ser nærmere på hvordan fri programvare kan benyttes til å svare på de store utfordringene som finnes innenfor virksomhetssøk. Mer konkret gjøres dette i tre steg:

- Først diskuteres tekniske muligheter; hvilke komponenter som kan brukes til hva, og hvordan et produkt eventuelt kan sveises sammen.
- Etterpå diskuteres andre forutsetninger, siden det ikke kun er tekniske årsaker til at fri programvare fremdeles har et stykke å gå innenfor virksomhetssøk.
- I tillegg finnes det en del problemer som i dag vanskelig lar seg løse, iallfall om det tenkte produktet skal være et ideelt FLOSS-prosjekt.

Helt til sist nevner jeg mulige konsekvenser.

Selv om det er for omfattende å løse problemene i en masteroppgave, er det viktig å være klar over utgangspunktet for oppgaven, som fremdeles er gyldig: Fri programvare har et stort potensiale i forhold til virksomhetssøk, og behovet for slike løsninger vil neppe avta med det første.

6.1 Tekniske muligheter

Avsnitt 2.4 på side 31 ga et lite innblikk i hva som er dagens situasjon innenfor virksomhetssøk. Det kom frem at lukket og åpen programvare konseptuelt stiller ganske ulikt i ES-markedet. Et komplett produkt, som Fast ESP, finnes ikke. Men Lucene [9] vil kunne utgjøre kjernen, kanskje sammen med andre søkeverktøy. Andre frie komponenter kan brukes for ulik funksjonalitet.

Men hva er målet? Én ting er å skape en komplett plattform som kan tilsvare de kommersielle, men det kan være meningsløst å leke hermegås. Det må være ønskelig å skape noe som er litt bredere enn dagens Lucene, litt dypere enn dagens Nutch [10] og litt mer konfigurerbart enn dagens Solr [11]. Poenget må til syvende og sist være å skape noe som har en generell verdi, og ikke *bare* spesialtilpassede løsninger som oftest er tilfellet med dagens bruk av Lucene.

6.1.1 Komponenter som kan brukes

Her følger en diskusjon omkring FLOSS-prosjekter som kan brukes i søk. Merk at det selvsagt er behov for en rekke øvrige komponenter, for eksempel logging, databaser, enhetstesting og annet, men disse er ikke tatt med.

Spring som rammeverk?

Spring [102] er et populært rammeverk for Java EE (tidligere J2EE), og kan på mange måter kalles en lettvekter i forhold til eldre rammeverk som EJB. Et slikt rammeverk brukes hovedsakelig til å lime sammen applikasjoner, i tillegg til at det gir en del nyttig funksjonalitet. Samtidig tvinger det ofte utviklerne til å følge en del god praksis, og det fordrer hensiktsmessig arkitektur. Spring er ikke noe unntak.

Det mest sentrale poenget i Spring er implementasjonen av IoC-mønsteret (inversion of control). I stedet for at objekter oppretter andre objekter som de er avhengige av, er det en ekstern prosess som tar seg av opprettelsen av slike avhengigheter. Ansvaret ligger i rammeverket, og alt er konfigurert i en XML-fil. Avhengigheter opprettes under kjøring. For å gjøre en lang historie kort, handler det om at utviklerne i større grad kan fokusere på POJOs (Plain Old Java Objects), med andre ord mye renere objekter som kun gjør det de egentlig skal. Boka Pro Spring [50] har en god innføring.

AOP, aspektorientert programmering, er en del av Spring. Paradigmet muliggjør i enkle ord at lag med funksjonalitet kan legges over en hel datamodell. For programvare i søkeverdenen vil dette ha store fordeler med hensyn til logging og sikkerhet. Spring er dessuten beregnet på

webapplikasjoner, og disse programmeres etter MVC-mønsteret (Model, View, Controller). Siden et administratorverktøy for en større søkeplattform bør lages som en webapplikasjon, er dette nok et punkt som taler til fordel for bruk av Spring.

Spring kan også tilføre støtte for transaksjoner via et slags abstraksjonslag. Transaksjoner kan ordnes både deklarativt og programmatisk [50]. Sannsynligvis vil dette kunne brukes til å bøte på Lucenes problemer i forhold til ACID. Som om ikke dette var nok, tilbyr Spring enkel skedulering. Dette kan være nyttig med hensyn til innsamling av data, blant annet.

Åpenbart kan Spring fylle mange behov for å lime sammen et større ES-prosjekt, men Spring har nok en del funksjonalitet som ikke vil være nødvendig å bruke. Heldigvis er Spring bra modularisert. Det finnes imidlertid alternativer, og for bittesmå søkeprosjekter er det mulig at Spring vil være å skyte spurv med kanon.

Alternative rammeverk

Hvis alt man ønsker å gjøre med et rammeverk er å bruke IoC (se ovenfor), er kanskje Apache Fortress [3] bedre egnet. Det er rett og slett en liten og enkel IoC-beholder uten noe særlig mer. For å lage webapplikasjonen til administratorverktøyet kan for eksempel rammeverket Apache Struts [12] benyttes. AspectJ [15] er en forlengelse av Java-språket, og kan brukes hvis aspektorientert programmering synes å ha noe for seg.

Behandling av aktuelle filformater

Nedenfor listes noen alternativer kan brukes som tolkere av forskjellige filformater i en søkeapplikasjon. Disse vil følgelig utgjøre en del av *gjenkjenningen*, som beskrevet i avsnitt 4.3.2 på side 74.

PDF. PDFBox [85] virker som et svært utbredt alternativ med hensyn til å lese og skrive PDF-filer i Java. Det bruker en BSD-lisens. Nutch og LIUS er to eksempler på søkeprodukter som allerede benytter seg av PDFBox.

Faktisk har PDFBox en spesialtilpasset Lucene-klasse: `org.pdfbox.searchengine.lucene.LucenePDFDocument` ekstraherer en rekke felter fra både data og metadata i dokumentet [84].

XML. Her er alternativene svært mange og nyanserte. JDOM [61] og Jaxen [60] er to Java-biblioteker som benyttes av blant andre LIUS, begge utgitt under apache-lignende lisenser. Et alternativ til JDOM er dom4j [34]. Jaxen er en motor for XPath [123] og brukes til å få

tilgang til deler av XML-dokumenter. To andre alternativer er SAX [96] og det noe eldre DOM [57], som begge er i bruk i Apache Nutch. Et sjette bibliotek er Apache XML Xerces [14].

HTML. Ulike XML-verktøy kan brukes også til dette. Problemet er imidlertid at HTML ikke stiller like strenge krav til syntaks. JTidy [64] og CyberNeko HTML Parser [29] er to alternative biblioteker som blant annet benyttes av LIUS.

Microsoft Office. Jakarta POI-prosjektet [7] kan brukes til tolkning av Excel-formatet. Til Word-filer er ikke biblioteket like bra, men Nutch bruker det allikevel i sine tillegg. LIUS bruker på sin side 'jxl' [58] for Excel-filer og 'textmining.org' for Word-filer. Disse prosjektene virker imidlertid litt lite oppdaterte.[1]

OpenOffice.org. Her er saken grei, siden OpenOffice.org bruker en godt spesifisert XML-standard.

Flash. Nutch har et tillegg som benytter JavaSWF2 [56] til å lese innholdet i Flash-filer. I sammenheng med virksomhetssøk er kanskje Flash mindre utbredt enn på internett.

Lydfiler. Java ID3 Tag Library [59] benyttes av Nutch til å trekke ut ID3-informasjon. ID3 er et format for metadata i lydfiler som *mp3*. Et alternativ er Tritonus [108]. Javas interne `javax.sound.sampled` kan dessuten brukes til å hente ut mer metadata fra lydfiler.

Andre. For komprimerte Zip-filer er det greit å bruke Javas egen `java.util.zip`. Tolkning av JavaBeans går også greit med Javas standardbibliotek. For rikt tekstformat (RTF) har Java en innebygget pakke som heter `javax.swing.text.rtf`. For å tolke JavaScript bruker Nutch Jakarta ORO [6], et fleksibelt bibliotek som kan trekke ut data fra tekstfiler. For eksempel kan den benyttes på RSS, LaTeXog VCard.

Forøvrig kan det være aktuelt å tolke metadata fra bilder, video og annen multimedia. Her stiller kommersielle løsninger ekstremt sterkt, ved at de tilbyr biblioteker for å lese alle tenkelige typer data. Det kan se ut som at fri programvare har en lang vei å gå, selv om det er umulig å spore opp alle tenkelige biblioteker og programsnutter som kan brukes.

Java-Source.net [83] er forøvrig en svært nyttig ressurs for å finne åpne biblioteker til nærmest ethvert bruk. Det er ikke hensiktsmessig å ramse opp alt som finnes her. De fleste av bibliotekene i listen over er valgt ut fordi de allerede er i bruk i enkelte frie søkeverktøy.

[1] Nettsiden for 'textmining.org' har vist seg vanskelig å nå etter gjentatte forsøk.

Dessuten er rekkefølgen ikke tilfeldig. PDF-filer, XML-filer og HTML-filer representerer nok sammen med Microsoft Office-filer brorparten av innhold i mange aktuelle virksomheter. På grunnlag av dette kan det være mulig å hevde at fri programvare dekker mange av de vanligste behovene.

Hva med resten av "mellomsteget"?

I seksjon 4.3.2 samt i figur 4.2 forklares det hvilke steg som er nødvendige; gjenkjenning, normalisering, utluking, redusering, utvelgelse og kategorisering. Eksisterer det komponenter av fri programvare som kan brukes til hele denne prosessen?

Mange av de viktigste komponentene er allerede innebygget i Lucene. Nutch tilbyr tillegg for en del av stegene. Påvisning av språk gjøres vanligvis ved hjelp av n-gram (se side 75). Nutch har klasser for dette i `/src/plugin`, og en rekke språkprofiler (deriblant norsk) følger med. Nutch' implementasjon av n-gram muliggjør også opprettelse av nye profiler og deteksjon av brødtekst som et frittstående program.

Normalisering og utluking av stoppord støttes også greit i Lucene, men det er opp til utvikleren å implementere mer komplette utgaver av disse. Ett eksempel er at Nutch har en liste over stoppord som brukes sammen med Lucenes filter. Dessverre er denne listen hardkodet inn og finnes bare på engelsk (dette er blitt rapportert som en bug i Nutch).

Med hensyn til redusering av ord til sin enkleste form, eksisterer det en fri ressursside som heter Snowball [99]. Her er blant annet reduseringsalgoritmer for ulike språk utarbeidet; porter-stemming [86], stemming for skandinaviske språk, og en del flere. Disse er allerede implementert av Lucene. Nutch har også implementasjoner for utvelgelse og kategorisering via synonymer eller andre ordlister.

Kommersielle aktører innen virksomhetssøk tilbyr langt bedre språkstøtte enn fri programvare. Ikke minst er støtten for asiatiske språk mye bedre hos Fast ESP. Lemmatisering er også noe som må utvikles fra bunnen dersom dette er ønskelig, men i de fleste sammenhenger vil det kanskje holde med stemming. Det største savnet er imidlertid muligheten for å konfigurere alle trinnene i "mellomsteget" enkelt og greit. Med dagens fri programvare er dette langt verre. Lucene har en god arkitektur, og implementasjon av moduler er mulig uten altfor mye kluss, men dette ligger egentlig på et relativt lavt abstraksjonsnivå. I et ferdig produkt for virksomhetssøk kan det være nyttig å ha et solid og enkelt verktøy for å styre alle trinnene, gjerne ved hjelp av enkel scripting, men gjerne også visuelt.

6.1.2 Mulige tilnærminger

Eksistensen av ulike generelle søkeløsninger basert på Lucene er et vitnesbyrd om at det er enkelt å bruke dette biblioteket. Det er lite hensiktsmessig å programmere noe som allerede finnes, så i denne seksjonen diskuteres ulike mulige tilnærminger for å kunne bringe kombinasjonen av virksomhetssøk og fri programvare opp på et høyere nivå.

Integrering mellom ulike produkter og å skape plattformer står derfor sentralt. I en artikkel snakker Jacob Ukelson om dette [111], og han nevner områder for forbedring i fremtiden. To av disse er

- bedre støtte for parsing av ustrukturerte data, og
- forbedret støtte for avbildning mellom ulike XML-skjemaer.

Det er mye som tyder på at det vil være fornuftig å la en rekke ulike produkter eller mindre moduler kommunisere. For at dette skal bli mulig, må modulene ha en felles plattform å kommunisere på. I noen tilfeller vil tjenesteorientert arkitektur (SOA) kunne gjøre jobben, men for høy ytelse vil en tettere integrering være nødvendig. Da kan løsningen være rammeverk, felles administrativ styring eller annen deklarativ sammenveving.

Generelt om arkitektur

Det er mulig å sette opp en skisse over hvilke biter som er nødvendige. Det er en kjennsgjerning at ved å bevege seg opp på et mer sofistikert nivå for en plattform, øker kompleksiteten betraktelig. Her er noe av det som kan være nødvendig:

- Lucene.
- Et rammeverk for å binde sammen komponenter. Dette behøver ikke være et spesielt rammeverk som eksempelvis Spring [102], men uansett må det finnes en *lim* som binder sammen komponentene på en konsistent måte.
- En måte å sende beskjeder på. Både under innmating og under søking bør en felles standard brukes, slik at ulike steg og moduler lett kan kombineres på forskjellige måter eller kanskje også sløyfes. Prosesser kan også holdes mer adskilt. Et forslag er å bruke et XML-format.
- Bindeledd. Det finnes gode crawlere, og forbindelse til filsystem og databaser er heller ikke noe problem. Utfordringen går nok heller ut på å utvikle en slags plugin-arkitektur som gjør at ulike bindeledd kan benyttes relativt enkelt. Stikkord her er integrasjon og sikkerhet.

- Køsystemer og skedulering. Eksempelvis må dokumenter som kommer inn fra bindeleddene samles i en kø for å prosesseres. Distribuering på flere noder bør være mulig. Dokumentmengden må derfor hele tiden kunne tilpasses; køen må ikke vokse seg for stor. Ved å bruke crawlerprosjektene Nutch eller LARM vil dette allerede være tatt hånd om, men det hjelper lite hvis andre og mer generelle løsninger skal utvikles for bindeleddene.
- En sentral modul som holder orden på noder i et oppsett med flere noder. I starten kan dette sløyfes, men det må uansett tas høyde for på sikt hvis løsningen skal være skalérbar eller redundant.
- Et styreverktøy må lages, gjerne som en webapplikasjon. Den kan blant annet styre konfigurasjon, oppstart og avstenging av noder og rapportering om status (overvåking).
- Verktøy for logging.
- Pakking og installasjon – funksjonalitet "ut av boksen".

Dette gir et bilde over hva løsninger innen fri programvare mangler i dag. De neste avsnittene går mer detaljert inn på ulike tilnærminger og diskuterer hva som mangler spesifikt i hvert tilfelle. Uansett er det viktig å prøve å holde arkitekturen enkel og elegant. En god start på dette er avgjørelsen om å holde seg til ett programmeringsspråk i all implementering, og ett lignende filformat for alle data – mest nærliggende er det å bruke henholdsvis Java og XML.

Utvikling av egen søketeknologi

Verity, Fast og andre kommersielle pionérer utviklet for mange år siden sine egne søketeknologier. Selv om svært mye av det teoretiske fundamentet ble lagt ytterligere mange år før disse suksesshistoriene oppstod, tok det altså noe tid. Kanskje var det markedet som bestemte tidspunktet? Uansett burde det muligens være grunnlag for å hevde at nye søkeselskaper og andre aktører innenfor virksomhetssøk burde utvikle egne IR-biblioteker i dag også. Dette viser seg ikke å være tilfellet. De fleste slike planer kan forkastes som ulønnsomme, med visse forbehold.

Aller først kommer argumentet om å "finne opp hjulet på ny". Det viser seg at fri programvare, med Apache Lucene i spissen, er kommet så langt at det ikke svarer seg å starte forfra. Ett aspekt er det politiske og prinsipielle spørsmålet om gjenbruk, men i tilfellet for IR-biblioteker er det direkte irrasjonelt å starte forfra i de aller fleste tilfeller. Dette har stadig flere, nye søkeaktører forstått, og de utvikler programpakker og tjenestetilbud på toppen av eksisterende fri programvare. Selv om

fagfeltet er relativt modent og det teoretiske fundamentet er på plass, har disse firmaene skjønt at det tar lang tid før en så ytelsesavhengig realisering som for eksempel Lucene blir moden. Det virker usannsynlig at noen skal klare å revolusjonere teknologien tilstrekkelig til at det rettferdiggjør å begynne om igjen i dag.

Dessuten ser markedet for programvare og søketeknologi annerledes ut nå enn det gjorde for femten år siden. Det blir etterhvert mindre fokus på selve IR-teknologien som finnes i bunnen. Oppmerksomheten rettes i stedet mot integrering, tilpassing og tjenester. En moderne organisasjon med behov for rask gjenfinning, ønsker nok i første rekke at produktet de investerer i skal fungere sammen med eksisterende infrastruktur og andre investeringer. Med andre ord kan det spekuleres i at etterspørselen etter ny IR-teknologi er synkende, mens etterspørselen etter gode og komplette søkeløsninger er voksende. Hvorvidt et dokument gjenfinnes i løpet av ett eller fem hundredels sekunder er interessant for Lucene-utviklere, men samtidig uvesentlig for de mest typiske brukere av virksomhetssøk.

Allikevel er ikke ytelse noe som skal stikkes under en stol. Svært snever eller spesiell ytelse og nye paradigmer kan være nyttige for forskning. Dessuten kan nye teknologiske konsepter behøves. Helt bestemte og smale forretningsområder kan nyte godt av nyvinninger innenfor informasjonsgjenfinning. Det er imidlertid tydelig hvordan dette problemområdet faller utenfor problemområdet som omhandler virksomhetssøk. Det skal i tillegg svært gode økonomiske utsikter til for å rettferdiggjøre at nyutvikling kan erstatte innkjøp eller anskaffelse av eksisterende teknologi, også om sistnevnte modifiseres kraftig.

Konklusjonen blir altså at organisasjoner kun i svært få tilfeller bør utvikle sin egen søketeknologi – iallfall på algoritmenivå som Lucene. Det skal helt spesielle krav til for at dette kan rettferdiggjøres, ikke minst for virksomhetssøk som i større grad dreier seg om forretningsproblemer. Siden markedet for søkeløsninger er voksende, oppstår det dog behov for teknologiske nyvinninger. Her er det som regel mest å hente på problemløsning på høyere nivå, for eksempel ved hjelp av spesialisering innenfor et bestemt segment eller en spesifikk bransje.

Hva med GPL? Avsnittet ovenfor hadde ikke vært gyldig dersom alle åpne IR-biblioteker hadde vært underlagt en restriktiv lisens. I seksjon 5.1.1 på side 97 ble det forklart hvordan utviklere av fri programvare kan legge strenge restriksjoner på bruken. Dersom Lucene hadde vært lisensiert under for eksempel GPL, hadde det satt en stopper for alle ideer om å bygge større systemer og mer overordnet søketeknologi, i hvert fall for firmaer som ønsker å selge lukket programvare basert på Lucene.

Kan Nutch og Solr kombineres?

Motivasjonen bak å kombinere disse to prosjektene er i første omgang å utnytte mest mulig av funksjonaliteten i begge. En del av det som trengs finnes både i Nutch og i Solr, men det er løst på forskjellige måter. Her er noen forslag:

1. Crawler fra Nutch. Den er relativt moden og egner seg for ekstremt store operasjoner. Kanskje kan lenkedatabasen også brukes?

2. Dokumenttolkere fra Nutch. Solr er ikke skapt for å lese inn ulike filformater, men i en total ES-pakke er det selvfølgelig ønskelig.

3. XML API fra Solr. Virksomhetssøk handler om integrering med andre systemer, og innmating/spørring via XML er ønskelig. Selvfølgelig er det også et ønske med ren Java API.

4. Enkle skjemaer fra Solr. Mye av grunnkonfigurasjonen i en ES-situasjon skjer ved tilpassingen av skjemaet eller skjemaene.

5. Web-administrering fra Solr. Er det mulig å utvide dette?

Eventuelt kan også integrasjon mot Hadoop [4] hentes med fra Nutch, men dette er kun nødvendig ved store dokumentmengder.[2]

Ethvert Lucene-indeks kan i teorien leses av enhver Lucene-applikasjon, gitt at filsystemet er av samme versjon. Derfor er det ikke noe i veien for at Nutch kan lese indekser som er laget av Solr, eller omvendt. Dessverre fører noen designvalg til at kombineringen ikke blir så triviell allikevel.

Lucene støtter opprettelse av nye felter i indekset dynamisk. Dette er en stor styrke som både Nutch og Solr vet å utnytte. Men tilnærmingen er svært ulik. Nutch har på sin side lagt opp til et fast skjema:

$$field \in \{\ldots, \text{'title'}, \text{'host'}, \text{'url'}, \text{'content'}, \ldots\} \tag{6.1}$$

Dette er praktisk med hensyn på søkemotorer for internett. Nye felter kan opprettes, men dette må gjøres i Java-koden ved å endre på en `filter()`-metode fra Lucene. Her er et eksempel (Lucene 1.4.3/1.9.1):

```
Document.add(Field field)
```
(6.2)

[2] Apaches inkarnasjon av Googles map/reduce er ifølge Nutch' hjemmeside nødvendig for ansamlinger med flere enn 100 millioner dokumenter, og derfor har jeg naturligvis ikke støtt på problematikken.

`Field` har flere konstruktører, og variasjonene mellom Lucene før og etter versjon 2.0 er formidable. Dette virker kanskje irrelevant teknisk, men hensikten er å demonstrere hvor lite fleksibel Nutch er med hensyn til opprettelse av nye felter. Solr definerer på sin side felter i et XML-skjema, og ingenting konfigureres i Java-koden.

Flere problemer oppstår som en følge av disse forskjellene:

- Crawleren fra Nutch er laget med tanke på det faste skjemaet. Den kan altså ikke plugges rett inn i Solr uten modifikasjoner.
- Enda viktigere er det at tolkerne til filformater (PDF, MS Office, med flere) som er utviklet for Nutch er også tilpasset disse predefinerte feltene.

Mye av poenget med å la Nutch ta seg av "venstresiden" (figur 2.1 på side 22) faller dermed bort. Positivt er det imidlertid at Nutch har en fleksibel arkitektur med tanke på bindeledd (connectors), og Nutch kan derfor med rimelig overkommelig innsats brukes til å lese fra andre kilder enn bare sin egen crawler.

Det finnes et ganske mye verre problem som kan bli en akilleshel for integrasjon av Nutch og Solr:

- Nutch bruker Lucene 1.9/2.0; Solr bruker Lucene 2.1.

En nyere versjon av Lucene har ingen problemer med å lese et eldre filsystem, men omvendt går rett og slett ikke. Dette utelukker enhver bruk av Nutch sin "høyreside" (figur 2.1) i en sammensmelting. Uansett måtte `schema.xml` i Solr sin "venstreside" ha vært tilpasset de faste feltene i Nutch, noe som medfører en litt for streng restriksjon.

Mer sannsynlig blir det altså å la Solr lese et indeks som Nutch har laget, men av grunner forklart over blir eneste gjenværende fordel ved denne tilnærmingen at XML API-et fra Solr kan tas i bruk.

Oppsummert virker det fornuftig ved første øyekast å la disse to prosjektene smeltes sammen. Problemene som oppstår i kjølvannet blir imidlertid så store at nesten alle fordeler forsvinner. En ting som ikke har vært diskutert her, er muligheten for å lage et langt større prosjekt som bygger over både Solr og Nutch, og som bruker elementer fra begge to. En slik løsning vil nok være brukbar; det viktigste blir nok å få utnyttet crawleren fra Nutch. De eksterne bibliotekene for å lese ulike filformater utnyttes muligens best ved å designe dem for Solr i utgangspunktet.

Et litt større overbygg for Solr vil måtte ta hensyn til avbildningen (se side 78), og dermed må hvert bibliotek tilpasses denne muligheten. En utfordring blir derfor å definere hvordan avbildningen skal foregå, slik at bibliotekene for filformatene kan lages med tanke på lett konfigurasjon – gjerne fra et webverktøy eller i en XML-fil.

Det kan godt tenkes at en kombinasjon av Nutch og Solr er fornuftig i mange prosjekter, men problemene blir kanskje større enn gevinsten i et tenkt, *generelt* ES-prosjekt. Andre veier virker lettere å gå.

Kombinere kommersielle og åpne søkeprodukter

I denne oppgaven diskuteres det i utgangspunkt hvordan fri programvare løser problemer innen virksomhetssøk – uten noen noen innblanding av proprietær programvare. Allikevel kommer jeg ikke utenom å nevne at det går an å kombinere både åpne og lukkede produkter i en større plattform. Dette er kanskje et aktuelt alternativ for virksomheter som ønsker å sy sine egne plattformer.

For eksempel kan Fast ESP kombineres med Solr. Under innmatingen av data kan XML styres inn i Solr i stedet for inn i Fasts egen motor. Det virker kanskje intuitivt meningsløst å bruke fri programvare når man allerede har betalt for noe proprietært, men det kan hende at en virksomhet ønsker å kunne bruke en distribuert modell hvor bare enkelte Fast-lisenser forefinnes. Kanskje skal også indekset være søkbart fra eksterne, Lucene-baserte systemer.[3]

Lukket og åpen programvare kan imidlertid først og fremst benyttes av lukkede løsninger. Det har seg nemlig slik at de fleste åpne lisenser (Apache og LGPL, for å nevne noen) tillater bruk i proprietær programvare. Det er svært utbredt å bruke fri programvare på denne måten, men det besvarer altså ikke problemstillingen i oppgaven.

Søkepakke med GNU/Linux som forbilde

GNU/Linux er et godt eksempel på en distribusjon, hvor hundrevis eller tusenvis av ulike programsnutter legges sammen i en pakke. Det store antallet av biter gir mange muligheter. Kan en Lucene-basert søkeløsning modelleres etter dette prinsippet?

Forslaget går altså ut på å la Lucene-pakken inneholde all tenkelig programvare som noen kan få bruk for, blant annet både Lucene Java [8], Nutch [10], Solr [11], Spring [102], Hibernate [53], log4j [71], Compass [27], LARM [5] og så videre. Med andre ord kan distribusjonen legge ved all programvare og alle biblioteker som noensinne kan tenkes brukt i sammenheng med virksomhetssøk.

Det blir altså opp til brukerne å sette sammen alt selv, og det kan i beste fall betegnes som en “ekspertløsning”. Fordelen er selvfølgelig at

[3] Merk at det er uklart hvordan en slik løsning vil fungere med hensyn til lisensbetingelser og produktstøtte fra Fast. Når programvare fra ulike leverandører skal kombineres, enten det er snakk om åpen eller lukket kildekode, må ofte slike forhold kartlegges.

fleksibiliteten og konfigurérbarheten blir utnyttet til det fulle, og brukerne (det vil si utviklerne) slipper å lete etter alle relevante biter til puslespillet sitt. Et annet ord for en slik løsning kan rett og slett være en ressurssamling.

Vil dette være levedyktig? Ekspertbrukere og utviklere er som regel godt kjent i terrenget hva angår den programvaren de skal utvikle. I tillegg er de gode på å søke etter informasjon og diskutere løsninger. Et forslag for å gjøre en Lucene-distribusjon som dette mer attraktiv, er å utvikle en rekke forslag til scenarier i form av enten dokumentasjon eller litt kode. Pakken kan derfor fungere som en samling av tips og triks.

En slik distribusjon kan rent praktisk ta form som en nettside. Faller hele ideen da til kort?

"Søk på boks"

Det er mulig å gå enda ett skritt videre fra å lage en utførlig søkedistribusjon. Det finnes flere kommersielle eksempler på såkalte svarte bokser (black box-prinsippet). Ideen bak konseptet er at en modul plasseres i et nettverksmiljø og settes opp nærmest én gang for alle. Hva som skjer på innsiden av boksen er helt avskjermet. Google Search Appliance følger denne filosofien. Er det mulig å herme etter Google og utvikle en "Lucene Search Appliance"?

Den svarte boksen kommer ferdig oppsatt, med alle delene sydd ferdig sammen. Hvilken tilnærming som velges av dem ovenfor (Lucene og Spring med flere, Nutch og Solr, eller noe annet) er ikke viktig i denne sammenhengen. Kanskje kan også flere av dem benyttes, alt etter hvilket *scenario* som velges under installasjonen. Det er nemlig en god idé å la en slik svart boks være basert noen få, typiske scenarier. Dette vil medføre at riktige moduler velges, at det meste av oppsatt blir gjort på forhånd, og at bare et minimalt sett med tilganger og oppførsel behøver å legges inn. Så kan dette gjøres med Lucene?

Fordelene med denne varianten er delvis åpenbare. Det er lett å komme igang, og oppsettet krever minimalt med kompetanse. Alt dette bidrar forhåpentligvis til at terskelen for å komme igang med virksomhetssøk i gitte tilfeller synker. Det skal kanskje ikke all verden med strategiske vedtak og diskusjoner til for å gå til et slikt skritt heller. I tillegg er det en fordel for noen virksomheter at boksen faktisk er en "open source"-boks.

Ulempene er imidlertid nokså tydelig å ense. Blant annet er minimalistisk konfigurasjon tveegget. Noe av bakgrunnen for å drive med virksomhetssøk kommer jo nettopp fra et ønske om å integrere systemer, samt å skreddersy løsninger. Det er selvfølgelig ikke mulig å oppnå perfekt integrering med en svart boks. For at integrering skal gå på noe som helst nivå, må det faktisk settes svært store krav til kvaliteten i

det pakkede produktet. En annen ulempe er at svarte bokser er utviklet for et kundesegment som muligens i utgangspunktet ønsker seg minst mulig bry; sømløs eller transparent teknologi, som klisjeen kaller det. Hvis årsaken er at brukerne mangler kompetanse, vil de garantert ha behov for brukerstøtte. Selv om FLOSS-entusiaster vil fremheve styrken i e-postlister og diskusjonsfora, vil nok den jevne, intetanende brukeren ønske seg annen type støtte. Et siste ankepunkt mot Lucene i en svart boks er at det kan bli utfordrende å bygge klynger.

Hvordan kan så "fri programvare på boks" ta seg ut? Flere tilnærminger bør være mulig. Det mest selvfølgelige er kanskje at løsningen kommer som et ferdig oppsatt operativsystem. I så fall må dette være basert på fri programvare – typisk GNU/Linux.

En GNU/Linux-distribusjon kan altså være mulig. Det finnes flere slike distribusjoner som bygger på black box-prinsippet, eksempelvis brannmurer, rutere og webtjenere. Brukerne må her installere operativsystemet selv, men det må være mulig å velge scenario under installasjonen.

Et alternativ kan være å lage en ferdig oppsatt virtuell maskin. Da vil ikke brukerne engang behøve å installere noe operativsystem, forutsatt at miljøet allerede kjører VMware [115] eller annen form for virtualiseringsprogramvare. Denne typen distribusjoner er også vanlige. En stor ulempe med denne tilnærmingen vil være ytelse og skalérbarhet. Middels store virksomhetssøk setter store krav til maskinvare, og det kan bety ulykke ikke å sette ES-systemet på egen maskinvare. Det kan derfor stilles spørsmålstegn til hele nytten ved en virtuell variant. De største bruksområdene kan derfor tenkes å bli innen testing, samt virksomhetssøk i svært liten skala.

En komplett løsning med både operativsystem, nødvendig søkeprogramvare og annen programvare, samt riktig maskinvare, kan synes som den beste veien å gå. Typisk bør løsningen da ha et firma i ryggen. Mange kunder kan nok tenkes å være interessert i en slik utgave, dersom de kan stole på kvaliteten. Spørsmålet blir da hvor unik denne tilnærmingen blir i det hele tatt, hvis poenget var å demonstrere hvordan virksomhetssøk og fri programvare som kombinasjon kan stå på egne ben i større grad enn i dag.

6.2 Andre forutsetninger

I kapittel 5 ble det gjort rede for at fri programvare er langt mer enn lisenser og åpenhet. Utfordringene med å oppnå suksess for FLOSS-prosjekter går derfor langt ut over det å åpne opp kildekoden. Tre grunnsteiner ble nevnt: opphavsrett og åndsrett, utviklingsmønstre og ressursforvaltning.

I tillegg tas det utgangspunkt i karakteristikken av virksomheter som ble gitt i seksjon 2.2 på side 23. Sammen med andre momenter medfører disse at andre forutsetninger enn de rent tekniske må legges til grunn, og her diskuteres noen av dem.

6.2.1 Fellesnevnere i virksomheter

Mange IT-løsninger skreddersys helt fra bunnen av. Dette kan gi gode resultater, men vil være dyrt å realisere – kanskje spesielt for mindre ressurssterke virksomheter. Derfor er det motsatte på mange måter idealet, nemlig at alt fungerer med minimalt med konfigurering. Å bevege seg mot dette idealet krever fellesnevnere. Brukere, altså virksomheter, bør ha en del sammenfallende behov.

I tilfellet med virksomhetssøk er det vanskeligere enn ellers å finne fellesnevnere. Noen typiske behov for virksomheter ble gitt på side 23. Jeg mener imidlertid at disse fellestrekkene hovedsakelig ikke betegner likheter mellom virksomheter, men snarere felter hvor virksomheter er forskjellige.

Et eksempel er punktet om individuelle oppfatninger om relevans og rangering. Det som er relevant for ett bruksområde, kan være helt ubrukelig i andre sammenhenger. Selvsagt ligger mye av styrken i at virksomhetssøk er konfigurerbart, spesielt med hensyn på relevans, men ulikhetene i konfigurasjon gjør det likevel vanskelig å finne felles grunn for hvilken type relevans som er generelt best.

Det finnes flere slike ulikheter, og resultatet blir at et samlende produkt vil bli nokså stort og omfattende hvis det skal kunne ta høyde for alle ulike scenarier og de vanligste konfigurasjonsmønstrene. Mye funksjonalitet må følge med i pakken, og gode administrasjonsverktøy eller grensesnitt må lages for enda flere moduler. Det blir ikke enkelt å lage et lite, nett produkt som er generelt. Da er det nok bedre å spesialtilpasse.

Det finnes selvsagt mange likheter, ikke minst på en del tekniske plan. Eksempelvis er det en selvfølge at en ES-pakke skal støtte traversering av lokale nettverk. Allikevel er bruksmønstrene og scenariene ulike, og dette krever mest konfigurering i eksisterende løsninger.

Behovene er komplekse

En viktig fellesnevner er at behovene ofte er relativt sammensatte.

Et kompliserende fellestrekk er behovet for å binde sammen resultater fra ulike kilder. Dette går litt utover bare det å samle inn fra de respektive kildene. Hvordan resultatene settes sammen i ulike kategorier for forskjellige bruksområder kan være vesentlig.

Ulik fordeling mellom strukturert og ustrukturert informasjon er også med på å gjøre oppgaven vanskelig. Enkelte virksomheter ønsker å in-

deksere informasjon som kun er strukturert. Apache Solr tar for eksempel utgangspunkt i at alt skal være strukturert ved tidspunktet for innmatingen, mens Apache Nutch fulltekstindekserer "kaotisk" materiale (skjønt web har en viss struktur). De forskjellige behovene langs skalaen for struktur av informasjon er nok en av de mest kritiske faktorene. Uten dette hadde det kanskje vært mye lettere å finne gode fellesnevnere for en bred, generell søkeplattform.

Andre kompliserende faktorer er sikkerhet og adgangskontroll, samt mangfoldighet i datakilder og -formater. Virksomhetssøk vil på mange måter kunne dra nytte av standardiseringer eksternt. Det blir lettere å finne fellesnevnere hvis virksomhetene enes i større grad om sikkerhet, filformater, kildesystemer og annet.

Markedet for virksomhetssøk og fri programvare bærer preg av denne erkjennelsen om fellesnevnere; de mest generelle produktene som Solr og Nutch har ikke kommet særlig langt, og det finnes en rekke spesialtilpassede søkeprodukter. En mulig tilnærming vil kanskje være å tilby scenarier i en ES-plattform og dermed finne en mellomting mellom det helt spesielle og det helt generelle.

6.2.2 Utvikling for virksomhetssøk

Selv om søk på nettet i dag er kjent for de aller fleste, ligger virksomhetssøk unektelig et stykke bak. Mange spår at dette vil endre seg i tiden fremover, men på hvilken måte vil dette skje? Vil ES bevege seg i en retning som vil gjøre det enklere for FLOSS-prosjekter å tiltrekke seg utviklere?

I seksjon 5.3 på side 105 kom det frem at synlighet i hackermiljøet er en viktig motivasjonsfaktor, spesielt for enkeltpersoner som bidrar innen fri programvare. Utviklere ønsker å oppnå status i et miljø hvor folk anerkjenner deres bragd. Men for at dette skal kunne skje, bør virksomhetssøk først oppnå en viss status innenfor hackermiljøet. Fagfeltet er riktignok på vei oppover. Allikevel er det ikke sikkert at dette vil øke statusen ES har i hackermiljøet.

Det ligger nemlig en fare i følgende: Selv om ES står overfor mange tekniske utfordringer, løser det i stor grad problemer innenfor forretning. Spørsmålet er derfor om "de riktige massene" vil anerkjenne den tekniske delen som interessant nok. Det er godt mulig at ES vil bevege seg i denne retningen, men det kan også godt skje at virksomhetssøk forblir et felt med fokus først og fremst på det forretningsmessige. Da vil i så fall virksomhetssøk ikke bli en del av "typiske hackerproblemer", noe som er en klar forutsetning dersom horisontale prosjektnett skal

fungere optimalt. Akkurat i dag er situasjonen nemlig slik at problemene søkes løst i et litt annet domene (hackermiljøet) enn problemdomenet (oftest forretning), og disse har tilsynelatende lite til felles.

Derimot er det en fordel at søk generelt er på vei oppover. Det er ikke usannsynlig at det vil komme drahjelp ved at stadig flere hackere blir interessert i søk gjennom nettsøk, og dermed ser potensialet i å løse stadig flere problemer ved hjelp av søketeknologi.

Oppsummert må søketeknologi bli mer kjent. I dag forbinder de fleste søk med Google. Flere entusiaster og programmerere må innse potensialet i teknologien, og begynne å betrakte søk som et grunnleggende felt – på lik linje med databaser. A propos databaser, så var det akkurat dette som skjedde da det virkelig løsnet for databaseteknologien på 80-tallet. Det tok noen år før relasjonsdatabasene ble kjent nok til at vellykkede, åpne databaseprosjekter fikk godt fotfeste. Forhåpentligvis går det den samme veien med søk, men ingenting er sikkert.

6.2.3 En grad av realisme

Noen ganger kan det være vanskelig å innse at utviklingsprosjekter må ha en viss grad av realisme. En person, gruppe eller et firma som begir seg ut på dette toktet må forstå alvoret. Først og fremst er antakelig arbeidsmengden ved et nyttig prosjekt stor. Samtidig er det ikke bare fristende, men strengt tatt helt nødvendig, i mange tilfeller å velge den økonomisk mest lukrative veien. Til slutt reiser det seg derfor et spørsmål om nytteverdien i et slikt prosjekt generelt. Er det virkelig verdt slitet? Realisme handler ofte om å svelge kameler, og i dette tilfellet skal minst tre av dem ned på tvers.

Stor arbeidsmengde

I løpet av denne oppgaven bør det ha blitt gitt et inntrykk av at virksomhetssøk ikke er enkelt. Selvfølgelig er det ikke et forbilde å lage et produkt så komplisert og komplekst som mulig. Snarere er det en kunst å kunne utvikle noe som er enkelt, men som allikevel kan svare på et sammensatt problem. Når det er sagt vil den totale arbeidsmengden ved et prosjekt eller produkt som foreslått i denne oppgaven antakelig være uoverskuelig. Så blir det opp til andre å finne ut om den også er uoverkommelig. Arbeidsmengder og krav vil være store innenfor både design, utvikling og testing. Virksomhetssøk er mer enn bare IR; men selv om mange av problemene innenfor IR er løst, må kunnskapen om disse tas med under produktutviklingen i virksomhetssøk.

Når produktet skal designes er det vesentlig med kunnskap om, og kjennskap til, en rekke områder og fagfelt. Spesielt sentralt står FLOSS og ES. Innenfor førstnevnte må designerne vite noe om systemutvikling,

og helst ha erfaring med arbeidsmetoder, miljø og horisontale prosjektnettverk. I tillegg må designet ta hensyn til lisensiering, uten at dette skal stoppe showet. Litt verre er det å kunne tilegne seg tilstrekkelig inngående kunnskap og erfaring rundt eksisterende prosjekter. Denne oppgaven berører bare overflaten. For effektivt å kunne sette sammen et komplett ES-system er ekspertkunnskap om disse en fordel. Dernest, innenfor ES, må de som designer ha bedre oversikt over aktuelle problemer som virksomheter søker å løse ved hjelp av søk, og erfaring med eksisterende systemer hos virksomheter vil være å foretrekke. Som om ikke dette var nok, må designerne forholde seg til alle de uløste problemene i ES-fagfeltet (se side 56), spesielt generering av testdata, rangering i heterogene og ikke-lenkede dokumenter, brukerscenarier og støyproblematikk.

Under selve utviklingen er det en utfordring å kunne bruke alle komponentene på en så generell måte som mulig, uten å måtte drasse med seg helhetlige pakker. Som et konkret eksempel kan Nutch nevnes. Det er ønskelig å bruke flest mulig av Nutch sine egenskaper, men brukt som en helhet er det ikke utenkelig at Nutch vil kunne bidra med mye unyttig, også. Riktignok er det tenkt at ES-produktet skal legge seg som en kappe over eksisterende programvare eller lime dem sammen, men hvis det skal bli konkurransedyktig må det være optimalisert i størst mulig grad. Konsekvensen blir at noen komponenter må skrives om, og det blir en kunst samtidig å holde bitene i en så generell tilstand som mulig. Som et resultat vil prosjektet kunne kreve mange utviklere og ta lang tid. Dette vil også muligens vanskeliggjøre oppstart av prosjektet for en enkeltperson, slik mange FLOSS-prosjekter tradisjonelt har kommet i gang.

Sist, men ikke minst, vil testingen ta tid. Selvfølgelig kan en testdrevet utvikling hjelpe til med å redusere omfanget. Men siden det dreier seg om en komplett søkepakke, vil testing på høyere plan kunne kreve mer oppmerksomhet. Mer spesifikt må produktet testes ut i ulike miljøer og etter gitte scenarier, helst så tidlig som mulig under utviklingen. Prosjektet skal tilby en del ny funksjonalitet, og denne må testes ekstra grundig. Transaksjonsstøtte etter ACID-prinsippet er et eksempel på en finesse som kanskje vil kreve grundigere testing enn en del annet. Sammenlagt er det ikke utenkelig at testing vil legge beslag på store ressurser.

Ingen av disse punktene – design, utvikling og testing – behøver å sette kjepper i hjulene ut fra denne fremstillingen. Litt sunn realisme skader nok allikevel ikke. Selv om det ikke vil være et krav at et produkt skal måtte bite over alt dette fra første stund, så skader det ikke å ha tenkt nøye igjennom det på forhånd.

Muligheten for (raskere) penger

La oss nå tenke oss at prosjektet er startet opp. Resultater har begynt å dukke opp, systemet er delvis implementert og ikke minst er det opparbeidet betydelig kompetanse hos utviklerne. Nøkkelpersoner kan ta med seg kunnskapen sin og andre ressurser, og "desertere" til det kommersielle markedet. Det vil alltid eksistere en realistisk fare for slik flukt. For det første er motivasjonsspørsmålet aktuelt, og det er videre mulig at dårlig kunnskap omkring virksomhetssøk i FLOSS-miljøet kan få innvirkning på den status produktet etterhvert får. Det at en del mindre firmaer nå satser på søk med fri programvare bærer kanskje et vitnesbyrd omkring det økonomiske aspektet. Vil kortsiktighet og mulighet for personlig vinning utgjøre en reell trussel?

På side 105 kom det frem at altruisme neppe styrker motivasjonen for deltakelse i FLOSS-prosjekter. Det ble videre vist at personlig læring og status er to viktige ingredienser. Hos firmaer som produserer fri programvare kan det også være at utviklerne rett og slett får betalt for jobben, og derfor ikke behøver å motivere seg for arbeid på fritiden. Høy grad av realisme, som diskutert her, er antakelig mer eller mindre et krav for at motivasjonsmekanismene skal fungere. Sagt med andre ord er det fare for at muligheten for personlig økonomisk gevinst kan overskygge FLOSS-prosjekter. For enslige entusiaster er kanskje spørsmålet mindre aktuelt, men for en ledelse i en bedrift som i utgangspunktet satset på FLOSS kan det stille seg annerledes.

Hvis andre tilbud lokker er det viktig å opprettholde et tilstrekkelig nivå av motivasjon. Her ligger det potensielt en fare i at momentet om personlig status kan svekkes. Hvis kunnskapen om virksomhetssøk er lav i generell forstand, og spesielt i det tradisjonelle hackermiljøet, kan en utvikler oppleve at vedkommende ikke oppnår den status som synes fortjent. Sagt annerledes er det en fare for at andre hackere ikke forstår bragden når sluttresultatet løser problemer som disse hackerne ikke visste om.

Svenske MySQL [79] er et godt eksempel på fri programvare som utvikles med dobbel lisensiering, av et firma som tilbyr både brukerstøtte og andre former for garantier. Databasen deres kan benyttes forholdsvis fritt, men i gitte sammenhenger må det betales, for eksempel når produktet utgjør en del av et kommersielt produkt. Dette tilfellet demonstrerer at det er fullt mulig å satse på fri programvare, men det økende antallet lukkede ES-løsninger basert på blant annet Lucene forteller kanskje at markedet er annerledes her. Det gjøres ikke forsøk på markedsanalyser. Poenget er heller å støtte opp om påstanden om at det er penger å tjene på kombinasjonen "FLOSS + ES".

En positiv observasjon er at det finnes tegn på at kunder ønsker seg billigere søkeløsninger, og at IBMs, Microsofts og Googles inntog i mar-

kedet er med på å presse prisene nedover [24].

Ønsket om en slags avkastning

Å få noe igjen for strevet handler ikke alltid om kroner og øre. Spørsmålet her er om det virkelig er verdt alt strevet og all innsatsen å utvikle en komplett, åpen ES-plattform. Dette punktet er nok et oppgjør med enkeltes oppfatning av altruisme. For at et slikt prosjekt skal kunne vare lenge og utvikle seg over tid, må deltakerne og ikke minst initiativtakerne føle at investeringen gir en reell og konkret avkastning – og virksomhetene, kundene om man vil, må følge med på bølgen.

I første omgang må markedet overbevises om at fri programvare holder bra nok kvalitet. Heldigvis er dette i ferd med å slå igjennom på en rekke områder, som for eksempel operativsystemer eller databasesystemer. Overbevisningen må etterhvert også kunne gjelde søkesystemer generelt og virksomhetssøk spesielt. Brukere må føle seg sikre på at ES-produktet og støtteapparatet rundt holder ønsket kvalitet. Som ledd i prosessen med å overbevise bør produktet vise til relativt god nytte ut av boksen. Kall det gjerne markedsføring, men en overbevisning må trolig til.

Når denne jobben er gjort, må produktet i tillegg tydelig vise at det er bedre enn dagens tilbud innen fri programvare. Å gjøre virksomhetssøk med fri programvare kan nemlig føre til en motsatt, negativ reaksjon mot et nytt produkt. Det vil være mulig å spørre seg selv hvorfor dagens Lucene, Nutch, Solr og litt integrasjon ikke vil gjøre en bra nok jobb. Dette er enda en grunn til at det vil være lurt å satse på god funksjonalitet rett ut av pakken, samtidig med at produktet må kunne konfigureres i like høy grad som den eksisterende jungelen av søkefragmenter.

De tekniske utfordringene innenfor virksomhetssøk er ikke til å stikke under en stol, og på visse områder ligger kommersielle aktører milevis foran, iallfall tilsynelatende. De kommersielle har også mye å gå på når det gjelder krigføring mot fri programvare. Hva skjer hvis et fantastisk nyttig og fritt ES-produkt plutselig dukker opp? Det er neppe utenkelig at de tunge og etablerte aktørene vil svare med gratisversjoner eller åpne, enkle utgaver av sine egne plattformer. Dette kan de gjøre uten å røpe altfor mye. IBM har allerede gått til dette steget med lanseringen av sin "yahoo-utgave" av OmniFind. Med andre ord: Et nytt og fantastisk produkt i mellomsjiktet vil neppe få markedet for seg selv.

Sammenlagt handler ROI-begrepet også om *hvem* som får noe igjen for strevet.[4] Utfallet for et prosjekt kan bli svært annerledes om et firma setter igang, enn om en gruppe med kamerater utvikler produktet på fritiden. Kanskje er markedsposisjonen til de etablerte så sterk at de

[4] ROI står for *return on investment*.

ikke vil ha noe som helst å frykte nærmest uansett hva som skjer. De beholder jo ofte sin markedsposisjon gjennom å tilby en uerstattelig, komplett plattform. En analogi kan trekkes over til Windows; Microsoft føler seg ikke særlig truet på kort sikt, selv om GNU/Linux på mange måter tilbyr overlegne tekniske løsninger.

Realisme, ikke pessimisme

Dette avsnittet har sett litt på den store utfordringen og arbeidsmengden som vil følge i kjølvannet av et tenkt prosjekt. Selv om virksomhetssøk ikke er trivielt, er nok ikke alt svart. Kan hende er det lysglimt i enden av tunnelen.

Store og komplekse produkter innenfor den frie sfæren har startet i det små. Arbeidsmengdene kan overkommes ved å bruke for eksempel smidige metoder [1, 100], som i høy grad lar seg kombinere med fri programvare. Den største realismen i prosjektet ligger derfor kanskje i at det overlever hvis det finnes en naturlig plass, og går dukken hvis ingen vil ha det. Å tvinge igjennom løsninger hører kanskje hjemme i noen proprietære tilnærminger, men neppe hos fri programvare.

6.3 Problematiske utfordringer

Det forrige avsnittet handlet om hvilke forutseninger som må ligge til rette for å lage et godt, rikt og generelt produkt for virksomhetssøk i fri programvare. Disse forutsetningene er overkommelige, men det finnes en del problemer som – iallfall i dag – er litt for vanskelige å løse.

6.3.1 Teknologiens rolle

Hvor stor rolle spiller egentlig teknologi?

Hvis man fokuserer på selve søketeknologien, ligger ikke Lucene bak de kommersielle aktørene i hverken ytelse eller skalérbarhet (som forklart i seksjon 4.6 på side 89). Både Solr og Nutch kan suppleres med Apache Hadoop, og i tester har dette vist seg å være svært effektivt. Kvalitet, som definert i seksjon 4.5 på side 86, er imidlertid avhengig av litt andre faktorer, spesielt hva som skjer ved innmatingen av data – ikke minst språkbehandling. Her ligger de kommersielle leverandørene foran, men hvor mye har dette egentlig å si? Spørsmålet kan også vris andre veien; fri programvare er generelt kjent for god teknisk kvalitet, men hvor mye har det egentlig å si?

Mye taler for at det ikke først og fremst er teknologien som er avgjørende innen virksomhetssøk. Det kan være at dette vil endre seg i fremtiden, i takt med at søketeknologi blir mye mer utbredt. Da vil det

nok være fornuftig å ta utgangspunkt i den teknologisk beste løsningen, især hvis mye av rammeverket rundt er mer eller mindre likt. Sagt på en annen måte vil virksomheter som har god prosessering eller strukturering av data i egne ledd, følgelig velge den søkemotoren som gir best ytelse.

Inntil virksomhetssøk er blitt mer kjent, vil nok de kommersielle ha et fortrinn. De er kanskje store og klumpete, men de gir stort sett alt virksomhetene trenger. Virksomheter trenger helhetlige løsninger. De kommersielle utnytter denne avhengigheten i såkalt "vendor lock-in". Dette betyr at det blir vanskelig for kunden å gå over til et annet produkt, eller vanskelig å skifte ut deler og moduler av søkeløsningen sin, med produkter fra konkurrenter. En del av denne strategien henger tett sammen med skriftlige avtaler, lisenser og brukerstøtte fra de kommersielle leverandørene.

Det hjelper altså ikke så mye at fri programvare yter bra. For mange regnes det faktisk nærmest som en selvfølge at teknologien er i øverste klasse. Om noen år kan det være at flere virksomheter føler seg sikre nok til å utvikle løsninger selv, og dermed benytte seg av fri programvare. Jeg tør spekulere i at en slik åpen praksis og erfaring med fri programvare er det som skal til for at flere hackere skal oppdage potensialet til å lage en åpen, moden og generell plattform for virksomhetssøk.

6.3.2 Produkter og løsninger

Mesteparten av ES-markedet ser ut til å handle om spesielle løsninger skreddersydd for bestemte behov.

Så lenge utsagnet over er sant, vil lisenskostnader spille en mindre rolle når det skal velges mellom proprietære eller åpne løsninger. Ofte vil den største delen av kostnadene gå til tilpassing eller helhetlig utvikling av et større system som bruker virksomhetssøk. Selv om lisenskostnadene kan være i hundretusenkronersklassen, eller kanskje mer, vil de kunne bli små i den store sammenhengen. Dessuten er det jo ofte slik at de kommersielle leverandørene har avdelinger med løsningsarkitekter og konsulenter som kan bistå i alle ledd.

Hvordan skal fri programvare – som produkt – kunne konkurrere mot dette? Svaret er selvfølgelig at fri programvare må ha et aktivt miljø med avanserte brukere og utviklere, såkalte horisontale prosjektnett. Inntil disse er blitt etablert godt nok, vil kanskje mange virksomheter kvie seg for å velge fri programvare. Helt i den andre enden vil nok fri programvare kunne få et soleklart fortrinn innenfor virksomhetssøk, men om utviklingen noengang vil komme så langt gjenstår å se.

6.3.3 Mennesker, kompetanse og kunnskap

Er mennesker villige til å satse på noe de kan svært lite om?

De litt mindre virksomhetene vil ofte ha litt enklere krav. For disse vil det være lettere å finne fellesnevnere og utvikle en bedre ES-plattform ved hjelp av fri programvare. Men hvis nå en slik plattform utvikles, hvordan skal virksomheter klare å ta den i bruk? Her er det et dilemma:

- Mindre virksomheter som kan bruke en mer komplett plattform, vil kunne vegre seg fordi de ikke kjenner til teknologien...
- ...og de som kjenner til teknologien ser nok potensialet i å bruke dagens fri programvare til å sy egne løsninger.

Det vil uansett være en terskel for å ta i bruk virksomhetssøk.

Jeg mener altså at kompetanse og kunnskap vil være mangelvare i mange tilfeller. Dette vil medføre at virksomheter oppsøker hjelp hos spesialister, for eksempel ved å leie inn konsulenter. I svært stor grad er det de proprietære og lukkede søkeplattformene som tilbyr konsulenttjenester og komplette løsninger. Man havner altså i en sirkel.

Mennesker, kompetanse og kunnskap er avgjørende også for fri programvare. I seksjon 5.3 på side 105 ble ideen om altruisme som motivasjonsfaktor tilbakevist; tvert imot er mye av motivasjonen bak open source å lære, å oppnå høyere status og lignende. Måten å gjøre dette på er neppe å drive markedstilpasset konsulentvirksomhet, men heller å lage noe som er *teknisk elegant*. Gjør dette kombinasjonen ES/FLOSS mindre attraktiv?

Dette kan i hvert fall stå som en hypotese. Jeg er ikke fremmed for at ES-terrenget vil bli mer teknisk i fremtiden. En katalysator for dette kan være klarere problemstillinger og bedre fellesnevnere (som følge av at ES-fagfeltet blir bedre kjent). Det kan også godt tenkes at virksomhetssøk ikke trenger å skifte fokus for å tiltrekke seg utviklere. Straks noe får stor nok oppmerksomhet, tror jeg mulighetene er gode for at utviklere ser mulighetene for en personlig *netto avkastning*.

6.4 Konsekvenser

På mange måter seiler fri programvare sin egen sjø. Det gjelder å vite hvilke deler av FLOSS-prinsippet som kan brukes til hvilke formål. Det tradisjonelle hackermiljøet kan være en kilde til misforståelser for bedrifter ukjent med fri programvare. De samme bedriftene må i tillegg skjønne hvordan FLOSS-prosjekter kan kontrolleres. Bedrifter som ønsker å satse på fri programvare må gjøre dette riktig. Betyr dette dermed at fri programvare bare kan løse en viss type problemer?

Den tradisjonelle hackerbevegelsen og dets opphav ble beskrevet på side 102. Det tilkommer stadig nye generasjoner av hackere, og disse vil sannsynligvis se på verden med nye øyne. Allikevel kan hackerbevegelsen virke som noe fremmed eller lukket. Bedrifter som ønsker å dra nytte av fri programvare må skjønne hvordan hackerbevegelsen fungerer, iallfall hvis de ønsker å utnytte det fulle potensialet som fri programvare har.

Full utnyttelse fås nemlig neppe uten bruk av horisontale prosjektnett (se side 113). Det er vanskelig å ha kontroll over disse nettene i utgangspunktet, så da gjelder det å vite hvordan de skal utnyttes. Kanskje må mange bedrifter innse at de ikke kan kontrollere programvareprosjekter uten at de samtidig gir slipp på kontrollen på visse områder, iallfall hvis de ønsker seg "ekte fri programvare", og ikke bare åpen kildekode.

Et motargument mot at fri programvare er i sin egen verden, er det faktum at mange firmaer satser på "open source" eller fri programvare. Det gjelder å holde tunga rett i munnen og filtrere markedsspråk fra innhold når man hører dette fra bedrifter. Det er forskjell på fri programvare og åpen kildekode, men enda viktigere er det å komme til bunns i *hvordan* fri programvare brukes i et gitt programvareprosjekt. Er det visse utviklingsmetodikker som benyttes? Hvilken kontroll har man gjennom styring eller lisenser?

OSI [82] (se historien om fri programvare fra side 102) har heldigvis bidratt til at FLOSS er blitt mer kjent og akseptert for mange bedrifter. De driver naturligvis også et opplysningsarbeid som hjelper med forståelsen rundt fenomenet. Men det endrer ikke på FLOSS' særegenhet, og dermed kanskje også at enkelte prosjekter er mer egnet som fri programvare enn andre. Hvilke typer passer best til å bli løst ved hjelp av fri programvare?

Eksisterer det en slags dobbeltmoral?

Rundt årtusenskiftet begynte fri programvare å få mer oppmerksomhet hos store bedrifter og i media, mye takket være OSIs innsats (som nevnt i forrige avsnitt). Rent personlig tviler jeg på at alle firmaer som skilter med åpen kildekode vet nok om hva fri programvare er i realiteten. Mange drar selvfølgelig direkte nytte, men på den andre siden kan det være at noen utnytter et motefenomen på feilaktig grunnlag.

Spørsmålet er om fri programvare blir bedre eller dårligere stilt ved slike "falske profeter". Det er langt fra utenkelig at tunge aktører med store økonomiske interesser innenfor de tradisjonelle distribusjons- og utviklingsmetodene, vil kunne motarbeide fri programvare med vilje. Jeg skal ikke begi meg ut på konkrete konspirasjonsteorier, jeg prøver bare å argumentere for at *grunnlaget* for slike konspirasjoner finnes.

Kapittel 7

Konklusjon

> The best thing about the future
> is that it comes one day at a time.
>
> — *Abraham Lincoln*

Den opprinnelige planen for denne masteroppgaven var å lage et program for virksomhetssøk. Det var litt uklart hva den konkrete problemstillingen skulle være, men etterhvert kom jeg frem til hypotesen som kan oppsummeres slik:

- Det må være mulig å utvikle en rikere, generell plattform for virksomhetssøk ved hjelp av eksisterende fri programvare.

Det viste seg at svaret i mange tilfeller vil være "ja", men det må tas en rekke forbehold:

- Det er vanskelig å lage et produkt uten noen klar målgruppe. Målgruppen "virksomheter" er i mange tilfeller ikke konkret nok.
- De store virksomhetene vil uansett legge den største innsatsen i tilpassing, og da er det lite hensiktsmessig at utgangspunktet deres er altfor spesifikt.
- De høyeste kostnadene for de store virksomhetene ligger ikke i selve produktet, men i tilpassingen.

Mye av styrken til fri programvare ligger i at hver programbit ofte fokuserer kun på én ting. Dette er for eksempel tilfellet med Apache Solr. Resten av programvaren rundt må virksomheten skrive selv.

I sterk kontrast til dette står de kommersielle aktørene. Fast ESP er et eksempel på en meget omfattende plattform som kan løse en rekke forskjellige oppgaver, og mange virksomheter vil ikke ha behov for alt som

tilbys. Her oppstår det et lite tankekors; skal fri programvare bringes nærmere de kommersielle produktene ved at de blir store og kompliserte? Jeg tror ikke dette er en ideell løsning, og på mange måter gjenspeiles det i hva som allerede finnes av FLOSS-prosjekter for virksomhetssøk i dag.

Det som skal til er i første rekke at virksomhetssøk blir mer kjent og utbredt, slik at flere utviklere vil ha et forhold til det og få erfaring med det. Det finnes mange tegn på at verden beveger seg i denne retningen. I takt med dette blir det forhåpentligvis lettere å videreutvikle de frie prosjektene. Husk at fri programvare ikke bare betyr å gjøre kildekoden åpen. Det handler i tillegg om ressursforvaltning og utviklingsmønstre.

Jeg luftet ideene om et mulig prosjekt på Solr-utviklernes e-postliste. En av utviklerne, Mike Klaas, svarte følgende:

> I don't think that anyone is arguing that this product shouldn't exist in the open-source world, just that it shouldn't be part of Solr's mandate. It sounds like a cool project (though the closer you get to "commercial product" the more important support, packaging, marketing, etc. become—some of which are very difficult to achieve in a purely open-source setting).

Dette fremhever to av poengene mine; at fri programvare ofte fokuserer på å gjøre én, enkel ting veldig bra, og at en bedre plattform for virksomhetssøk vil være vanskelig å realisere som et godt FLOSS-prosjekt.

Oppsummering

Diskusjonen i denne oppgaven kan oppsummeres i tre punkter. Det første er at det finnes en hel del eksisterende fri programvare som kan brukes innen virksomhetssøk, og at disse kan kombineres på ulike måter. I tillegg må en rekke andre forutseninger være på plass for at et bedre, generelt ES-produkt skal kunne utvikles. Til sist finnes det en del problematiske utfordringer som gjør alt litt for vanskelig – iallfall i denne omgang.

Det finnes en del mer programvare for virksomhetssøk enn jeg var klar over. Ikke minst finnes Apache Solr – et særdeles lovende produkt som det absolutt bør bygges videre på. Kanskje bør noe av lærdommen fra Apache Nutch tas med, men som diskusjonen viste lar disse produktene seg litt vanskselig kombinere. Andre fremgangsmåter er å bruke rammeverk som for eksempel Spring. Det store utvalget av åpne biblioteker for tekstanalyse, bindeledd og lignende er i hvert fall til god hjelp.

Siden det allerede finnes flere prosjekter som har den funksjonaliteten jeg i utgangspunktet hadde tenkt å utvikle for denne oppgaven, blir det mer interessant å bringe fri programvare for virksomhetssøk enda et steg videre. Dette er svært krevende med tanke på å finne fellesnevnere i virksomheter, og jeg stiller også spørsmålstegn ved hvor lett det er å skape en aktiv brukerbase all den tid virksomhetssøk er relativt lite kjent. I tillegg er situasjonen litt preget av at mindre aktører lett kan tjene penger på å spesialtilpasse eller videreutvikle fri programvare for virksomhetssøk, og derfor ikke velger å utvikle en åpen plattform.

De største problemene ser allikevel ut til å være at virksomhetssøk er preget av skreddersøm og løsninger, snarere enn produkter. Det blir til syvende og sist mennesker, kompetanse og kunnskap som blir avgjørende.

Jeg påstår at det er umulig å utvikle en god og helhetlig plattform i omfanget av en masteroppgave. Naturligvis var det ikke meningen å lage et altomfattende produkt, men funnene viser også at det er vanskelig å komme videre i det hele tatt – *i generell retning*. Således hadde denne oppgaven vært bedre tjent med å fokusere på et bestemt problem innenfor fri programvare og virksomhetssøk, og kanskje tilpasse Apache Solr til et bestemt problem. Men dette var selvfølgelig ikke mulig – Solr eksisterte ikke engang da denne oppgaven ble påbegynt. Det sier litt om hvor raskt utviklingen går innen for dette feltet, og jeg tror at svært mye mer vil skje i årene fremover.

Betyr det egentlig så mye?

Det er noen ting som rett og slett bare ikke går fordi problemet søkes løst i fremmede domener. Dette krever litt forklaring.

I løpet av denne oppgaven har det flere ganger blitt nevnt at virksomhetssøk er for lite kjent. En konsekvens av dette er at hackerkulturen heller ikke kjenner til søketeknologien like mye som en del andre fagområder.

Et optimalt FLOSS-prosjekt utnytter horisontale prosjektnett bestående av hackere, som regel geografisk spredt. Disse jobber mye ut fra egen motivasjon og løser problemer i sin hverdag. Alternativt løser de problemer i andre hackeres hverdag, rett og slett for å bli synlige og dermed oppnå status.

Virksomhetssøk handler først og fremst om bedrifter, og om problemer som finnes i forretningsverdenen. Her er fokuset mer på lønnsomhet. Disse to domenene synes altså å ligge litt langt fra hverandre.

Imidlertid er det fullt mulig for virksomheter å utvikle sine egne, åpne løsninger for virksomhetssøk. Fri programvare *krever* på ingen måte verdensomspennende nettverk av mer eller mindre frivillige, noe som

særlig har vært bevist i det siste. Apache Lucene er mer populært enn noen gang før, og stadig mer innen virksomhetssøk.

Imidlertid skal det et tilstrekkelig antall bedrifter, systemutviklere og problemer til for at oppmerksomheten blir stor nok. Når det skjer, vil tiden kanskje bli moden for å løfte kombinasjonen av virksomhetssøk og fri programvare til det neste nivået.

Bibliografi

[1] The agile manifesto.
http://agilemanifesto.org/.

[2] ANAND, V. Seagate outlines the future of storage. *Hardware Zone* (2006). http://www.hardwarezone.com/articles/view.php?cid=30&id=1805.

[3] Apache Fortress – rammeverk for å lage IoC-beholdere.
http://excalibur.apache.org/fortress/.

[4] Apache Hadoop – rammeverk for clustering; implementasjon av map/reduce og et distribuert filsystem.
http://lucene.apache.org/hadoop/.

[5] Apache Jakarta LARM – søkemotor basert på Apache Lucene.
http://larm.sourceforge.net/.

[6] Apache Jakarta ORO – Java API for avansert tekstbehandling og regulære uttrykk.
http://jakarta.apache.org/oro/.

[7] Apache Jakarta POI – Java API for Microsoft Office-filformat.
http://jakarta.apache.org/poi/.

[8] Apache Lucene Java – Java-implementasjon av Apache Lucene.
http://lucene.apache.org/java/.

[9] Apache Lucene – IR-bibliotek.
http://lucene.apache.org/.

[10] Apache Nutch – søkemotor for web basert på Apache Lucene Java.
http://lucene.apache.org/nutch/.

[11] Apache Solr – enterprise search (ES) basert på Apache Lucene Java.
http://lucene.apache.org/solr/.

[12] Apache Struts – rammeverk for å lage webapplikasjoner i Java.
http://struts.apache.org/.

[13] Apache Tomcat – Java servlet container.
http://tomcat.apache.org/.

[14] Apache Xerces2 (Java) – Java API for XML.
http://xerces.apache.org/xerces2-j/.

[15] AspectJ – utvidelse av Java-språket for aspektorientert programmering (AOP).
http://www.eclipse.org/aspectj/.

[16] Autonomy: Enterprise Search.
http://www.autonomy.com/.

[17] BACKUS, J. W. Can programming be liberated from the von neumann style? A functional style and its algebra of programs. *Communications of the ACM 21*, 8 (aug 1977), 613–641.

[18] BAEZA-YATES, R. OG RIBEIRO-NETO, B. *Modern Information Retrieval*. ACM Press, New York, NY, USA, 1999.

[19] BLUMBERG, R. OG ATRE, S. The problem with unstructured data. *DM Review* (feb 2003).

[20] BONDI, A. B. Characteristics of scalability and their impact on performance. Fra *WOSP '00: Proceedings of the 2nd international workshop on Software and performance* (New York, NY, USA, 2000), ACM Press, s. 195–203.

[21] BRIN, S. OG PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 107–117.

[22] BROOKS, FREDERICK P., J. The mythical man-month. Fra *The Mythical Man-Month: Essays on Software Engineering*, anniversary ed. Addison-Wesley Publishing Company, Inc., Reading, MA, USA, 1995, s. 13–28.

[23] BROOKS, FREDERICK P., J. Ten pounds in a five-pound sack. Fra *The Mythical Man-Month: Essays on Software Engineering*, anniversary ed. Addison-Wesley Publishing Company, Inc., Reading, MA, USA, 1995, s. 97–106.

[24] BROWN, M., MOORE, C. OG FOSSNER, L. The forrester wave: Enterprise search platforms, Q2 2006. http://api-quebec.ca/_doc/infolettres/2006/septembre/the_forrester_wave%_enterprise_search_platforms_q2_2006.pdf, 2006.

[25] Carrot^2 – rammeverk for å bygge klynger av søkemotorer.
http://www.carrot2.org/.

[26] CLAUSEN, H. FAST søkemotor som indeksaksessmetode i en relasjonsdatabase. Hovedoppgave, Institutt for informatikk, Universitetet i Oslo, 2004.

[27] Compass – rammeverk for å ta i bruk Apache Lucene i applikasjoner basert på Spring og Hibernate. http://www.opensymphony.com/compass/.

[28] CREESE, G. Information scarcity to information overload. *DM Review Magazine* (jan 2007). http://www.dmreview.com/find_content.php?content_id=1072477.

[29] CyberNeko – Java API for HTML. http://people.apache.org/~andyc/neko/doc/html/.

[30] DBSight: Full-text database search platform/engine. http://www.dbsight.net/.

[31] DEFAZIO, S., DAOUD, A., SMITH, L. A. OG SRINIVASAN, J. Integrating IR and RDBMS using cooperative indexing. Fra *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 1995), ACM Press, s. 84–92.

[32] What is del.icio.us? http://del.icio.us/about/.

[33] DMITRIEV, P. A., EIRON, N., FONTOURA, M. OG SHEKITA, E. Using annotations in enterprise search. Fra *WWW '06: Proceedings of the 15th international conference on World Wide Web* (New York, NY, USA, 2006), ACM Press, s. 811–817.

[34] dom4j – Java API for XML. http://www.dom4j.org/.

[35] Eclipse – åpen utviklingsplattform og IDE spesielt for Java. http://www.eclipse.org/.

[36] Egothor – Java search engine and supplemental libraries. http://www.egothor.org/.

[37] Endeca: Enterprise Search. http://endeca.com/.

[38] Fast Search & Transfer: Enterprise Search. http://www.fastsearch.com/.

[39] Free software foundation (fsf). http://www.fsf.org/.

[40] FREUND, L. OG TOMS, E. G. Enterprise search behaviour of software engineers. Fra *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2006), ACM Press, s. 645–646.

[41] FREYNE, J., FARZAN, R., BRUSILOVSKY, P., SMYTH, B. OG COYLE, M. Collecting community wisdom: integrating social search & social navigation. Fra *IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces* (New York, NY, USA, 2007), ACM Press, s. 52–61.

[42] GARCIA-MOLINA, H., ULLMAN, J. D. OG WIDOM, J. D. *Database Systems: The Complete Book*. Pearson Education Limited, Upper Saddle River, NJ, USA, 2002.

[43] GATES, B. An open letter to hobbyists. `http://www.blinkenlights.com/classiccmp/gateswhine.html`.

[44] GJERULL, N. F. Open source development in developing countries: The HISP case in Ethiopia. Masteroppgave, Institutt for informatikk, Universitetet i Oslo, 2006.

[45] GNU Licenses. `http://www.gnu.org/licenses/`.

[46] GNU: The Free Software Definition. `http://www.gnu.org/philosophy/free-sw.html`.

[47] GNU: What is Copyleft? `http://www.gnu.org/copyleft/`.

[48] Google (søkemotor). `http://www.google.com/`.

[49] GOSPODNETIC, O. OG HATCHER, E. *Lucene in Action*. Manning Publications Co., Greenwich, CT, USA, 2005.

[50] HARROP, R. OG MACHACEK, J. *Pro Spring*. Apress, Berkeley, CA 94710, USA, 2005.

[51] HAWKING, D. Challenges in enterprise search. Fra *ADC '04: Proceedings of the fifteenth Australasian database conference* (Darlinghurst, Australia, Australia, 2004), Australian Computer Society, Inc., s. 15–24.

[52] HENNESSY, J. The future of systems research. *Computer 32*, 8 (1999), 27–33.

[53] Hibernate – rammeverk for avbildning (mapping) mellom relasjonsdatabaser og programobjekter i Java og .NET.
http://www.hibernate.org/.

[54] ht://Dig – åpen søkemotor skrevet i C++.
http://www.htdig.org/.

[55] iBATIS – rammeverk for avbildning (mapping) mellom relasjonsdatabaser og programobjekter.
http://ibatis.apache.org/.

[56] JavaSWF2 – JAVA API for Flash.
http://www.anotherbigidea.com/javaswf/.

[57] Java DOM API.
http://www.mozilla.org/projects/blackwood/dom/.

[58] Java Excel API (jxl).
http://www.andykhan.com/jexcelapi/.

[59] Java ID3 Tag Library.
http://javamusictag.sourceforge.net/.

[60] Jaxen – universell Java-motor for XPath.
http://jaxen.codehaus.org/.

[61] JDOM – Java API for XML.
http://www.jdom.org/.

[62] JIRA – verktøy for bug tracking, saksbehandling og prosjektstyring.
http://www.atlassian.com/software/jira/.

[63] JOGALEKAR, P. OG WOODSIDE, M. Evaluating the scalability of distributed systems. Fra *Proceedings of the Thirty-First Hawaii International Conference on System Sciences* (1998), vol. 7, IEEE Computer, s. 524–531.

[64] JTidy – Java API for HTML.
http://jtidy.sourceforge.net/.

[65] karakoram.
http://cephas.net/projects/karakoram/.

[66] Kvasir (norsk søkemotor).
http://www.kvasir.no/.

[67] LANCASHIRE, D. The fading altruism of open source development. *First Monday 6*, 12 (2001). http://firstmonday.org/issues/issue6_12/lancashire/index.html.

[68] LERNER, J. OG TIROLE, J. Economic perspectives on open source. Fra *Perspectives on Free and Open Source Software*, J. Feller, B. Fitzgerald, S. A. Hissam, og K. R. Cusumano, Eds. The MIT Press, Cambridge, MA, USA, 2005, s. 47–78.

[69] LIE, Å. K. Fritt søk med FAST søkemotor integrert i PostgreSQL relasjonsdatabase. Hovedoppgave, Institutt for informatikk, Universitetet i Oslo, 2004.

[70] Lius (lucene index update and search). http://sourceforge.net/projects/lius/.

[71] Log4j. http://logging.apache.org/log4j/.

[72] LOHR, S. John W. Backus, 82, Fortran developer, dies. *The New York Times* (2007). http://www.nytimes.com/2007/03/20/business/20backus.html.

[73] Lucene benchmarks. http://lucene.apache.org/java/docs/benchmarks.html.

[74] MAIER, M. W. OG RECHTIN, E. *The Art of Systems Architecting*, second ed. CRC Press LLC, Boca Raton, FL, USA, 2002.

[75] MANNING, C. D., RAGHAVAN, P. OG SCHÜTZE, H. *Introduction to Information Retrieval*. Cambridge University Press, 2007. Ennå ikke offisielt utgitt. Siste versjon ligger på http://www-csli.stanford.edu/~schuetze/information-retrieval-book.html.

[76] MA, L., SU, Z., PAN, Y., ZHANG, L. OG LIU, T. RStar: an RDF storage and query system for enterprise resource management. Fra *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management* (New York, NY, USA, 2004), ACM Press, s. 484–491.

[77] MILLEN, D. R., FEINBERG, J. OG KERR, B. Dogear: Social bookmarking in the enterprise. Fra *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems* (New York, NY, USA, 2006), ACM Press, s. 111–120.

[78] MUKHERJEE, R. OG MAO, J. Enterprise search: Tough stuff. *Queue 2*, 2 (2004), 36–46.

[79] MySQL. http://www.mysql.com/.

[80] Newegg.com (amerikansk nettbutikk).
`http://www.newegg.com/`.

[81] Novell SUSE Linux Enterprise.
`http://www.novell.com/linux/`.

[82] Open Source Initiative.
`http://opensource.org/`.

[83] Open source software in Java.
`http://java-source.net/`.

[84] PDFBox: Java documentation.
`http://www.pdfbox.org/javadoc/index.html`.

[85] PDFBox – Java API for PDF.
`http://www.pdfbox.org/`.

[86] PORTER, M. F. An algorithm for suffix stripping. *Program 14*, 3 (1980), 130–137.

[87] PostgreSQL.
`http://www.postgresql.org/`.

[88] PRIEBE, T. OG PERNUL, G. Towards integrative enterprise knowledge portals. Fra *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management* (New York, NY, USA, 2003), ACM Press, s. 216–223.

[89] RAYMOND, E. A brief history of hackerdom. Fra *The Cathedral and the Bazaar: Musings on Linux and Open Source by and Accidental Revolutionary*, revised ed. O'Reilly Media, Inc., Sebastopol, CA, USA, 2001, s. 1–17.

[90] RAYMOND, E. The cathedral and the bazaar. Fra *The Cathedral and the Bazaar: Musings on Linux and Open Source by and Accidental Revolutionary*, revised ed. O'Reilly Media, Inc., Sebastopol, CA, USA, 2001, s. 19–63.

[91] RAYMOND, E. Homesteading the noosphere. Fra *The Cathedral and the Bazaar: Musings on Linux and Open Source by and Accidental Revolutionary*, revised ed. O'Reilly Media, Inc., Sebastopol, CA, USA, 2001, s. 65–111.

[92] Red Hat Linux.
`http://www.redhat.com/`.

[93] Red Piranha community edition.
`http://red-piranha.sourceforge.net/`.

[94] Regain.
http://regain.sourceforge.net/.

[95] SAND, K. A. Question answering using syntactic patterns in a contextual search engine. Masteroppgave, Institutt for informatikk, Universitetet i Oslo, 2006.

[96] SAX – Java API for XML.
http://www.saxproject.org/.

[97] SCHMID, P. OG ROOS, A. 15 years of hard drive history: Capacities outran performance. *Tom's Hardware Guide* (2006). http://www.tomshardware.com/2006/11/27/15-years-of-hard-drive-history/.

[98] Sesam (norsk søkemotor).
http://www.sesam.no/.

[99] Snowball – språk for prosessering av strenger, spesielt designet for å lage stemming-algoritmer; inneholder implementasjoner av stemmere for en del vanlige språk.
http://snowball.tartarus.org/.

[100] SOMMERVILLE, I. *Software Engineering*, seventh ed. Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England, 2004.

[101] SourceForge.
http://sourceforge.net/.

[102] Spring Framework.
http://www.springframework.org/.

[103] STALLMAN, R. The free software movement and the future of freedom (forelesning i Zagreb, 9. mars 2006). http://fsfeurope.org/documents/rms-fs-2006-03-09.en.html.

[104] STENMARK, D. Searching the intranet: Corporate users and their queries. Fra *ASIST '05: Proceedings of the 68th Annual Meeting of the American Society for Information Science and Technology* (2005).

[105] Sun Microsystems: Free and Open Source Java.
http://www.sun.com/software/opensource/java/.

[106] SUREKA, A. Making unstructured data findable using tagging and annotation. *BI Report* (mai 2006).

[107] THOMPSON, E. Gartner: Strategic technologies for 2006 and 2016. http://www.ergogroup.no/upload/plenum_edmund_thompsonpdf.pdf. Presentasjon i IT-tinget Tønsberg 20.09.2006.

[108] Tritonus – Java API for lydbehandling. http://tritonus.org/.

[109] TSOI, A. C., HAGENBUCHNER, M. OG SCARSELLI, F. Computing customized page ranks. *ACM Trans. Inter. Tech. 6*, 4 (2006), 381–414.

[110] TSOI, A. C. Structure of the internet? Fra *Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing* (2001), IEEE Conference Proceeding, s. 449–452.

[111] UKELSON, J. Combining structured, semistructured and unstructured data in business applications. *DM Direct* (des 2006). http://www.dmreview.com/find_content.php?content_id=1069202.

[112] BCG. The Boston Consulting Group/OSTG hacker survey (release 0.73). Fra *O'Reilly Open Source Conference* (2002). http://www.ostg.com/bcg/.

[113] VALMOT, O. R. Sjefforsker i FAST tar oppgjør med Google. *Teknisk ukeblad 153*, 17 (2006), 6–7.

[114] VALMOT, O. R. Søkemotoren overtar for databasen. *Teknisk ukeblad 153*, 18 (2006), 40–42.

[115] Vmware, Inc. http://www.vmware.com/.

[116] VON HIPPEL, E. Open source software projects as user innovation networks. Fra *Perspectives on Free and Open Source Software*, J. Feller, B. Fitzgerald, S. A. Hissam, og K. R. Cusumano, Eds. The MIT Press, Cambridge, MA, USA, 2005, s. 267–278.

[117] W3C: Resource Description Framework (RDF). http://www.w3.org/RDF/.

[118] WALTER, C. Kryder's law. *Scientific American 293*, 2 (2005), 32–33.

[119] WHEELER, D. A. Why open source software / free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers! http://www.dwheeler.com/oss_fs_why.html, nov 2005.

[120] Wikia Search. http://search.wikia.com/.

[121] Wikipedia.
http://www.wikipedia.org/.

[122] WILLIAMS, S. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly & Associates, Inc., 2002. HTML-versjon (http://www.faifzilla.org/).

[123] XML Path Language (XPath).
http://www.w3.org/TR/xpath.

[124] YouTube – sosialt nettsted for deling av video.
http://www.youtube.com/.

[125] Zilverline.
http://www.zilverline.org/.