Forord

Ole Andreas Grytdal
Bjørn Kristian Hilstad

Denne hovedoppgaven er utført som en siste del av utdanningen som leder fram til tittelen Sivilingeniør ved Institutt for Datateknikk og Telematikk, Norges Tekniske Høgskole i tidsrommet 5 september 1995 til 19 januar 1996. Oppgaven er formulert etter et forslag fra Gruppe for Algoritmekonstruksjon ved IDT.

Vi vil takke faglærer professor Arne Halaas og veilederne Tor Egge og Øystein Grøvlen for assistanse under oppgaven.

Vi vil ellers takke Asgaut Eng og Harald Evensen som har kommet med nyttige tips og triks under programmeringen. Takk også til Jan Tore Markussen for sitt flotte bilde på forsiden av denne diplomen.

Tilslutt vil vi takke de andre medstudentene ved IDT for en kjempefin tid og et godt miljø, samt alle avbrytelsene på kontoret.

Trondheim, fredag 19 januar 1996

Ole Andreas Grytdal   Bjørn Kristian Hilstad

Sammendrag

Ole Andreas Grytdal
Bjørn Hilstad

En oppsummering av oppgaven

Denne oppgaven omhandler utviklingen av en ny søkemaskin for WWW dokumenter. Et program henter automatisk inn sider fra WWW servere over hele Norge og lagrer disse på lokal disk.
I motsetning til tradisjonelle søkesystemer på WWW, skal vi ikke bruke indeksbaserte metoder for å gjøre dokumentmassen søkbar. Vi benytter et MS160-kort som er utviklet ved Norges Tekniske Høyskole av Professor Arne Halaas. Kortet tilbyr ultra-hurtig søking i tekst og kan søke i dokumentmassen med en hastighet av 160 Mb/s. En begrensning med kortet i denne sammenhengen er at det kun har plass til 160 Mb i minnet. Dette blir fort for lite selv om vi bare retter tjenesten vår mot norske WWW-dokumenter.
Søketjenesten er implementert og kjører på en Intel 486DX2 66 som kjører NetBSD 1.0. Søkemulighetene er noe begrenset, men ordsøk, setningsøk, trunkert søk, domenebegrensning, case sensitive og case insensitive søk og resultatrangering (vha. vekttildeling) fungerer.
En WWW-robot som kan hente inn ca. 4000 dokumenter om dagen ble utviklet til dokumentinnsamlingen. Et slikt verktøy legger relativt stor belastning på nettverksressurer. Når vi i tillegg vet at det er flere titalls lignende programmer i aktivitet på internettet, skjønner vi at det kan bli nødvendig med andre søke og - innsamlingsmetoder i fremtiden. På flere hold jobbes det med alternativer til slik søking. Blant disse tar vi for oss GlimpseHTTP og Hyper-G.

Innholdsfortegnelse

Forord 1
Sammendrag 3
Kapittel 1 : Innledning 17

1.0 Introduksjon 18
- 1.1 Oppgave tekst : 18
- 1.2 Diplom gruppen og veiledere 18
- 1.3 Rapport innhold 18

2.0 World-Wide Web 19
- 2.1 Internet 19
- 2.2 Hva er WWW 21
- 2.3 Hypertext og hypermedia 21
- 2.4 Hva finnes på WWW 22
- 2.5 Bruk av World Wide Web 23

Kapittel 2 : Dagens situasjon : Søkemaskiner og World Wide Web. 27

3.0 Introduksjon 28

4.0 Bruk av World Wide Web i dag 28
- 4.1 Hva er en robot i denne sammenheng ? 29

5.0 Oversikt over dagens søkemaskiner 30
- 5.1 HARVEST 30
  - 5.1.1 Generell informasjon 30
  - 5.1.2 Historie / bakgrunn 31
  - 5.1.3 Datainnsamlingsprosess 31
  - 5.1.4 Indekseringsprosess 31
  - 5.1.5 Søkemuligheter 32
- 5.2 Infoseek 32
  - 5.2.1 Generell informasjon 32
  - 5.2.2 Historie / bakgrunn 32
  - 5.2.3 Datainnsamlingsprosess og indekseringsprosess 33
  - 5.2.4 Søkemuligheter 33
- 5.3 OpenText 33
  - 5.3.1 Generell informasjon 33
  - 5.3.2 Historie / bakgrunn 34
  - 5.3.3 Datainnsamlingsprosess 34
  - 5.3.4 Indekseringsprosess 35
  - 5.3.5 Søkemuligheter 35
- 5.4 Inktomi 36
  - 5.4.1 Generell informasjon 36
  - 5.4.2 Historie / bakgrunn 36
  - 5.4.3 Datainnsamlingsprosess og Indekseringsprosess 36
  - 5.4.4 Søkemuligheter 37

5.5 AltaVista 37
5.5.1 Generell informasjon 37
5.5.2 Historie / bakgrunn 37
5.5.3 Datainnsamlingsprosess og Indekseringsprosess 38
5.5.4 Søkemuligheter 38
5.6 ALIWEB 39
5.6.1 Generell informasjon 39
5.6.2 Historie / bakgrunn 39
5.6.3 Datainnsamlingsprosess og Indekseringsprosess 40
5.6.4 Søkemuligheter 40
5.7 LYCOS 41
5.7.1 Generell informasjon 41
5.7.2 Historie / bakgrunn 41
5.7.3 Datainnsamlingsprosess og Indekseringsprosess 41
5.7.4 Søkemuligheter 43
5.8 WEBCRAWLER 43
5.8.1 Generell informasjon 43
5.8.2 Historie / bakgrunn 43
5.8.3 Datainnsamlingsprosess 44
5.8.4 Indekseringsprosess 44
5.8.5 Søkemuligheter 44
5.9 MOMSpider 45
5.9.1 Generell informasjon 45
5.9.2 Historie / bakgrunn 45
5.10 Fish Search 45
5.10.1 Generell informasjon 45
5.10.2 Historie / bakgrunn 46
5.10.3 Søkemuligheter 46
5.11 World Wide Web Wanderer 47
5.11.1 Historie / bakgrunn 47
5.12 World Wide Web Worm 47
5.12.1 Generell informasjon 47
5.12.2 Historie / bakgrunn 48
5.12.3 Datainnsammlingsprosess og indekseringsprosess 48
5.12.4 Søkemuligheter 48
5.13 Yahoo 48
5.13.1 Generell informasjon 48
5.13.2 Historie / bakgrunn 48
5.13.3 Datainnsamlingsprosess og Indekseringsprosess 49
5.13.4 Søkemuligheter 49
5.14 Kvasir 50
5.14.1 Generell informasjon 50
5.14.2 Historie / bakgrunn 50
5.14.3 Søkemuligheter 50
5.15 Origo Index 50
5.15.1 Generell informasjon 50
5.15.2 Historie / bakgrunn 51
5.15.3 Datainnsamlingsprosess 51
5.15.4 Søkemuligheter 51
5.16 Hvilket søkesystem er best ? 51
5.17 Motivasjon for oppgaven 53

6.0 Hvor stort er WWW? 54
6.1 Definisjoner 54
6.2 Størrelsen av WWW 55
6.2.1 Undersøkelse gjort av Lycos 55
6.2.2 Vårt estimat 57
7.0 Fremtiden og søking på World Wide Web 58
7.1 En ny generasjon servere 60
Kapittel 3 : En introduksjon til MS160 61
8.0 Ms160 62
8.1 Historie 62
8.2 Hva er MS160 62
8.3 Funksjonell beskrivelse av Ms160 62
8.4 Datapumpens funksjonalittet 63
8.5 Oppsett av MS 160 64
8.5.1 Data Ruter konfigurasjon 64
8.5.2 Oppsett av søkevinduene 68
8.5.3 Rapporterings modus 69
8.5.4 Konfigurasjon av treff masken 69
8.6 Oppkjøring av MS160 71
Kapittel 4 : Kravspesifikasjon for WWW-robot 73
9.0 Introduksjon 74
9.1 Hensikten med dette dokumentet 74
9.2 Organisering av dokumentet 74
10.0 Generell systembeskrivelse 75
10.1 System kontekst 75
10.2 System komponenter 77
10.2.1 Kontroller 77
10.2.2 Dokumentslave 77
10.2.3 Konfigurasjonsmodul 78
10.2.4 CERN LibWWW for Perl 78
10.2.5 Rapporteringsmodul 79
10.2.6 Eget bibliotek 79
10.2.7 Andre hjelpeprogrammer 79
10.3 Datafiler 80
10.3.1 Logg-fil 80
10.3.2 URL-fil 80
10.3.3 Resterende-fil 80
10.3.4 Robot-fil 81
10.3.5 Unngå-fil 81
11.0 Ikke-funksjonelle krav 81
11.1 Hardware krav 81
11.2 Software krav 82
11.3 Andre krav 82
12.0 Funksjonelle krav 83

12.1 Idenifiserbarhet 83
12.2 Ressursvennlighet 84

13.0 Konfigurasjon av WWW-roboten 86
13.1 Kommandolinjeparametre 87
13.2 Konfigurasjonsfilformat 87
13.2.1 Name 89
13.2.2 FileRoot 89
13.2.3 StartURL 89
13.2.4 MaxDepth 89
13.2.5 Timeout 89
13.2.6 Excludefile 89
13.2.7 Niceness 89
13.2.8 Schemes 90
13.2.9 OperatorEmail 90
13.2.10 Domains 90
13.2.11 Filetypes 90
13.3 Defaultverdier 91

14.0 Implementasjonskrav 92
14.1 Hardware 92
14.2 Software 92
14.3 Enkelt vedlikehold 92
14.3.1 Lesbar kode 92
14.3.2 Uavhengighet mellom software moduler 92
14.3.3 Systemstrukturen 92

Kapittel 5 : Kravspesifikasjon for MS160 søkemaskin 95

15.0 Generell system beskrivelse 96
15.1 System kontekst 96
15.2 System komponenter 96
15.3 Søkemaskinens komponenter 99
15.3.1 Ekstern program del 100

16.0 Ikke funksjonelle krav 100
16.1 Hardware krav 100
16.2 Software krav 101
16.3 Andre krav 101

17.0 Funksjonelle krav 102
17.1 Søkemaskinen 102
17.1.1 Fler-kriterie søk 102
17.1.2 Domene søk 104
17.1.3 Wildcards på ord og eksakt søk 104
17.1.4 Maksimum treff 104
17.1.5 Samtidige søk 105
17.2 Database oppslag 106
17.3 Postprosessering av innsamlede data 106
17.4 Søkegrensesnitt 107
17.5 Resultat presentasjon 109

18.0 Implementation krav 110
18.1 Hardware 110
18.2 Software 110
Kapittel 6 : Konstruksjons-spesifikasjon for WWW-roboten 111
19.0 introduksjon 112
20.0 Funksjonsbeskrivelser 112
20.1 robot.pl 112
20.1.1 sig_int_handler 112
20.1.2 logg_message 112
20.1.3 check_URL 112
20.1.4 logg_URL 113
20.1.5 is_html 113
20.1.6 sjekk_avoid 113
20.1.7 sjekk_domener 113
20.1.8 sjekk_schemes 113
20.1.9 sjekk_URL 114
20.1.10 sjekk_URL_queue 114
20.1.11 lagre_URL 114
20.1.12 traverser_link 115
20.1.13 check_speed 115
20.1.14 load_restfile 115
20.2 OURConfig.pl 115
20.2.1 parse_config_file 116
20.2.2 parse_command_line 116
20.3 OURlib.pl 116
20.3.1 ip_to_hostname 116
20.3.2 hostname_ti_ip 116
20.3.3 timestamp 116
20.4 OURreport.pl 117
20.4.1 send_email 117
20.4.2 add_to_report 117
20.5 filesort.pl 117
20.5.1 domaincompare 117
20.6 filemake.pl 117
20.7 domainfile.pl 118
20.7.1 finn_start 118
20.7.2 finn_slutt 118
20.7.3 domaincompare 118
20.8 statgen.pl 119
Kapittel 7 : Konstruksjons-spesifikasjon av søkemasin 121
21.0 Innledning 122
22.0 Bruk av gammel kildekode 122
22.1 Ftpsearch 122

22.2 Software driver 122
22.3 Datastruktur 122
22.3.1 WindowS 123
22.3.2 HitS 123
22.3.3 HitInfoS 123
22.3.4 QueueS 123
22.3.5 RouteS 123
22.3.6 RangeS 123
22.3.7 MSConfigS 124
22.3.8 RequestS 124
23.0 Nye program moduler 126
23.1 Postprosessering 126
23.2 Innlasting av data på kort 127
23.3 Start av søkeprogram 128
23.3.1 CGI-skript 128
23.3.2 Tolking av søkekriteriene 128
23.4 Oppsett av søkemaskinen 129
23.4.1 Vindu oppsett 129
23.4.2 Duplisering av vinduer 130
23.4.3 Oppsett av HitMask 131
23.5 Søket 133
23.5.1 Gjennomgang av datamengden 133
23.6 Oppslag av treff 133
23.6.1 Beregning av offset for oppslag 133
23.6.2 Domenebegrensning 133
23.6.3 Oppslag av URL på basis av offset i fil 134
23.6.4 Første og beste søk 136
23.6.5 Logg funksjon 136
Kapittel 8 : Implementasjons-beskrivelse 137
24.0 Implementasjon av søkemaskin 138
24.1 Implementering 138
24.2 Hva er implementert og testrapport 138
24.2.1 Innlastnings program 138
24.2.2 Konverterings program for etter prosessering 139
24.2.3 Domene søk 139
24.2.4 Wildcards 140
24.2.5 Maksimum antall treff 140
24.2.6 Låsemekanimse 141
24.2.7 Oppslag i DBM-fil 141
24.2.8 Søkegrensesnitt 141
24.2.9 Minst 1 av N søkekriteria 142
24.2.10 Testresultater av minst 1 av N søkekriteria 143
24.2.11 Beste resultater / Første resultater 143
24.2.12 Tidstempling av søk 144
24.2.13 Bruk av offset fil i minne eller på DBM-fil 144
24.3 Hva som ikke ble implementert 145
24.3.1 Flere søkekriteria 145
24.3.2 Logg funksjon 145
24.3.3 Søkegrensesnittet 145

25.0 Forbedringer av søkemaskinen 146
25.0.1 Tolker og bedre uttrykk 146
25.0.2 Bedre resultatlisting 146
25.0.3 Bedre spesifisering av hva en vil søke i 146
26.0 Forbedringer av roboten 147
Prosjektevaluering 149
27.0 Prosjektevaluering 150
27.1 Prosjekt organisering 150
27.2 Faser i diplomoppgaven 150
27.2.1 Tidsskjema 150
27.2.2 Forstudium 150
27.2.3 Kravspesifikasjon 151
27.2.4 Konstruksjonsspesifikasjon 151
27.2.5 Implementering 151
27.3 IDT som arbeidsplass 151
Konklusjon 153
28.0 Konklusjon 154
Referanseliste 155
Appendix A 157
29.0 Artistic license for LibWWW-perl 157
Appendix B 161
30.0 Robot Exclusion Standard 161
Appendix C 165
31.0 Utdrag fra HTTP spesifikasjonen ver 1.0 165
Appendix D 179
32.0 Kort brukerveiledning for systemet 179
Appendix E 181
33.0 Kildekoden til WWW roboten, relaterte programmer og søkemaskinen (MS160 grensesnitt) 181

Figurliste

FIGURE 1. Internet tilknytning verden over 20
FIGURE 2. FTP, WWW og Gopher traffikk 20
FIGURE 3. Hypertekst oppslag 22
FIGURE 4. Http forespørsel og svar 23
FIGURE 5. HTML-formatering 24
FIGURE 6. En oppdeling av Cyberspace 55
FIGURE 7. Lycos beregningsgrunnlag for størrelsen av WWW 56
FIGURE 8. MS 160 oversikt 63
FIGURE 9. En oversikt over datareservoaret 64
FIGURE 10. Åtte data strømmer hver til et vindu i parallell 65
FIGURE 11. Fire data strømmer hver til to vinduer i parallell 65
FIGURE 12. To data strømmer til fire vinduer i parallell 66
FIGURE 13. En data strøm til åtte vinduer i parallell 66
FIGURE 14. Vindus konfigurasjon 68
FIGURE 15. Hit Mask konfigurasjon 70
FIGURE 16. WWW-robotens komponenter 78
FIGURE 17. WWW-klient og server 97
FIGURE 18. Oversikt over MS160 søkemaskin sine moduler 98
FIGURE 19. Software moduler i søkemaskinen 99
FIGURE 20. Match latency problem 103
FIGURE 21. Maksimum treff eksempel 105
FIGURE 22. WWW-søkegrensesnitt 109
FIGURE 23. Oversikt over datastrukturen til program / driver 125
FIGURE 24. Vindus innlastning 127
FIGURE 25. Oppslag i DBM-fil 134

Tabellliste

TABLE 1. Resultater fra WWW Wanderer 47
TABLE 2. Rutings og duplikat tabell 67
TABLE 3. Kommandolinjeparametre 87
TABLE 4. 90
TABLE 5. Duplisering av vinduer 130
TABLE 6. Antall skifte operasjoner 132
TABLE 7. Test resultater av innlastings program 138
TABLE 8. Testresultater av konverterings program 139
TABLE 9. Testresultater av domene søk 139
TABLE 10. Testresultater av Wildcard søk 140
TABLE 11. Testresultater av Maksimum antall treff 140
TABLE 12. Testresultater av låsemekanismen 141
TABLE 13. Testresultater av oppslag i DBM-fil 141
TABLE 14. 142
TABLE 15. Testresultater av minst 1 av N søkekriteria 143
TABLE 16. Testresultater av beste resultat og første resultat 143

Innledning

Ole Andreas Grytdal
Bjørn Kristian Hilstad

I de siste årene har World Wide Web eksplodert i omfang og informasjonsmengden øker hver dag uten at noen riktig vet hvor mye, eller hva som befinner seg i den. Søkemaskiner og roboter er en måte å lage en oversikt over den distribuerte datamengden rundt om i verden, som vi skal se nærmere på i denne oppgaven.

Søk i så store datamengder som finnes på WWW, krever raske og ikke minst presise søkealgoritmer. MS160 er en god løsning som er svært god på fritekst søk. Det kan derfor være interessant å se hvordan den kan brukes for å lage en søkbar database over noe av innholdet på World Wide Web. Derfor vil vi utvikle en søkemaskin som vil gi brukere i Norge rask tilgang til all informasjon som ligger på WWW-servere i Norge.

1.0 Introduksjon

I dette kapittelet skal vi presentere oppgave teksten, litt om prosjekt deltagerene og en oversikt over de andre kapitlene i denne rapporten.

1.1 Oppgave tekst :

Oppgave teksten som var gitt lød som følgende :

Det skal lages en tjener for søking med MS160 i materiale lagret på World Wide Web. I oppgaven vil det blant annet inngå utvikling av metoder for å innhenting og lagring av søkbart materiale. I tillegg kommer utvikling av WWW-grensesnitt og implementering av søking ved hjelp av MS 160. Problemet har mange paralleller til Archie-tjeneren som tidligere er utviklet.

1.2 Diplom gruppen og veiledere

De som har vært med på denne diplom oppgaven er :

- Arbeidsgiver : Norges Tekniske Høgskole
- Faglærer : Arne Halaas
- Veiledere : Tor Egge og Øystein Grovlen
- Diplom studenter : Bjørn Kristian Hilstad og Ole Andreas Grytdal

1.3 Rapport innhold

- Kapittel 1 : Innledning : Generelt om oppgaven og om WWW.
- Kapittel 2: Dagens situasjon : søkemaskiner og WWW. Et forstudie om forskjellige søkemaskiner
- Kapittel 3: Introduksjon til MS160. Kort innføring i MS160
- Kapittel 4: Kravspesifikasjon for WWW-robot
- Kapittel 5: Kravspesifikasjon for MS160 søkemaskin
- Kapittel 6: Konstruksjons-spesifikasjon for WWW-robot
- Kapittel 7: Konstruksjons-spesifikasjon for søkemaskin
- Kapittel 8: Implementasjon-beskrivelse
- Kapittel 9: Prosjekt evaluering
- Kapittel 10: Konklusjon
- Referanse liste
- Appendikser
- Kildekode

2.0 World-Wide Web

Dagens samfunn går fra det papirbaserte til papirløse samfunn, der den enkelte person vil få presentert mer og mer informasjon via sin datamaskin og Internet. World Wide Web, forkortet WWW, er et internasjonalt og gigantisk informasjons system som man kan benytte på det nå så kjente Internet.

2.1 Internet

Internet er et nettverk av mange små nettverk som idag har knyttet sammen mange millioner brukere. Det har ingen sentralisert organisasjon, som samordner disse nettene mot hverandre. Det eneste er en organiasjon som tildeler unike adresserom til maskiner på Internet. Internet begynte som så mange andre ting som forsknings prosjekt ved det amerikanske militæret. Det hele startet på slutten av 60-tallet med utvikling av ARPANET og derfra gikk utviklingen sakte i 70-årene helt fram til 1982 da TCP/IP[1] ble utviklet for bruk på Internet. I perioden 1984 - 1988 gikk tallet fra antall maskiner knyttet til nettet fra 1000 til 10000, og idag er tallet på tilknyttede maskiner et sted mellom 5 og 10 millioner. [1]Antall brukere er også vanskelig å estimere, men et sted mellom 30 og 70 millioner er vel ikke urimelig. Som man ser av Figur 1, “Internet tilknytning verden over,” på side 20 så har nesten hele verden tilknytning til internet eller i det minste elektronisk post. Tallet på antall brukere kommer til å stige voldssomt framover også, da flere og flere tjenester knyttes opp mot Internet via WWW. Dette kan man se en estimert kurve av i Figur 2, “FTP, WWW og Gopher traffikk,” på side 20.

Via Internet kan man f.eks. :

- sende elektronisk post
- abonnere på forskjellige nyhets grupper,
- lete etter filer og hente disse filene ned
- “surfe” over World Wide Web og finne informasjon av nyttig og unyttig art.

1. TCP/IP er Transmission Control Protocol/Internet Protocol. På Internet kan datapakker forflytte seg mellom maskiner over en dataprotokoll kalt TCP/IP.

FIGURE 1. Internet tilknytning verden over

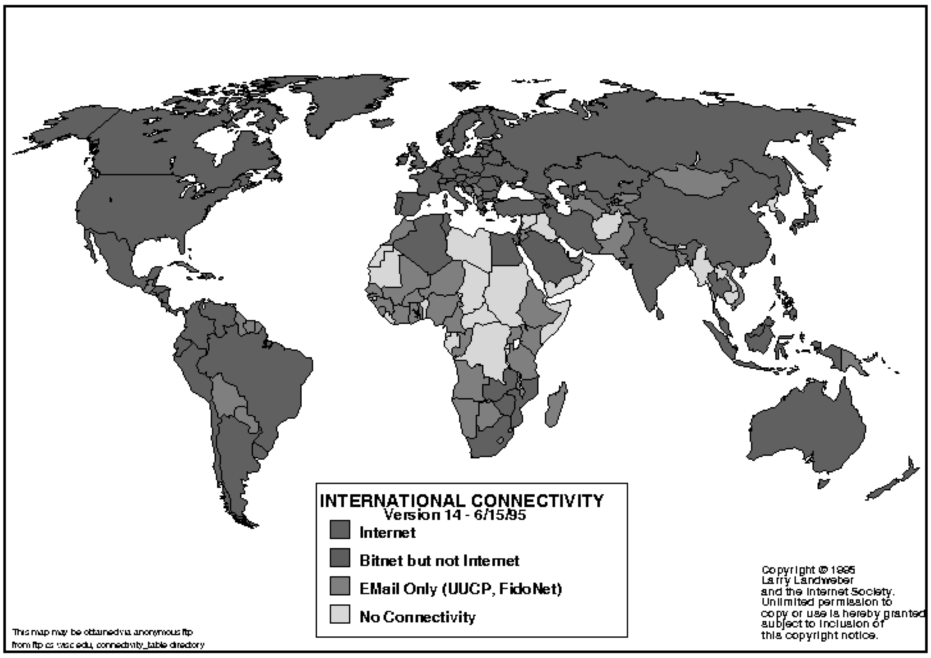
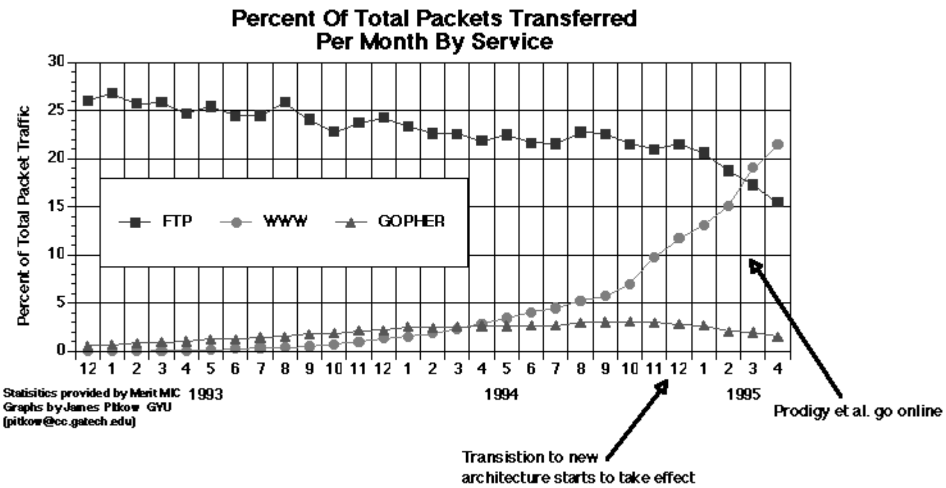


FIGURE 2. FTP, WWW og Gopher traffikk



2.2 Hva er WWW

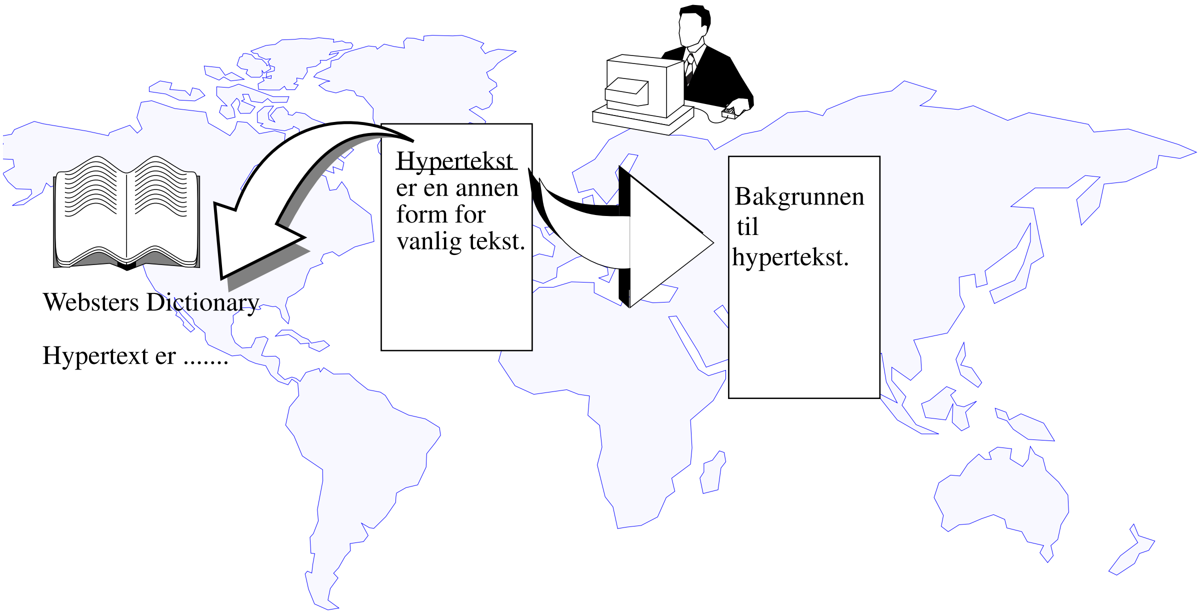
World Wide Web har som overordnet mål "å kunne gi universell adgang til en verden av dokumenter"[2]. I begynnelsen så var WWW et forsknings prosjekt utviklet ved CERN, European Particle Physics Laboratory. Prosjektet gikk ut på å finne en enkel måte å utveksle dokumenter på innen miljøet i CERN. Prosjektet gikk videre, og utviklet seg til noe man sikkert ikke hadde forestilt seg på den tiden. De fleste dokumenter som idag er tilgjengelig via WWW, kan man se på som deler av en infostruktur - en informasjondatabase med en spesial designet struktur. En infostruktur inneholder en stor og variert mengde informasjon spredt over distribuerte servere som hyperlinkede filer. Problemet med hyperlenker, er at hvis man flytter/sletter en fil / server, må alle referanser til denne filen oppdateres noe som kan være vanskelig. Noe av hensikten med å lage en søkemaskin er å luke bort dette problemet, men også å få en total oversikt over det omfangsrike nettet.

2.3 Hypertext og hypermedia

For at WWW skal være enkelt å bruke, benyttes klikk og pek metoden. Det kan da være greit å kunne klikke på en tekst for å få vite mer. I WWW benyttes noe som kalles **hypertekst**. Hypertekst er i hovedsak det samme som vanlig tekst, men "inne i" teksten er det en peker til et annet dokument som kan befinne seg en helt annen plass i verden. Dette blir forklart bedre litt senere. Pekeren vil være en entydig adresse til en spesiell maskin og absolutt sti til dokumentet. Disse nye dokumentene kan ha nye lenker til andre dokumenter og slik fortsetter det. Et lite eksempel kan sees i Figur 3, "Hypertekst oppslag," på side 22. Her kan man ved å trykke på Hypertekst i et norsk dokument, finne lenker til andre plasser som i f.eks Websters Dictionary. Man kaller gjerne også slike hypertekst lenker for **hyperlenker**.

Et begrep til er **hypermedia,** som er noe ala hypertekst, men mer en kombinasjon av hypertekst og multimedia. I hypermedia dokumenter er det ikke bare referanser til andre dokumenter, men også til lyd, bilder eller filmklipp. Bilder kan ha lenker "under" seg, ved at man f.eks klikker på bildet av en hund så vil bjeffing eller informasjon om den spesielle hunderasen blir presentert for brukeren.

FIGURE 3. Hypertekst oppslag



2.4 Hva finnes på WWW

Det som finnes av informasjon eller kan aksesseres via en www-klient er :

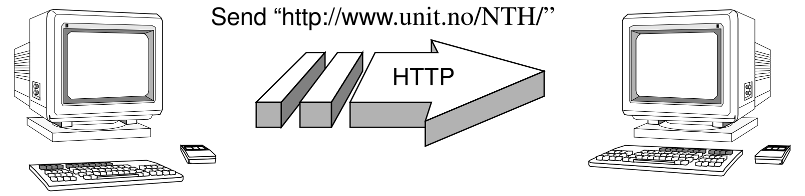
- HTML formaterte hypertekst og hypermedia dokumenter
- Alt som kan aksesseres via gopher
- Filer fra anonyme ftp servere
- Alle nyhetsguppene på USENET
- Alt som er tilgjengelig via TELNET
- Archie søk (Søk i anonym FTP baser)
- Veronica søk (Søk i Gopher tjenester)

2.5 Bruk av World Wide Web

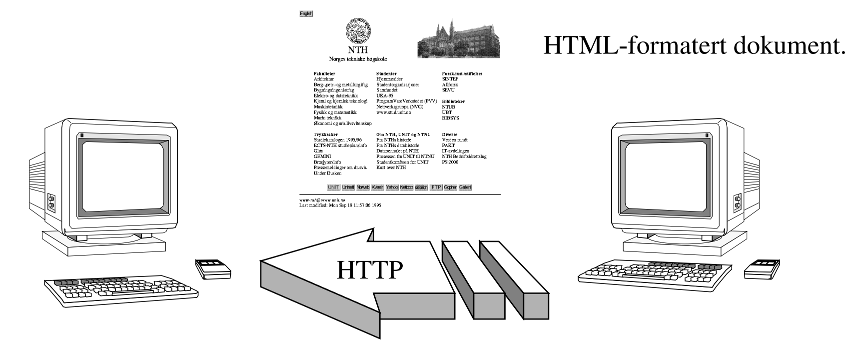
For å kunne benytte seg av WWW, må en maskin kjøre en WWW server. Server programmet tar imot forespørseler som en bruker sender fra sin klient. Den mest vanlige WWW-klienten er Netscape, som har de viktigste funksjonene innebygd for å kunne utnytte mulighetene i WWW.

Protokollen som benyttes mellom klient og server i WWW kalles **Hypertext Transfer Protocol**, **HTTP**. Alle Web klienter og servere må kunne "snakke" HTTP for å kunne sende og motta hypertekst dokumenter. Mens HTTP er prokollen som utveksles mellom klient og server, er **Hypertext Markup Language**, **HTML**, det språket som benyttes for å lage hypertekst dokumenter. Et eksempel på denne utvekslingen kan en se i Figur 4, "Http forespørsel og svar," på side 23.

FIGURE 4. Http forespørsel og svar



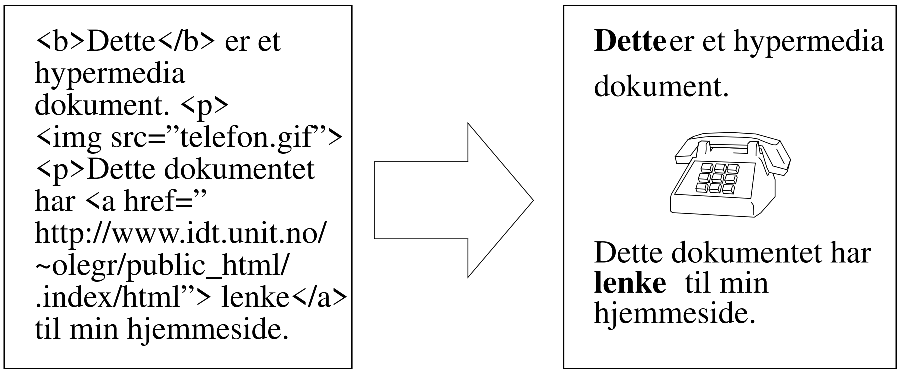
Klienten sender over en HTTP beskjed til maskinen som kjører WWW server og spør etter et dokument.



WWW serveren svarer med å sende et HTML dokument tilbake til klienten. Dokumentet dekodes og vises til brukeren på skjermen.

HTML kan benyttes for skriftelig å framstille og formatere dokumenter til bruk på WWW. Språk som HTML og som følger standarden til Standard Generalized Markup Language (SGML) kan tillate forfattere av WWW dokumenter og skille formaterings informasjon fra dokument informasjon; brukere av LATEX kjenner seg nok igjen her. Et lite eksempel er vist i Figur 5, “HTML-formatering,” på side 24.

FIGURE 5. HTML-formatering



Dokument skrevet i HTML

Dokument presentert for leser.

HTML ble påbegynt i 1991 og hadde et begrenset sett med kommandoer, men er idag utvidet med flere kommandoer for layout messig å kunne virke som et godt presentasjonmiddel.

World Wide Web benytter noe som kalles Uniform Resource Locators (URLer) for å representere hypermedia lenker og lenker til andre nettverks tjenester. Det er mulig å representere alle mulige former for forbindelser via URL. Den første delen av en URL, det før :// , sier hva slags forbindelse som skal brukes, mens det som kommer etterpå er adressen dit man skal hente dokument fra. En URL vil alltid være entydig hvis en benytter seg av IP adressen og ikke navnet som er knyttet til IP’en. Noen ganger kan det nemlig lages et navne-alias for et maskin navn. Et eksempel her kan være “www.idt.unit.no” som egentlig er et alias for “peon.idt.unit.no”, som også er det samme som IP “129.241.111.13”. Et annet problem som kan dukke opp er at en maskin har flere nettverkskort noen som gjør at et navn kan knyttes til flere ip adresser, noe som gjør det vanskelig å håndtere når en skal skape unike forhold mellom navn og IP adresse.

I tillegg vil stien som man benytter etter adressen til WWW-serveren også være unik. Problemet er å oppdage at to dokumenter er like selvom de har forskjellig sti og/eller ligger på forskjellig server.

Under er noen eksempler på forskjellige URLer.

- file://www.idt.unit.no/~olegr/public_html/ole.gif
  Denne henter en fil fra den oppgitte adressen og kan vise bildet på skjermen hvis man har assosiert noe med endelsen .gif.
- http://www.idt.unit.no/~olegr/public_html/.index.html
  Kobler seg opp mot en HTTP server og henter en HTML fil.
- ftp://ftp.funet.fi/README
  Setter opp en FTP forbindelse til ftp.funet.fi og henter file README
- telnet://iq.pvv.unit.no:4000
  Setter opp en telnet forbindelse til iq.pvv.unit.no mot port 4000
- newsrc://news.unit.no
  Går til USENET nyhetene ved å koble seg opp mot en NNTP server[1] og returnerer de nyhetene man har i sin newsrc fil
  news://idt.fritt-forum
  Ved å skrive slik vil man direkte sette seg opp mot en newsgruppe via en NNTP-server

De fleste WWW klienter tillater å koble seg direkte opp mot de forskjellige URLene.

1. NNTP-server betyr News Network Transfer Protocol

Dagens situasjon : Søkemaskiner og World Wide Web.

Ole Andreas Grytdal
Bjørn Kristian Hilstad

Dokumenet gir en oversikt over hvordan WWW brukes i dag, hvordan man oppdager nye ressurser, oversikt over hva som finnes av søkemuligheter på WWW idag og antyder dagens størrelse og mulig vekst for WWW.

3.0 Introduksjon

Tenk deg at du befinner deg i et bibliotek og er på jakt etter en spesiell bok. Plutselig oppdager du at biblioteket ikke har noe katalogiseringssystem. Å finne spesifikke dokumenter eller informasjon på WWW kan være like vanskelig. I utgangspunktet er den eneste muligheten man har til å navigere på WWW å følge lenker fra et dokument til et annet. Hadde det ikke vært for det faktum at WWW endres for hver time som går, ville det nok vært mulig å klassifisert dokumentene og på den måten ha en logisk og navigerbar struktur. Det er gjort endel forsøk på nettopp dette (Yahoo i USA, Kvasir og Index i Norge), men det kreves mye arbeid både for å opprette og vedlikeholde slike databaser.

Samtidig er WWW også desentralisert og mangfoldig, og dette gjør det enda vanskeligere å finne den informasjonen man er på jakt etter ved manuell navigering. Denne problemstillingen blir ofte kalt ressurs-oppdagelses problemet.

4.0 Bruk av World Wide Web i dag

Hypertext og hyperlenker hørtes helt utrolig ut når Ted Nelson foreslo dem i sin bok *Cumputer Lib* fra 1974. Når WWW brakte oss hyperlenker i 1991, syntes det å oppfylle løftene om universell aksess til data spredt over hele verden.
Etterhvert har virkeligheten hentet oss inn. Et skred av servere, dokumenter og hyperlenker har sammen med WWW's eksponentielle vekst, gjort WWW mindre brukbart til virkelig arbeid enn hva som opprinnelig var meningen. Det kreves også mye arbeid for å vedlikeholde en dyanmisk server. Med mange dokumentendringer og nye dokumenter fra dag til dag, blir det mye 'ekstra' arbeid med å oppdatere hyperlenker. Det finnes komersielle softwarepakker som gjør denne jobben lettere (f.eks. CyberLeaf fra InterLeaf Corp.), men det hadde vært bedre om det var integrert i selve WWW. For å gjøre WWW lettere å navigere i, kan man enten endre basisen for WWW (se seksjon 7.1 på side 60) eller bygge opp søkbare databaser med oversikt over "alle" dokumenter på WWW.

Undersøkelser viser at brukerne av WWW opererer i to distinkte modi. Enten leter de etter spesifikk informasjon eller så kikker de bare etter noe nytt og interessant (såkalt surfing).
Det er gjort mange forsøk på å indeksere WWW og slike indekser vil i størst grad støtte brukere som leter etter spesifikk informasjon, men vil også kunne hjelpe "surfere" ved å henvise til de nyeste sidene som

er oppdaget.

4.1 Hva er en robot i denne sammenheng ?

Ordet robot ble først brukt av den tsjekkiske forfatteren K. Capek (1890 - 1938). I vanlig forstand betyr det enten en menneskelignende maskin som kan gjøre menneskearbeid, eller en person som arbeider automatisk uten å tenke.

En WWW robot er et program som automatisk kan hente ned og behandle dokumenter fra WWW. Dette gjør den selvfølgelig på en robotaktig måte, dvs. den velger hele tiden nye dokumenter å hente og fortsetter til det ikke er flere igjen (eller andre kriterier er oppfylt). Slike programmer kalles også ofte for en crawler, spider eller worm. En WWW-robot er også en måte å løse ressurs-oppdagelses problemet på. De kan tilby hurtige søkemuligheter ved å opprettholde en indeks over WWW, som er tilgjengelig for WWW-brukere via enkle grensesnitt. Det finnnes flere typer roboter/søketjenester tilgjengelige på WWW i dag. Vi kan grovt sett dele dem inn i 3 kategorier :

- Roboter som kjøres av en operatør og henter ned ett og ett dokument av gangen (eller flere samtidig dersom de er virkelig effektive). Det bestemmes lokalt hvilke dokumenter som skal hentes og hvor mye av dokumentet som skal hentes. Roboten vil hele tiden følge lenker i dokumenter den finner, slik at den stadig henter dokumenter som er lenger unna utgangspunktet. Det bygges også en lokal indeks over alle dokumentene som er hentet inn. Denne er søkbar for andre brukere av WWW.
- Roboter som kjøres når man får en forespørsel fra en bruker. Disse robotene får sine søkedata fra en bruker og starter så søket ut i fra disse. Typiske data som må oppgis av brukeren er et dokument som roboten skal starte fra og hvilke ord som det skal søkes etter. Roboten starter så sin traversering av WWW og samler inn relevante dokumenter. Denne metoden benevnes ofte automatisk navigering “on-demand”. Denne typen roboter bruker ofte lang tid på å gi brukeren et svar tilbake, men svaret vil derimot være basert på helt ferske dokumenter og på denne måten gi et mer oppdatert bilde av situasjonen. Siden slike roboter ikke nødvendigvis henter dokumenter som ligger langt unna utgangspunktet (pga. tids og- dybdebegrensinger), bør brukeren ha en viss ide om hvor på WWW han kan finne informasjonen han ønsker. Roboten bør også ha en viss form for intelligens, slik at den hele tiden bruker søkekriteriene for å velge vei videre (mao. den bør opptre mer som en agent). Slike roboter egner seg f.eks. godt til å lete på en bestemt server etter angitt informasjon. Det finnes forøvrig få roboter av denne typen.

- ALIWEB (Archie Like Indexing in the WEB). ALIWEB prøver å benytte ideene bak Archie[3] til å gjøre WWW søkbart. Archie brukes i dag for å muliggjøre søk i ftp og gopher servere. ALIWEB foreslår at de som er ansvarlig for en WWW-server skal ha en oversikt over alle filer som befinner seg på sin(e) server(e) og gjøre disse tilgjengelige slik at et program kan hente disse og kombinere de til en søkbar database. En mer utfyllende beskrivelse følger senere i kapittelet.

Felles for de fleste er at de skal bygge opp en ressurs-oppdagelses database.

5.0 Oversikt over dagens søkemaskiner

I dette kapitlet skal vi forsøke å gi en oversikt over en del søkemaskiner som finnes på WWW i dag. Vi vil stort sett begrense omtalen til de største og mest kjente søkemaskinene, men vil også nevne andre søkemaskiner som har spesielle egenskaper i forhold til andre.

I generell informasjon tabellene nedenfor lister vi opp følgende punkter :

- **URL** : URL for tjenesten
- **Eier / ansvarlig** : Eier eller ansvarlig for tjenesten
- **Sted** : Hvor tjenesten er lokalisert rent fysisk
- **Protokoller** : Hvilke protokoller som er tilgjengelige (http, gopher, news, ftp osv.) å søke i
- **Størrelse** : Hvor stor del av WWW som er søkbart på tjenesten
- **Kommersiell** : Er tjenesten gratis ? Reklamefinansiert ?

5.1 HARVEST

5.1.1 Generell informasjon

URL	http://harvest.cs.colorado.edu
Eier / ansvarlig	Internet Research Task Force Research Group on Resource Discovery
Sted	University of Colorado
Protokoller	HTTP
Størrelse	ca. 45000 URLer (Desember 1995)
Kommersiell	Nei

5.1.2 Historie / bakgrunn

Harvest prosjektet er mye mer enn bare en WWW-robot. Harvest er et informasjon oppdagelse og -aksess system som består av flere subsystemer. Det tilbyr en skalerbar infrastruktur for å bygge, distribuere og indeksere informasjon, samt metoder for å aksessere WWW informasjon. Harvest prosjektet har som mål å finne en metode for å lage indekser fordelt mellom flere servere og samtidig ha en mest mulig effektiv bruk av disse serverene. Utviklingsarbeidet er primært støttet av Advanced Research Projects Agency.
Blant subsystemene i Harvest kan vi nevne :

- Gatherer som samler inn indekserbar informasjon
- Broker som tilbyr et fleksibelt grensesnitt til denne informasjonen
- Glimpse er indekserings/lete systemet
- Replicator sørger for å speile informasjonen som alle Brokers har
- Object Cache skal finne den raskeste serveren for enhver forespørsel

5.1.3 Datainnsamlingsprosess

Harvest har et svært effektivt innsamlingssystem takket være en kombinasjon av et optimalisert innsamlingsprogram og muligheten til å dele innhentet informasjon mellom flere innsamlere. Vanligvis hentes all informasjon inn med samme utgangspunkt (dvs. innsamlingsprosessen kjøres bare fra et sted), men Gatherer kan kjøres lokalt flere steder samtidig. Gatherer kan selvfølgelig også hente inn informasjon via nettverksprotokoller (FTP, GOPHER og HTTP). Essence er et system som brukes av Gatherer for å takle forskjellige dataformater. Essence kan f.eks. dekomprimere *tar*-filer, finne forfatter og tittel innenfor Latex dokumenter og symboler i objekt filer. Typegjenkjenningen og hvordan de forskjellige typene skal takles er konfigurerbart.

5.1.4 Indekseringsprosess

Harvest har et definert grensesnitt mellom søkeprosessen og indekseringsprosessen, slik at det kan støtte flere forskjellige søkemaskiner. De viktigste kravene er at de må støtte logiske operatorer (boolske kombinasjoner) og inkrementell oppdatering av indeksen. Foreløpig støtter Harvest kommersielle og gratis versjoner av WAIS, Glimpse og Nebula. Glimpse og Nebula er spesielt utviklet for Harvest, men bare Glimpse er med i den nåværende distribusjonen. Glimpse bruker svært små indekser (2-4% av indeksert tekst), og har både hurtig og inkrementell indeksering. Den støtter også boolske uttrykk, regulære uttrykk og unøyaktige søk.

5.1.5 Søkemuligheter

Siden Harvest kan støtte nesten hvilken som helst søkemaskin, kan man i prinsippet få svært avanserte søkemuligheter. Glimpse, som brukes i dag, gir brukeren disse mulighetene :

- “case sensitive” og “case insensiteve” søk
- boolske kombinasjoner av nøkkelord
- regulære uttrykk
- ustrukturerte eller strukturerte søk (kan begrense hvilke felter som skal sjekkes)
- substring søk, ordsøk (eksakte treff) og fler-ord søk (f.eks. “Vegard Ulvang”)
- spesifisere antall tillatte stavefeil
- begrense antall returnerte treff
- rangering av treffene

5.2 Infoseek

5.2.1 Generell informasjon

URL	http://www.infoseek.com
Eier / ansvar-lig	Infoseek Corporation (eies av Sun Microsystems)
Sted	Santa Clara, California
Protokoller	HTTP, Usenet News (NNTP), FTP, Gopher
Størrelse	ca. 1 000 000 WWW sider og 15 000 Newsgrupper (Dec. 1995)
Kommersiell	Ja

5.2.2 Historie / bakgrunn

Selskapet Infoseek ble opprettet i januar 1994 og de lanserte sine tjenester 13. februar 1995. De tilbyr to forskjellige tjenester :

- Infoseek Guide er en gratis tjeneste (sponsor betalt) som tilbyr søking i WWW-sider, Newsgrupper og Usenet FAQs (frequently asked questions). WWW-sidene er organisert i et hierarki, slik at manuell søking skal være mulig. Infoseek har inngått et samarbeide med Netscape, slik at denne tjenesten er tilgjengelig via “Net Search”

knappen i Netscape Navigator (den mest populære WWW klienten per idag).
- InfoSeek Professional er en abonnementbasert tjeneste, som i tillegg til tjenestene til Infoseek Guide, tilbyr søking i mange komersielle dokumentsamlinger (bl.a. Business Wire, ComputerWorld, InfoWorld, Newswire, Newsbytes m.fl.).

5.2.3 Datainnsamlingsprosess og indekseringsprosess

Infoseek tilbyr ingen detaljer om teknologien som benyttes.

5.2.4 Søkemuligheter

Infoseeks søkemaskin har følgende søkemuligheter :

- kan spesifisere at ord skal forekomme i nærheten av hverandre (men ikke hvor nærme)
- kan angi egennavn og skille dem med komma (f.eks. Bill Clinton, Al Gore)
- kan spesifisere at ord skal forekomme ved siden av hverandre ("computer graphics")
- kan spesifisere at to ord ikke skal være adskilt av mer enn ett ord (user-interface)
- kan angi ord som ikke skal forekomme og ord som må forekomme (f.eks. IBM -Compaq og chip +Pentium)

men støtter ikke :

- case sensitive og case insensitive søk
- boolske operatorer
- eksakt søking (alt er substreng søk)

5.3 OpenText

5.3.1 Generell informasjon

URL	http://www.opentext.com:8080
Eier / ansvarlig	Open Text Corporaion
Sted	Open Text Corp ligger i Waterloo, Ontario, Canada, mens serverene står i Mountain View,California
Protokoller	HTTP, FTP, Gopher

Størrelse	15,436,712 URler (November 1995) (disse URLene er ikke nødvendigvis unike. 10 like URLer teller som 10 URLer i tallet ovenfor)
Kommersiell	Tjensten er gratis, men sidene har annonsører

5.3.2 Historie / bakgrunn

Open Text Corporation ble opprettet i 1991 i Ontario, Canada. Indekserings og- søkemaskinen (Open Text 5) som benyttes, er basert på teknologi utviklet på Center for Text Research ved Universitetet i Waterloo, Ontario. Teknologien benyttes både i akademiske og statlige institusjoner, samt forskjellige kommersielle foretak (bl.a. Yahoo). Systemet som brukes av Open Text på WWW idag, er basert på et SGML (Standard Generalized Markup Language) indekseringssystem, som ble laget for å indeksere Oxford English Dictinary. HTML er et subsett av SGML, slik at dette systemet er spesielt godt egnet til strukturerte søk i HTML tekst (i tillegg støtter den strukturerte søk i 40 andre filformater).
Open Text indeksen ble lansert på Internet 29.mars 1995 og har idag mellom 100 000 og 200 000 forespørsler hver dag.

5.3.3 Datainnsamlingsprosess

Som de fleste andre roboter, henter Open Text inn dokumenter ved å følge lenker i WWW sider. De kjører flere roboter samtidig på forskjellige maskiner og dette gir dem kapasitet til å hente inn rundt 50 000 sider om dagen (parallellitet med løs kopling). Roboten henter inn og lagrer hele HTML-dokumenter, og ikke bare f.eks. tittel og de viktigste ordene, som noen andre roboter gjør.
Kapasiteten deles mellom innhenting av nye sider og gamle sider som kan ha endret seg (En bedre løsning ville være å sjekke om dokumentet hadde blitt oppdatert uten å hente inn hele dokumentet. Dette kunne gjøres ved å bruke HTTP HEAD og sjekke den nye Last-modified dataoen mot den gamle Last-modified datoen. HTTP HEAD henter bare inn HTTP headeren og ikke selve dokumentet. Det hadde også vært mulig å utføre et vilkårlig HTTP request med “HTTP If-modified-since satt til den datoen man sist hentet inn dokumentet. Se appendix C for mer informasjon om HTTP protokollen).
Roboten håndterer ikke problemet med forskjellige URLer som peker til samme sted (f.eks. http://www.idt.unit.no/ og http://peon.idt.unit.no). Dette innbærer at man kan få flere referanser til den samme siden i resultatlistingen.
Open Text roboten støtter Robot Exclusion Standard.

5.3.4 Indekseringsprosess

Open Text Indeks bruker sitt eget Open Text 5 system til å indeksere og søke i teksten. I følge Open Text er dette et av de raskeste som finnes på markedet. Blant noen av mulighetene som Open Text 5 har, kan vi nevne :

- kan indeksere over 40 forskjellige filtyper (som f.eks. forskjellige tekstbehandlingsformater, SGML, HTML og PDF(Adobe))
- indekserer hvert eneste ord (inkluderer også såkalte "stopp"-ord som f.eks. 'og', 'det' osv.)
- kjennskap til strukturen i forskjellige dokumenttyper og kan trekke ut informasjon fra de forskjellige strukturelementene
- støtte for mange språk. Støtter både europeiske og ikke-europeiske språk (japansk og arabisk)
- Parallel Execution Monitor gjør det mulig å sende forespørsler til flere servere samtidig (distribuert og parallellt søk), og få tilbake ett samlet svar

Mange av disse egenskapene skyldes at Open Text bruker en patentert streng søking arkitektur, og ikke den mer vanlige inverterte ord indeksene (inverted word index).

5.3.5 Søkemuligheter

Open Text er en av de mest avanserte søkeverktøyene som er tilgjengelig på WWW idag. Den tilbyr følgende søkemuligheter :

- lete etter et ord, flere ord eller en setning
- lete etter ord og/eller setninger koblet sammen med boolske operatorer (AND, OR, BUT NOT)
- lete etter ord og/eller setninger koblet sammen med nærhetskriterium ('i nærheten av', 'fulgt av')
- begrense søket til spesielle deler av en WWW side (f.eks. tittel, første heading, URLen, oppsummeringen (nøkkelord) eller hele siden)
- tildele et ord eller en setning en verdi (vekt). Dokumentene med de viktigste ordene vil da komme øverst i resultatlisten
- fra resultatsiden kan man enten slå opp på den aktuelle siden eller slå opp i Open Texts versjon av dokumentet som viser hvor i dokumentet man traff med søket
- fra resultatsiden kan man for alle dokumentene man fikk treff på, lete etter lignende dokumenter. Dette fungerer ved at dokumentteksten blir analysert, slik at man finnner ordene som forekommer med størst frekvens. Deretter finner man andre sider hvor de samme

ordene forekommer med høy(est) frekvens. Hvor "riktig" de nye treffene blir, vil variere kraftig, siden man ikke analyser semantikken (betydningen av ordene, i den sammenhengen de forekommer), men bare forekomster av ord.

5.4 Inktomi

5.4.1 Generell informasjon

URL	http://inktomi.berkeley.edu/
Eier / ansvarlig	Berkeley University ved Prof.Eric Brewer og grad. student Paul Gauthier
Sted	Berkeley, California, USA
Protokoller	HTTP
Størrelse	2 800 000 URLer
Kommersiell	Nei (ARPA har delvis finansiert prosjektet, Sun har donert maskiner)

5.4.2 Historie / bakgrunn

Inktomi bruker flere vanlige arbeidsstasjoner til å lage en skalerbar WWW server. Inktomi distribuerer både belastning og database over flere SparcStation10 maskiner. Kapasiteten til systemet skalerer lineært i forhold til antall maskiner, slik at det lett kan utvides. Databasen er fordelt over minnet til alle maskinene som fungerer som en felles cache for systemet. På denne måten kan Inktomi ha en svært rask responstid uten å ha behov for maskiner med enorme mengder minne. Bruk av flere maskiner gir også høy tilgjengelighet. Inktomi vil være operativ selv om 3 av 4 maskiner ikke fungerer (med mindre ytelse). For å fordele belastningen mellom maskinene bruker man HTTP's redirect funksjon, dvs. at når man aksesserer Inktomi[http://inktomi.berkeley.edu] får man tilbake en omdirigering (redirection) til den serveren som er minst belastet i øyeblikket. De fleste WWW klienter håndterer idag omdirigeringer ved å gå direkte til siden som angis i "HTTP Location" feltet (se appendix C).

5.4.3 Datainnsamlingsprosess og Indekseringsprosess

Svært få detaljer er tilgjengelig om datainnsamling og indeksering hos Inktomi. De få detaljene som var tilgjengelige var :

- alle ord som er kortere enn tre ord og 'vanlige' ord (is, are, the, osv.) blir ignorert. Det ikke mulig å søke etter disse.
- vanlige endelser blir fjernet fra ord før søk utføres.

5.4.4 Søkemuligheter

Søkemulighetene på Inktomi er relativt begrenset. De viktigste funksjonene er :

- kan søke på opptil 10 ord
- + og - kan brukes foran ord for å indikere at ordet må eller ikke må forekomme i dokumentene. Dette gir samme funksjonalitet som boolske uttrykk
- foretar en automatisk trunkering av søkeord. Fjerner vanlige endelser fra søkekriterier slik at man istedet søker på ordstammen til de oppgitte ordene

Inktomi har bl.a. ikke

- case sensitive søk
- søking innenfor visse felter i dokumentene (f.eks. tittel)
- mulighet til å spesifisere at ord skal være i nærheten av hverandre eller etter hverandre
- mulighet til å søke på annet enn bokstaver (tall og spesialtegn er fjernet)
- mulighet til å søke på småord siden disse er fjernet

5.5 AltaVista

5.5.1 Generell informasjon

URL	http://altavista.digital.com
Eier / ansvarlig	Digital Equipment Corporation
Sted	Palo Alto, California, USA
Protokoller	HTTP, Usenet News (NNTP)
Størrelse	16 000 000 URLer 13 000 Usenet Newsgrupper
Kommersiell	Tjenesten er gratis.

5.5.2 Historie / bakgrunn

Den 15. desember 1995 lanserte Digital Equipment Corporation sin løsning til søking på WWW. Den har i sin korte levetid vist seg å være en av de raskeste og mest innholdsrike som er tilgjengelig. Samtidig har den også svært gode søkemuligheter. Roboten som brukes er ifølge DEC den raskeste som finnes, og kan hente inn opptil 2.5 millioner sider om dagen. Dette er en stor forbedring i forhold til

tidligere roboter. Med denne hastigheten vil den kunne indeksere hele WWW på kort tid. En av årsakene til at det er mulig for DEC å hente inn så mange dokumenter om dagen, er at de i stor grad benytter sitt eget fiberoptiske nettverk.
Systemet kjører på maskiner som benytter DEC's egen 64-bits Alpha prosessor og fungerer sådan ypperlig som reklame for disse.

5.5.3 Datainnsamlingsprosess og Indekseringsprosess

Siden DEC har patentsøknader inne på teknologien som benyttes, har de ikke gjort tekniske detaljer tilgjengelig for allmennheten. I den sparsommelige informasjonen som er tilgjengelig fant vi følgende fakta :

- kan hente inn opptil 2.5 millioner dokumenter om dagen (dvs. 29 dokumenter i sekundet)
- kjører mange roboter samtidig
- støtter Robot Exclusion Standard
- indekserer *alle* ord i dokumentene (forkaster alt som ikke er bokstaver eller tall inkl. punktumm komma osv. I praksis regnes disse som ordskille/space)

5.5.4 Søkemuligheter

Altavista har svært mange søkemuligheter og tilbyr bl.a. :

- søking etter ord og setninger. Setninger må skrives i anførselstegn.
- boolske uttrykk (AND, OR, NOT) som kan nøstes ved bruk av paranteser
- + og - operatorer som indikerer at et ord/setning, må eller ikke må forekomme i dokumentene
- søk etter trunkerte ord f.eks. norg* (må ha minst tre tegn foran stjerne)
- nærhetsoperator (NEAR) som spesifiserer at to ord/setninger må forekomme i nærheten av hverandre (innenfor en avstand av 10 ord)
- ikke-alfabetiske tegn binder ord sammen slik at http://www.idt.unit.no tolkes som "http www idt unit no" og vil treffe alle URLer som peker til IDT
- kan søke på struktur i dokumenter. Ved søk i WWW-sider kan man søke i tittel, dokumentets URL, lenker i dokumentene og navnet på maskinen som dokumentet ligger på (på denne måten kan man begrense hvilke domener (internett adresser) dokumentene man søker i skal ligge i. F.eks ved å spesifisere host:idt.unit.no kan man søke kun i dokumenter som ligger under idt.unit.no domenet)

- ved søk i Usenet News artikler kan man på samme måte som i forrige punkt, spesifisere hvilke deler av artikkelen man skal søke i. Alternativene er 'from', 'subject', 'newsgroups', 'summary' og 'keywords'.
- søkeord som skrives med små bokstaver oppfattes som case insensitive. Skriver man derimot en eller flere store bokstaver, blir søket case sensitivt
- resultatene returneres i rangert rekkefølge. Rangeringen skjer etter følgende kriterier (i prioritert rekkefølge):
  1) dokument hvor treff forekommer i tittel eller de første linjene
  2) dokument hvor søkeordene forekommer i nærheten hverandre
  3) dokument som innholder flere av ordene/setningene som det søktes etter
  4) dokument som innholder flere kopier av etterspurte ord
- kan velge format på resultatlistingen (standard, kompakt, detaljert og antall treff)

5.6 ALIWEB

5.6.1 Generell informasjon

URL	http://web.nexor.co.uk:80/public/aliweb/aliweb.html
Eier / ansvarlig	Nexor Ltd
Sted	Nottingham, England
Protokoller	HTTP
Størrelse	ca. 85 registrerte servere og 500 index filer (se forklaring nedenfor)
Kommersiell	Nei

5.6.2 Historie / bakgrunn

ALIWEB (Archie Like Indexing in the Web) er et system som muliggjør automatisk innhenting og prosessering av WWW indekser. Dagens ALIWEB implementasjon henter indeksfiler fra registrerte servere med jevne mellomrom og kombinerer disse til en søkbar database.
Årsaken til at ALIWEB ble opprettet er at opphavsmannen (Martijn Koster ved Nexor) så farer ved bruk av roboter og ville ha et alternativ. En av hans WWW servere ble "angrepet" av en robot (roboten hentet ett dokument per sekund) slik at serveren ble overbelastet. Han tenkte så ut et ressursvennligere system og implementerte ALIWEB. ALIWEBs database kan være like oppdatert som en robots database,

men uten at den har traversert WWW. En annen fordel med ALIWEB, er at det ikke vil eksistere “døde” lenker i databasen, siden denne baserer seg på filer som er generert lokalt på hver server. Det er nettopp disse indeksfilene som er ALIWEBs største problem fordi det kreves en god del arbeid å opprette og vedlikeholde dem. Derfor er ikke ALIWEB en spesielt populær søketjeneste idag. Et annet problem med ALIWEB er at det ikke eksisterer noen komite som spesifiserer et rammeverk. Dette tillater hurtig utvikling, men det mangler et forum for diskusjon og fastsettelse av implementasjonsstandarder.

5.6.3 Datainnsamlingsprosess og Indekseringsprosess

Indeksene som hentes inn fra de registrerte serverene er enkeltfiler slik at det kreves lite ressurser til dette. Registreringen foregår via et enkelt WWW-skjema på Nexor, som fylles ut med bl.a. servernavn, indeksfilnavn og hvor ofte indeksen blir oppdatert. Indeks filene som brukes i Archie systemet (seksjon 4.1 på side 29) lages enkelt og automatisk ved en rekursiv fillistekommando. I ALIWEB må disse listene lages for hånd av serveransvarlig. Dette fører ofte til at de bare inkluderer de viktigste dokumentene. Det er da meningen at man skal kunne finne fram til selve tjenesten via ALIWEB og så bruke dokumentstrukturen til å finne spesifikk informasjon. Indeksfilene følger et format spesifisert av Internet Anonymous FTP Archives (IAFA) Working Group.
Dagens versjon av ALIWEB er implementert som en samling av Perl skripts. Innhentingsprosessen går gjennom listen over registrerte servere og sjekker om det skal hentes inn en ny kopi av indeksen. Til slutt kombinerer den alle indeksfilene til en stor fil som har nesten det samme formatet som de enkelte indeksfilene.

5.6.4 Søkemuligheter

ALIWEB har følgende søkemuligheter :

- substreng søk, eksakt søk og regulære uttrykk
- case-sensitive og case-insensitive søk
- søke i gitte felter i indeksfilene (tittel, beskrivelser, nøkkelord, URLer). Kan også angi hvilke av disse som man ønsker vist ved evt. treff
- begrense søket til et gitt domene (adresse omeråde)
- stoppe søket etter angitt antall treff

Selv om ALIWEB har mange søkeopsjoner, gir den som oftest svært få nyttige treff til svar. Dette skyldes at den har et svært tynt datagrunnlag, siden den bare har oppsummeringer og nøkkelord som beskriver filene. Det er også få servere som er registrert hos ALI-WEB.

5.7 LYCOS

5.7.1 Generell informasjon

URL	http://www.lycos.com
Eier / ansvarlig	Carnegie Mellon University
Sted	Pittsburgh, USA
Protokoller	HTTP, FTP, Gopher
Størrelse	11,745,521 URLer og103,059 servere.
	Antall URLer i Lycos databasen inkluderer også lenker som ikke er hentet ned. Dersom Lycos henter en URL som inneholder lenker til 99 andre URLer, teller dette som 100 URLer i tallet ovenfor)
Kommersiell	Tjenesten er gratis for internettbrukere. Reklame-finansiert. Har også lisensiert tjenesten til Microsoft Network (MSN)

5.7.2 Historie / bakgrunn

Lycos startet som et prosjekt på Carnegie Mellon University (Center for Machine Translation) i midten av 1994, og har nå vokst til å bli det største og mest brukte søkeverktøyet på WWW (minst 3 500 000 søk per uke). Lycos påstår selv at de har indeksert 91% av WWW, men dette er en sannhet med modifikasjoner (se *Størrelse* punktet under Generell informasjon ovenfor).
Selskapet CMG Information Services Inc kjøpte rettighetene til Lycos i juni 1995. Lycos har også lisensiert ut sine tjenester til flere store selskaper, bl.a. Microsoft, Cybersearch og NlightN (enorm database).
Dr. Michael M. Mauldin som er hovedmannen bak Lycos, jobber nå i et underselskap av CMG som heter Lycos Inc. De står for drift, implementasjon og videre forskning på Lycos-systemet.

5.7.3 Datainnsamlingsprosess og Indekseringsprosess

Lycos roboten er hovedsaklig skrevt i Perl, men bruker et C program basert på CERN's libwww til å hente URLer. Den bruker en randomisert søkealgoritme og lagrer URLene den har hentet i en DBM (Data Base Management) fil (assoiativ array i Perl). Den henter inn HTTP, FTP og Gopher dokumenter og overser Telnet, Mailto og WAIS. Den overser også filer som begynner med "/dev/tty" eller ender på bokstaver som antyder et binært format (en fullstendig liste er tilgjenelig men ikke relevant for dette dokumentet). Når den skal prioritere hvilke lenker som skal hentes inn, prioriterer den dokumenter som har flere lenker til seg og dokumenter med korte URLer (for å få hentet inn toppnivået på serverene). Randomiseringen består i å velge mellom HTTP, FTP eller Gopher lenker. Lycos bruker en datareduksjons algoritme for å begrense hvor mye data som lagres om hver dokument. Lycos lagrer :

- Tittel
- Header og subheadere
- 100 ord med størst vekt (beregnet med Term. freq. / Inverse doc freq.)
- 20 første linjene
- størrelse i bytes
- antall ord

I tillegg har Lycos en frekvensliste for alle ordene den har lest.
Lycos prøver også å slå sammen dokumenter som ligger under hverandre til ett dokument i sin database. Dette gjøres ved å kombinere dataene for flere dokumenter til ett innslag i Lycos databasen (man kan visualisere dette ved å se for seg en trestruktur som trimmes (slår sammen flere løvnoder til en løvnode)). Databasen var tidligere delt opp i to deler (the Big Catalog og the Small Catalog). Den ene innholdt sammendrag av alle dokumentene som var hentet inn og analysert, mens den andre innholdt alle dokumentene som ikke var analysert ennå i tillegg til sammendragene.
Lycos systemet kan hente inn ca. 50 000 dokumenter om dagen.
Lycos støtter også Robot Exclusion Standard.
Lycos har også en meget bra liste over de 250 mest populære dokumentene på WWW. Denne er basert på deres tall over antall lenker som peker til en gitt side. De har i tillegg kategorisert de mest populære sidene etter emne.
I begynnelsen brukte Lycos et indekseringssystem som het SCOUT (implementert i forbindelse med SCOUT prosjektet ved CMU [4]), men dette har nå skiftet navn til Pursuit. Pursuit er et C program som bruker diskbasert invertert filinnhentingssystem og et enkelt vektleggingssystem for å rangere dokumentene. Pursuit gir dokumentene poeng basert på hvor langt inn i dokumentet et treff ord forekommer (treff i tittelen gir høyere poeng enn treff lenger inn i dokumentet).
Pursuit kan håndtere samlinger på flere millioner dokumenter. Den støtter også boolske uttrykk og kan tillate stavefeil.
Lycos fordeler lasten mellom serverene sine, slik at alle brukere skal få best mulig respons.

5.7.4 Søkemuligheter

Lycos har meget bra søkemuligheter selv om de ikke benytter seg av alle mulighetene i SCOUT/Pursuit. Disse mulighetene er tilgjengelige idag :

- AND mellom alle ord, OR mellom alle ord eller alt fra 2 til 7 treff av n oppgitte ord
- kan sette parameter som angir hvor gode treffene skal være (loose match, fair match, good match, close match, strong match)
- angi antall dokumenter per resultatside
- kan bestemme mengden informasjon som blir returnert om hvert dokument (standard, detaljert, oppsummering)

5.8 WEBCRAWLER

5.8.1 Generell informasjon

URL	http://www.webcrawler.com
Eier / ansvarlig	Startet på Universitetet i Washington, USA. Ble kjøpt opp av America Online 1. juni 1995
Sted	San Francisco, USA
Protokoller	HTTP
Størrelse	74 434 servere (40644 HTTP, 7276 FTP, 6666 Gopher og 4925 andre (telnet, rlogin, tn3270og wais))
	250 000 besøkte dokumenter og 1 800 000 ubesøkte dokumenter
Kommersiell	Tjenesten er gratis. Annonsører på sidene.

5.8.2 Historie / bakgrunn

Webcrawler begynte som Brian Pinkertons forskningsprosjekt på Department of Comuter and Engineering ved University of Washington i Seattle. Systemet ble kjøpt opp av America Online [5] den 1. juni 1995, men er fortsatt en gratis tjeneste. Prosjektet hadde som mål å tilby en hurtig og gratis søketjeneste av god kvalitet. Webcrawler fungerer på to forskjellige måter :

- den bygger en indeks over dokumenter den finner på WWW
- den kan opptre som en agent, ved å lete etter dokumenter på forespørsel fra en bruker. Den vil da traversere WWW for brukeren og forsøke å gjøre intelligente valg ut i fra informasjonen brukeren oppga (Denne modusen er ikke tilgjengelig for “vanlige” brukere).

5.8.3 Datainnsamlingsprosess

Webcrawler starter med et sett dokumenter og finner alle utgående lenker fra disse. Lenkene besøkes i bredde-først rekkefølge, men samtidig prøver den å besøke den eldste lenken hele tiden. Datainnsamlingsprosessen henter inn flere dokumenter samtidig (vanligvis bruker den fra 5 til 10 "agenter" samtidig). Webcrawleren forsøker å bygge en bredest mulig indeks. Hver gang den finner en ny server vil denne bli plassert først i listen over lenker som skal besøkes. Den forsøker også å være ressursvennlig ved å spre forespørslene mest mulig over de serverene den kjenner til (den itererer over listen av kjente servere og henter ett dokument fra hver per iterasjon).
I "agent"-modus (se forrige seksjon) er målet å finne nye dokumenter som stemmer best mulig overens med brukerens kriterier. Først prosesseres kriteriene mot databasen for å finne dokumenter som kan være et passende utgangspunkt. De ubesøkte lenkene i disse dokumentene som passer best, følges så videre. Når nye dokumenter er hentet inn kjøres et nytt søk i databasen med brukerens kriterier. Resultatene av dette søket sorteres og prosessen starter på nytt. Slik fortsetter den til tilstrekkelig med nye dokumenter er funnet eller en tidsbegrensning nås.
Webcrawler er skrevet i C og Objective-C, kjører under NextStep og bruker en modifisert versjon av WWW biblioteket fra CERN. Webcrawler støtter *ikke* Robot Exclusion Standard.

5.8.4 Indekseringsprosess

Databasen inneholder en liste over alle dokumenter Webcrawleren har besøkt, men også alle som den har sett referanser til men ikke besøkt ennå. I tillegg inneholder den en indeks over innholdet i alle dokumenter som er hentet inn. Indeksen er invertert slik at det blir raskere å slå opp i den. Når man slår opp på et ord får man tilbake en liste med pekere til dokumenter som inneholder dette ordet. Mer komplekse søk håndteres ved å kombinere dokumentlister for flere ord med vanlige sett operatorer. Indeksen bruker en "vector-space" modell [6].

5.8.5 Søkemuligheter

Søkemulighetene på Webcrawler er ganske begrenset. Den følgende korte listen oppsummerer disse :

- kan skrive inn en liste med ord og kombinere disse med enten OR eller AND operator (kan bare velge en operator, slik at alle ordene kombineres med enten AND eller OR)
- kan velge hvor mange dokumenter som skal resturneres som resultat
- sorterer resultatene etter dokumentenes tildelte vekt

5.9 MOMSpider

5.9.1 Generell informasjon

URL	Gjelder ikke
Eier / ansvarlig	University of California ved Roy Fielding
Sted	Gjelder ikke
Protokoller	HTTP
Størrelse	Gjelder ikke
Kommersiell	Gjelder ikke

5.9.2 Historie / bakgrunn

MOMspider er ikke et søkesystem, men et vedlikeholdsverktøy. Grunnen til at vi omtaler det her, er at vi har fått mange av ideene til vår robot fra MOMspider.
MOMspider tar for seg problemet med inkonsistente infostrukturer på WWW. Dette er et av de største problemene med WWW idag. Dokumenter er ofte knyttet sammen over flere servere med flere eiere. Dokumentene er også dynamisk og det oppstår svært ofte “løse” pekere i infostrukturen (URLer som peker på et dokument som er flyttet/fjernet). Etterhvert som infostrukturen vokser blir den enda vanskeligere å vedlikeholde. Vedlikeholdsarbeidet er også avhengig av at man sjekker loggfilene på serverene og retter opp feil som er registrert der. Dette kan være et svært tidkrevende og lite givende arbeid.
MOMspider tilbyr en delvis automatisert vedlikeholdsmetode. Den kan traversere en infostruktur, sjekke om de har endret seg og merke seg endringer som krever korrigeringer. Den lager også et dokument hvor den oppsummerer infostrukturen den har traversert. Dette dokumentet er et hypertekst-dokument og vil være til stor hjelp ved vedlikehold av infostrukturen.
I framtidens WWW vil det forhåpentligvis ikke være behov for verktøy som MOMspider (se seksjon 7.0 på side 58).

5.10 Fish Search

5.10.1 Generell informasjon

URL	http://www.win.tue.nl/bin/fish-search
Eier / ansvarlig	dr. P.M.E. De Bra og drs. R. Post ved Eindhoven University of Technology i Nederland

Sted	Eindhoven, Nederland (men kan også kompileres og kjøres lokalt fra andre steder)
Protokoller	HTTP
Størrelse	Gjelder ikke
Kommersiell	Nei

5.10.2 Historie / bakgrunn

Fish-search er et klient-basert søkeverktøy som tilbyr automatisk navigering i WWW. Den jobber mer eller mindre på samme måten som et menneske gjør når det søker i WWW sider, bortsett fra at det går mye fortere.
Fish-search henter inn startdokumentet og leter etter relevant informasjon i forhold til hva brukeren spesifiserte (Startdokumentet er også oppgitt av brukeren. En indekstjeneste som f.eks. Yahoo kan være et godt utgangspunkt). I tillegg lagrer fish-search alle lenker som den finner i dokumentet slik at disse kan besøkes senere dersom de er relevante for søket. Algoritmen som fish-search bruker, forsøker å finne dokumenter som er spredd utover forskjellige deler av WWW. Den favoriserer også naboer til relevante lenker, og lenker som ligger under relevante lenker.

5.10.3 Søkemuligheter

Fish search har et rikt utvalg av søkeopsjoner. En oversikt følger nedenfor :

- case sensitive og case insensitive søk
- AND funksjon (finn alle nøkkelord)
- kan lete i søkbare indekser (ISINDEX i HTML) for å finne relevante lenker
- kan settes opp til bare å lete i HTML dokumenter (ikke vanlige tekst-filer)
- kan lete med utgangspunkt i en brukers “hot-list” (bokmerker)
- kan begrense domenene som det skal søkes i
- kan spesifisere dybde og bredde på søket
- kan angi maksimum antall svar og en timeout verdi (angir maksimumverdi for hvor lenge fish-search skal lete)

5.11 World Wide Web Wanderer

5.11.1 Historie / bakgrunn

WWW Wanderer er et program som ble implementert av Matthew Gray med det formål å måle hvor hurtig WWW vokser. Målingene har pågått fra juni 1993 fram til januar 1996. WWW Wanderer er en enkel robot som bare registrerer nye WWW servere. Resultatene er oppsummert i tabellen nedenfor.

TABLE 1. Resultater fra WWW Wanderer

Month	# of Websites	% .com sites	Hosts per Web Server
6/93	130	1.5	13 000
12/93	623	4.6	3 475
6/94	2 738	13,5	1 095
12/94	10 022	18.3	451
6/95	23 500	31.3	270
1/96	90 000	50.2	100

Vi ser bl.a. at i andre halvdel av 1993 så fordoblet WWW seg på 3 måneder. I dag er denne ca. 5 måneder.
Den tredje kolonnen viser prosentvis vekst av kommersielle WWW servere (.com domenet i USA).
Vi ser også at hver hundrede maskin på internettet i dag er en WWW server.

5.12 World Wide Web Worm

5.12.1 Generell informasjon

URL	http://guano.cs.colorado.edu/wwww/
Eier / ansvar-lig	University of Colorado ved Prof. Oliver A. McBryan
Sted	Boulder, Colorado, USA
Protokoller	HTTP
Størrelse	3 000 000 URLer
Kommersiell	Nei

5.12.2 Historie / bakgrunn

WWW Worm var en av de første robotene som fantes, og hadde en database på 110 000 URLer tidlig i 1994. Den har i dag vokst til 3 000 000 URLer. Den skiller seg ut fra andre roboter ved at den bare indekserer lenker og URLer i dokumentene den henter inn. Allikevel gir den svært gode søkeresultater og ble utropt til beste navigeringsverktøy i 1994 (Best Navigational Aid 1994) i en konkurranse.

5.12.3 Datainnsammlingsprosess og indekseringsprosess

WWW Worm traverserer WWW fra et utganspunkt (som alle andre roboter), men lagrer bare lenkene den finner i dokuementene. For en gitt lenke lagres : 1) lenken som pekte til denne lenken 2) den klikkbare hyperteksten til lenken 3) lenkens URL og tittel.
WWW Worm støtter Robot Exclusion Standard og begrenser traverseringen ved å angi en maksimumsdybde.
Selve søkemekanismen er basert på UNIX kommandoen *egrep*. Søkegrensesnittet oversetter brukeren søkeuttrykk til en tilsvarende *egrep* søkestreng. Standard *egrep* regulære uttrykk og “wild cards” er tillatt.

5.12.4 Søkemuligheter

- lete i URLer i databasen, lete i tittelfeltet, lete i dokument addresser
- enkle boolske uttrykk (AND mellom alle ord, eller OR mellom alle ord)
- antall returnerte dokumenter

5.13 Yahoo

5.13.1 Generell informasjon

URL	http://www,yahoo.com
Eier / ansvarlig	Yahoo!
Sted	Mountain View, Californina, USA
Protokoller	HTTP
Størrelse	ca. 150 000 URLer
Kommersiell	Reklamefinansiert

5.13.2 Historie / bakgrunn

Yahoo (http://www.yahoo.com) er en annen type søkbar database på nettet. Yahoo er basert i USA og katalogiserer WWW-servere over hele verden (med hovedvekt på USA). Forskjellen mellom denne og

andre søkbare WWW-databaser, er at de ikke bruker roboter til å hente inn dokumenter fra WWW. Yahoo katalogen er beregnet på brukere som vet hva de leter etter ,og kjenner Yahoo så godt at de vet hvilken kategori de vil finne det i. Det kreves også at brukeren har god tid til å navigere seg gjennom hierarkiet for å finne fram, da det ikke alltid er like logisk oppbygd.
Den 18. sept. inngikk Yahoo en avtale med Open Text Corp. (se seksjon 5.3 på side 33) som innebærer at Yahoo skal bruke Open Text 5 indekserings og -søkesystem. Dette forbedret søkemulighetene på Yahoo, som tidligere bare hadde manuell søking (navigering i hierarkiet).
Yahoo har ca. 300 000 besøkende om dagen og kjører på en blanding av SGI Indy og Intel Pentium maskiner koblet til en T3 linje (45 Mbit/s).

5.13.3 Datainnsamlingsprosess og Indekseringsprosess

URLer samles hovedsaklig inn med hjelp fra brukerene. Man fyller da ut et skjema, hvor man i tillegg til tittel og URL, foreslår en kategori som URLen skal plasseres under (en del annen informasjon kan også oppgis, men det er ikke påkrevd).
Yahoo har også en form for automatisk registrering, ved at de automatisk sjekker steder hvor nye URLer vanligvis blir annonsert (f.eks. visse Usenet News-grupper).
Yahoo har begynt å bruke Open Text 5 systemet fra Open Text Inc. for å gjøre hele søk i databasen sin enklere og mindre tidkrevende (det gamle Yahoo systemet med hierarki fungerer som en innholdsfortegnelse, mens Open Text 5 systemet gir det en indeks i tillegg). Se seksjon 5.3.4 på side 35 for mer informasjon om Open Text 5.

5.13.4 Søkemuligheter

Søking på Yahoo er enten manuell (hierarki navigering) eller automatisk. Ved automatisk søking oppgir brukeren en del nøkkelord og kan i tillegg velge blant :

- treff når alle nøkkelordene er funnet(AND) eller minst en av nøklene er funnet (OR)
- nøkkelordene kan tolkes som substrenger eller hele ord
- kan velge antall treff som returneres per resultatside

5.14 Kvasir

5.14.1 Generell informasjon

URL	http://www.sn.no/kvasir/
Eier / ansvarlig	Schibstednett A/S
Sted	Oslo, Norge
Protokoller	HTTP, FTP
Størrelse	4438 URLer (Jan. 96)
Kommersiell	Tjenesten er gratis. Inneholder reklame.

5.14.2 Historie / bakgrunn

Kvasir, som drives av Schibstednett (tidligere Oslonett A/S), baserer seg i utgangspunktet på at brukerene av WWW skal registrere URLer i deres database på samme måte som Yahoo (se seksjon 5.13 på side 48). Disse URLene blir deretter klassifisert og plassert i et hierarki som skal være lettere for mennesker å navigere i. Hierarkiet på Kvasir var i begynnelsen et eneste stort rot, men dette har blitt noe forbedret etter at Schibstednett gikk inn som eier.

5.14.3 Søkemuligheter

I Kvasir kan man velge om man vil søke i URLer, titler, beskrivelser og nøkkelord. I tillegg kan man begrense søket til et eller flere skandinaviske land. Man kan også begrense søket ved å angi at oppføringene som returneres må være nyere enn et visst antall dager.

5.15 Origo Index

5.15.1 Generell informasjon

URL	http://origo.telenor.bo/index/
Eier / ansvarlig	Telenor Media A/S
Sted	Oslo
Protokoller	HTTP, FTP, Gopher, Usenet News
Størrelse	5 663 oppføringer (Jan. 96)
Kommersiell	Tjenesten er gratis.

5.15.2 Historie / bakgrunn

Index drives av Telenor Media A/S og baserer seg på lignende prinsipper som Kvasir og Yahoo. Index har et team av personer (såkalte nettjegere) som har som oppgave å finne interressante steder på WWW, beskrive dem og katalogisere dem i et hierarki. Dette hierarkiet er bygget opp etter kjente katalogiseringsprinsipper og drar nytte av erfaringer gjort i bl.a. bibliotekbransjen.
Index er på mange måter bedre enn Kvasir :

- de satser mer aktivt på å skaffe seg nye oppføringer
- personene som legger inn oppføringer jobber med visse emner og holder seg oppdaterte innefor disse
- inneholder mer informasjon om hver oppføring
- mer omfattende søkemuligheter
- bedre oppbygd hierarki

5.15.3 Datainnsamlingsprosess

Nettjegerne på Index er hovedansvarlige for å legge inn nye oppføringer i databasen, men det er også mulig for andre å foreslå oppføringer. Ved ny oppføring må man oppgi en god del informasjon bl.a. tittel, URL, kategori (plass i hierarkiet på Index), beskrivelse, nøkkelord mm.

5.15.4 Søkemuligheter

Ved søking i Index kan man velge :

- søk i en eller flere av : tittel, URL, beskrivelse eller nøkkelord
- treff ved enten : minst ett av søkeordene finnes, alle søkeordene finnes eller alle søkeordene finnes etter hverandre
- tolke søkeordene som substreng eller hele ord
- skille mellom små og store bokstaver (case sensitve og case insensitive search)
- begrense søket til et eller flere skandinaviske land
- søke bare i angitt del av katalogen (angir kategori og søker bare i de som ligger under eller i denne)

5.16 Hvilket søkesystem er best ?

Etter å ha skrevet om og testet de viktigste søkesystemene som er tilgjengelige på WWW idag, har vi prøvd å objektivt vurdere hvilket som er det beste. “Information retrieval” verktøy måles ofte etter to kriterier : presisjon (precision) og relevans (recall) [6]. Med presisjon mener vi hvor godt dokumentene som returneres stemmer overens med søkekriteriene. Med relevans (recall) måler vi hvor stor del av dokumentene som returneres som er relevante dokumenter i forhold til søkekriteriene. Et eksempel følger : en dokumentdatabase inneholder ti dokumenter, fem handler om koffein. Et søk etter "koffeinfri kaffe" som returnerer 4 dokumenter om koffein og to andre, har en presisjon på 0.66 og en relevans på 0.80.

Relevansgraden er stort sett bra i alle søkeverktøyene. Det er som oftest ikke noe problem å finne nok relevante dokumenter. Problemet er presisjonen, siden alle systemene vi har prøvd returnerer veldig mange falske treff. Dokumenter som returneres etter søk på nøkkelord, trenger bare inneholde nøkkelordet et eller flere steder i teksten. Veldig ofte er flesteparten av dokumentene ikke relatert til det som brukeren søkte etter. Ved å veie dokumentene etter f.eks. antall nøkkelordforekomster og hvor i dokumentet treffene forekom, blir resultatene bedre, men irrelevante dokumenter forekommer fremdeles.

En annen faktor som begrenser presisjonen av søkene, er at brukerene ikke oppgir presise nok søkekriterier. Generelt kan vi si at søkene blir mer presise desto flere nøkkelord som blir lagt til. Undersøkelser som er gjort på Lycos viser at 95% av søkene bare innholder ett søkeord, Tilsvarende undersøkelse ved Webcrawler sier at brukerne i snitt bruker 1.5 søkeord. Dette er i de fleste tilfeller ikke nok til gi et presist resultat tilbake.

Våre forsøk med med søkesystemene i dette kapitlet gir Altavista (se seksjon 5.5 på side 37) som en klar vinner både når det gjelder presisjon og relevans. Den har en veldig stor database med dokumenter, samtidig som den har et godt utvalg av søkeopsjoner slik at det er mulig å spesifisere relativt presise søkekriterier. I tillegg kan den søke både i WWW og Usenet News.
Av de nest-beste søkesystemene kan vi nevne Lycos, Open Text og InfoSeek. Disse er også svært bra, men stiller ikke helt i samme klasse som Altavista både når det gjelder presisjon og relevans.
Et fellestrekk ved alle søkemaskinene som vi har nevnt, er at de ikke har spesielt avanserte kombinatorikk muligheter. Tester som vi har utført indikerer at de fleste søker gjennom datamengden flere ganger, for så å kombinere resultatet av alle søkene til en liste over dokumenter som tilfredstiller søkekriteriene (f.eks. hvis man søker etter tre ord med AND mellom, vil de fleste søke tre ganger i datamengden og derettet kombinere resultatene til ett gyldig resultat). Dette kan vi gjøre bedre på vårt system ved å utnytte MS160-brikken avanserte kombinatorikk muligheter.

Det største problemet for søkemaskinene vi har testet, er at de av og til er så godt som utilgjengelige fra Norge pga. nettets belastning.

5.17 Motivasjon for oppgaven

Etter å ha studert andre søkeverktøy på WWW og kommet fram til at de til en viss grad tilfredsstiller krav til god informasjonssinnhenting (presisjon og relevans), kan man stille seg spørsmålet om det er noen vits å lage en til. Nedenfor følger punktvise lister med argumenter både for og imot. Vi antar hele tiden at vårt søkesystem kun er innrettet mot norske WWW ressurser.

Argumenter for :

- å lete med de amerikanske søkesystemene er av og til vanskelig fordi de ikke alltid er tilgjengelige fra Norge pga. høy belastning på nettet
- en robot basert i Norge vil kunne bygge en database over norske WWW ressurser raskere enn roboter som er basert i utlandet. Den vil også kunne vedlikeholde databasen raskere, slik at den er mere oppdatert enn tilsvarende databser i utlandet
- databasen kan bli mer komplett i forhold til utenlandske databaser, fordi vi konsentrerer oss kun om Norge mens andre henter inn dokumenter fra hele verden. F.eks. hvis vår robot har samme dybdebegrensning som en basert i utlandet, er sjansene store for at vår robot henter inn flere dokumenter fra Norge, siden den utenlandske antakelig vil gå via andre land før den kommer til Norge.
- ikke alle utenlandske søkesystemer håndterer norske bokstaver. Dette håper vi at vi skal få til med vårt system.
- når datamengdene blir så store som de er på WWW (med så stor vekst), virker det naturlig å opprette spesialiserte søketjenester
- fin anledning til å vise hva MS160 kortet kan brukes til ovenfor det norske WWW publikum

Argumenter mot :

- finnes gode søkesystemer fra før (men kanskje ikke gode nok)
- MS160 kortet har i dag ikke stor nok kapasitet til å ha alle WWW dokumenter i Norge i minnet (på ett kort)

6.0 Hvor stort er WWW?

6.1 Definisjoner

Før vi kan besvare dette spørsmålet, må vi definere hva vi mener med WWW. Vi velger å definere WWW som alle dokumenter som befinner seg i ett av de følgende områdene :

- FTP område
- Gopher område
- HTTP område

Følgende områder regnes ikke som WWW :

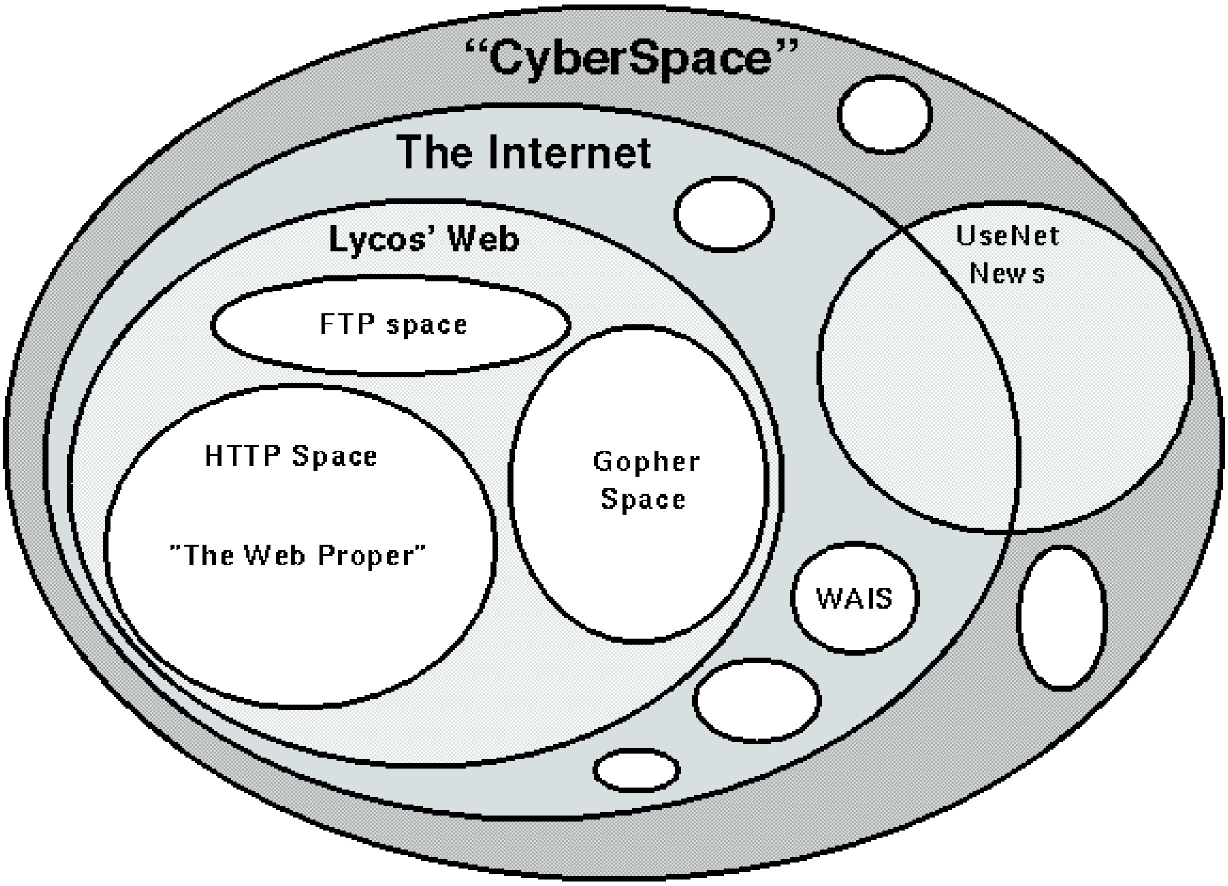
- WAIS databaser
- USENET News
- TELNET tjenester
- E-mail

Vår søketjeneste vil i første omgang være beregnet kun norske dokumenter som befinner seg på HTTP området. Dette er primært pga. plasshensyn.

Vi vil også overse URLer som genereres av CGI-skript/programmer. I praksis vil dette skje ved at vi overser URLer som inneholder enten et spørsmålstegn (?) eller et likhetstegn (=), da disse primært brukes i forbindelse med CGI skript. Vi vil også overse URLer som har tilstandsinformasjon kodet inn. Et eks. på dette kan være spill som spilles over internett (f.eks http://info.gte.com/cgi-bin/mines?8Wm----t-----7Z0G2H02f00Ga5. Denne har dog ? i URLen og faller bort av den grunn.).

Figuren under er hentet fra Lycos og forsøker å gi en illustrasjon av hva som menes med “cyberspace” og hva som er WWW for oss.

FIGURE 6. En oppdeling av Cyberspace



Som figuren viser har vi valgt å kun betrakte HTTP området i første omgang.

6.2 Størrelsen av WWW

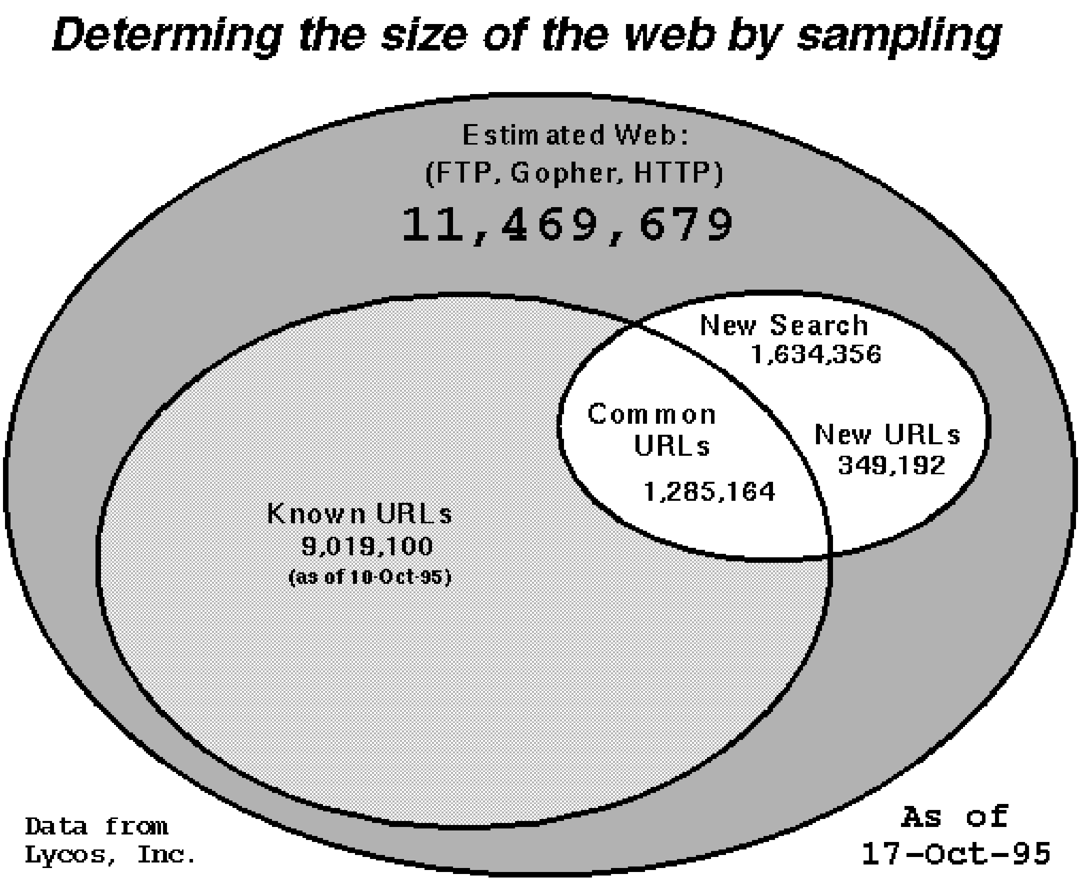
For å estimere størrelsen av WWW vil vi ta utgangspunkt i en undersøkelse som er gjort tidligere. I tillegg skal vi vha. egne data forsøke å estimere størrelsen av WWW i Norge.

6.2.1 Undersøkelse gjort av Lycos

Lycos er den søketjenesten som per idag har flest URLer i sin database. Lycos henter inn nye dokumenter kontinuerlig og slår sammen disse dataene med de som ligger i databasen fra før. For å estimere størrelsen av WWW har de tatt alle data som er hentet inn i løpet av en uke (de antar at disse er et tilfeldig utvalg av WWW i sin helhet) og sammenlignet disse med de som ligger i den "gamle" databasen.

Sammenligningen består i å multiplisere størrelsen av det gamle datasettet med forholdet mellom størrelsen på det nye datasettet og størrelsen på snittet av det nye og gamle datasettet. Figur 7 på side 56 gir en illustrasjon av dette.

FIGURE 7. Lycos beregningsgrunnlag for størrelsen av WWW



Den 4. april 1995 hadde Lycos 2,687,146 URLer i det "gamle" datasettet. En ny innhenting hadde da gitt 680,011 URLer, hvorav 408,839 fantes i det gamle datasettet. Dette gir et forhold på 1.508 og et estimat på 1.508 * 2,687,146 = 4.051 millioner URLer på WWW.

Den 17 oktober hadde Lycos 9,019,000 URLer i det "gamle" data settet. En ny innhenting hadde da gitt 1,634,356 URLer, hvorav 1,285,164 fantes i det gamle datasettet. Dette gir et forhold på 1.272

og et estimat på 1.272 * 9,019,000 = 11.469 millioner URLer på WWW.

Blant andre relevante data kan vi nevne at Lycos har hentet dokumenter fra 80,985 unike HTTP servere og at gjennomsnittstørrelsen for dokumentene er 7,874 bytes.

Lycos estimat for størrelsen av WWW blir da 11.469 mill * 7,874 bytes = 84.1 gigabytes.

Den største feilkilden er at stikkprøvene som tas i løpet av en uke ikke er virkelig tilfeldig blanding av URLer. Lycos velger de mest populære dokumentene først, slik at et nytt søk har en tendens til å overlappe det gamle datasettet mer en et virkelig tilfeldig utvalg ville ha gjort. Denne feilkilden innebærer at forholdet som er beregnet ovenfor er mindre enn det burde være (siden nevneren er større enn for et virkelig tilfeldig utvalg). 84.1 gigabytes blir derfor en nedre grense for estimatet.

6.2.2 Vårt estimat

Vår datamengde er forskjellig fra den Lycos har på flere områder. Vi må ta hensyn til at vi har svært begrenset med plass pga. minnebegrensningene til MS160 kortet. Denne begrensningen vil kanskje bli borte (eller bli mindre) i fremtiden, etterhvert som nye versjoner av kortet blir utviklet. Pga. av denne begrensningen velger vi å bare hente inn dokumenter fra HTTP servere, i motsetning til Lycos som henter dokumenter fra HTTP, FTP og GOPHER servere. Selv med denne begrensningen vil vi overstige MS160 kortets minnebegrensning på et tidlig stadium.
Vi henter selvfølgelig bare inn rene tekst/html dokumenter, siden det er disse som er aktuelle å søke i.
Lycos henter inn dokumenter fra hele verden. Vi velger å holde oss i Norge og skal prøve å oppnå størst mulig dekningsgrad her.
Med disse begrensningene skal det være mulig å få store deler av det som ligger på WWW i Norge inn på minnet som er tilgjengelig.

Lycos gir et estimat av størrelsen av WWW ved å ta tilfeldige prøver av WWW og sammenligne disse. Vår robot opererer på en annen måte slik at dette ikke er mulig for oss. Roboten vår tar ikke tilfeldige prøver av WWW, men følger lenker fra sitt utgangspunkt i følge en bredde-først algoritme [7]. Vi må derfor bruke en annen metode for å komme fram til et estimat. Vi tar utgangspunkt i følgende formel :

Ligning for beregning av størrelsen av WWW i Norge **(EQ 1)**

$$\text{Størrelse av WWW i Norge} = \text{ant. servere i Norge} \cdot \text{gj.snittlig antall doks per server} \cdot \text{gj.snittlig dok.størrelse}$$

Det største usikkerhetsmomentet ved denne formelen er antall WWW servere i Norge. Her må vi bare ta utgangspunkt i hva roboten vår har funnet. Foreløpige tall viser at det er ca. 300 WWW servere i Norge. Dette tallet vil øke etterhvert som roboten finner nye. Gjennomsnittlig antall dokumenter per server og gjennomsnittlig dokumentstørrelse finner vi enkelt ved å studere dataene som er samlet inn. Etter en tidlig kjøring av roboten fant vi fram til følgende tall :

- antall WWW servere : 434
- gjennomsnittlig antall dokumenter per server : 47.81
- gjennomsnittlig dokumentstørrelse : 4575 byes

Dette gir oss en samlet størrelse på WWW i Norge på ca. 93 Mb. Senere kjøringer viste at dette var et altfor lavt tall. Spesielt var gjennomsnittsstørrelsen på dokumenter og gjennomsnittlig antall dokumenter per server altfor lavt. Etter flere kjøringer økte disse tallene slik at ny samlet størrelse på WWW i Norge ble 950 Mb (maksimumsverdi).

7.0 Fremtiden og søking på World Wide Web

WWW er idag blitt veldig stort og vokser fremdeles med stor hastighet. De beste robotene har greid å lage indekser som dekker en stor del av WWW, men hvis man ser på vekstraten virker det nesten umulig å bygge en ordentlig katalog/indeks som dekker hele WWW. Agent-lignende verktøy som f.eks. fish-search (se seksjon 5.10 på side 45) får også problemer med å finne relevante dokumenter, fordi det hele begynner å ligne på et “nåla i høystakken”-scenario.

Selv om man har tilstrekkelig med tid tilgjengelig, slik robotene har, kan det allikevel være vanskelig å få tak i den informasjonen man ønsker. Det er alltid deler av WWW som ikke er tilgjengelig, selv fra mange forskjellige utgangspunkter og det finnes også servere som ingen peker på ennå. I tillegg er også mye informasjon skjult bak klikkbare bilder og skjemaer (antall mulige koordinater på et klikkbart bilde er altfor stort til at man kan prøve alle kombinasjoner for å finne informasjonen bak dem).

Kjernen i problemet ligger allikevel et annet sted. Tilgang til søkemuligheter på en server burde tilbys av den serveren, og ikke av andre servere et helt annet sted på nettet.
Denne ideen motiverte menneskene som står bak GlimpseHTTP (se seksjon 5.1 på side 30) til å utvikle et skjema (HTML Forms) basert søkegrensesnitt, slik at all informasjon som er lagret på en Glimpse-HTTP-server er søkbart via et dette. GlimpseHTTP har de vanlige søkemulighetene (boolske uttrykk, substreng osv.) i tillegg til støtte for regulære uttrykk. GlimpseHTTP databasene inneholder all informasjon som ligger lagret på en server, også den som normalt er skjult bak kilkkbare bilder og skjemaer.
En database som GlimpseHTTP kan vedlikeholdes daglig (automatisk). En komplett og oppdatert liste over alle nodene på en server som inneholder en gitt streng, kan hentes inn i løpet av få sekunder. Dette kunne man ikke fått til med dagens søkesystemer (robot-basert), siden disse aldri er oppdaterte eller komplette.
For å søke gjennom hele WWW (eller store deler av det) etter en streng eller et uttrykk, kunne man bruke en algoritme lignende den fish-search bruker. I stedet for å hoppe fra dokument til dokument, kunne man hoppe fra server til server. For å oppnå dette, måtte serverene tilby en liste over noder som er relevante i forhold til søkekriteriene, samt en liste over alle andre servere den kjenner til. Man kunne da stille disse serverene det samme kravet : “ Gi meg en liste over dokumenter som inneholder denne søkestrengen og en liste over alle WWW servere du kjenner til”.

Å få til denne typen informasjonssøking på hele WWW er ikke så vanskelig som det kanskje høres ut. De finnes bare noen få WWW servere som har stor brukermasse. De mest populære er NCSA og CERN, men også Netscape og Apache øker i popularitet[8] (totalt finnes det nærmere 100 forskjellige WWW-server pakker). Dersom disse kunne inkludere et “GlimpseHTTP”-lignende verktøy og en standard for å aksessere dette, ville implementasjonen av av scenarioet diskutert ovenfor bli relativt enkel. Dagens indeks-databaser kunne erstattes av cacher som innholder de nyeste svarene til vanlige spørsmål. De kunne også innholde lister over servere som de kjente til.

De ufullstendige og utdaterte svarene man får fra dagens indeks-databaser, holder til mange bruksområder ennå en tid framover. For helt oppdaterte søk i en liten del av WWW, er verktøy som fish-search og andre agenter fremdeles brukbare. Men hvis man ønsker en informasjonstjeneste med høy kvalitet, er det klient baserte fish-search programmet for tregt, og den utdaterte annenhåndsinformasjonen i indeksdatabasene for upålitelig. I tillegg er det flere og flere (systemadministratorer) som har motforestillinger til robotenes bruk av nettverksressuerser. Ved å installere programvare på hver server som tilbyr oppdatert og komplett informasjon om sin del av nettet, kan WWW bli søkbart igjen.

7.1 En ny generasjon servere

Når det gjelder fremtiden til selve WWW, mener mange at en del av basisteknologien er klar for utbytting. HTTP protokollen i seg selv er ganske ineffektiv og HTML språket ødelegges av mange “ad-hoc” tillegg fra de forskjellige klient leverandørene.
Hyper-G representerer den nye generasjonen av informasjonssystemer på Internet. Det tilbyr virkelig hypermedia (kan f.eks. ha lenker mellom to animasjonsfiler, men har også verktøy for å vedlikeholde, strukturere og produsere dokumenter. Hyper-G garanterer også automatisk hyperlenke konsistens, gode tekstsøkingsmuligheter, et Unix aktig sikkerhetssystem og grensesnitt mot Gopher og WWW klienter. Hyper-G er fullt ut kompatibel med dagens Internett teknologi og tilbyr helt transparent tilgang til WWW og Gopher. Når det gjelder søkemuligheter har Hyper-G et konsistent grensesnitt for søking i alle typer tjenester (HTTP, WAIS, Gopher). Med Hyper-G servere kan man til enhver tid vite hvor i “Cyberspace”man befinner seg, og også få tegnet dette opp på skjermen. Lenkene i Hyper-G dokumenter er toveis lenker slik at man kan følge dem begge veier. Lenkene lagres heller ikke i dokumentene, slik som i vanlig HTML, men i egne databaser. Dette innebærer at lenker kan knyttes til filer av hvilken som helst type, uten at dokumentformatet må vite noe mer om lenken eller hvordan den fungerer.
Ovenfor har vi bare trukket fram noen av fordelene med Hyper-G servere fremfor dagens WWW servere. Hyper-G er ikke perfekt (f.eks. takler den ikke dårlig HTML kode på en tilfredsstillende måte. Men Hyper-G er stabilt, kraftig og fremfor alt tilgjengelig. Flere store bedrifter og organisasjoner har skiftet til Hyper-G allerede (f.eks. European Space Agency, ESA) og siden den også støtter dagens standarder kan den overta som “motor” på WWW uten store problemer.
For mere informasjon om Hyper-G se referanser [9] og [10].

En introduksjon til MS160

Ole Andreas Grytdal
Bjørn Kristian Hilstad

I dette kapittelet skal vi gi en introduksjon til MS160 kortet.

8.0 Ms160

I dette kapittelet skal vi ta for oss MS160 kortet, slik at leseren kan se litt mer om fordelene og ulempene til MS160 kortet.

8.1 Historie

MS 160 kortet har blitt utviklet over lengere tid. Arbeidet begynte på 1980-tallet da Arne Halaas fikk ideen til en slik chip, da han tok et "studieår" i Tyskland for å studere VLSI design. Men det var ikke før i 1992 at MS 160 brikken som den er idag så dagens lys. Arne Halaas hadde da skapt et firma sammen med Kåre Gisvold og Torstein Heggebø.
Det var i samarbeid med Nordic VLSI og Turbit at MS160 brikken ble fullført høsten 1992. Man så at det kunne være et behov for et spesial kort som var dyktig på tekst og database søk og "pattern matching".

8.2 Hva er MS160

MS 160 er en meget rask, ikke numerisk co-prosessor som kan jobbe i parallell med tekst gjenkjenning. Den opererer på en frekvens på 20 MHz og kan søke igjennom 160 MB/s. Dette tilsvarer ca 64000 A4 sider på et sekund, der en kan finne igjen deler av en hel setning. Man kan også lage mer komplekse søk, men til da en lavere hastighet, 20 MB/s. [11]

Det er komplekse søk som gir MS160 en så stor fordel framfør vanlige søk som gjøres i programvare. Den kan :

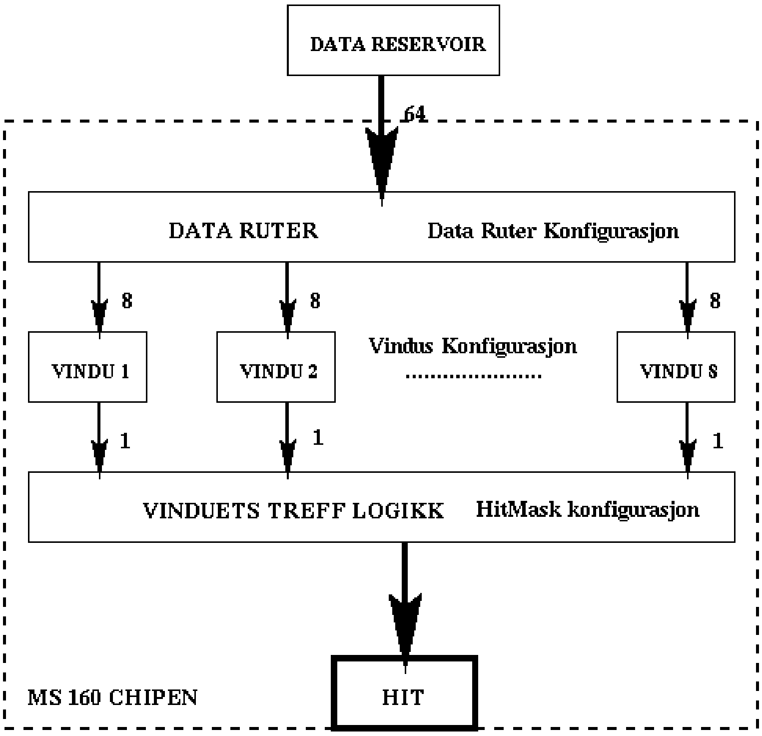
- finne ord som står innenfor en viss avstand f.eks 3 ord som skal gi et treff innenfor en avstand av 100 tegn.
- behandle søkeord som substrenger slik at en ikke behøver lage egne algoritmer for substreng søk.

8.3 Funksjonell beskrivelse av Ms160

I Figur 8, "MS 160 oversikt," på side 63. kan en se en oversikt over de viktigste delene som en MS160 består av. Datareservoaret inneholder alle dataene som skal til MS160 kortet og er egentlig ikke en del av MS160. Den fører data til en data fordeler som distribuerer data til de forskjellige vinduene. Datastrømmen kan strømme på endel forskjellige måter som vi skal ta for oss i "Data Ruter konfigurasjon" på side 64. MS160 består av 8 data vinduer, som hver inneholder et 32 byte shift register. Det er dette vinduet man setter søkekriteriet i, og som man kan konfigurere på endel måter som vi skal beskrive senere i

“Oppsett av søkevinduene” på side 68. Ved et treff i vinduet, blir det sendt melding til vinduets treff logikk del, som ut fra visse satte kriterier KAN genere et treff.

FIGURE 8. MS 160 oversikt



8.4 Datapumpens funksjonalittet

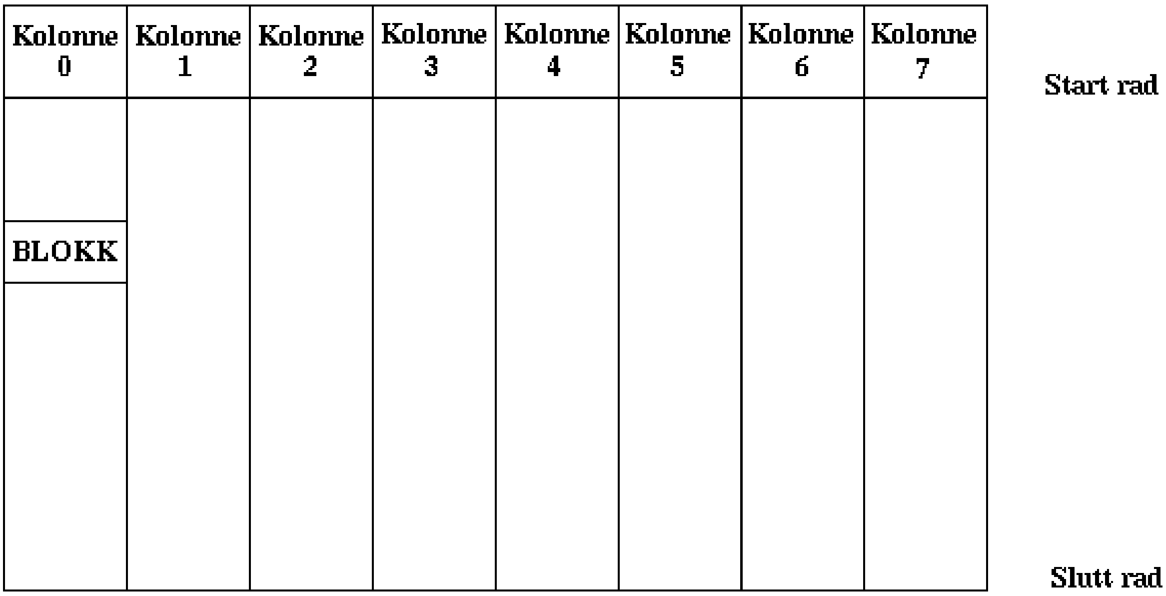
Datareservoaret er delt opp i åtte kolonner som deler RAM’en på MS 160 i like store deler. Kolonnene er oppdelt i blokker på 1 K størrelse og åtte blokker som går over alle kolonnene er en rad. Figur 9, “En oversikt over datareservoaret,” på side 64 viser dette.

Antall rader man kan ha er :

$$\text{Antall rader} = \frac{\text{Antall MB på kort}}{8KB} - 1$$

Skriving til datareservoaret skjer kolonnevis; dvs den fyller inn data i kolonne 0 først, og deretter videre til de andre kolonnene. Det er slik device-driveren virker; for nærmere informasjon henvises til [11]. Ved et datasøk settes start og slutt rad og datapumpen vil dytte disse dataene gjennom, og sammenligne de mot søkekriteriene.

FIGURE 9. En oversikt over datareservoaret



8.5 Oppsett av MS 160

For å få satt opp MS 160 riktig må en passe på å konfigurere kortet riktig. Som nevnt før er det tre ting som er viktig:

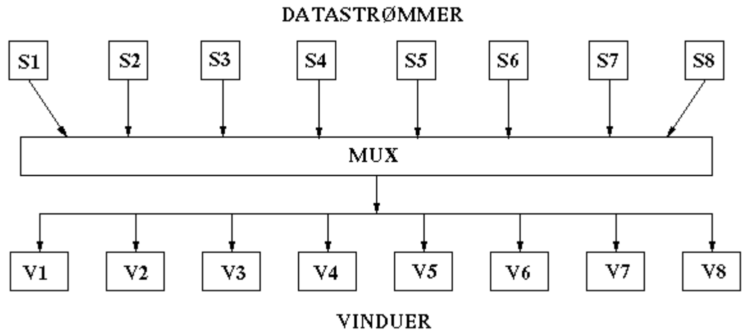
- Data ruter konfigurasjon
- Vindus konfigurasjon
- HitMask konfigurasjon

8.5.1 Data Ruter konfigurasjon

Data ruter oppsettet avhenger mye av hvordan distribusjon av søkekriteriene skal fordeles i vindusoppsettet. Det er mange måter å rute data på til vinduene, alt avhengig av hvor mange søkekriterier man har eller hvor stor datamengden er. Under skal vi vise endel eksempler som vil kunne forekomme ved MS 160 ruting.

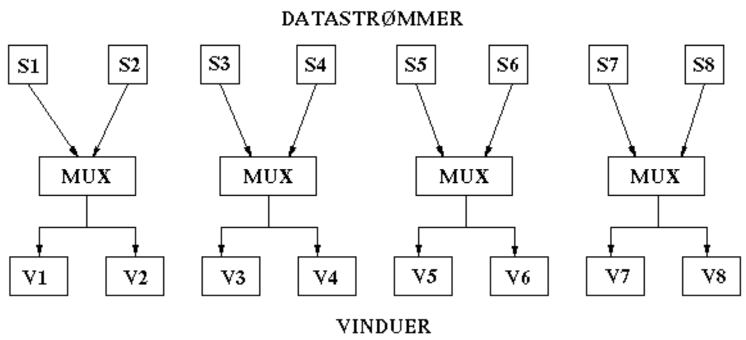
1) Åtte data strømmer hver til et vindu i parallell. Ved kun **ET** kriterie blir denne måten brukt. Det holder å sende data gjennom datapumpen en gang og man kan få et søk på 160 MB/s ved å duplisere søkekriteria til alle vinduene.

FIGURE 10. Åtte data strømmer hver til et vindu i parallell



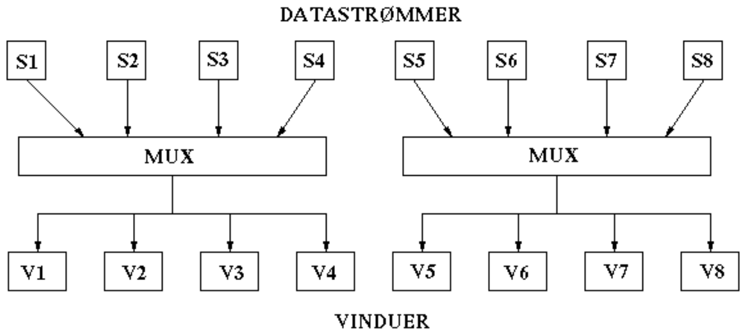
2) Fire data strømmer hver til to vinduer i parallell. Denne rutingen brukes når det er to kriteria som skal oppfylles. Man må sende data gjennom datapumpen to ganger, noe som senker datagjennomstrømmingen til 80 MB/s.

FIGURE 11. Fire data strømmer hver til to vinduer i parallell



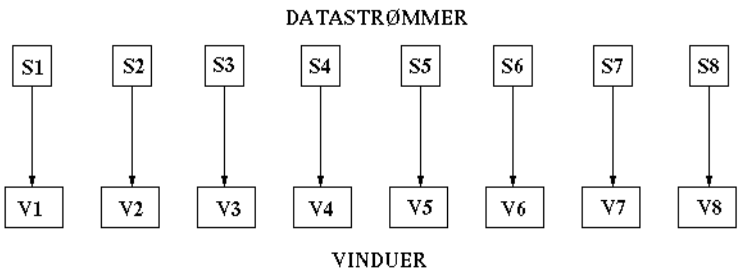
3) To data strømmer til fire vinduer i parallell. Denne rutingen brukes når det er tre eller fire kriteria som man ønsker å søke på. Nå må man sende data gjennom datapumpen fire ganger og datagjennomstrømningsraten senkes til 40 MB/s.

FIGURE 12. To data strømmer til fire vinduer i parallell



4) En data strøm til åtte vinduer i parallell. Denne rutingen brukes når det er enten fem, seks, syv eller åtte kriteria som det skal søkes på. Data må sendes gjennom datapumpen åtte ganger nå og dette senker datagjennomstrømmingen til 20 MB/s.

FIGURE 13. En data strøm til åtte vinduer i parallell



Det er også mulig å kombinere søk serielt for å skape store "supervinduer". Vinduene kan da bli større enn 32 byte slik at en kan søke på lenger ord, men normalt i tekst søk vil ikke dette være nødvendig.

En oversikt kan sees i Tabell 2, "Rutings og duplikat tabell," på side 67.

TABLE 2. Rutings og duplikat tabell

Antall kriteria	Antall duplikater	Antall gjennom-ganger i data-pumpa	Datagjenn omstrøm-ning i MB/s	Rutings metode
1	7	1	160	1
2	3	2	80	2
3	1	4	40	3
4	1	4	40	3
5	0	8	20	4
6	0	8	20	4
7	0	8	20	4
8	0	8	20	4

8.5.2 Oppsett av søkevinduene

Ved søk i MS160 må man sette opp søkevinduene med de kriteriene som det skal søkes på. For å få et så bra søk som mulig i parallell, bør man kopiere søkekriteriene til ubrukte vinduer slik at man får maksimal uttelling ved et data gjennomløp. Som man kan lese ut av Tabell 2, “Rutings og duplikat tabell,” på side 67 ser man hvor mange vinduer en kan kopiere til og hvor mange datagjennomløp en da trenger.

I et MS160 søk, må disse verdiene settes :

1. En 32 byte øvre grense på søkekriteriet.
2. En 32 byte nedre grense på søkekriteriet.
3. En 32 bit felt separator maske.
4. “Match latency”.
5. “Record lenght”.

De to første kriteriene må settes for hvert vindu. Disse spesifiserer tekststrengen man søker etter. Grunnen til at det er to tekst strenger er at man kan søke på både små og store bokstaver og samtidig også alle kombinasjoner i mellom øvre og nedre grense. Når man fyller inn søkeordet inn i den 32 byte lange strengen bør en sette resten av verdiene i strengen til “Don’t care”. Dette gjør man ved å sette verdien 255 (0xFFh) inn i resten av cellene i øvre grense og verdien 0 inn i resten av cellene til nedre grense. Nedenfor er det vist et eksempel der dette bedre kommer frem.

FIGURE 14. Vindus konfigurasjon

Øvre grense : DIPLOM************************

Nedre grense: diplom--------------------------

Match Latency : 1

Field Seperation Mask : 11111111111111111111111111111111

Record Length : 1

I dette tilfellet er * det samme som verdien 255, og - tegnet er verdien 0. Hvis en konfigurer alle vinduene slik, vil den lete etter alle substrenger som har DIPLOM / diplom i seg. Men ved å gjøre det på denne måten vil en få en masse falske treff, bla. eiplom, fiplom etc etc. For å få et eksakt treff bør en derfor sette det samme i øvre og nedre grense. Problemet som oppstår da er at en KUN kan søke på enten DIPLOM eller diplom, hvis en ikke trikser litt med tegnsettet, noe som er forklart i senere i oppgaven. I en søkemaskin er det viktig å få valgt "CASE-sensitivt" og "in-case-sensitivt" søk, derfor valgte vi å få til denne funksjonen.

"Match latency" forteller hvor lenge MS160 skal huske et hit. Når en søker etter to eller flere ord i en tekst, settes denne verdien til et tall som er større en antall ord som det skal søkes på. Søker man f.eks på to ord og match latency er satt til 50 og disse to ordene forekommer innen en avstand av 50 tegn, vil et treff registreres.

"Record lenght" brukes når en søker i database og hver post har en fast lengde.

"Field seperation bit mask" brukes til å spesifisere et felt i datamengden.

8.5.3 Rapporterings modus

Når MS160 har funnet et eller flere treff, er det flere måter å få tilbakemelding på.

- Count / Report : Ved Count satt, vil MS160 kun telle opp antall treff og ikke lage en Treff liste som den vil gjøre hvis man velger Report.
- Continue / Stop : I Stop modus venter man på en bekreftelse av et innlest treff, men i continue modus leses alle treff inn med en gang.
- Flank / Hits : Hvis Flank modus er valgt, vil MS160 kun registrere et treff hvis det er flere kriteria som slår til innenfor størrelsen på Match Latency. I Hits modus vil alle treff bli registrert.

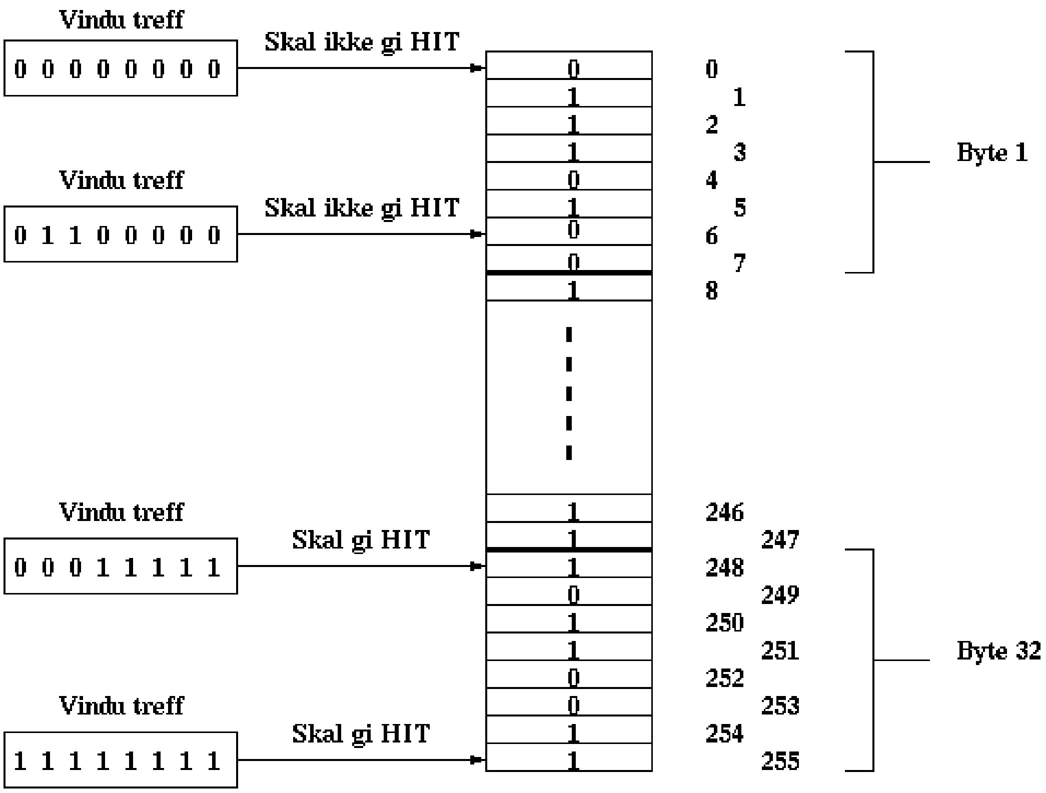
Til vårt bruk vil vi bruke modusene Report , Continue og Flank.

8.5.4 Konfigurasjon av treff masken

Når vinduene i MS160 skaper et treff, er det en følge av hva som er satt opp som vinduets treffkriterium. Kortet har åtte vinduer, noe som gir 256 mulige treff kombinasjoner. Disse treff kombinasjonene settes via et 32 byte register (32*8 = 256). Treffkombinasjonene settes da som enkelt bit i treff masken og sier noe om hvilke kombinasjoner som skal være gyldige.

Som et eksempel med et kriteria som kan bli kopiert til alle vinduene, settes alle bitene i treff masken til 1, bortsett fra det alle første bitet som settes til 0. 0 i bitposisjon 0 betyr at hvis det ikke er noen kriterium som passer i noen av vinduene, så skal ikke et treff registreres. 1 i posisjon 255, betyr at hvis alle vinduene og dets søkekriterier samsvarer skal et treff genereres. Dette er også vist i Figur 15, “Hit Mask konfigurasjon,” på side 70.

FIGURE 15. Hit Mask konfigurasjon



8.6 Oppkjøring av MS160

Ved oppkjøring og søk på MS160 gjør man slik :

1. Laster inn data til MS160 kortets RAM. Ved lite data (mindre enn en kolonne) kan man duplisere dataene i RAM slik at parallell søk vil gå mye raskere.
2. Sette opp søkekriterier til vinduene, søkemodus, treff masken og rutings informasjon.
3. Starte datapumpen.
4. Lese ut treff og maske bort mulige falske treff.
5. Finne URLene til de treff som er gyldige
6. Skrive ut resultatene.

Dette er i hovedtrekk også hvordan søkemaskinen kommer til å bli oppbygd.

Kravspesifikasjon for WWW-robot

Ole Andreas Grytdal
Bjørn Kristian Hilstad

"requirements are about understanding a problem prior to proposing a solution..."

9.0 Introduksjon

9.1 Hensikten med dette dokumentet

Dette dokumentet spesifiserer krav til WWW-roboten vår. Vi beskriver funksjonaliteten systemet tilbyr, men hovedfokus er på hvordan oppgavene skal utføres. Vi definerer krav for :

Hva WWW-roboten skal gjøre.

Hvordan brukeren kan kommunisere med programmet (konfigurasjon).

Hvordan WWW-roboten bruker andre subsystemer (programpakker).

Hva som skal til for å implementere og kjøre WWW-roboten.

9.2 Organisering av dokumentet

Kravspesifikasjonen består av 5 hoveddeler :

Seksjonen “Generell systembeskrivelse” gir først og fremst en overordnet beskrivelse av WWW-roboten, men sier også litt om hvordan bestemte oppgaver blir utført.

Seksjonen “Ikke-funksjonelle krav” beskriver kravene som WWW-roboten stiller til sine omgivelse, dvs. hva slags programvare og maskinvare som må være tilgjengelig for at alt skal fungere tilfredstillende.

Seksjonen “Funksjonelle krav” spesifiserer alle funksjoner WWW-roboten må ha og vil også begrunne de valg vi har gjort ved valg av funksjonalitet.

Seksjonen “Konfigurasjon” beskriver alle de konfigurerbare delene av WWW-roboten og formatet på konfigurasjonsfil.

Seksjonen “Implementasjonskrav” lister opp software og hardware som trengs under implementasjon, samt en del regler som skal følges når man skriver kildekode.

10.0 Generell systembeskrivelse

En WWW-robot har som oppgave å traversere en størst mulig del av WWW. Dokumentene som hentes ned lagres lokalt, slik at det er mulig å lage indekser og søke i dokumentmassen. I dette tilfellet skal vi i stedet for å indeksere dokumentene benytte en eller flere “MS160 Search Engines” for å muliggjøre søk. Med dette ønsker vi å tilby hurtigere og mer sofistikerte søkemuligheter enn de som allerede er tilgjengelige på WWW idag.
Vi skal nå i korte trekk skissere fremgangsmåten WWW-roboten benytter når den traverserer WWW. Det første steget i prosessen er å velge et utgangspunkt i WWW. Det betyr at man velger et spesifikt dokument (en URL) som man skal starte traverseringen i. Alle lenker (URL’er) i dette rotdokumentet plukkes ut og lagres for senere traversering. Prosessen fortsetter rekursivt inntil visse stoppkriterier er oppfylt. Etter en evaluering av søketjenester på WWW idag ,se Seksjon 5.0 på side 30, er det to ting som fremstår som vitale egenskaper : bredde først søk og robot-aktig oppførsel. Tilsammen gjør disse to egenskapene at vi får et veldig bredt nedslagsfelt for roboten og vi får med flest mulig servere i søket vårt.

10.1 System kontekst

Roboter har nå blitt brukt på WWW i ca. 2 år. I løpet av denne perioden har de gjort nytte for seg ved å tilby en etterspurt tjeneste, men har også gjort skade ved å overbelaste nettverk (men dette er som oftest når roboten er dårlig implementert og ikke følger visse basisregler).
Roboter kan brukes til utføre nyttige oppgaver som :

Statistisk analyse
Den første roboten på WWW [12] ble kjørt for å telle antall WWW servere. Andre statistiske opplysninger kan være f.eks. gjennomsnittlig antall dokumenter per server, gjennomsnittstørrelse av en WWW side, graden av interkonnektivitet i dokumenter osv.

Vedlikehold av WWW sider
En av de største problemene ved å vedlikeholde en hypertekst struktur, er at referanser til andre sider (lenker) kan bli ødelagt når den refererte siden flyttes eller fjernes. Det finnes per i dag ingen generell mekanisme som automatisk varsler alle som har lenker til en side om at lenken er endret/fjernet. Noen servere, som f.eks. CERN HTTPD, logger alle forespørsler som ikke er vellykkede (som et resultat av en “død” lenke) sammen med en referanse til siden hvor feilen oppstod.

Dette gjør det mulig å rette feilene i lenkene, men er ikke særlig praktisk. I virkeligheten er det som oftest dokumentforfatterne som oppdager "døde" lenker i sine egne dokumenter. Det er mao. sjelden at andre tar seg bryet med å si i fra til dokumentansvarlig/forfatter når de oppdager "døde" lenker i andres dokumenter.
En robot som automatisk sjekker alle lenker og gir beskjed om "døde" lenker, kan være til stor hjelp når man skal vedlikeholde en hypertekststruktur. En robot kan også være behjelpelig med å vedlikeholde innholdet av dokumentene, ved f.eks. å sjekke at HTML standarden blir fulgt. Denne funksjonaliteten burde også være integrert i HTML-editorer, slik at man kunne sjekke alle dokumenter hver gang de ble endret og rette feil umiddelbart.

- **Speiling av WWW sider**
  Speiling (mirroring) er en populær teknikk som ofte brukes til i forbindelse med FTP arkiver. En "mirror-site" kopierer hele eller deler av katalogstruktur rekursivt via FTP, og henter så regelmessig de dokumenter som har endret seg eller kommet til. På denne måten får man delt belastningen mellom flere FTP servere, redundans i datalagringen slik at man lettere takler feil på enkeltservere og får hurtigere, lokal aksess på flere steder (og off-line aksess).
  Speiling av WWW sider kan implementeres vha. en robot, men vi har ikke funnet noen softwarepakke som tilbyr dette på en tilfredstillende måte. Det finnes riktignok roboter som kan hente et subtre av WWW sider, men de har ingen mekanismer for å oppdatere bare sider som er endret. Et annet problem som er unikt i WWW sammenheng, er at lenker i de kopierte dokumentene må endres. Dersom de referer til lenker som er kopiert, må de endres til å peke på kopien i stedet for orginalen. Dersom relative lenker peker til sider som ikke er kopiert må også disse endres, dvs. de må ekspanderes til en absolutt URL. Behovet for å speile WWW sider vil kanskje forsvinne etterhvert som gode såkalte "caching" servere blir utviklet. De må kunne tilby selektiv oppdatering av dokumenter og kunne garantere at et dokument som ligger i cachen er oppdatert.
- **Ressursoppdagelse**
  Den mest spennende bruken av roboter (sett fra vanlige WWW brukeres synsvinkel) er uten tvil ressursoppdaging. Når informasjonsmengdene blir så store at mennekser ikke lenger kan ha oversikten, er det bedre å la en robot gjøre jobben. Det finnes flere roboter i dag som har indeksert store deler av WWW (90 - 95%) og som tilbyr søkemuligheter i denne databasen.
  På denne måten kan en vanlig WWW bruker kombinere manuell leting med søking i en database, når han er på jakt etter informasjon. Selv om man ikke alltid finner akkurat det man leter etter i en slik database, finner man som oftest relaterte sider som kan peke videre til ønsket informasjon.

En annen fordel med slike databaser er at de kan oppdateres automatisk med jevne mellomrom, slik at “døde” lenker kan fjernes og nye kan komme til. På denne måten vil databasen til enhver tid være (nogenlunde) konsistent, men vil samtidig vokse i et stort tempo og kreve svært stor plass (xx Gb).

- **Kombinert bruk**
  En robot kan selvfølgelig utføre mer enn en av de oppgavene som er nevnt ovenfor. Et eksempel på dette er RBSE Spider (ref. http://rbse.jsc.nasa.gov/eichmann/urlsearch.html) som foretar statistisk analyse av dokumenter den henter og tilbyr en søkbar database. Foreløpig er det ikke så mange roboter som gjør annet enn ressursoppdagelse, men dette vil nok endre seg etterhvert.

10.2 System komponenter

I dette avsnittet skal vi beskrive de forskjellige modulene WWW-roboten vil bestå av. I tillegg kommer en beskrivelse av alle datafiler/loggfiler som blir laget under kjøring og programmer som brukes til preprosessering av dataene.

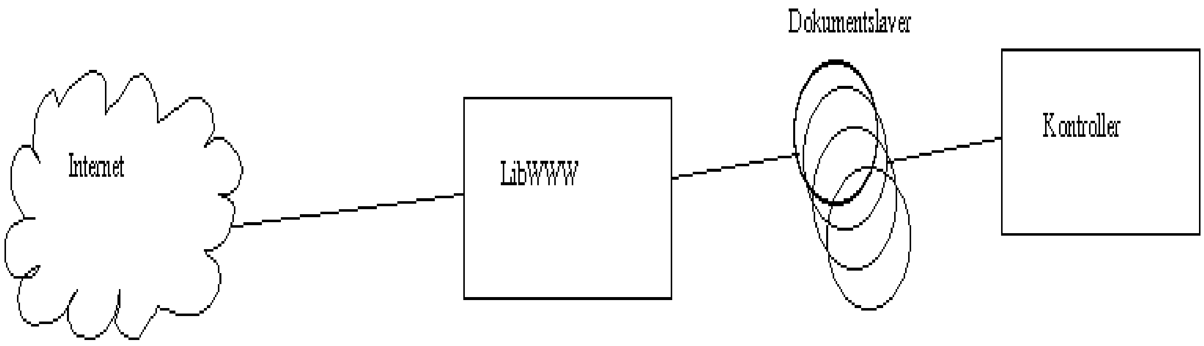
10.2.1 Kontroller

Kontrolleren vil bruke opplysningene som ligger i instruksjonsfilen (tolket av konfigurasjonsmodulet) , til å bestemme hvilke URL’er som skal hentes ned. Den skal også holde orden på dokumentslavene (Seksjon 10.2.2 på side 77), dvs. den skal vite status for hver enkelt dokumentslave til enhver tid og fordele oppgaver på dem.
I tillegg vil kontrolleren være ansvarlig for å lagre dokumentene som hentes inn på riktig sted i filsystemet.

10.2.2 Dokumentslave

En dokumentslave har en oppgave, nemlig å hente ned et angitt dokument fra WWW. Man vil da kunne ha et ubestemt antall dokumentslaver som henter ned dokumenter fra flere forskjellige servere samtidig.
Det vil kun være aktuelt å ha dokumentslave per server av ytelsesmessige grunner. HTTP protokollen er (i sin nåværende form) svært ineffektiv, og undersøkelser (ref) har vist at mange samtidige forespørsler til samme servere (ala Netscape multi-threading), belaster serveren unødvendig mye. Figur 16 på side 78 viser en oversikt over hovedkomponentene i systemet.

FIGURE 16. WWW-robotens komponenter



10.2.3 Konfigurasjonsmodul

Konfigurasjonsmodulet (OURConfig.pl) innholder alle konfigurerbare parametre for WWW-roboten. Defaultverdier er satt i dette modulet og i tillegg blir det implementert to andre metoder for å konfigurere roboten der. Den ene metoden vil tolke instruksjonsfilen og lagre oppgavene (med parametre) i en liste. Den andre metoden tolker kommandolinjen og setter interne variable på grunnlag av den.

10.2.4 CERN LibWWW for Perl

Følgende er hentet fra <URL http://www.ics.uci.edu/pub/websoft/libwww-perl/> og som er LibWWW's hjemmeside.

"libwww-perl is a library of Perl4 packages which provides a simple and consistent programming interface to the World-Wide Web. This library is being developed as a collaborative effort to assist the further development of useful WWW clients and tools. The library is based on version 4.036 of the Perl Programming Language.

libwww-perl is freely available as described in the Artistic License which accompanies the standard distribution. The libwww-perl software architecture and standard distribution package are copyrighted by the University of California for the sole purpose of retaining consistency and coherence in the distribution of the library. Contributions to the library are strongly encouraged and will be included in the standard distribution with full citation to the developers.

Support for the initial development and distribution of libwww-perl has been provided by the Arcadia Project at UCI, part of the larger Arcadia Consortium for research in software engineering environments."

10.2.5 Rapporteringsmodul

Dette modulet skal generere og sende en rapport til den som er ansvarlig for kjøring av roboten, når roboten har fullført (eller avbrutt) en kjøring. Rapporten skal inneholde en oppsummering for hver oppgave i instruksjonsfilen. Aktuelle data for hver oppgave kan være :

- Status (fullført/avbrutt)
- Suksessrate (antall vellykkede forespørsler i forhold til totalt antall forespørsler)
- Antall Kb/Mb som er hentet ned
- Gjennomsnittlig forsinkelse på forespørsler
- Gjennomsnittlig dokumentstørrelse

Mer data kan legges til etterhvert som vi finner det nødvendig.

10.2.6 Eget bibliotek

Under utviklingen av WWW-roboten, kommer vi til å samle en del generelle funksjoner i et eget bibliotek. Dette vil være funksjoner som blir brukt av flere programmoduler, som f.eks. en funksjon som koverterer IP-nummer til maskinnavn.

10.2.7 Andre hjelpeprogrammer

- Statistikk pakke
  Dette programmet skal generere statistisk informasjon om databasen vår. Aktuell informasjon er :
  Antall dokumenter per server
  Gjennomsnittlig dokumentstørrelse for hver server
  Største og minste dokument for hver server
  Total dokumentmasse for hver server

  Gjennomsnittlig dokumentstørrelse for hele databasen
  Antall servere i databasen
  Antall dokumenter i databasen
  Gjennomsnittlig antall dokumenter per server
  Største og minste dokument i databasen
  Serverene med færrest og flest dokumenter
  Serverene med størst og minst dokumentmasse
- URLsorteringsprogram
  Dette programmet skal ta en liste over alle filer som ligge ri databasen og sortere den etter domenet til maskinen som dokumentene befinner seg på. Dette gjør det mulig å laste opp dataene på MS160-kortet sortert på domene. Vi kan da begrense hvor på kortet det skal søkes hvis brukeren ønsker å lete i dokumentene under et spesielt domene.

- Program som lager fil som skal lastes på MS160 kortet og grunnlag for nødvendige DBM-filer (Data Base Management).
  Programmet må slå sammen alle HTML/text-filene i databasen til en stor tekstfil. HTML-kode fjernes for å begrense størrelsen på fila og siden det ikke er spesielt interresant å søke etter. Et unntak fra dette er lenker (HREF), som også lastes opp på MS160-kortet. En fil som inneholder alle dokumenttitler blir også generert.
  Programmet generer også grunnlaget for DBM filer som skal brukes av selve søkeprogrammet. Dette omfatter en fil som inneholder alle URLer og en fil som sier hvor hver enkelt URL ligger i den store datafila som lastes inn på MS160-kortet.
- Program som lager fil med informasjon om hvor de forskjellige domenene befinner seg på MS160-kortet
  Programmet går igjennom alle URLer som er samlet inn og trekker ut alle unike maskinnavn og subdomener. Så finner det ut hvor i datafilen (på MS160-kortet) dokumentene under hvert domene ligger samlet. Denne informasjonen lagres til en fil som er grunnlaget for en DBM fil som brukes av programmet som søker på MS160-kortet.

10.3 Datafiler

En lite beskrivelse av samtlige datafiler følger.

10.3.1 Logg-fil

Denne filen inneholder alle meldinger som roboten generer under kjøring. Dette omfatter både feilmeldinger og generell informasjon om robotens bevegelser (og avgjørelser som tas underveis.

10.3.2 URL-fil

Denne filen innholder en liste over alle URL'er som roboten har gjort et vellykket forsøk på å hente ned.

10.3.3 Resterende-fil

Denne filen innholder alle URL'er som roboten ikke fikk besøkt under siste kjøring. Dersom roboten fullfører oppgavene sine uavbrutt, vil denne filen være tom. Hvis roboten blir avbrutt av bruker under kjøring, vil den skrive alle URLer som ligger i kø (dvs. de skal hentes) til denne filen.

10.3.4 Robot-fil

Filen skal inneholde en liste over alle servere som er sjekket i henhold til Robot Exclusion Standard [appendix B]. Det vil være mulighet for å angi hvor ofte disse skal sjekkes (i tilfelle endringer).

10.3.5 Unngå-fil

Denne filen skal inneholde en liste over URLer som roboten vår skal unngå når den traverserer WWW. Dette kan f.eks. være servere som ikke ønsker besøk av roboter, men vi vil også prøve å unngå genererte sider (produsert av et program/skript). Grunnen til dette er at programmene som genererer slike sider ofte trenger innparametre som må velges av en bruker. Uten innparametre returneres som oftest bare en side med feilmeldinger som er lite nyttig for roboten. Lenker som referer til søkbare indekser (ISINDEX eller FORMS) skal heller ikke følges av samme grunn som ovenfor.

11.0 Ikke-funksjonelle krav

Her beskriver vi krav til WWW-roboten som hverken er funksjonelle eller konkrete, men allikevel viktige å huske på når roboten implementeres.

11.1 Hardware krav

For at WWW-roboten skal kunne kjøre tilfredstillende, er den avhengig av at visse hardwarekrav er oppfylt. Nedenfor følger en prioritert liste over hva som kreves.

- Store mengder diskplass til WWW-sider som hentes ned. Nøyaktig hvor mye plass som trengs er vanskelig å si. Det bestemmes av hvor store deler av WWW som skal lagres. I utgangspunktet har vi sagt at vi skal holde oss innenfor Norge og vi regner med at vi da vil trenge ca. 1 gigabyte. Plassbehovet vil stadig øke, siden WWW vokser jevnt og trutt.
- Den største flaskehalsen i programmet er all nettaksessen. For å gjøre denne flaskehalsen minst mulig, bør man ha en hurtig nettforbindelse (T1/T3)
- Under kjøring vil WWW-roboten kunne bruke en god del minne. Hovedårsaken til dette er alle nye URL’er som oppdages lagres i en kø som kan bli veldig stor. Dersom dette skulle vise seg å bli et stort problem, vil vi vurdere å lagre køen på disk, for så å hente inn URL’ene etterhvert som det er behov for dem.
- En maskin med “normal” prosessorkraft.

11.2 Software krav

WWW-roboten skal i sin helhet skrives i Perl og man trenger da Perl 4.036 eller nyere for å kjøre den.
I tillegg vil vi benytte LibWWW som er et bibliotek med bl.a. HTTP relaterte funksjoner. Vi velger å benytte LibWWW versjon 0.40 for perl. Se appendix A for Artistic License som beskriver bruksregler for LibWWW.

11.3 Andre krav

Siden hver kjøring av WWW-roboten er svært tidkrevende, ønsker vi selvfølgelig at den skal være så pålitelig som mulig. Dette innebærer at roboten må kunne takle de fleste feilsituasjoner som kan oppstå på egen hånd. Den må f.eks. ikke påvirkes av at WWW-servere kræsjer eller feilmeldinger fra WWW-servere. For å få programmet så feilfritt som mulig, er vi nødt til å teste det grundig. Slike tester foretas på lokalt plan, slik at vi ikke skal overbelaste WWW-servere og nett mer enn nødvendig.
Det er også ønskelig at WWW-roboten skal være så ressursvennlig som mulig. For at vi ikke skal overbelaste WWW-servere med for mange forespørsler i løpet av en kort periode (såkalt “rapid firing”), vil roboten ta en pause ved mellom etterfølgende forespørsler til en og samme server.
En av årsakene til at vi valgte perl som implementeringsspråk, er at det er svært portabelt. Perl finnes for så godt som alle UNIX operativsystemer som er i bruk i dag. I tillegg finnes det også for Windows/DOS/MacOS/AmigaOS. Roboten vil da kunne kjøre på en stor mengde forskjellige plattformer.
Vi ønsker også at roboten skal være enkel å kjøre. Konfigureringen bør derfor være så enkel som mulig, uten at det går utover mulighetene.
Når det gjelder ytelse, så har vi ikke satt noen strenge krav. Det vil uansett ta lang tid å samle inn dokumenter pga. nettytelse og krav om at vi ikke skal overbelaste servere.

12.0 Funksjonelle krav

En foreløpig liste over spesifikke krav til www-roboten :

12.1 Idenifiserbarhet

- **Identifisering av roboten**
  HTTP har et *User-Agent* felt der man kan identifisere en WWW-browser. Siden roboten er en slags browser, bruker vi dette feltet til å angi navn på roboten (f.eks. OURobot 0.1 - libwww-perl 0.40). På denne måten kan de som er ansvarlig for en server skille roboter fra vanlige brukere. Roboten bør også kjøres fra en maskin som er registrert i DNS (Domain Name Server), slik at det er lettere å identifisere den. Dette vil også indikere hvor roboten kjøres fra.
- **Identifisering av ansvarlig person**
  HTTP har et *From* felt som kan brukes til å identifisere personen som bruker en WWW-browser. I dette feltet skal vi angi e-mail adressen til den som er ansvarlig for roboten under kjøring. Dette gjør at andre (spesielt de som er ansvarlig for andre servere) kan kontakte riktig person dersom problemer oppstår (erfaringsmessig får man også en hyggeligere start på en evt. dialog dersom man er lett å spore opp).
- **Ekstra informasjon**
  De som er ansvarlige for en server, kan ønske å vite hvorfor deres server blir truffet av en robot. Dette kan vi på en enkel måte fortelle dem ved å bruke HTTP feltet *Referer*. Dette feltet indikerer hvilket dokument som refererte til nåværende dokuement.
- **Informer folk som berøres**
  Det bør postes en beskjed på USENET gruppen *comp.infosystems.www.providers* før en større kjøring av roboten startes. Denne leses av mange server-ansvarlige rundt om i verden. Det finnes en liste over aktive roboter på *http://web.nexor.co.uk/mak/doc/robots/active.html* som oppdateres regelmessig. Dersom noen lurer på aksesser fra en spesiell maskin, kan man sjekke denne.
  Dersom man velger å traversere *en* spesiell server, bør man informere de som er ansvarlige for denne.
- **Vær tilstede**
  Roboten bør ikke kjøre uten tilsyn. Det kan hende at nettet ikke takler den ekstra belastningen eller at roboten gjør skade på andre måter (eller at den irriterer mange brukere av en server). Siden det bare er den som kjører roboten som kan stanse den (i tillegg til sys. adm.), bør man være logget på den maskinen roboten kjøres fra, slik at man er tilgjengelige via f.eks. *talk.*

12.2 Ressursvennlighet

- **Moderat tempo**
  Roboten bør kjøres i et moderat tempo. I praksis kan roboten takle hundrevis av dokumenter i minuttet, men dette belaster servere unødvendig mye. Vi kan enten ta en pause mellom hver forespørsel eller rotere forespørslene mellom flere servere slik at vi aldri henter to dokumenter etter hverandre fra samme server.
  Her velger vi følgende løsning :

7. Vi kan hente n dokumenter fra samme server med m sekunders mellomrom.
8. Etter at n dokumenter er hentet, skal vi ta en lengre pause (f.eks. 4*m) før vi henter flere dokumenter fra serveren.

- **Bredde først søk**
  Vi skal bruke bredde-først søk når vi traverserer WWW. På denne måten forsikrer vi oss om at hver eneste server (med nyttig innhold) vi besøker, har flere sider representert i søkemassen. Denne dekningen er viktig for brukere siden det er lettere å navigere innenfor en server enn mellom servere. Hvis vårt søkeverktøy kan identifisere servere som har nyttig for en bruker, så kan brukeren stort sett navigere innefor serveren på egenhånd for å finne det han ønsker.
  En annen positiv effekt av bredde-først søk, er at vi får delt belastningen mellom flere servere.
- **Angi de filtyper som ønskes**
  HTTP har et *Accept* felt som kan brukes til å spesifisere hvilke datatyper vi kan håndtere. Dersom vi bare ønsker tekstdokumenter, kan vi spesifisere *text/html* i dette feltet. Ikke alle servere støtter/ bruker dette feltet, men dersom de gjør det, sender de bare dokumenter av angitt type. Vi slipper da å forkaste filer som vi ikke kan håndtere.
- **Hent *kun* de filtyper som ønskes**
  Vi bør også benytte oss av filendelser i den grad det er mulig. Lenker som refererer til f.eks. *".ps", ".zip", ".Z", ".gif"* osv., hentes ikke dersom man bare ønsker tekst (som vi gjør). Dette er ikke den moderne måten å gjøre dette på (se forrige punket, *Accept*), men det er ennå mange servere som ikke støtter HTTP *Accept* feltet.
  Siden vi ikke ønsker å hente den samme siden flere ganger, følger vi ikke relative lenker (f.eks. *HREF=#seksjon_X*).
- **Protokoller**
  Vi skal begrense hvilke protokoller som støttes. I prinsippet ønsker

vi bare å følge lenker av typen *http*, men vi vil åpne for flere protokoller i instruksjonsfilformatet.

- **Sjekk URLer**
URLer skal sjekkes før de følges. F.eks. kan det være viktig å sjekke at vi ikke mister en *"/"* når vi parser og setter sammen igjen en URL.
- **Sjekk resultater (HTTP responskode)**
Vi skal sjekke responskodene som kommer tilbake etter en HTTP forespørsel. Dersom det er en feilmelding bør dette indikeres i rapporten som sendes til robotoperatøren ved endt kjøring (angi sted hvor feilmeldingen ligger lagret).
- **Unngå repetisjon**
Før et dokument hentes ned, skal vi sjekke om vi har hentet det ned tidligere. Dette gjøres ved å sammenligne URLer.
Enda viktigere er det å sjekke maskinadresser. F.eks. er *www.idt.unit.no* den samme maskinen som *peon.idt.unit.no* og *129.241.111.13*. Vi vil bruke IP-nummer for å utføre denne sjekken (da dette *skal* være unikt).
- **Aksesser servere når belastningen er minst**
Servere bør aksesseres når belastningen er minst. Når dette er kan man finne ut ved å kontakte serveransvarlig.
- **Ikke kjør for ofte**
Roboten må ikke kjøres for ofte. Hva dette innebærer blir en vurderingssak, men 4 ganger i året kan synes passe. I tillegg bør man prøve å dra nytte av data man har samlet inn tidligere. Man bør bl.a. vite hvilke steder man bør unngå og sørge for at roboten gjør det. I tillegg bør man prøve å starte på et sted hvor man forventer å finne mange nye lenker, slik at man finner mange nye lenker i begynnelsen av søket. Hvis man da må avbryte roboten av en eller annen grunn, har man allikevel en gevinst ved kjøringen.
- **Ikke prøv alle lenker**
En del dokumenter er søkbare (ISINDEX, ref. HTML standard) eller innholder FORMS (ref. HTML standard). Slike lenker bør ikke følges videre.
I tillegg bør man bygge inn logikk slik at man unngår andre former for "programgenererte" sider. Man bør da f.eks. unngå URLer som har *"cgibin"*, *"cgi-bin"*, *".pl"* ,*"?"*og *"bin"* i spesielle deler av seg (gjelder både store og små bokstaver, men man må vise litt forsiktighet slik at man ikke mister URLer som egentlig er helt ufarlige å følge. Eksempelet ovenfor nevner forøvrig bare et utvalg av tilfeller der man bør være forsiktig med å følge en URL.
- **Logg**
Tilstrekkelig logging må til. I tillegg bør man også føre litt statistikk (siden man allikevel har dataene).

- **Robot Exclusion Standard**
  Robot Exclusion Standard skal støttes (se appendix B).
- **Lagringsstruktur**
  Vi velger å lagre alle dokumenter som hentes ned under følgende struktur :

1. For hver ny server skal det opprettes en katalog med serverens IP-nummer som navn. Portnummer kan komme i tillegg siden det kan finnes flere servere på en og samme maskin.
2. Alle dokumenter som hentes fra serveren skal legges under denne katalogen på samme måte som de ligger på serveren.

- **Domenebegrensning**
  Roboten kan begrenses til et eller flere angitte domener (Seksjon 13.0 på side 86). Dette kan være nyttig både under testing og senere dersom man ønsker å holde seg innen et visst geografisk område (f.eks. Norge). Det kan også være nyttig mht. begrenset lagringsplass.

13.0 Konfigurasjon av WWW-roboten

WWW-roboten har en mengde kofigurasjonsopsjoner som kan settes av den som installerer systemet eller av brukeren. Slik kan man kontrollere traverseringsprosessen og tilbakemeldinger fra roboten. Disse opsjonene kan settes på tre steder (listet opp i presedens rekkefølge (dvs. metode 1 har fortrinnsrett over metode 2) :

3. Noen opsjoner kan skrives inn på kommandolinjen.
4. Instruksjoner for traverseringen kan legges inn i en konfigurasjonsfil, eventuelt sammen med verdier som kan erstatte deafultverdiene i 3.
5. Defaultverdier for mange av de konfigurerbare parametrene er lagt inn i OURConfig.pl

13.1 Kommandolinjeparametre

Kommandolinjeparametre slik vi tenker oss de foreløpig er oppsummert i tabell 1.

TABLE 3. Kommandolinjeparametre

Opsjoner	Forklaring	Defaultverdi
-h	Hjelp. Vis en oversikt over kommandolinjeparametre.	
-e errorfil	Skriv feilmeldinger (i append modus) til den oppgitte fil. Anbefales brukt dersom roboten skal kjøres i en lengre periode. Feilmeldinger blir skrevet direkte til fila (uten buffer) , slik at man kan overvåke kjøringen av roboten via fila. Dersom man ikke oppgir noen verdi her, blir feilmeldinger skrevet til STDERR.	STDERR
-o utfil	Skriv robotens logg og resultater til oppgit fil. Dersom filen eksisterer vil den bli flyttet til utfil.bak før en ny fil opprettes. Anbefales brukt dersom roboten skal kjøres i en lengre periode. Logglmeldinger blir skrevet direkte til fila (uten buffer) , slik at man kan overvåke kjøringen av roboten via fila. Dersom man ikke oppgir noen verdi her, blir loggmeldinger skrevet til STDOUT.	SDTOUT
-i instruksjonsfil	Leser instruksjoner fra oppgitt fil. Instruksjonsfila forteller hvilke opsjoner som skal settes og hvilke oppgaver som skal utføres under kjøring.	
-d	Maksimum traverseringsdybde.	20
-a unngåfil	Leser og skriver URL som skal unngås til oppgitt fil. Hvis filen allerede eksisterer blir den flyttet til unngåfil.bak for ny fil opprettes	
-s serverfil	Leser og skriver navn på servere som er sjekket for /robots.txt fil til oppgitt fil. Hvis filen allerede eksisterer blir den flyttet til serverfil.bak for ny fil opprettes	
-b basefil	Leser og skriver navn på servere som er besøkt av roboten. Hvis filen allerede eksisterer blir den flyttet til basefill.bak for ny fil opprettes	

13.2 Konfigurasjonsfilformat

I konfigurasjonfilen ligger den informasjonen som WWW-roboten må ha før den kan starte en traversering. Nedenfor følger et eksempel på hvordan en konfigurasjonsfil kan se ut.

```
#Konfigurasjonsfil template
#Task1
{
 Name UNIT
 FileRoot /home/vier/b/bkh/diplom/crawler/data/
 StartURL http://www.unit.no/
 MaxDepth 20
 Timeout 30
 Excludefile avoid.txt
 Niceness 0
 Schemes http
 OperatorEmail bkh@idt.unit.no
 Domains unit.no
 Filetypes html
}

#Task2
{
 Name UiO
 FileRoot /home/vier/b/bkh/diplom/crawler/data
 StartURL http://www.uio.no
 MaxDepth 20
 Timeout 30
 Excludefile avoid.txt
 Niceness 0
 Schemes http,ftp
 OperatorEmail bkh@idt.unit.no
 Domains uio.no,uib.no
 Filetypes html,ps
}
```

Den foretrukne måten å gi roboten instruksjoner på, er via konfigurasjonsfilen. Ved oppstart vil roboten lese konfigurasjonsfilen og sette verdien av interne variabler ut i fra disse. Navenet på konfigurasjonsfilen gis av kommandolinjeopsjon -i eller en defaultverdi.

En konfigurasjonfil består av flere oppgaver/task. Oppgavene blir utført i den rekkefølgen de er listet opp i filen. Ved påbegynnelse av en ny oppgave, vil alle konfigurasjonsparamtre bli satt i henhold til det som er oppgitt. Dersom noen verdier ikke er oppgitt, vil defaultverdier bli brukt istedet for disse.

Format på konfigurasjonsfilen er ganske strengt. Tomme linjer og linjer som begynner med #, blir oversett (linjer som begynner med # er kommentarer). Alle instruksjoner skal skrives på en linje (dvs. en instruksjon pr .linje) uansett lengde på instriksjonen. En oppgave skal begynne med en { på en linje for seg selv og avsluttes med en } (på en linje for seg selv).

En forklaring av de forskjellige parametrene følger nedenfor.

13.2.1 Name

Navn på oppgaven. Brukes stort sett bare til logging og rapportering. Alle ASC-II tegn kan brukes.

13.2.2 FileRoot

Her skal man oppgi fullstendig sti til en katalog på lokal disk, der man ønsker at roboten skal legge filene som blir hentet ned. Loggfiler og rapportfiler blir også lagt her.

13.2.3 StartURL

Her skal man oppgi en URL som er utgangpunktet for denne oppgaven. Det kan være hensiktsmessig å velge f.eks. rotURl'en til en server dersom man ønsker å få med seg flest mulig dokumenter (men stort sett ikke nødvendig).

13.2.4 MaxDepth

Her skal man oppgi maksimum traverseringsdybde for roboten. Vi tenker oss at hvert dokument på WWW er en node og alle lenkene er kanter i en graf. MaxDepth angir hvor dypt i grafen (ut i fra rotdokumentet) man skal gå.

13.2.5 Timeout

Timout angir hvor lenge roboten skal vente på å få et dokument fra serveren (dvs. tid i sekunder fra HTTP GET blir sendt til dokumentet kommer tilbake fra serveren). Defaultverdi er 30 sekunder.

13.2.6 Excludefile

Angir fullstendig sti til filen som inneholder liste over de serverne, som vha. Robot Exclusion Standard, har gitt uttrykk for at roboter ikke er ønsket. Denne filen blir oppdatert av roboten etterhvert som den kommer over flere servere som skal ekskluderes.

13.2.7 Niceness

Her kan man angi en verdi som vil påvirke robotens oppførsel i stor grad. Man oppgir et tall mellom 0 og n som skal indikere hvor "nett-etisk" man ønsker at roboten skal oppføre seg (se seksjon 4 for en nærmere spesifisering av “nett-etikk”). Nedenfor følger en foreløpig liste over hvilke kjøringsmodi som er aktuelle.

TABLE 4.

Niceness verdi	Beskrivelse
0	Normal, snilt modus
1	
2	
3	Rapid fire, ressursgris

13.2.8 Schemes

Her skal man angi de protokoller som roboten skal ta hensyn til under kjøring. Man kan angi f.eks. http, ftp, gopher, wais osv. Protokollene skal skilles med et komma. Defaultverdi er http.

13.2.9 OperatorEmail

E-mail addressen til den som er ansvarlig for roboten under kjøring. Når kjøringen er ferdig, sendes en rapport til denne adressen.

13.2.10 Domains

Her kan man angi et filter (filtre) som skal spesifisere hvordan man vil begrense traverseringen. Dersom man oppgir fullt hostnavn til en server, vil roboten holde seg til denne serveren under kjøring. Man kan også spesifisere ufullstendige hostnavn, f.eks. unit.no. Roboten vil da kun sende forespørsler til servere i domenet unit.no.
Dersom ingen verdi er oppgitt, vil traverseringen bli ubegrenset.

13.2.11 Filetypes

Her skal man angi de filtypene som roboten skal hente ned. Roboten vil bruke HTTP-feltet “Content-Type” for å sjekke hva som blir returnert fra serveren. Filendelser blir brukt som en første test på filtype (f.eks vil .html indikere text/html dokument). Man skal angi filtypene på standard MIME format (se appendix C). Et eksempel kan være “text/html”, som indikerer at roboten bare skal hente HTML filer. Dette er også default. Dersom man vil oppgi flere typer, skal disse skilles med et komma.

13.3 Defaultverdier

Defaultverdier ligger i modulet OURConfig.pl. Verdiene i OURConfig.pl er enten verdier som vi/andre har ment er fornuftige, eller verdier som bør settes for å tilpasse roboten til spesielle kjøringsmiljøer. Et eksempel på det første er maksimum traverseringdybde, som er satt til 20. Et eksempel på det andre er semdmail kommando, som kan variere fra system til system. Denne brukes når roboten skal sende elektronisk post til operatøren. Defaultverdien for denne er "/usr/lib/sendmail".

Andre defaultverdier som ligger i OURConfig.pl er :

- Maksimum traverseringsdybde
- Timeout verdi (maksimum antall sekunder fra forespørsel til respons).
- Maksimum antall etterfølgende forespørsler til en server før man bør ta en (lang) pause.
- Antall sekunder man venter ved en lang pause.
- Antall sekunder mellom etterfølgende forespørsler til en server.
- Navn på fil som roboten skal se etter når den sjekker servere for Robot Exclusion Standard. Denne bør ikke endres med mindre Robot Exclusion Standard endres. Se forøvrig appendix B for en beskrivelse av Robot Exclusion Standard.

14.0 Implementasjonskrav

Her beskriver vi hva som trengs av hardware og software for å implementere WWW-roboten, og også hvordan vi vil prøve å gjøre systemet mulig å vedlikeholde og videreutvikle.

14.1 Hardware

En vilkårlig UNIX maskin.

14.2 Software

Perl4.036 , LibWWW 0.40, RCS, editor.

14.3 Enkelt vedlikehold

Vi ønsker å implementere et godt strukturert system som skal være enkelt å vedlikeholde. For å oppnå dette vil vi :

- Skrive kildekode som er enkel å lese og forstå
- Forsøke å oppnå stor grad av uavhengighet mellom software komponenter
- Lage en god software struktur, med en logisk komponentoppbygning.

14.3.1 Lesbar kode

Vi skal følge et par retningslinjer for å skrive lesbar kildekode. Det er vikrig å gi variabler og funksjoner meningsfylte navn. I tillegg skal vi skrive gode kommentarer der det er nødvendig (triviell kode bør i stor grad ikke kommenteres). Vi vil også følge en mal for hvordan funksjoner og moduler skal se ut.

14.3.2 Uavhengighet mellom software moduler

Vi ønsker å ha en stor grad av modularitet i systemet, for å forenkle endringer og utvidelser. Vi kan komme til å benytte noen globale tilstandsvariabler (av hensyn til konfigureringen), men vil prøve å begrense dette i størst mulig grad.

14.3.3 Systemstrukturen

For å lage en god systemstruktur skal vi prøve å følge følgende retningslinjer :

- Vi ønsker en løs kopling og høy kohesjon i koden. For å oppnå dette vil prøve å bruke objektorienterte prinsipper, spesielt innkapsling/informasjonsskjulig. Vi ønsker ikke at data som brukes i et modul (perl : package) skal være mulig å endre fra et annet modul (mest mulige lokale variabler).

- Vi prøver å minimalisere “fan-out” og ha høy “fan-in” fordi vi ønsker at koden skal være mest mulig gjenbrukbar.
- Prøve å holde “scope-of-effect” innfor “scope-of-control” for alle subprogrammer/funksjoner. Dette betyr at alle tilstandsvariabler i subprogram A, bare påvirker andre subprogram som er direkte underlagt A (dvs. alle subprogrammer som ligger under A i kallkjeden).

Kravspesifikasjon for MS160 søkemaskin

Ole Andreas Grytdal
Bjørn Kristian Hilstad

I dette kapittelet beskriver vi kravene til vår søkemaskin

15.0 Generell system beskrivelse

Denne generelle systembeskrivelsen gir en oversikt over de delene som søkemaskinen består av.

Et av hovedmålene med oppgaven var å lage noe som fungerer og kan brukes i fremtiden.

Applikasjonen som er konstruert har ikke alle de finessene vi ønsket oss, men det er allikevel nok til å dekke 95 % av de søketilfellene som brukerne benytter.[13]

15.1 System kontekst

World Wide Web er ufattelig stort. For å kunne søke i WWW, må man ha en eller annen form for indeks. For å lage dette ble det lagd roboter for å innhente informasjon, og søkemaskiner for å kunne søke i datamengden. Så godt som alle søkemaskiner bruker idag en indeks for å kunne slå opp data. Et MS160-basert system vil ikke gjøre dette. Hvordan de andre søkemaskinene virker er nærmere beskrevet i Seksjon 5.0 på side 30. I OURSpider prøver vi også å få lagt inn nye funksjoner som dagens søkemaskiner ikke har / eller som går sakte i software, men som MS160 løser i hardware. Videre vil nok noen av funksjonene kun være beskrevet og ikke implementert, alt avhengig av tiden som er til rådighet. Endel av tiden har gått med til å forstå driveren som er laget for kortet og hvordan selve kortet opererer på data.

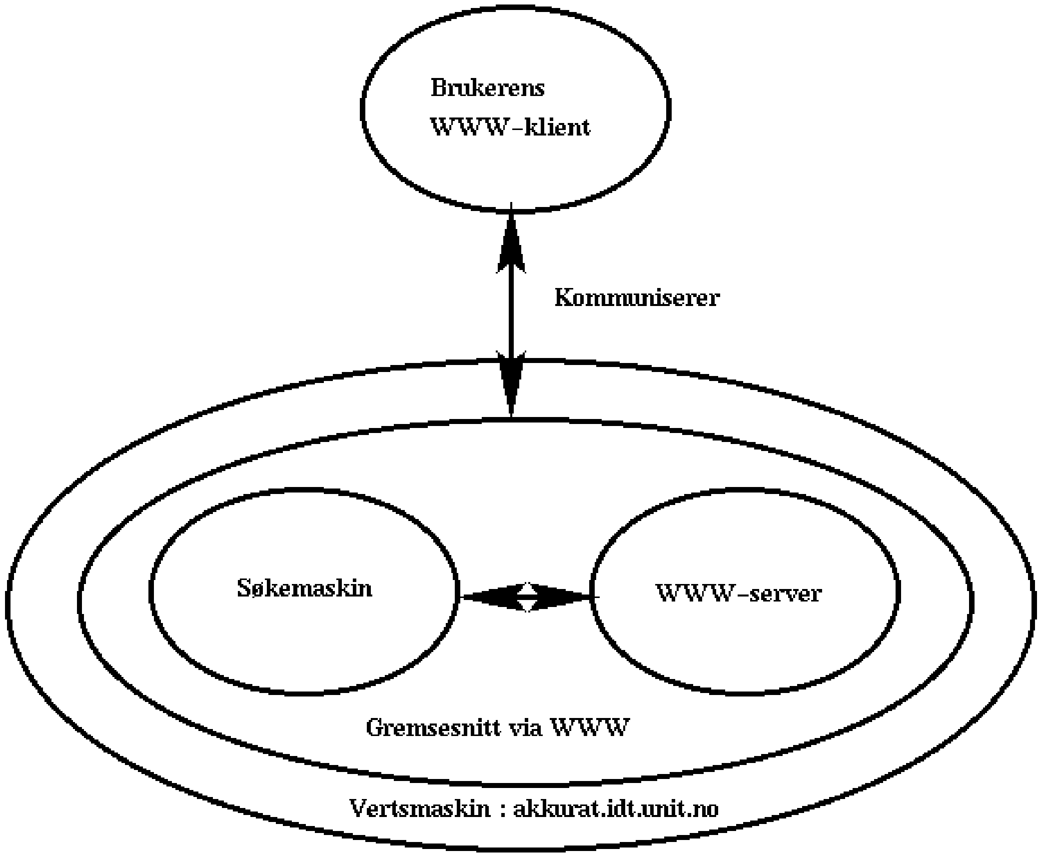
15.2 System komponenter

En oversikt over hoveddelene i samspillet mellom klient og server kan man se i Figur 17, “WWW-klient og server,” på side 97. Som man ser er det endel moduler som samspiller med hverandre for at søkemaskinen skal kunne fungere. De er :

- Brukerens www klient - Brukeren som skal søke i OURSpider må benytte seg av en www-klient for å oversende forespørselen til søkemaskinen
- Vertsmaskin - akkurat.idt.unit.no fungerer som vertsmaskin for forespørslene. Den kjører både www-server daemon og vår søkemaskin.

- Grensesnitt via www - Det brukes et grensesnitt via www mot søkemaskinen for å oversende søkekriteriene, og det samme grensesnittet sender svar tilbake til klienten som forespurte
- WWW-server : Denne tar imot forespørsler fra www-klienten og sender tilbake fornuftige data.
- Søkemaskin : Behandler søkedata og sender tilbake svar via www-serveren.

FIGURE 17. WWW-klient og server



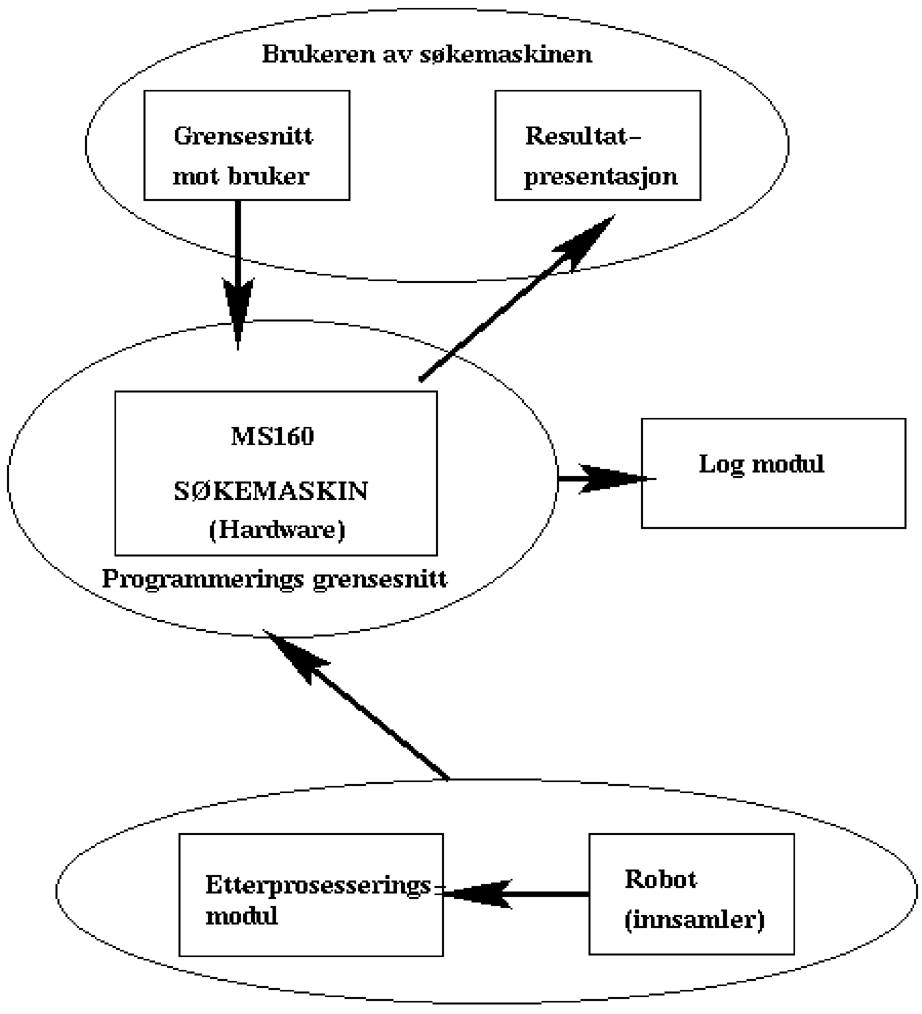
De deler som vår søkemaskin skal bestå av kan en se i Figur 18, “Oversikt over MS160 søkemaskin sine moduler,” på side 98.

Som en kan se av figuren, finnes følgende komponenter i vårt system:

- Søkemotoren - et hardware kort som kan søke gjennom 160 MB/s i fritekst.

- Søkemotorens programmerings grensesnitt - program som støtter opp mot søkemotorens hardware.
- Log modul - holder rede på antall aksesser og statistikk for søke-maskinen.
- Grensesnitt mot bruker - Utfyllings skjema der bruker skal fylle inn sine søkekriterie
- Resultat presentasjon - Presenterer søkeresultat til brukeren på en oversiktelig og grei måte.
- Etterprosesseringsmodul - Behandler data som er innsamlet fra roboten.
- Robot - Samler inn data fra forskjellige world wide web servere rundt omkring i Norge.

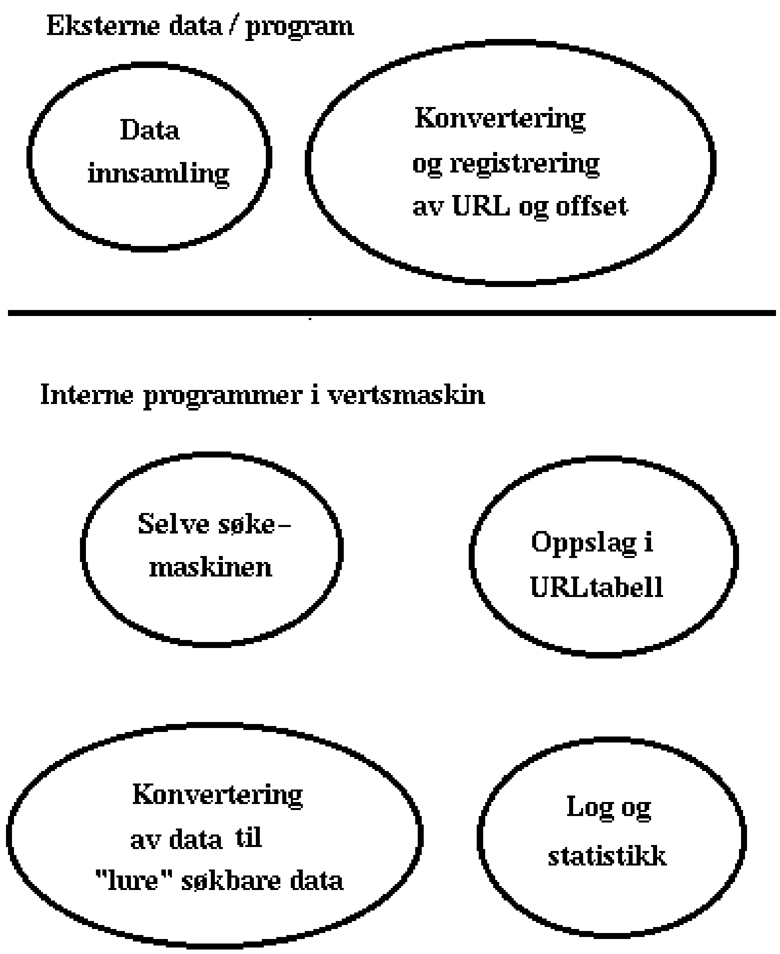
FIGURE 18. Oversikt over MS160 søkemaskin sine moduler



15.3 Søkemaskinens komponenter

I den applikasjonen som vi skulle lage for å søke gjennom tekst er det endel komponenter som er verdt å merke seg som man kan se i Figur 19, "Software moduler i søkemaskinen," på side 99.

FIGURE 19. Software moduler i søkemaskinen



Vi har skilt endel av modulene fra hverandre som eksterne og interne moduler, da de skiller seg klart fra hverandre. De har heller ikke noe med hverandre å gjøre under kjøring av selve søkeprogrammet.

Vi vil presentere en kort oversikt over funksjonaliteten til hver modul.

15.3.1 Ekstern program del

Den eksterne programdelen har med innhenting av data og postprosessering av disse dataene. Datainnsamlingen er allerede beskrevet før, så vi vil kort beskrive postprosesserings modulene. Vi vil senere beskrive funksjonaliteten og kravene til modulene.

- Postprosesseringsmodulen slår alle innsamlede data sammen til en fil. I tillegg lages det to databasefiler,en med hver URL som er lagret, en annen som har offset[1]'en til hver URL lagret.
- Selve søkemaskinen dytter data gjennom kortet og plukker ut de treff brukeren ønsker.
- Det må gjøres et oppslag i URL-tabell for å finne ut hvilken URL som hører til ordet man fant.
- For at man skal kunne utnytte mulighetene i MS160 best mulig bør en endre litt på måten man lagrer data på. Dette kommer av måten MS160 bruker sine øvre og nedre grense.
- Loggemodulen skal lage statistikk over antall søk, hvor fort det går og andre helt unyttige oversikter.

16.0 Ikke funksjonelle krav

I dette kapittelet vil vi beskrive krav til søkemaskinen som ikke er funksjonelle, men viktige å følge når søkemaskinen skal implementeres.

16.1 Hardware krav

For at OURSpider skal kunne fungere tilfredstillende må følgende hardware krav være oppfyllt:

- Minst et MS160 søkekort med mye RAM. Hvert kort kan ha 160 MB ram på kortet. Det bør være mye ram på kortet, da størrelsen på teksten som roboten henter inn kan bli ganske stor. Aller helst bør det være flere kort som man kan lagre tekst på, for å dekke hele Norge.
- En pc som kan ha et eller flere MS160 kort installert. Det bør minst være en 486/66 eller bedre og ha nok hardisk plass til operativsystem, innsamlede data, databasefiler og www-server. Selve søkingen vil ikke gå noe merkbart fortere med raskere pc, men oppslag til harddisk vil kunne gå raskere med en raskere kontroller/ raskere harddisk.

1. Offset sier noe om hvor langt ute i den store fila noe starter på.

- Nettet som søkemaskinen er koblet opp mot bør være forholdsvis raskt slik at det kan sende et hurtig svar tilbake til klienten når søket er over. Dette kan bli en flaskehals , hvis det blir mange oppslag og derfor mye å overføre.

16.2 Software krav

I tillegg til hardwarekravene, må visse krav på software siden også være oppfylt.

- Driveren som vi bruker for å kommunisere med kortet er skrevet for operativ systemet NetBSD. Driveren er skrevet av Tor Egge, vår veileder.
- Operativsystemet, NetBSD, kan kjøre på en vanlig pc og kan hentes på mange FTP baser rundt omkring.
- Det må kjøres en www-server på maskinen. Vi har valgt å bruke apache versjon 1.0.0.[14] WWW-serveren er lett konfigurerbar, er skrevet i C og er bare å kompilere opp.

16.3 Andre krav

Ved bruk av søkemaskinen må både www-server og vår søkemodul være stabil. WWW-serveren er nå kommet i versjon 1.0.0 etter lenge å ha være i beta / versjon 0.8.x. Vi regner derfor med at denne vil holde seg stabil og takle last på nettet og antall forespørsler på en god måte. Når det gjelder vår søkemodul, må den testes på endel forskjellige tilfeller for å få den til å fungere på de krav vi har satt under punktet om funksjonelle krav.

17.0 Funksjonelle krav

Nedenfor er en liste over de krav vi har satt til søkemaskinen og dens moduler.

17.1 Søkemaskinen

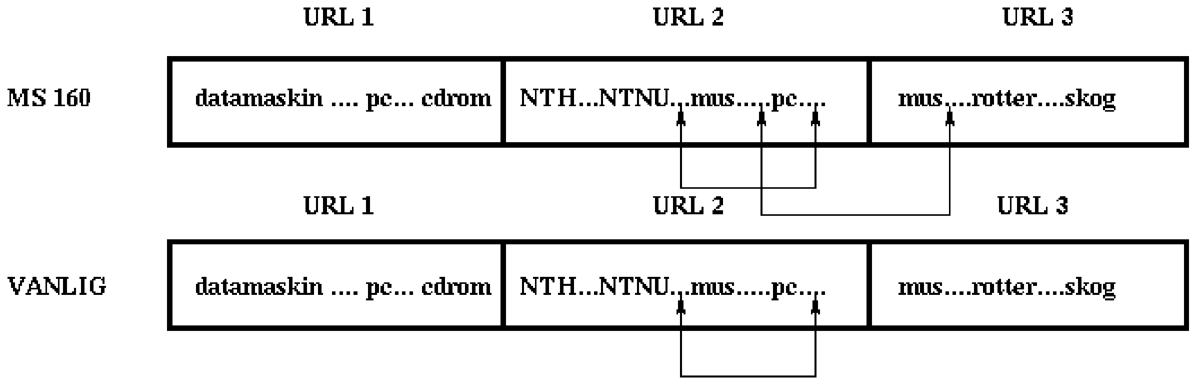
Til søkemaskinen har vi satt visse krav til hva den skal kunne klare å søke på. Kravene er som følger :

17.1.1 Fler-kriterie søk

Brukeren skal kunne søke på et eller flere kriterier i et og samme søk. MS160 kortet har muligheter for at dette kan gjøres enkelt via dens programvare på selve kortet. I den forbindelse vil det være ønskelig å kunne bruke flere kriterier sammen for å begrense søkeresultatene.

- **Søk innenfor en bestemt avstand**
  Når man søker på flere kriterier kan det være interesseant å søke innenfor en bestemt avstand, f.eks for å finne kriteriene innenfor et og samme dokument. Dvs. at hvis man finner alle ordene innefor den gitte avstanden, skal det genereres et treff. Dette er noe som MS160 kan utføre i sin programvare. Problemet med dette er at vi ikke kan være sikre på om dette virkelig **ER** innenfor samme dokument. Hvis dette virkelig skal sjekkes på, må dette etterprosesseres og vi kan **IKKE** bruke MS160 sin muligheter. MS160 genererer kun treff hvis alle kriterene er innenfor "Match Latency". Skal man være sikker på at de virkelig må ligge innenfor samme dokument, må en finne hvert ord for seg og sjekke om de ligger innenfor samme URL. Dette krever endel ekstra sjekk. Sannsynligheten for at fornuftige samsvarende ord ligger i samme dokument er nokså stor og at dette ikke lager falske treff, ved å kombinere to forskjellige URLer. For å illustrere dette bedre kan man se på Figur 20, "Match latency problem," på side 103. Der ser man at i MS160 sitt tilfelle, vil den gi to treff i URL 2 og 3, mens i det vanlige tilfellet vil gi kun et treff i URL 2.

FIGURE 20. Match latency problem



- **Boolske uttrykk**
  Ved søk med flere kriterier kan det være fornuftig med boolske uttrykk. Man kan bruke operatorene AND , OR og NOT. Helst bør man kunne kombinere operatorene slik :
  Finn : (pc AND mus) or datamaskin
  Dette vil gi treff på dokument med enten ordene pc og mus i seg eller dokument med ordet datamaskin. Det må da lages en tolker som kan gå igjennom søkesetningen og finne uttrykk og hvordan setnings oppbygningen skal være.
  Det må selvsagt være bruk av paranteses eller lignnende for å skille tvetydige uttrykk som :
  pc AND mus OR datamaskin , slik at man vet om dette betyr :
  (pc AND mus) OR datamaskin eller pc AND (mus OR datamaskin) noe som vil gi ganske forskjellige resultater.

  NOT operatoren brukes der en vil ha bort uaktuelle treff. Et eksempel på dette kan være : fotball NOT 1.divisjon , hvis en vil ha treff som omhandler fotball, men ikke noe med 1.divisjon å gjøre.

- **M av N treff**
  Istedet for å bruke boolske algebraer ved fler-kriterie søk, kan det være bedre å søke på minst M av N.
  Man oppgir da hvor mange kriterier man vil treffe på utfra de kriteriene en har skrevet inn. Et “m av n” søk kan også på en måte erstatte simpel boolsk algebra. Hvis en ikke har lagd en parser som kan ta parenteser og bare kan utføre A AND B eller A OR B, kan det være like greit å bruke “m av n” søk.
  A AND B = 2 av 2 søk og A OR B = 1 av 2 søk

17.1.2 Domene søk

Ved søk kan man begrense søk til innenfor et visst domene. Innenfor et domene betyr det samme som innenfor et visst navneområde i alle de navn som er knyttet opp til ip adressene. De data som er innsamlet er kun innenfor *.no domenet, men det kan være ønskelig å begrense søk til innen for f.eks *.unit.no. På den måten vil søkingen gå raskere og en slipper endel uønskede svar.

17.1.3 Wildcards på ord og eksakt søk

MS160 kortet vil automatisk kunne gi substring søk. Noe som kan være fordelaktig er å vite at ordet begynner med det en skriver, at det ikke skal være en bit midt i et ord eller at man vil ha en nøyaktig samsvarighet i det en søker på.

Eksempel : Hvis man søker etter ordet ole vil en at dette kun skal treffe ordet **ole** og ikke slå til på ord som sk**ole**.

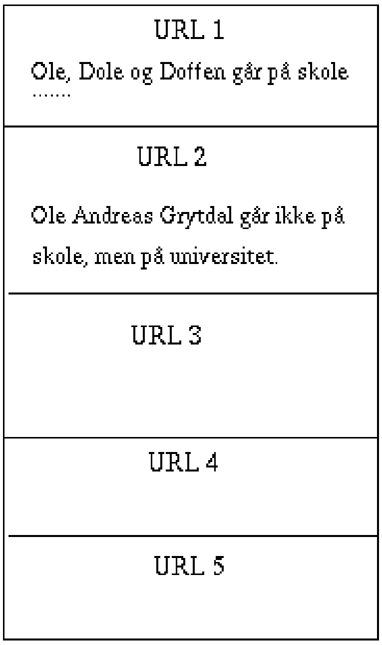
En annen ting kan være at man vil benytte wildcards som *,stjerne. Man kan søke på ord som lamp* og vil da ha alle substringer som begynner med lamp. Dvs. at ord som tak**lamp**e ikke skal gi treff, men ord som **lamp**e skal kunne gi treff.

17.1.4 Maksimum treff

Ved søk kan man oppgi hvor mange treff en vil ha rapportert tilbake. MS160 vil rapportere disse treffene tilbake til brukeren, men vil ikke si noe om hvor mange som er innenfor hvert dokument. Dette stiller oss i en kinkig situasjon. Ved søk i søkemaskinen er en selvsagt opptatt av å få rapportert et visst antall maksimums dokumenter. Vi må da regne med at et ord kan forekomme flere ganger innenfor et visst dokument og justere for dette når vi søker på kortet. Hvis en velger for mange ganger antall treff, bruker søkemaskinen lenger tid på å finne fram til alle treffene og sjekke om de virkelig ligger innenfor samme dokument. Velger en derimot en for liten verdi av på antall ganger maksimum treff, kan en ende opp med at man får for få treff.

Hvis man velger f.eks å få tre treff, dvs tre forskjellige dokumenter, som inneholder substrengen ole, ser man av Figur 21, “Maksimum treff eksempel,” på side 105 at man kun vil få rapportert **et** dokument fordi alle treffene blir registrert i **Ole**, D**ole** og sk**ole**. For å bøte på dette kan man la maksimum treff være f.eks 4 ganger så stor som det opprinnelige kriteriet. Da vil også andre dokument kunne bli registrert.

FIGURE 21. Maksimum treff eksempel



17.1.5 Samtidige søk

Ved søk i søkemaskinen låser man ressursene på MS160-kortet. Hvis to brukere prøver å søke samtidig, vil den siste måtte vente til den første har gitt fra seg ressursene. Vi må lage en sikker måte å lage et køsystem på slik at brukeren slipper å utføre søket på nytt, men heller på vente på tur. Det går ofte bort litt tid i nett-overføring, visning i www-klient osv., så det er ikke sikkert brukeren får et helt riktig inntrykk av hvor forsinkelsen ligger hen. Vi vil derfor lagre tid for start av søk og hvor lang tid det tok før brukeren fikk svar. Dvs. hvor lang tid det tok før www-serveren mottok svaret og søkemaskinen hadde funnet fram til svarene. Resten av tiden kommer an på belastning på nettet, www-serveren eller klientmaskinen.

17.2 Database oppslag

Når søkemaskinen har funnet fram til de treff som den skulle ut ifra de gitte kriteriene, må man finne ut fra hvilken URL de kom fra. Vi ønsker et så raskt oppslag (få aksesser) som mulig og vil bruke DBM-filer (Data Base Managment) for å oppnå dette. DBM-filer muliggjør å finne et innslag med bare et til to aksesser til filen. For å klare dette benytter den en hashing[1] tabell.

Vi vil benytte to slike DBM-filer for å finne URLen. En fil for å søke oss fram til riktig offset, og en fil for å ta nøkkelen som ga oss den rette offset'en og slå opp riktig URL. Grunnen til at vi må ha to DBM-filer er at vi ikke ønsker å lese inn Nøkkelfilen med offset inn i minnet ved oppstart. Siden søkeprogrammet blir startet og lastet inn i minnet når man trykker på søkeknappen på www-siden, kan vi ikke ha nøkkelfilen i minnet som man kunne ønske. Det er søking i denne som kommer til å ta det meste av tiden i et søk.

Det som kunne vært ønskelig var å ha en prosess som gikk i minnet med offset verdiene og la en protokoll kommunisere mellom denne prosessen og vår søkeprosess.

17.3 Postprosessering av innsamlede data

Etter at alle data er samlet, vil en postprosessering av endel tabeller være nødvendig. Dette må gjøres for at vi skal få så raske oppslag som mulig ved å lage forskjellige DBM-filer.

De DBM-filer som lages er :

- Nøkkelfil
  Filnavn : "KeyFil.dbm". Innholder en nøkkel som skal gi starten på hver url som er lagret.
- UrlFil
  Filnavn: "URLFil.dbm". Inneholder en nøkkel som skal gi navnet til hver url som er lagret.
- Tittelfil
  Filnavn: "TittelFil.dbm". Inneholder en nøkkel som skal gi tittelen til hver url som er lagret.
- Domenefil
  Filnavn: "DomeneFil.dbm". Inneholder en nøkkel som skal gi området til et visst domene.

1. hashing tabell er en tabell som ved en gitt nøkkel vil gi et unikt svar tilbake.

17.4 Søkegrensesnitt

Søkegrensesnittet er det som brukeren blir presentert for, når han kobler seg opp mot maskinen som kjører www-serveren. For å gjøre dette kobler han seg opp mot url: http://akkurat.idt.unit.no

Grensesnittet skal være enkelt å forstå og være ferdig utfylt med de verdier vi mener er gunstige. Dette kan en se i Figur 22, “WWW-søkegrensesnitt,” på side 109. Vi vil i de neste punktene beskrive hva de forskjellige feltene og valgknapper er.

- **Finn** - felt
  I dette feltet skal brukeren kunne spesifisere hva han ønsker å søke etter. Han skal kunne søke på flere kriterier ved å adskille de ved hjelp av mellomrom eller han kan søke på et utsagn ved å sette utsagnet i ““. Et eksempel på dette kan være :
  Finn: “Diplom oppgave” NTH
  For å gi treff på dokumenter som har noe med “Diplom oppgave” og/eller NTH å gjøre. Et søkeuttrykk som er omsluttet av “ ” kan kun være 32 tegn langt. Dette er pga. vindus størrelsen på MS160 kortet. Hvis en skulle hatt større uttrykk enn 32 tegn kan en slå sammen to eller flere vinduer i MS160 til “supervinduer”, men vi kommer ikke til å lage dette.
- **Avstand** - valgknapp
  Denne knappen spesifiserer antall tegn som treffene skal finnes innen.

  Eksempel:
  Finn : “diplom oppgave” NTH - Avstand : 200

  Dette eksemplet gir et treff hvis den finner begge ord innenfor et område på 200 tegn.

- **Minste antall kriteria** - felt
  I dette feltet oppgis antall kriteria som MINST må være oppfyllt for at man skal få et treff.Vi legger inn en feilsjekk slik at vi ikke kan søke etter fler kriterier enn det som er skrevet inn av brukeren. Et “M av N”-søk som dette kan delvis erstatte et boolsk søk. Vi lager denne måten å søke på istedetfor å ha med boolske operatorer som OR og AND.
  Eksempel:
  Finn : “diplom oppgave” NTH - Minst antall kriteria : 1

  Dette er en typisk OR funksjon der en enten kan få treff på “diplom oppgave”, NTH eller begge to.

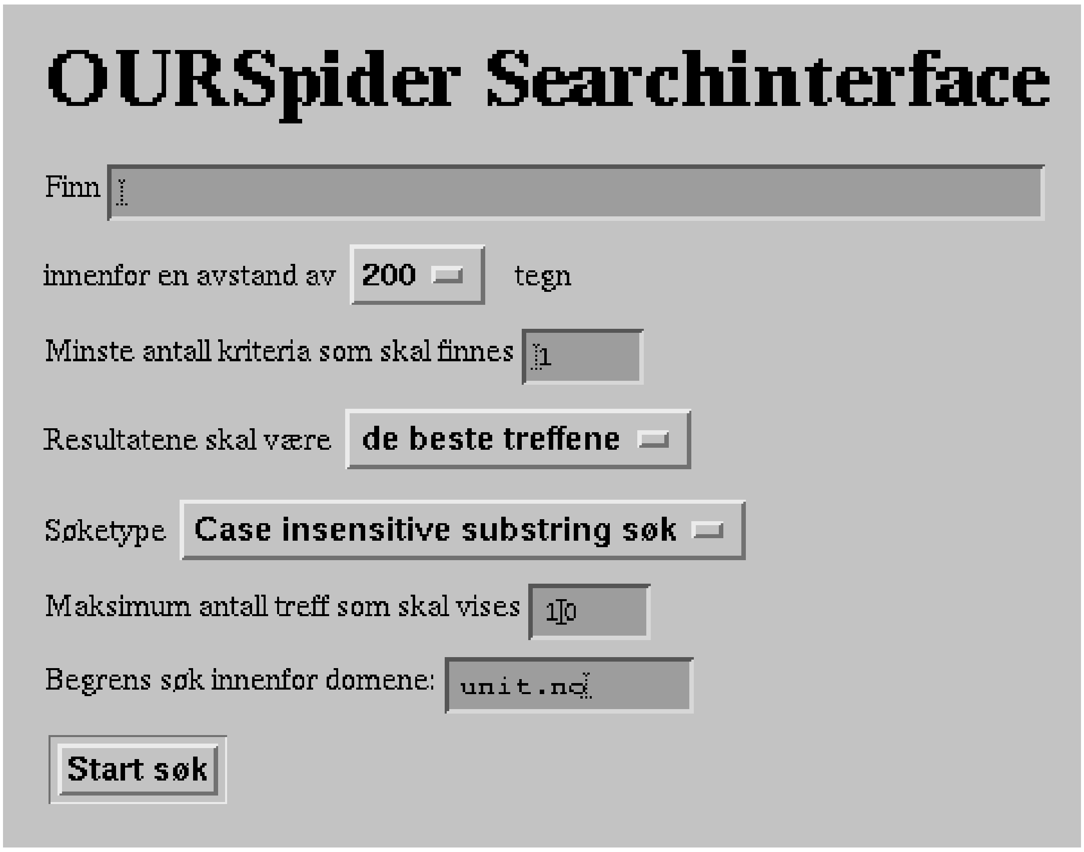
- **Resultat modus** - valg knapp
  Denne knappen har to muligheter. Enten “de beste treffene” eller “de første treffene”. Dette gjenspeiler seg i hvor mange treff brukeren har valgt skal være gyldig, X. Ved beste treff, vil alle treff bli registrert, blir sortert på vekt og skrevet ut. De X antall med høyest vekt blir da skrevet. Dette kommer til å ta lenger tid enn “de første treffene” da den må slå opp alle treffene før den kan sortere.
  “De første treffene” derimot vil stoppe etter at den har funnet og lokalisert X antall treff.
  Det er altså en avveining brukeren må ta. Vi har satt “de beste treffene” i utgangspunktet. Dette mener vi er fornuftig for brukeren som uansett må sjekke hva url’ene inneholder.
- **Søketype** - valgknapp
  Under søketype kommer vi til å ha to valg. Enten “case insensitv” eller “case sensitiv” substreng søk. Dette gir brukeren muligheten til å få et eksakt treff (case sensitivt) til hva han skrev i søkekriteriene.

  Eksempel:
  Finn: “Diplom opPgaVe” - søketype : case insensitiv
  Denne vil finne alle ord som samsvarer med “diplom oppgave” i både store og små bokstaver.

- **Maksimum antall treff** - felt
  Dette feltet kan brukeren fylle ut for å få maksimale antall treff som han vil ha ha.
- **Domene begrensing** - felt
  I dette feltet kan brukeren skrive inn det domenet søket skal utføres innen. Man skriver kun inn de deler i domenenavnet man vil søke på.

  Eksempel:
  Finn : “diplom oppgave” - Domene : unit.no
  Dette vil finne “diplom oppgave” innenfor alle maskiner innenfor unit.no domenet. Dvs, *.*.unit.no, der * kan være et eller annet navn. Hvis en oppgir et navn med stjerne, kutter rutinen bort dette.
- **Start søk** - knapp
  Idet brukeren trykker på denne, overføres parameterene som ligger i grensesnittet over til søkeprogrammet som startes, søker og gir de treff som er funnet tilbake til brukeren.

FIGURE 22. WWW-søkegrensesnitt



17.5 Resultat presentasjon

Når søkemaskinen har lokalisert alle treffene som søkeren har bedt om, skal de presenteres på en grei og oversiktelig måte. De treffene som kommer skal ha følgende informasjon:

- **Vekt**
  Vekten anslår hvor det er flest treff blant søkekriteriene som er oppgitt. Dette gir en bedre "korrekt-treff-indikasjon" for brukeren.
- **Tittel**
  Tittelen til URLen skal vises og være en hyperlenke til det orginale dokumentet. Tittelen vil gi brukeren enda en indikasjon på om dokumentet har relevant betydning.
- **URL**
  Ved siden av tittelfeltet vil også url stå som en hyperlenke.
- **Tid**
  Tiden som er brukt på søk og oppslag på disk vil bli presentert for brukeren.

18.0 Implementasjons krav

I denne delen vil vi kort nevne implementasjons kravene for søkemaskinen.

18.1 Hardware

I utviklingsfasen av programmet, måtte vi jobbe på maskinen som kjørte NetBSD. I utgangspunktet kunne vi ha sittet på en hvilken som helst UNIX-maskin og logget inn på akkurat.idt.unit.no som var maskinen som hadde søkekortet, men vi syntes det var greit å sitte på selve maskinen under utviklingen.

18.2 Software

De software komponentene som var nødvendige under utviklingen var :

- En c kompiler, vi brukte gcc.
- Gnus debugger - gdb
- Software driveren for kortet
- Operativsystem NetBSD
- Netscape - en www-klient for testing av søkemaskin,
- emacs - editoren vi skrev programmet i.

Ellers gjelder de samme kravene for søkemaskinen som for roboten som er nevnt under : Seksjon 14.3 på side 92.

Konstruksjons-spesifikasjon for WWW-roboten

Bjørn Hilstad og Ole Andreas Grytdal

Hensikten med dette dokumentet er å spesifisere detaljer om hvordan roboten skal imlementeres

19.0 introduksjon

20.0 Funksjonsbeskrivelser

I denne seksjonen skal vi ta for oss hvert enkelt programmodul og beskrive funksjonene som finnes i hver av dem. Navnene på seksjonene nedenfor skal gjenspeile filnavn og pakkenavn (Perl : package).

20.1 robot.pl

Dette er selve hovedmodulen i WWW-roboten vår. Det inneholder de fleste funksjonene som trengs for å hente inn og sjekke HTML-sider. Den starter med å lese instruksjonsfil og kommandolinjeparametre, og setter så opp funksjoner som tar seg av systemfeil (signaler fra systemet). Deretter henter den inn startdokumentet og plukker ut lenkene den finner her. Lenkene blir så sjekket for å finne ut om de skal legges i kø for senere traversering eller ikke. Slik vil den fortsette til den har nådd maskimum traverseringsdybde eller køen blir tom.

20.1.1 sig_int_handler

- Innparametere : Ingen
- Returverdi : Ingen (returnerer aldri)

Denne funksjonen skal forsøke å rydde opp og redde data ved systemfeil. Den blir startet hvis roboten mottat enten SIG_INT, SIG_TERM, SIG_QUIT, SIG_HUP ("man signal" for mer informasjone om signalene). Først skrives en beskjed til loggfilen og så tømmes innholdet av traverseringskøen ut på en fil (default : OURrest). Når dette er gjort avsluttes programmet.

20.1.2 logg_message

- Innparametere : Tekststreng som skal logges
- Returverdi : 1->logget, 0->feil oppstod

Funksjonen legger til en tekststreng på slutten av loggfilen (default : OURlog)

20.1.3 check_URL

- Innparametere : en URL
- Returverdi : 1-> ny URL, 0->har vært her

Funksjonen sjekker databasen for å se om en angitt URL allerede er hentet inn. Databasen ligger lagret i en DBM-fil (default : OUR_URLs.[dir|pag]) så operasjonen krever to diskaksesser. Skriver beskjed til loggen dersom URLen allerede er besøkt.

20.1.4 logg_URL

- Innparametere : en URL
- Returverdi : 1->ok, 0->feil oppstod

Funksjonen skriver en URL til DBM-filen som innholder besøkte URLer. Den skriver også URLen til en tekstfil (default : OUR_URLs), slik at man lett kan følge med hvor roboten befinner seg i øyeblikket (med kommandoen “tail -f OUR_URLs”)

20.1.5 is_html

- Innparametere : en URL og tilhørende HTTP header
- Returverdi : 1->URL er text/html, 0->ikke text/html

Funksjonen finner ut om en URL er av type text/html eller ikke. Til dette bruker den HTTP headeren som vi fikk tilbake når vi hentet URLen. Dersom det er en ukjent filtype, legges den inn i våre MIME-typer (Multipurpose Internet Mail Extensions) med sammen med filendelsen til URLen.

20.1.6 sjekk_avoid

- Innparametere : en URL
- Returverdi : 1->ok, URL skal ikke unngås, o->unngå URLen

Funksjonen sjekker den angitte URLen mot en fil som inneholder en liste over URLer som skal unngås (f.eks. fordi de er et “svart hull” eller ikke inneholder informasjon som vi vil ha i basen).

20.1.7 sjekk_domener

- Innparametere : oppgavenummer og en URL
- Returverdi : 1->URLen er gyldig i henhold til oppgavens domenebegrensning, 0->ikke gyldig URL for denne oppgaven

Funksjonen sjekker om angitt URL er innenfor domenebegrensningen til angitt oppgave. Skriver beskjed til loggfil dersom URLen ikke inneholder gyldig domene.

20.1.8 sjekk_schemes

- Innparametere : oppgavenummer og en URL
- Returverdi : 1->URLen er gyldig i henhold til oppgavens protokoll-begrensning, 0->ikke gyldig URL for denne oppgaven

Funksjonen sjekker om angitt URL er innenfor protokollbegrensningen (HTTP, FTP, osv.) til angitt oppgave. Skriver beskjed til loggfil dersom URLen ikke inneholder gyldig protkoll.

20.1.9 sjekk_URL

- Innparametere : en URL
- Returverdi : korrigert og sjekket URL hvis URLen er ok, tom streng dersom URLen ikke skal besøkes

Funksjonen sjekker om angitt URL inneholder elementer som gjør at den ikke skal hentes inn. Den sørger også for at URlen bare består av små bokstaver. Dersom URLen inneholder et ?-tegn (antyder parametre til et skript) eller et #-tegn (antyder relativ URL, dvs. bare en peker til en seksjon lenger ned i et dokumentet vi allerede har hentet inn) skal den ikke hentes inn. Vi sjekker også at filen ikke inneholder en av følgende filendelser/elementer : cgi-bin cgibin cgi .pl .gif .jpg .jpeg .au .aiff .aifc .wav .ps .gz .zip .tar .mpg .mpeg .mov .avi .exe .qt .lha .lzh .dvi .arj .rgb .uue cgi_bin .mv .xpm .xbm .z .tif .tiff .tga .mid .tex .hqx .sit .voc .mod .xm .s3m .raw .ppt .eps .fli .flc .pdf .mp2. Disse elementene tyder på at filen er på et binært format som vi ikke ønsker i databasen vår. Listen er på ingen måte fullstendig, men er dekkende nok til at vi nesten aldri opplever uønskede binærfiler i databasen vår.

Til slutt sjekker vi at maskinnavnet i URLen er et ekte maskinnavn og ikke et alias. Dersom maskinnavnet er på IP-nummer form konverteres dette til maskinnavn. Begge deler gjøres ved å bruke ‘host’- kommandoen i Unix (via OURlib som inneholder funksjonene som tolker tilbakemeldingene fra ‘host’).

20.1.10 sjekk_URL_queue

- Innparametere : en URL
- Returverdi : 1->URlen er ikke i køen fra før, 0->URLen ligger i køen

Funksjonen sjekker om angitt URL allerede ligger i traverseringskøen. Dette sjekkes ved å gå igjennom DBM-filen som køen ligger lagret i.

20.1.11 lagre_URL

- Innparametere : oppgavenr, en URL, innholdet i et dokument
- Returverdi : 1->lagret filen, 0->feil oppstod

Funksjonen lagrer angitt URL på riktig sted på lokal disk. URLen blir plukket fra hverandre slik at man har de enkelte bestanddelene hver for seg. Dersom URLen ike har et filnavn blir den tildelt filnavnet “index.html”. Filen lagres under en katalog som har et navn som er en sammenslåing av : lokal sti hvor alle dokumenter skal lagre, servernavn (med portnummer) og katalognavnene på serveren. F.eks. vil “http://peon.idt.unit.no/IDT/index.html” bli lagret som “lokal_sti/peon.idt.unit.no/IDT/index.html”.

20.1.12 traverser_link

- Innparametere : oppgavenummer, en URL, foreldredokumentet til URLen (dvs. dokumentet som refererte oss til denne URLen)
- Returverdi : HTTP responskode

Funksjonen setter HTTP-headeren til riktige verdier, slik at vi identifiserer oss overfor serveren. Deretter henter den inn angitt URL. Dersom man får tilbake et omdirigeringssvar (redirection, HTTP kode 301 eller 302) vil den følge omdirigeringen og hente dette dokumentet. Noen statistiske data om filen som hentes ned lagres også. Tilslutt kaller den en funksjon som lagrer dokumentet (se seksjon 20.1.11 på side 114), trekker ut alle lenker som finnes i dokumentet og lagrer disse i køer (disse blir senere sjekket før de evt. havner i traverseringskøen).

20.1.13 check_speed

- Innparametere : en URL
- Returverdi : ingenting

Funksjonen sjekker at vi ikke henter dokumenter for fort og at vi ikke henter for mange fra en server etter hverandre. Ved slike tilfeller vil den ta et visst antall sekunders pause. Ved for mange etterfølgende forespørsler til en server tar man en relativt lang pause, ellers tar man en kortere pause mellom hvert dokument.

20.1.14 load_restfile

- Innparametere : navn på restfil
- Returverdi :antall URLer som ble lastet inn

Laster inn URLer som har blitt dumpet ut på fil når roboten har blitt avbrutt. Disse legges inn i traverseringskøen slik at vi kan fortsette traverseringen der den ble avbrutt.

20.2 OURConfig.pl

Dette modulet inneholder alle konfiguerebare verdier for alle oppgavene som angis i instruksjonsfilen. Den innholder også default-verdier for bl.a. pausetider og traverseringsdybde. Kommandoen som brukes for å sende e-mail til operatøren er også lagret her.

20.2.1 parse_config_file

- Innparametere : Ingen
- Returverdi : Ingen

Funksjonen leser instruksjonsfilen (default : OUR_instructions) og legger verdiene den finner inn i lister. Listene inneholder ett element for hver oppgave.

20.2.2 parse_command_line

- Innparametere : Ingen
- Returverdi : Ingen

Funksjonen tolker kommandolinjen som gis inn når roboten startes, og setter variable i henhold til disse. Kommandolinjen har høyere prioritet enn instruksjonfil.

20.3 OURlib.pl

Dette modulet inneholder en samling små nyttige funksjoner som vi kommer til å bruke i roboten.

20.3.1 ip_to_hostname

- Innparametere : IP-nr
- Returverdi :servernavn

Konverterer et IP-nummer til et maskinnavn vha. ‘host’ kommandoen.

20.3.2 hostname_ti_ip

- Innparametere : servernavn
- Returverdi : IP-nr

Konverterer et maskinnavn til et IP-nummer vha. ‘host’ kommandoen.

20.3.3 timestamp

- Innparametere : Ingen
- Returverdi : tidsstempel

Funksjonen henter lokal tid fra maskinen vi kjører på og setter dette sammen til et tidsstempel som inneholder dato og klokkeslett.

20.4 OURreport.pl

Modulet innholder noen enkle funksjoner for å lagre informasjon om traverseringsprosessen og sende dette med e-mail til operatøren.

20.4.1 send_email

- Innparametere : mottaker adresse, tittel
- Returverdi :Ingen

Sender en e-mail til operatøren med beskjed om at en oppgave er ferdig. Inkluderer noen data om selve oppgaven, f.eks. hvor mange dokumenter som er hentet ned og totalstørrelsen av alle dokumentene.

20.4.2 add_to_report

- Innparametere : oppgavenummer, rapporttekst
- Returverdi : fullstendig rapporttekst

Legger en tekststreng til rapporten.

20.5 filesort.pl

Dette programmet leser inn en tekstfil som inneholder alle URLer i databasen vår og sorterer disse etter hvilket internett domene dokumentet er hentet fra.

20.5.1 domaincompare

- Innparametere : to URLer
- Returverdi : -1, 0 eller 1 avhengig av hvilken URL som ansees som størst

Sammenligner to URLer og returnerer en verdi basert på sammenligningen (1->a større enn b, 0 a er lik b, -1 -> a mindre enn b). Domenene sammenlignes ved å splitte opp maskinnavnene og sammenligne dem bit for bit (mest omfattende domene del sammenlignes først, dvs. den bakerste delen av domenene)

20.6 filemake.pl

Programmet leser inn en sortert versjon av alle URLene i databasen vår og genererer følgende filer :

6. en fil som er en sammenslåing av alle dokumentene hvor all HTML kode er fjernet (bortsett fra lenker)
7. en fil som inneholder alle URLene

8. en fil som inneholder hver enkelt fils posisjon innenfor filen nevnt i punkt 1
9. en fil som inneholder alle titler fra HTML dokumentene

Alle filene blir skrevet til i henhold til rekkefølgen på den sorterte listen med alle URler i (dvs. linje 1 i tittelfilen er tittel på URLen som er på linje 1 i URLfilen osv.).

20.7 domainfile.pl

Dette programmet skal generere en fil som inneholder alle domener og subdomener i URLene som finnes i databasen vår. For hvert domene/subdomene skal man vite hvor i hoveddatafilen (alle dokumentene slått sammen til en stor fil) domenet befinner seg. Programmet tar utgangspunkt i noen av filene som ble generert av programmet beskrevet i forrige seksjon. Første del består i å oppdage alle domener og subdomener. Deretter må man finne begynnelse og sluttposisjoner for domenet innenfor hoveddatafilen. Dette gjøres ved å gå gjennom alle URLene fra toppen av listen til man finner den første som passer og så starte fra slutten av URL-listen til man finner slutten på domenet (posisjonene man finner her brukes til å slå opp i filen med posisjonene til alle URLene, som gir URLens posisjon innenfor hoveddatafilen). Prosessen repeteres for alle domener/subdomener. En oversikt over hvilke domener som er prosessert finnes, slik at man ikke gjør samme oppgaven flere ganger.

20.7.1 finn_start

- Innparametere : et domene/subdomene
- Returverdi : startposisjon

Finner startposisjon på angitt domene ved å utføre prosessen beskrevet i seksjon 20.7 på side 118.

20.7.2 finn_slutt

- Innparametere : et domene/subdomene
- Returverdi : sluttposisjon

Finner sluttposisjon på angitt domene ved å utføre prosessen beskrevet i seksjon 20.7 på side 118.

20.7.3 domaincompare

- Innparametere : to domener
- Returverdi :1->domenene er ikke like, 0->domenene er like

Sammenligner to domener på samme måte som beskrevet i seksjon 20.5.1 på side 117.

20.8 statgen.pl

Dette programmet går igjennom alle filene i databasen vår og genererer en del statistiske data ut i fra disse. Følgende data genereres :

- Antall dokumenter per server
- Gjennomsnittlig dokumentstørrelse for hver server
- Største og minste dokument for hver server
- Total dokumentmasse for hver server
- Gjennomsnittlig dokumentstørrelse for hele databasen
- Antall servere i databasen
- Antall dokumenter i databasen
- Gjennomsnittlig antall dokumenter per server
- Største og minste dokument i databasen
- Serverene med færrest og flest dokumenter
- Serverene med størst og minst dokumentmasse

Konstruksjons-spesifikasjon av søkemasin

Ole Andreas Grytdal
Bjørn Kristian Hilstad

Algoritmer og løsninger til vår søkemaskin.

21.0 Innledning

I denne delen vil vi ta for oss algoritmer og hvordan søkemaskinen blir implementert. Det er dette dokumentet som ligger til grunn når implementeringen skal utføres.

22.0 Bruk av gammel kildekode

Under utviklingen av søkemaskinen brukte vi eksempler fra programmet “ftpsearch”. [15]

22.1 Ftpsearch

Ftpsearch ble utviklet av Hugo Eide Gunnarsen i 1993 og siden videreutviklet av Tor Egge. Programmet het opprinnelig Archie, men skiftet navn pga. copyright bestemmelser. Ftpsearch kan ta et argument inn og finne filnavn rundt omkring på en mengde ftp baser rundt om i verden.

22.2 Software driver

Tor Egge har skrevet / portet en driver for NetBSD som vi har brukt i denne oppgaven. Denne driveren oversetter våre c-kall til maskinkode kall som MS160 benytter seg av.

For å kunne bruke driveren må filen “ms160se.h” inkluderes i c-programmene.

22.3 Datastruktur

Sammen med ftpsearch programmet er det laget en datastruktur som vi har holdt oss til. Vi har lagt til endel i datastrukturen og fjernet endel som vi ikke fant nødvendig. En oversikt over strukturen kan en se i Figur 23, “Oversikt over datastrukturen til program / driver,” på side 125. Vi viser her de viktigste variable og strukturer som er verdt å merke seg. Vi vil kort kommentere hva de brukes til nedenfor.

De strukturer som brukes er :

22.3.1 WindowS

WindowS brukes for å sette opp vinduene. Den har følgende variable som er viktige:

- short MatchLatency
- char Upper[32] - øvre grense av søkekriteriet
- char Lower[32] - nedre grense av søkekriteriet

22.3.2 HitS

- long Result - Posisjon i kolonnen som ble truffet
- char HitMask (MatchPattern) - Sier noe om hvilken kolonne som hadde et treff

22.3.3 HitInfoS

- long NumHits - Antall treff som er blitt funnet etter en gjennomgang av datapumpen
- long HitsLeft - Antall treff som er igjen
- HitS *Hits - Peker til HitS strukturen

22.3.4 QueueS

Denne strukturen hjelper til med å holde rede på hvilke kriterie vi søker og jobber på.

- char search[1024] - Inneholder søkekriteriet
- WindowS Window - Inneholder informasjon om vinduene vi søker på.

22.3.5 RouteS

Denne strukturen brukes spesielt hvis en skal ha multiple kriterie søk, da en må sette rutingen for hver gjennomgang av datapumpen.

- short Serial
- short Parallel
- short Folding

22.3.6 RangeS

Denne strukturen brukes for å sette start og stopp av det området som det skal søkes på.

- short Start
- short Stop

22.3.7 MSConfigS

Denne strukturen inneholder alle data som skal til for å konfigurere MS160 kortet korrekt for et søk.

- RouteS route - route må konfigureres med alle tre variablene. For et et-kriterie søk settes alle til null.
- WindowS Windows[8] - Definerer åtte vinduer av WindowS strukturen, slik at en lett kan få satt forskjellige verdier til hvert av de åtte vinduene.
- char HitMask[32] - HitMask definerer en 256 bit tabell som skal definere hvilke vinduer som skal gi gyldige treff. Denne er forklart nærmere i Seksjon 8.5.4 på side 69.
- char mode - Sier hvilken modus MS160 skal kjøre. Forklart nærmere i Seksjon 8.5.3 på side 69.

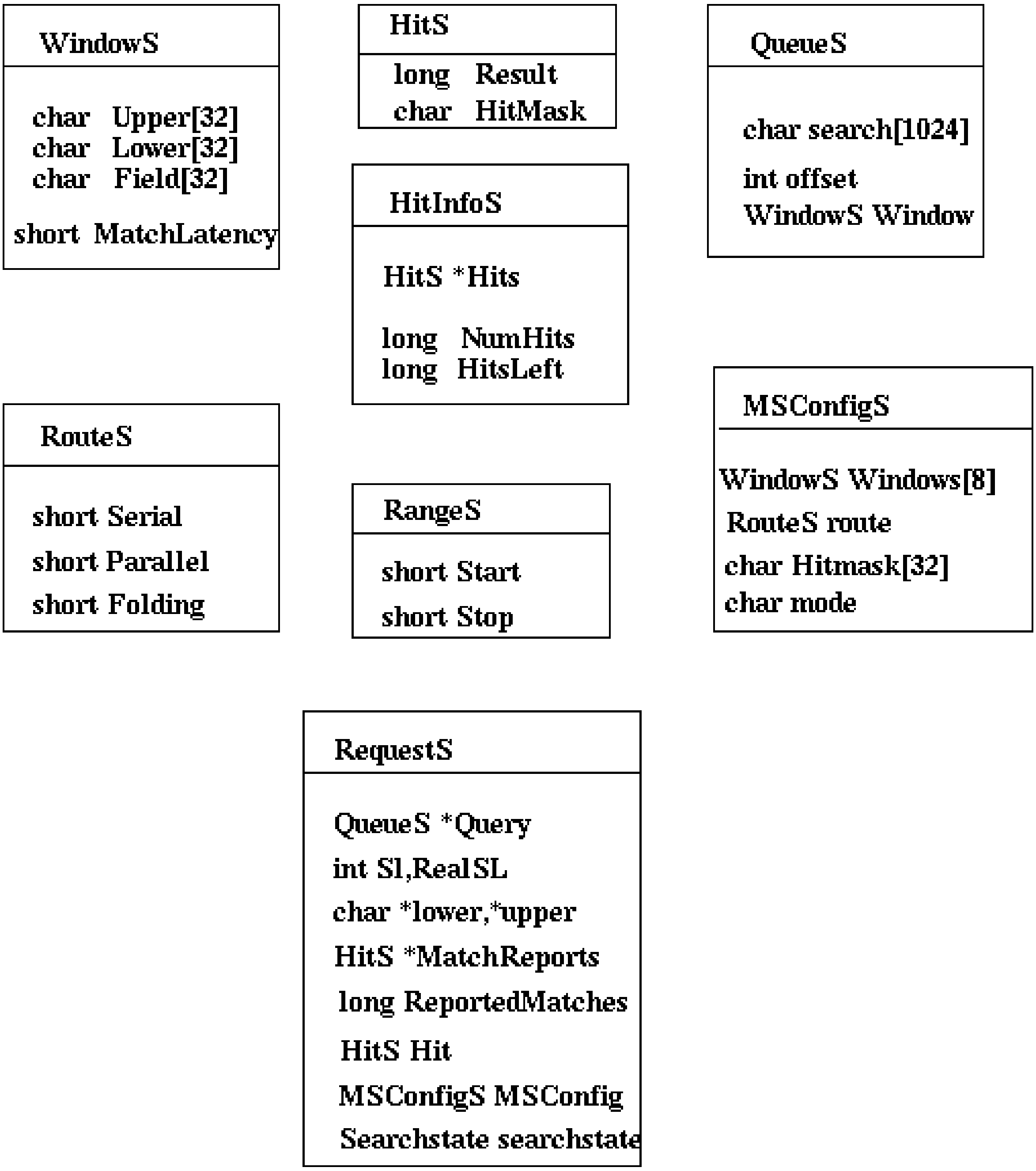
22.3.8 RequestS

Denne strukturen inneholder alle de strukturer og variable som man bruker til å sette opp et søk, sende over søk og ta i mot rapporterte treff.

- QueueS *Query
- int SL - Lengden på søkekriteriet
- char *lower, *upper - pekere som inneholder nedre og øvre grense
- HitS *MatchReports - peker av type HitS som inneholder de treff som MS160 rapporterer tilbake
- long ReportedMatches - Teller over antall rapporterte treff
- Searchstate searchstate - En “enum” som holder rede på om det er “in case eller case” sensitivt søk.

Dette er i hovedsak de strukturer vi har brukt fra den opprinnelige koden. Videre skal vi beskrive de rutiner og strukturer som vi skal implementere.

FIGURE 23. Oversikt over datastrukturen til program / driver



23.0 Nye program moduler

I de neste sidene vil vi beskrive de moduler vi vil skrive.

Vi vil her vise gangen fra vi får data som roboten har samlet inn, til resultat presenteres for brukeren ved et søk.

23.1 Postprosessering

Etter at data er samlet inn fra roboten må vi prosessere de filer som den har generert. De filer som vi skal bruke videre i prosessen er :

- DataFil - En stor fil som inneholder **inneholdet** til alle URLene som er samlet inn, sortert på server adresser (domene navn). Alle HTML kode er fjernet.
- URLFil - En fil som gjenspeiler seg i DataFil ved at den inneholder alle URLer som finnes i DataFil. Skriver antall elementer til disk.
- KeyFil - Fil som gjenspeiler seg i DataFil ved at den inneholder starten på hver URL som finnes i DataFil.
- TittelFil - Fil som gjenspeiler seg i DataFil ved at den inneholder tittelen til hver URL som finnes i DataFil.
- DomeneFil - Inneholder alle domenene og subdomener og det område som de dekker i DataFil.

De fire siste filene må konverteres til DBM format, da vi har kompatibilitets problemer mellom PERL og C sin DBM-format når den kjører på forskjellige plattformer som SunOS og NetBSD. Derfor konverteres alle filene på maskinen som kjører NetBSD.

Måten vi omformer filene , URLFil, KeyFil og TittelFil er som i følgende algoritme:

Åpne DBMfil for skriving og tekst fil for lesing
nøkkel = 0
for alle innslag i tekstfilen
{
skriv nøkkel og innslag fra tekstfil til DBMfil
nøkkel++
}

I domene fil lagrer vi domene navnet som nøkkel og området som domenet er over som innslaget.

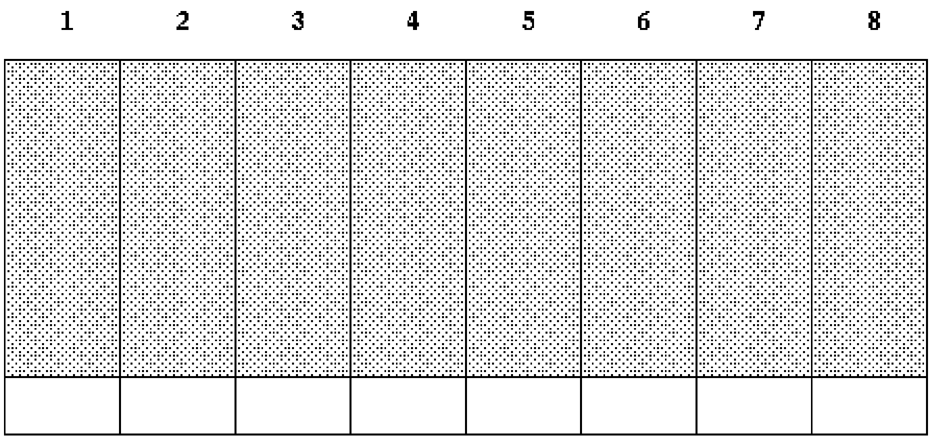
Konverterings algoritmene skal også lagre hvor mange unike nøkler vi har, da dette trengs i søkealgoritmen. Dette kan selvsagt finnes ut ved å traversere DBM-filen, men det er mye raskere å slå opp direkte i en fil.

23.2 Innlasting av data på kort

Etter at alle data er etter-prosessert, må vi laste DataFilen inn i minnet på MS160 kortet. Datafilen må være mindre enn størrelsen på minnet på kortet. Vi kommer ikke til å konstruere en søkemaskin / innlaster program som har støtte for flere kort av gangen. Hvis en skal dele opp datamengden over flere kort, må en passe på å få tilpasset de tilhørende datafilene til den store datafilen slik at man kan finne tilbake til hvilket kriterie som hørte til hvilket treff.

Ved innlasting på kortet deler vi opp datafilen i åtte like store deler og fordeler hver del inn på kortet slik som vist på Figur 24, “Vindus innlastning,” på side 127. Hvis datafilen er mindre enn antall megabyte i en kolonne , lastes og distribueres datafilen over alle åtte kolonnene.

FIGURE 24. Vindus innlastning



Algoritmen for dette blir :

Åpne datafil, finn filstørrelse

kolonne_data = filstørrelse/8

Åpne ms device

Slett alt som ligger i minnet

for hver kolonne

les inn kolonne data

Feilen vi får ved å gjøre det på denne måten er at vi kan kutte opp en URL over to kolonner. Vi kommer ikke til å ha noen sjekk på dette, men nevner det her. Problemet som kan oppstå er, hvis en skal søke på 2 eller fler kriterier og bruke "MatchLatency" for å finne treff innenfor en viss avstand. Ved et-kriterie søk har denne oppdeling ingen betydning. Noe av denne problemstillingen og løsningen vil vi komme tilbake til senere.

23.3 Start av søkeprogram

23.3.1 CGI-skript

Start av søkeprogrammet skjer ved at brukeren trykker på "Start søk" knappen som han ser på via sin www-klient. For å få en interaksjon mellom bruker og program brukes noe som kalles CGI-skript.

CGI (Common Gateway Interface) skal skape en port mellom input fra brukeren og tilbakemeldingen som et eventuelt skript skal ta seg av. De fleste nye www-servere har støtte for CGI-programmering. Eksempler på CGI skript kan være "image-maps" som man kan klikke på og ut fra det velge en URL, eller fylle ut et skjema der det man skriver inn påvirker og bestemmer hva som kommer tilbake.

Vi vil benytte et C-bibliotek som har CGI og HTML støtte. Dette er kompilert opp for NetBSD. Biblioteksfilen heter : cgihtml.a , men headerfilene som må inkluderes er :

- cgi-lib.h - Inneholder kall for å hente innhente verdier fra www-dokumentet
- html-lib.h - Inneholder kall for å støtte opp om utskrift i HTML-format.

Ved kall av et C-program blir endel parametere overført til den kjørbare filen fra www-dokumentet. Disse leses inn via kallet **cgi_val** som legger det i en variabel av typen **llist**.

23.3.2 Tolking av søkekriteriene

Etter innskriving av alle søkeparameterene må disse tolkes. De fleste feltene er enkle, men vi vil ta for oss feltet med søkekriteriene.

Dette feltet må splittes opp i de forskjellige delene og legges inn som forskjellige søkeuttrykk. I tillegg til dette må man tolke betydningen av søkekriteriene noe som beskrives nærmere nedenfor.

Hvert søkekriterie kan være av typen :

- Enkelt ord - ordet som skrives skal være en delstreng eller eksakt treff.
- Wildcard - Hvis ordet har en stjerne etter seg betyr dette at det ordet skal **begynne** med det som står, men likevel være en substreng av resten.
  Eksempel : ole* . Ordet skal begynne med ole, og vil derfor ikke gi treff ved f.eks sk**ole**.
- Uttrykk - Ord som står inne i anførselstegn skal tolkes som et uttrykk. Dette mener vi kan være lurt å ha med, da det gir en klarere indikasjon om man har funnet det man ønsker, hvis man får et treff.
  Eksempel : “Diplom oppgave”

Nedenfor er en algoritme for å splitte opp og undersøke kriteriene.

```
Løp igjennom den store søkestrengen
    hvis treff på “ finn neste “
        la det være et uttrykk og sett skille
        antall_kriterier++
    hvis treff på mellomrom
        la ordet mellom skille og treff være kriterie
        antall_kriterier++
for antall_kritierier
    hvis sjekk ord avsluttes av *
        legg inn et mellomrom foran ord og fjern stjerne
```

De andre feltene / knappene er legges bare inn som enkelt variable og trenger ingen prosessering.

23.4 Oppsett av søkemaskinen

I kapittelet om MS160 har vi skrevet om hva som trengs for å sette opp MS160 til et korrekt søk. Vi vil her beskrive metodene vi vil bruke for å konfigurere kortet korrekt.

23.4.1 Vindu oppsett

Hvert vindu har en øvre og nedre grense som må settes. MS160 har den muligheten at den kan søke mellom disse grensene, noe som skaper endel trøbbel hvis en skal ha case insensitivt søk.
Eksempel:

Øvre grense : diplom
Nedre grense: DIPLOM

Vi har skrevet det slik da ASCII- verdien til små bokstaver er høyere enn for store bokstaver. Det som er problemet her er at vi vil få treff på ALLE kombinasjoner mellom grensene , slik som eiplom,fiplom etc. Vi vil benytte en algortime som ble laget for ftpsearch programmet for å omgå dette.

Måten teksten blir lagret på blir endret fra å være abc..yzABC..XYZ til aAbBcC...xXyYzZ. Dette gjør at vi omgår det nevnte problemet. Naturligvis må data på kortet endres på samme måten som søkekriteriene.

MatchLatency for hvert vindu blir satt til det brukeren oppgir på www-siden.

23.4.2 Duplisering av vinduer

Ved søk i MS160 er det lurt å utnytte alle vinduene til søk. Derfor bør en duplisere antall søkekriterier til alle vinduene. Duplisering av dataene gjør at søk i parallell kan utføres, skjer i henhold til Tabell 5 på side 130.

TABLE 5. Duplisering av vinduer

Antall søke-kriterier	Antall dupli-seringer	Antall gjennom-ganger av data-mengden
1	7	1
2	3	2
3	1	4
4	1	4
5	0	8
6	0	8
7	0	8
8	0	8

23.4.3 Oppsett av HitMask

På bakgrunn av antall kriterier og hvor mange treff en vil ha kan en bygge opp en HitMask[1]. Vi vil se på de algoritmer vi benytter for å gjøre dette. Vi kan dele opp beregning av HitMask i tre deler.

- Beregning av gyldige treffmønstre
- Duplisering av de gyldige mønstrene i henhold til vindus dupliseringen
- Sette de mønstrene som vi har kalkulert oss fram til inn i den 256 bit lange HitMasken.

Beregning av gyldige treffmønstre

En generell rutine for dette kan være vanskelig å vise her. Hvis man skal beregne HitMasken for boolske algebraer, kan man benytte seg av sannhetstabeller for å kalkulere slutt uttrykket.

Vi har sett på metoden M av N treff [12]som gjør det enklere å kalkulere de gyldige kombinasjonene, siden vi ikke har konstruert en tolker som tar boolske uttrykk.

Antall gyldige kombinasjoner for M av N treff vil bli :

$$\text{Antall kombinasjoner} = \sum_{\text{Antall vindus kombinasjoner}} 2^{\text{Vindus Nummer}}$$

Et eksempel:
Vi vil ha at minst 1 av 2 treff skal være gyldige. Vi legger kriteriene inn i vinduene slik at de fyller opp vindu 0 og 1. Av formelen ser vi at vi får : $2^0 + 2^1 = 3$ kombinasjoner som vil gi gyldige treff. Enkel resonnement ville også gitt samme svar, men hva med å finne 5 av 8 gyldige kombinsjoner ?

Duplisering av gyldige treffmønstre

På samme måte som med vindusduplisering må også de gyldige treffmønstrene dupliseres på samme måte. Dette kan man gjøre ved for hvert gyldig treffmønster, skifte denne et visst antall ganger i henhold

1. HitMask er forklart i kappitelet om MS160

til Tabell 6 på side 132. Tilsammen får vi alle de gyldige kombinasjoner som kan gi treff ved et søk.

TABLE 6. Antall skifte operasjoner

Antall vinduer	Antall kopier	Antall ganger å skifte	Antall posisjoner å skifte
1	7	7	1
2	3	3	2
3	1	1	4
4	1	1	4
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0

Oppbygging av HitMask

Tilslutt må man sette hvert av de 256 bitene i HitMask i henhold til de kombinasjoner en nå har beregnet seg fram til. Algoritmen for å sette bitene er som følger [12] :

```
Mask[32], Maske[256]
for treff_bit=0;treff_bit<256;treff_bit++
  for teller =0;teller<antall_kombinasjoner;teller++
    Maske[((255-g_komb[teller] & treff_bit) | g_komb[teller]] = 1
for (byte=0;byte<32;byte++)
  for(bit=0;bit<8;bit++)
    if (Maske[Masketeller] )
      Mask[byte]= 2^bit | Mask[byte]
      Masketeller++
```

Maske inneholder en oversikt over de 256 mulig kombinasjonene og om det er treff i de fra de to foregående punktene.
Mask er selve HitMask'en som skal overføres til kortet.
g_komb er en tabell som inneholder verdien til de gyldige kombinasjonene.

23.5 Søket

Selve søket er en liten algortime, nå som alt er satt opp og konfigurert. Vi må sette opp området som MS160 kortet skal søke innen også sette i gang datapumpen som dytter dataene gjennom kortet.

23.5.1 Gjennomgang av datamengden

Antall ganger man må dytte data gjennom kortet avhenger av hvor mange søkekriterier en har, Se Tabell 5 på side 130. I tillegg må man endre rutingstabellen som styrer datastrømmene til vinduene. [11]

Når datapumpen har stoppet, kan en lese ut eventuelle treff. Det er viktig å lagre unna de treff en har før en fortsetter på neste søk.

En feil ved kortet er at etter et treff blir de 14 neste tegnene liggende i en “skygge” og blir ikke gjennomsøkt. Ved et treff, må en derfor sette i gang et søk akkurat rett etter det første treffet. Dette gjør man for å gardere seg mot at en mister eventuell treff som kan ligge rett etter ordet man fikk treff på.

23.6 Oppslag av treff

Etter at alle treff er registrert, må man slå opp den riktige URL som treffet svarer til. For å slå opp den riktige URLen må man ha verdien til treffmønsteret for å se hvilket vindu som ga treff, og offset i kolonnen.

23.6.1 Beregning av offset for oppslag

Vi har registrert alle treffene, så det er bare å starte å lese ut treffmønster og offset. Treffmønsteret sier noe om hvilket vindu som ha treff og sammen med datarutingen kan vi si hvilken kolonne treffet ble funnet i. Når en har kolonne, offset i kolonnen og vet hvor mye data som er i hver kolonne kan en beregne seg til hvor treffet er i den store datafilen.

Dette gjøres ut fra følgende formel :

offset til fil = kolonne*antall_bytes_per_kolonne+kolonne_offset

23.6.2 Domenebegrensning

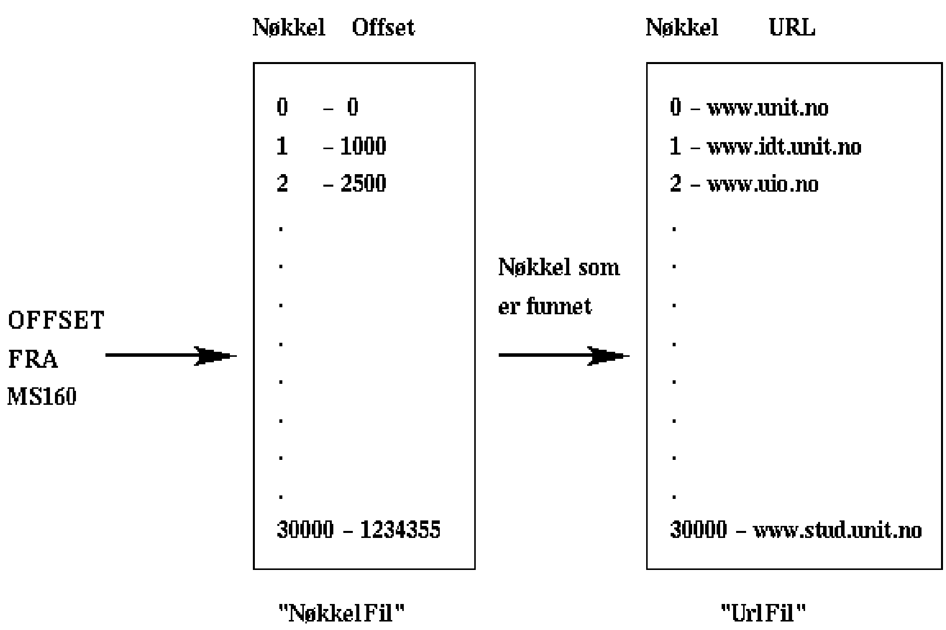
Når vi har funnet offseten som ordet ligger på, sjekker vi om ordet skal være gyldig i henhold til den domenebegrensningen. Dette gjør vi ved at vi før selve søket starter, leser inn området for domenet slik at vi har start og slutt-adressen for domenet. Hvis offset'en som er funnet ligger innenfor start og slutt adressen lar vi det være et gyldig treff.

23.6.3 Oppslag av URL på basis av offset i fil

Vi vil benytte oss av to DBM-filer for å finne URLen. En fil for å søke oss fram til riktig offset, og en fil for å ta nøkkelen som ga oss den rette offset'en og slå opp riktig URL. I Figur 25, "Oppslag i DBM-fil," på side 134, kan en se forløpet ved et oppslag.

Vi tar den offset vi får fra kortet, binærsøker i "Nøkkelfil" på offset slik at vi tilslutt vil finne hvilken nøkkel som vi skal bruke for å slå opp i "UrlFil". Vi regner med at dette er den beste måten å slå opp på da DBM-oppslag gir et eller to aksesser og binærsøk i en fil på mellom 16384 og 65535 offset'er vil gi henholdvis maksimalt mellom 14 - 16 oppslag. Måten vi gjør det på er kanskje litt bakvendt, da vi prøver oss fram med forskjellige nøkler og sjekker om ønsket offset er innen for det resultatet som søket gir.

FIGURE 25. Oppslag i DBM-fil



Algoritmen for søket blir :

nedre = 0 element i fil, øvre er siste element i fil
hvis vi ikke har treff

midt = (nedre+øvre)/2

les inn offset fra dbmfil med midt som nøkkel

hvis innlest_offset>ønsket offset

les inn offset med midt -1 som nøkkel

hvis ny innlest offset<ønsket offset

VI HAR ET TREFF!

nøkkel = midt -1

øvre = midt

eller hvis innlest_offset<ønsket offset

les inn offset med midt+1 som nøkkel

hvis ny innlest offset>ønsket offset

VI HAR ET TREFF!

nøkkel = midt

nedre = midt

eller hvis innlest_offset = ønsket offset

VI HAR ET TREFF!

nøkkel = midt

Bruk nøkkel verdi til å slå opp URL i URLfil og dens tilhørende tittel i tittelfilen, samle alle urler i en tabell, sorter på vekt og hvis til brukeren

Grunnen til at vi må ha to DBM-filer er at vi ikke ønsker å lese inn Nøkkelfilen inn i minnet ved oppstart. Siden søkeprogrammet blir startet og lastet inn i minnet når man trykker på søkeknappen på www-siden, ønsker vi ikke i tillegg å laste inn hele nøkkelfilen da vi regner emd at dette kommer til å ta mer tid enn å binærsøke på selve filen.

Det er binærsøkingen som kommer til å ta det meste av tiden i et søk.

Det som kunne vært ønskelig var å ha en prosess som gikk i minnet med offset verdiene og la en protokoll kommunisere mellom denne prosessen og vår søkeprosess. Dette ser vi på som ikke gjennomførbart i vår oppgave rent tidsmessig.

23.6.4 Første og beste søk

Opsjonen første og beste søk virker på to forskjellige måter.

Første søk vil ta alle de første treffene den finner, i samsvar med maksimalt antall treff. Dette er tidssparende for brukeren hvis det er mange søk han skal gå igjennom.

Beste søk vil gå igjennom alle treff den finner for så å sortere de etter mest vektlagt dokument. Et høyt vektlagt dokument har en høy forekomst av treffet i seg. Den vil så rapportere de beste dokumentene tilbake til brukeren. Mer at dette tar lenger tid, da den må slå opp alle-treffene for å få lagd oversikten.

23.6.5 Logg funksjon

Logg funksjonen lagrer disse data på Logfil.

- Tid for søk
- Antall og hvilke søkekriteria
- Host
- Total søketid

Alle data bortsett fra host kan tas rett fra søkeprogrammet. Maskinen brukeren satt på overføres via et CGI-skript.

Implementasjons-beskrivelse

Ole Andreas Grytdal
Bjørn Kristian Hilstad

Dette dokumentet oppsummerer implementasjonsfasen og ser på hva vi har implementert og hvilke mulige forbedringer vi ser for oss.

24.0 Implementasjon av søkemaskin

I dette kapittelet vil vi ta for oss hva som har blitt implementert og forskjellige tester som har blitt utført på programmer og funksjoner.

24.1 Implementering

Vi brukte endel tid på å sette oss ned å forstå driveren til MS160 kortet og hvordan kallene skulle utføres for at vi skulle få noe respons. Da det ikke foreligger noen spesiell dokumentasjon på driveren utover kildekoden, måtte vi ty til eksempler på bruk fra programmet ftpsearch og også spørre vår veileder Tor Egge som også er medforfatter av ftpsearch.

Det dukket opp endel problemer der enkelte ble løst, mens andre enda er et uløst mysterium. Vi har derfor måttet gå bort fra noe av det vi beskrev i kravspesifikasjon og konstruksjonspesifikasjon.

24.2 Hva er implementert og testrapport

De deler som er implementert og vi har kjørt tester på er :

24.2.1 Innlastnings program

Innlastings programmet ble tidlig ferdig og kunne testes ut på et tidlig tidspunkt. Testresultatene kan leses nedenfor.

TABLE 7. Test resultater av innlastings program

Forutsetning	Forventet handling	Bemerkning	Dato:
Ingen innfil spesifisert	Avslutter kontrollert og gir brukeren beskjed	OK	20.11.95
Feil filnavn oppgitt	Avslutter og gir brukeren beskjed om at navnet ikke kan bli funnet	OK	20.11.95

TABLE 7. Test resultater av innlastings program

Forutsetning	Forventet handling	Bemerkning	Dato:
Filen er for stor for minnet på kortet	Avslutter og gir brukeren beskjed	OK	20.11.95
Filen passer inn i en kolonne	Distribuerer filen i hver kolonne	Fikk noen små-feil og ble aldri implementert da vi som regel vil får større filer enn en kolonne-størrelse	20.11.95

24.2.2 Konverterings program for etter prosessering

Alle konverterings programmene fungerer tilfredstillende og konverter korrekt fra tekstlig format til DBM-format. Vi fant ut hva grunnen var til at vi ikke kunne bruke DBM-filene som ble laget av perlskriptet. Vi kjørte perl-skriptet og c-programmene på to forskjellige arkitekturer og maskiner. Årsaken ligger nok i big-endian og little-endian problematikken som viser seg når man forsøker å overføre data mellom to datasystemer som har prosessorer som ikke snakker samme språk (forskjell i representering av data på byte nivå).

TABLE 8. Testresultater av konverterings program

Forutsetning	Forventet handling	Bemerkning	Dato
Tekstfilen mangler	**Avslutter program**	OK	4.1.96
Oppdatering av nye filer	Skriver over de gamle filene	OK	4.1.96

24.2.3 Domene søk

Domene begrensning virker tilfredsstillende. Litt dårlig feilsjekk, men det var pga. tidsnød at dette ikke ble rettet på.

TABLE 9. Testresultater av domene søk

Forutsetning	Forventet Handling	Bemerkning	Dato
Ingen verdier	Gir tilbake melding til brukeren om å oppgi et domene. Avslutter.	OK	10.1.96
Gyldig domene	Finner område til domenet	OK	10.1.96

TABLE 9. Testresultater av domene søk

Forutsetning	Forventet Handling	Bemerkning	Dato
Ugyldig domene	Gir tilbakemelding til brukeren om et ugyldig domene. Avslutter.	OK	10.1.96
Egentlig gyldig domene, men har * foran slik : *.unit.no	Gir beskjed om at domenet er feil. Avslutter.	Burde nok fjernet * og . men dette er ikke implementert	10.1.96

24.2.4 Wildcards

Implementert og virker tilfredsstillende.

TABLE 10. Testresultater av Wildcard søk

Forutsetning	Forventet resultat	Bemerkning	Dato
Inn: Grytd*	Fjerner * og legger til mellomrom foran Grytd	OK	5.1.96
Inn : "Ole Grytd*"	Fjerner ikke * innenfor et uttrykk. Brukes for å kunne søke på stjerne.	OK	5.1.96

24.2.5 Maksimum antall treff

Funksjonen er testet og implementert.

TABLE 11. Testresultater av Maksimum antall treff

Forutsetning	Forventet resultat	Bemerkning	Dato
Input : tall	MaxResults blir riktig satt	OK	5.1.96
Input : tegn	MaxResults blir 0	OK	5.1.96
Input : tall større enn antall MAX definerte i søkemaskinen	Maxresults blir satt til maksimalt antall som er definert.	OK	5.1.96

24.2.6 Låsemekanimse

Funksjonen er testet og implementert. Et skript for å fjerne låsefil hvis prosessen dør eller henger er ikke lagd. Dette KAN henge hele systemet.

TABLE 12. Testresultater av låsemekanismen

Forutsetning	Forventet resultat	Bemerkning	Dato
Start av program	Lager låse fil	OK	5.1.96
Avslutter program	Sletter låsefil	OK	5.1.96
Søke mens en annen søker samtidig	Venter til låsefilen er slettet.	OK	10.1.96
Program krasj, mens låsefil er satt	Låsefilen forblir og kan blokkere hele systemet	Løsning se forslag i kons.spek. Ikke fikset.	10.1.96

24.2.7 Oppslag i DBM-fil

Oppslagene i de to DBM-filene fungerer tilfredstillende.

TABLE 13. Testresultater av oppslag i DBM-fil

Forutsetning	Forventet resultat	Bemerkning	Dato
Riktig offset inn	Søker korrekt ved binær søk og slår opp rett url og tittel.	OK	10.1.96
Offset er større en størrelsen på datafil	Vil ikke finne noe ved binærsøk og returne. Feiler da på å slå opp url og tittel.	Tenkt tilfelle. I virkligheten vil aldri dette skje.	10.1.96

24.2.8 Søkegrensesnitt

Søke grensesnittet er skrevet i html. CGI-skriptet som overfører parametere er laget.

TABLE 14.

Forutsetning	Forventet resultat	Bemerkning	Dato
Oppslag på url til søkesiden.	Gir brukeren et skjema der han kan fylle inn de rette verdien for et søk.	OK	23.11.95
Trykk på “Start søk” knapp	Kaller CGI-skriptet og starter søkemaskinen.	OK	10.1.96
Søk ferdig	Viser søkeresultater til bruk-eren.	OK	10.1.96

24.2.9 Minst 1 av N søkekriteria

24.2.10 Testresultater av minst 1 av N søkekriteria

Denne er implementert og testet. Merk at det står kun Minst 1 av N søkekriteria. Se ellers hva som ikke har blitt implementert.

TABLE 15. Testresultater av minst 1 av N søkekriteria

Forutsetning	Forventet resultat	Bemerkning	Dato
Ingen kriteria oppgitt	Ingen treff. Programmet ber om minst et input.	OK	10.1.96
1 kriteria oppgitt, skal finne minst 1.	Finner alle forekomster.	OK	10.1.96
2 eller fler kriteria oppgitt og skal finne minst 1.	Finner alle dokumenter som har kriteriene. Vanlig OR funksjon av kriteriene.	OK	10.1.96
2 eller fler kriterier og man skal finne minst 2 eller fler	Skulle melde forekomster av treffene.	VIRKER IKKE!	18.1.96
Antall søkekriterier en skal finner er fler enn de oppgitte kriteria	Melder fra til bruker at antall kriterier en ønsker er for mange.	OK	10.1.96

24.2.11 Beste resultater / Første resultater

Denne rutinen virker og er implementert.

TABLE 16. Testresultater av beste resultat og første resultat

Forutsetning	Forventet resultat	Bemerkning	Dato
Første resultat	De X antall første resultatene blir returnert. Søket er raskt.	OK	10.1.96
Beste resultat	De X antall beste resultatene kommer tilbake, sortert på vekten til dokumentet.	OK	10.1.96

24.2.12 Tidstempling av søk

Vi lagrer tid for søk, oppslag, ventetid i tilfelle kortet er opptatt og totaltid for hele søket.

24.2.13 Bruk av offset fil i minne eller på DBM-fil

Slik vi skrev i konstruksjonsspesifikasjonen så bruker vi en DBM-fil som brukes i binær søk. Vi har testet og implementert begge metodene og kommet fram til følgende konklusjon.

- **DBM-fil** - DBM-fil bruker ca 0.15 sek for å slå opp en treff.
- **Laste inn fil i minnet** - På en fil med ca 30000 URLer tok det 2.5 sekunder å laste inn filer.

Vi ser her at vi kan slå opp ca. 20 treff før vi tas igjen av binærsøk i minnet.

Som regel kommer URL filen til å bli endel større enn dette også, og da vil det da enda lenger tid å laste inn filen i minnet. Vi regner derfor med at vårt valg har vært det beste ut fra de forutsetninger vi har tatt.

24.3 Hva som ikke ble implementert

Det er også enkelte ting som ikke ble implementert i henhold til krav- og konstruksjonspesifikajon.

24.3.1 Flere søkekriteria

Etter å ha prøvd å få dette til, gikk vi bort fra dette. Vi kalkulerte Hit-Masken riktig, satte dataruting korrekt, men ved tilbake lesing av treffene fikk vi ikke helt de resultater vi ønsket. Vi regner med at dette har med falske treff og at vi ikke beregner oss riktig tilbake til den rette offset vha data rutingen. Det ble for ustabilt til at vi ville ha dette med i den ferdige versjonen av søkemaskinen.

Som følge av dette søker vi kun på et søkekriteria av gangen og lagrer unna de treff som dette gir. Vi mister da muligheten til å bruke Match-Latency for å finne treff som ligger innenfor et visst område. Istedet må vi prosessere på dette i etterhånd.

En ting som kommer ut av dette er at vi slipper å gå igjennom datamengden så mange ganger som vi måtte ellers når vi ruter data. Vi slipper unna med antall gjennomganger av datamengden lik antall kriterier. Ved f.eks fem kriterier, trenger vi kun fem gjennomløp, men på den andre måten hadde vi trengt åtte gjennomløp.

24.3.2 Logg funksjon

Grunnet tidspress på slutten ble denne muligheten nedprioritet og ikke implementert. Det skulle ikke være noe problem å legge inn denne opsjonen ved et senere tidspunkt.

24.3.3 Søkegrensesnittet

Søkegrensesnittet som er vist i kravspesifikasjonen har med elementer som ikke har blitt implementert i den siste versjonen. Søke grensesnittet vil derfor kunne se litt annerledes ut enn det som er avbildet i kravspesifikasjonen.

25.0 Forbedringer av søkemaskinen

De rutinene vi ikke implementert kan sees på forbedringer av søkemaskinen. Dette er rutiner i henhold til krav spesifikasjonen. Andre forbedringer er nevnt nedenfor.

25.0.1 Tolker og bedre uttrykk

Vi kunne tenke oss en tolker av søkeuttrykket slik at man kan bruke regulære eller boolske uttrykk. De to uttrykkene vil være veldig kraftig når det gjelder søk i fritekst på MS160. Et boolsk uttrykk kan f.eks være :
Finn : (olegr OR Grytdal) AND (diplomoppgave AND NTH)

25.0.2 Bedre resultatlisting

En bedre resultatutlisting for å presentere treffene til brukeren på en bedre måte. Av informasjon vi vil ha med kan nevnes:

- Filstørrelse på dokumentet.
- Når dokumentet er innhentet sist.
- Dele søkeresultatet over flere sider.

25.0.3 Bedre spesifisering av hva en vil søke i

Vi ønsker å bestemme mer hva en kan søke på i dokumentet. Det man f.eks kan søke på er :

- Tittel feltet
- Lenker i dokumentet. Slik det er idag, tolkes alt som det lå i ett dokument.
- Lenker til et dokument
- Dokumenter som omhandler en spesiell maskin

26.0 Forbedringer av roboten

Roboten har stort sett fungert tilfredsstillende den tiden den har vært operativ. En del småfeil har oppstått når roboten har truffet på URLer som inneholdt ting vi ikke hadde forutsett. Dette har vi alltid greid å rette på og roboten er nå ganske robust. Det første problemet var at den brukte altfor mye ressurser (les : minne) på maskinen den kjørte på. Dette har vi forbedret ved å legge mest mulig av dataene som brukes under traverseringen på disk (i DBM-filer).
Etter å ha observert roboten under mange kjøringer har vi funnet en del finesser (og noen påkrevde forbedringer) som vil gjøre roboten bedre. Vi lister de opp i prioritert rekkefølge her :

- "Last-modified" dato må lagres for hvert dokument slik at vi kan vurdere om et dokument skal hentes inn på nytt eller ikke. Ellers bør man vurdere å lagre hele HTTP headeren som blir returnert for hvert dokument da det ligger en del nyttig informasjon i den.
- Roboten bør kunne bli hurtigere. Ved normal kjøring (ressurs-vennlig) henter den inn ca. 4000 dokumenter om dagen. Dette kan forbedres betraktelig med en lurere algoritme for innhenting. Slik det er nå er det for mye døtid (ren venting) under traversering. Ved f.eks. å ha en URL kø per server vil man kunne hente inn et dokument fra hver kø om gangen (må kjøre flere innhentingsprosesser samtidig)
- I DBM-filene lagres alle URLene i klartekst. Dersom man pakket disse sammen til binærformat (perl : pack og unpack) ville databasen bli mindre og kanskje raskere og slå opp i.
- I stedet for å slavisk følge lenker i den rekkefølgen de oppdages vil man kunne redusere belastningen på serverene ved å hente inn en vilkårlig utvalgt URL hele tiden. Vi bruker jo allerede en DBM-fil (returnerer elementer i en pseudo random rekkefølge. Rekkefølgen bestemmes av hashing formelen som benyttes) til køen så dette skulle være enkelt å legge til som en opsjon.
- Skrive om deler av roboten fra perl til C for å øke effektiviteten. Vi vil nødig gi slipp på perl egenskaper i tekstprosesseringsdelen, men selve dokumentinnhentingen ville nok bli raskere med et C program.
- Telle opp antall lenker som peker på URLene vi treffer på. Dette er en triviell endring i kildekoden (et par linjer ekstra). Grunnen til at vi ikke har lagt det til, er at vi allerede har en stor database hvor vi ikke har talt lenker.
- Bedre håndtering av HTML IMAGEMAPS. Grunnet en feil i libwww-0.40 som ikke skiller ut imagemap lenker fra andre lenker, forsøker roboten å hente inn slike. Dette gir en feilmelding tilbake

fra serveren om at vi ikke oppga koordinater. Denne blir lagret i dokumentbasen vår.

Prosjektevaluering

Ole Andreas Grytdal
Bjørn Kristian Hilstad

Våre erfaringer fra diplomen.

27.0 Prosjektevaluering

Vi vil i denne delen av rapporten se tilbake på faser av diplomoppgaven og evaluere vår arbeidsmetode.

27.1 Prosjekt organisering

Siden diplomgruppen var på to personer, fordelte vi oppgavene oss i mellom med hvert vårt spesialfelt. En tok seg av roboten og innsamling av data, mens den andre så på MS160 og konstruerte søkemaskinen. Selvom det i utgangspunktet er to frittstående moduler, kommer en ikke vekk fra at man må samarbeide om å få de to modulene til å virke sammen.

Det ble avholdt møter med veiledere for å fortelle om progresjon i oppgaven og for vår del stille spørsmål om ting vi syntes var uklart. Ellers hadde vi kontakt via e-post som vel var den raskeste måten å få svar på, når vi lurte på noe.

27.2 Faser i diplomoppgaven

27.2.1 Tidsskjema

Tidlig i oppgave fasen satte vi opp et tidsskjema for hele diplomen. Antall belastningstimer som er satt av diplomen i uken er 48 timer. En jevn fordeling fra prosjektstart 5.september til prosjekt slutt 12 januar var målet vårt.

Dette synes vi at vi klarte bra, men mot slutten av prosjektet fikk vi enkelte problemer med koden til søkemaskinen, så vi fikk en ukes utsettelse til 19 januar.

27.2.2 Forstudium

Før vi kunne begynne å skrive forstudie rapporter om MS160 og de forskjellige søkemaskinene, var det nødvendig å samle inn endel relevant informasjon. Informasjon om de forskjellige søkemaskinene kunne vi finne på de forskjellige www-basene, mens data om MS160 leste vi ut av spesifikasjon håndboken. I tillegg måtte fant vi fram til en del annen interessant litteratur om søkealgortimer, informasjons innhenting og hvordan roboter skal oppføre seg på Internet.

Forstudiumet ble brukt til å sette seg grundig inn i oppgaven, slik at det kunne lage et godt og bredt grunnlag for den videre gangen i oppgaven.

27.2.3 Kravspesifikasjon

Etter at vi fikk fordypet oss grundig i bakgrunnstoffet og selve oppgaven, fikk vi satt ned på papiret hva prosjektmålene var. Ut fra disse målene skrev vi ned de krav vi ville stille til vårt system. Vi prøvde å være realistiske med de krav vi satt oss, men vi ville også gjerne vise hvilke muligheter systemet vårt kunne ha. Derfor ble noe av det som ble beskrevet i kravspesifikasjonen ikke implementert pga tidsnød. Dette kan selvfølgelig være nyttig informasjon for andre som skal gjøre gjenbruk av vår oppgave.

27.2.4 Konstruksjonsspesifikasjon

I konstruksjonsfasen ble de ideer vi hadde funnet, satt ut i live ved å konstruere de algoritmer og strukturer vi fant nødvendig. Konstruksjonsfasen ble nært knyttet opp mot implementeringen, da vi ofte testet ut om de rutiner vi lagde virkelig var gjennomførbare.

27.2.5 Implementering

Det var under implementeringen at tiden virkelig begynte å fly. Vi fikk enkelte problemer med søkemaskinen, og vi måtte gå bort fra noen av algoritmene vi konstruerte og heller legge inn andre måter å løse problemet på.

27.3 IDT som arbeidsplass

Under jobbing med diplomoppgaven fikk vi tildelt et eget kontor, der vi fikk satt inn PCen som hadde MS160 kort og koblet den på nett. I tillegg sto det en UNIX terminal på kontoret slik at vi fikk jobbe på hver vår maskin.

Vi må nesten takke drift på for at vi fikk være slike “disk-griser” når det gjaldt innsamling av data. Vi fikk ikke montert opp den eksterne harddisken, som var koblet på PCen, til IDTs vanlige maskiner, så vi måtte kopiere ned de data som trengtes etterhvert.

Konklusjon

Bjørn Hilstad og Ole Andreas Grytdal

“Og alle var enige om det hadde vært en fin tur.....”

28.0 Konklusjon

Under prosjektets gang har vi støtt på mange vanskeligheter. En del har vi lært av og løst, andre fant vi ikke ut av. Det viste seg at programmeringsarbeidet tok mye mer tid enn vi hadde håpet på. Dokumentasjonsarbeidet har stort sett gått greit, men det har vært litt hektisk i perioder.
Vårt opprinnelige mål med oppgaven var å lage en løsning som kanskje kunne brukes i praksis ved prosjektets slutt. Vi fikk ikke implementert alle søkemulighetene som vi hadde håpet, men vi mener allikevel at resultatet er et nyttig verktøy som kan brukes til søking. Datainnsamlingsverktøyet fungerer meget bra, men vi kunne ha ønsket oss at det var litt raskere (ca. 4000 dokumenter om dagen slik det er nå).

Vi har lært mye om mange ting i løpet av tiden vi har jobbet med oppgaven. Vi har lært nye ting om WWW, HTTP protokollen CGI og ikke minst MS160-brikken. I tillegg har vi fått mer erfaring i programmering i Perl og C.
Når det gjelder litteratur, har vi lest svært mye om søking på WWW og søkemetoder generelt. Vi har også sett litt på nye systemer som kanskje vil overta for dagens WWW-standarder om få år. Vi føler at det har gitt oss litt innsikt i hva fremtiden vil bringe av distribuerte informasjonssystemer.

Vi føler fremdeles at det er mye som kan gjøres med MS160-brikken innenfor søking på WWW. Vi oppdaget en del begrensninger, men mange av disse kan omgås ved å legge dataene bedre til rette. Siden HTML er et svært strukturbasert språk, hadde det vært ønskelig å kunne søke på bestemte elementer i strukturen. Dette er ikke uten videre mulig med MS160-brikken, men dersom man på forhånd kan skille ut strukturelementene i separate deler er det fullt mulig.
MS160-brikkens største begrensning i denne sammenhengen vil være lagringskapasiteten. Vi mener at brikken vil kunne gjøre stor nytte for seg i et ALIWEB[16] lignende system. Der vil datamengden være veldig redusert fordi systemet bare inneholer summariske oversikter over dokumenter. I slike systemer vil nok MS160-brikken kunne brukes til søking på WWW med like stor suksess som den benyttes i ftpsearch systemet.

Referanseliste

Ole Andreas Grytdal
Bjørn Kristian Hilstad

[1] - Growth of Internet
http://www.mit.edu:8001/aft/sipb/user/mkgray/ht/web-growth.html

[2] - Hughes : Entering the World-Wide Web: A guide to Cyberspace

[3] - Archie, Bunyip Homepage
http://www.bunyip.com

[4] - Scout Project Home
http://fuzine.mt.cs.cmu.edu/scout/home.html

[5] - American Online
http://www.aol.com

[6] Salton G: Automatic text processing: The Transformation, analysis and retrival of information by computer

[7] Comren, T : Introduction to Algorithms, p470

[8] Web Servers Survey
http://www.proper.com/www/servers_survey.html

[9] Byte Magazine, November 1995 p59-64

[10] Hyper-G Homepage
http://hyperg.ivcm.tu-graz.ac.at/hyperg;sk=D88264AA

[11] MS160 Search Engine User Manual

[12] Fundamental Software for Fuzzy Information Retrieval Based on New Hardware
Eirik Stephansen og Kjell-Sverre Jerijaervi

[13] Lycos Homepage
http://www.lycos.com/whatsnew.html

[14] Apaches HomePage
http://www.apache.org

[15] FTPsearch
http:/www.ftpsearch.no

[16] AliWEB
http://web.nexor.co.uk/mak/doc/aliweb/paper/paper.html