ANVENDT INFORMATIKK

OPPTRYKK AV STUDENTARBEIDER VED
HØGSKOLEN I MOLDE
INSTITUTT FOR INFORMATIKK

DESEMBER 2004
NUMMER 1

REDAKSJON:
JUDITH MOLKA-DANIELSEN
HALVARD ARNTZEN

INNHOLD

Anmerkninger fra redaktørene:

Innholdet i dette notatet består av fire utvalgte studentprosjekter ved institutt for informatikk i studieåret 2004. Det er lærerne på de enkelte kurs som har stått for utvelgelsen. De utvalgte arbeidene har ikke vært rettet eller modifisert av redaksjonen og dermed er innholdet presentert i original form slik de ble levert av studentene.

Prosjektene er ordnet etter hvilket nivå kursene ligger på, fra 1. semester i bachelorstudiet til masternivå. Prosjekt 1 handler om filsikring og apache webserver. I prosjekt 2 sammenlignes læringsmetodikk i utdanningen i programmering med læringsmetodikk brukt for utdanning av flygeledere. Prosjekt 3 beskriver en databaseapplikasjon for å håndtere bestilling av flybilletter. Temaet i prosjekt 4 er metoder for å studere topologiske egenskaper i store nettverk.

1. IN115: Filsikring i en aktuell produksjonskontekst av Jan Arne Janssen

«Filsikring i en aktuell produksjonskontekst»
In 115 – Nettsystemer – Øving 4

Jan Arne Janssen

Introduksjon

I denne oppgaven skal vi se på hvordan vi kan sette filsikring i en aktuell produksjonskontekst. Til dette skal vi bruke Apache webserver. Vi kommer til å lære hvordan en starter apache webserver, og hvordan en kan sjekke om serveren er oppe og går. Vi kommer til å se nærmere på prosesser som har sammenheng med webserveren, og hvilke porter som blir åpnet når vi starter serveren. Vi skal prøve å finne filene som prosessene til webserveren bruker, og se litt på metainformasjonen til disse. Vi skal se litt på statisk materiale som blir vist i weblesere, og hvilke rettigheter httpd må ha for å kunne lese filer med statisk materiale. Dynamisk materiale er noe som ofte blir brukt i sammenheng med websider. Vi skal se litt på hva som skjer når webleseren prøver å få fram dynamisk materiale. Til sist skal vi se litt på hvordan vi kan konfigurere serveren, og se litt på feil som kan skje og hvordan vi kan finne disse.

Apache Webserver

Apache webserver er en av de mest populære webserverne å bruke i Unix/Linux miljø. Nesten alle Linux distribusjoner kommer med en installasjonspakke av Apache. Denne pakken blir sjelden installert automatisk, du må som regel når du installerer Linux merke av at du vil ha denne pakken installert. Du kan også på et senere tidspunkt velge å installere denne pakken hvis du skulle ha bruk for den.
Har du Apache installert, så er det bare å starte webserveren. Men for å kunne gjøre dette må du vite hvilke kommandoer du kan bruke. Ved å skrive Apachectl --help i kommandopromt, får vi opp en liste over en del kommandoer vi kan bruke. De to viktigste er **apachectl** *start* og **apchectl** *stop*, som starter og stopper serveren. Vi skriver inn start kommandoen, og får bekreftet at serveren har blitt startet.

```
Insurrectica2:/# apachectl start
[Tue Oct  5 12:33:55 2004] [alert] apache: Could not determine the server's fully
qualified domain name, using 127.0.0.1 for ServerName
/usr/sbin/apachectl start: httpd started
```

For å sjekke at webserveren går kan vi åpne en webleser og skrive inn URL http://*host*. Da vil vi få opp index-siden som ligger i mappen /var/www. Vi kan og sjekke dette ved å skrive kommandoen **netstat** *-an|grep tcp*. Vi vil da få opp en liste over TCP sokler, og hvilke porter disse prosessene bruker. Ut ifra listen kan vi se en prosess som bruker port 80, dette er porten som webserveren bruker. Skriver vi **netstat** *-a|grep tcp* vil vi få et litt mer forståelig resultat hvis du ikke er så kjent med hvilke prosesser som bruker hvilken port. Da vil portnummeret bli byttet ut med navnet til prosessen. I dette tilfellet vil port 80 bli byttet ut med *www*, og de fleste vil forstå at dette har med internett å gjøre.

```
Insurrectica2:~# netstat -an|grep tcp
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:427           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1013            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:631             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:2654            0.0.0.0:*               LISTEN
```

Prosesser

Videre skal vi se på hvilke prosesser som arbeider og utfører serverarbeidet. Vi skal finne ut hvilke TCP porter som er åpnet og hvilken tilstand disse er i.
Før og etter vi har startet webserveren, kan vi bruke kommandoen **ps** -Af. Da vil vi få opp en lang liste over alle prosesser som foregår på maskinen. Vi kan sammenligne listen før og etter vi har startet webserveren, og finne ut hvilke prosesser som starter i sammenheng med serveren. Vi kan se ut ifra resultatene at det er seks prosesser som startes i sammenheng med webserveren. I listen kan vi og se hvilken UID prosessene har. I vårt tilfelle kan vi se at den første prosessen har UID «Root» og de andre har «www-data» som UID.

```
UID        PID  PPID  C STIME TTY          TIME CMD
root      1681     1  0 17:22 ?        00:00:00 /usr/sbin/apache
www-data  1682  1681  0 17:22 ?        00:00:00 /usr/sbin/apache
www-data  1683  1681  0 17:22 ?        00:00:00 /usr/sbin/apache
www-data  1684  1681  0 17:22 ?        00:00:00 /usr/sbin/apache
www-data  1685  1681  0 17:22 ?        00:00:00 /usr/sbin/apache
www-data  1686  1681  0 17:22 ?        00:00:00 /usr/sbin/apache
```

For å finne hvilke TCP porter som blir åpnet i sammenheng med webserveren, bruker vi kommandoen netstat -an|grep tcp. Da vil vi få opp samme resultat som i innledningen, og vi kan se at port 80 har blitt åpnet, som er standard port til http. Vi kan og ut i fra den listen se at tilstand til port 80 er «LISTEN». Dette betyr at prosessen bare ligger og venter på at en annen prosess skal kalle på den. Dette vil som regel skje når vi skal hente fram en webside i en «browser».
Stopper vi webserveren og sjekker prosess-statusen og statusen på TCP portene, kan vi bekrefte at alle prosessene og portene som har sammenheng med serveren blir lukket.

Filer

Filer er det som driver prosessene, uten disse kunne ikke webserveren fungere. Disse filene ligger en plass i filsystemet, problemet kan være å vite hvor dette er. En kan enten lese dokumentasjonen som følger med Apache, der en helt sikkert finner ut hvor serverens «rot» er lagt. Har en lest litt generell dokumentasjon om Linux, vil en vite at programfiler blir lagt i */etc* katalogen. Går en inn i denne katalogen vil en finne en lang liste med kataloger, blant annet en katalog som heter *apache*. Går en inn her ligger filene som webserveren bruker. Stopper vi webserveren, kan vi se at filene enda ligger der og ikke forsvinner når vi stopper programmet. Ved å bruke kommandoen ls -li vil vi få opp en detaljert liste over filene som ligger der med metainformasjon.

```
Insurrectica2:/etc/apache# ls -ls
total 45
   Insurrectica2:/etc/apache# ls -ls
total 45
  29148 -rw-r--r--    1 root     root          285 2004-10-01 22:09 access.conf
  29147 -rw-r--r--    1 root     root        35086 2004-10-01 22:10 httpd.conf
  29139 lrwxrwxrwx    1 root     root           15 2004-10-01 22:09 mime.types ->
/etc/mime.types
  29149 -rw-r--r--    1 root     root          297 2004-10-01 22:10 srm.conf
```

Leserettigheter og Statisk materiale

Vi skal nå verifisere at httpd trenger leserettigheter til de html filene han skal lese. Httpd prosessen blir forøvrig på mitt system vist som en av mange «apache» prosesser, så jeg får ikke opp «httpd» i prosess statusen. Index html filen på mitt system finner en i katalogen var/www, og det er denne filen som blir hentet når en skriver http://*host* i webleseren. For å sjekke om httpd må ha leserettighet, bruker vi denne index filen, og fjerner rettigheter fra den.
Jeg ble litt overrasket over resultatet jeg fikk etter å ha gjennomført testen. Jeg startet med å fjerne leserettighetene til meg selv, og prøvde å åpne filen i webleseren. Dette gikk til min store forundring helt uten problem. Etter å ha testet litt mer med å fjerne og legge til rettigheter, kom jeg fram til at den eneste rettigheten som har noe å si for om filen kan bli lest av webleseren er leserrettigheten til «andre» brukere. Leserettigheten til meg selv eller andre i gruppen har ingenting å si. Jeg vet ikke hvorfor dette er slik, men httpd krever ihvertfall at det er leserettighet hos denne gruppen. Dette kan sikkert ha noe å gjøre med at alle som skal lese html filen gjennom en webleser, er andre brukere på nettet som ikke er registrert på mitt system. Httpd har ikke protokoller innebygd for å sjekke hvem som eier filen og hvilke i min gruppe som har rettighet til å lese den. Httpd konsentrerer seg bare om «andre» har rettigheter, og hvis filen har denne rettigheten betyr det at absolutt alle får lov å lese denne filen.
For hver gang jeg hadde fjernet leserettigheten til «andre» kom det opp en feilmelding i webleseren som sa følgende *«Forbidden - You don't have permission to access /index.html on this server».*

Videre testet jeg om httpd måtte ha «x» rettighet til «www» mappen som html filen ligger. Etter hva jeg fant ut om leserettighetene til html filen, tenkte jeg at det kanskje kunne være noe lignende her. Etter å fjernet «x» rettighetene fra de 3 forskjellige gruppene, fant jeg ut at her fungerer alt slik som jeg var vant til. Fjernet jeg «x» rettigheten til meg selv fikk jeg bare den statiske feilmeldingen. Fjernet jeg «x» rettighetene til alle andre en meg selv, fikk jeg lese filen uten problem. Sannsynligvis vil ingen som prøver å lese html filen fra internett få tilgang. Siden mest sannsynlig alle skal lese html filen, så har «www» mappen «x» rettigheter på alle gruppene som standard.

Vi lager oss en ny undermappe som heter «um», og kan nesten verifisere at det ikke ligger noe der. Det eneste som ligger der, er to skjulte filer «.» og «..». Begge disse filene blir opprettet i alle mapper. «.» filen er bare en referanse til selve mappen som den ligger i. «..» filen inneholder referanser til mappen over, slik at du kan komme deg tilbake til forrige mappe. Denne filen blir brukt av **cd** kommandoen når du skal gå tilbake til forrige mappe. En skriver bare **cd ..** så vil du gå en mappe tilbake. Skriver du http://host/um, slik at du kommer inn i «um» mappen, vil du se et «arkiv» av det som ligger i denne mappen. Dette skjer fordi det ikke ligger noe index html fil her. Det eneste som ligger der som vi kan se, er en peker til bake til mappen som «um» ligger i, i realiteten er dette «..» filen som vi ser.
Vi prøver å fjerne «r» og «x» rettigheter fra mappen for å finne ut hva som er minimum for at vi skal kunne se innholdet i mappen. Vi finner ut at vi bare trenger «r» og «x» rettighet på «andre» grupper for å kunne se innholdet av mappen gjennom webleseren.

Vi lager en ny html fil i mappen «um» med navn «a.html», og prøver å finne ut hva som er minst nødvendige rettighet for at en webleser skal kunne se denne siden. Vi fjerner både «x» og «r» rettigheter, og finner ut det samme som vi gjør med index filen i «www» mappen. Vi trenger bare leserettighet på «andre» brukere for at en skal kunne se denne filen gjennom en webleser.

Mv er en kommando som en bruker hvis en vil flytte en fil til en annen mappe, eller hvis en vil gi filen nytt navn. Vi skal nå gi «a.html» navnet «index.html», dette fører til at vi ikke trenger å oppgi filnavnet når vi skal skrive en url i en webleser. For eksempel trenger vi bare oppgi http://host/um, så vil index filen bli vist automatisk i webleseren. Vi skriver kommandoen **mv** *a.html index.html* for å endre navnet på filen. Gjennomfører vi en **ls** kommando før og etter vi endret navnet på filen, kan vi se at inode-nummeret er det samme. Filene som vi ser som a.html eller index.html, er bare pekere til de virkelige filene. Inode-nummeret forteller hvilken fil pekeren tilhører. Når vi bruker **mv** kommandoen forandrer vi bare navnet på pekeren og ikke den egentlige filen, derfor vil inode-nummeret hele tiden være det samme. Hvis vi sletter pekeren som vi ser, vil vi bare slette denne pekeren og ikke selve filen. Derfor vil det og være mulig å hente frem igjen filer som vi angivelig har slettet.

```
Insurrectica2:/# cd var/www/um
Insurrectica2:/var/www/um# ls -li
total 4
  55475 -rw-rw--w-    1 root     root           77 2004-10-06 13:48 a.html
Insurrectica2:/var/www/um# mv a.html index.html
Insurrectica2:/var/www/um# ls -li
total 4
  55475 -rw-rw--w-    1 root     root           77 2004-10-06 13:48 index.html
```

Dynamisk materiale

Dynamisk materiale gjør at resultat som vi får opp i webleseren, kan være forskjellig fra gang til gang. Som regel må en kjøre et script som gjennomfører en del kommandoer som henter fram og viser det dynamiske materialet. Disse scriptene må kjøres gjennom en «interpreter» for at de skal fungere. Når vi skal hente frem dynamiske sider gjennom en webleser, vil httpd starte en «interpreter» som leser scriptet. Hvis det er cgi-script som skal kjøres, blir «bash» startet som «interpreter».
I utgangspunktet må bash ha «r» og «x» rettigheter på script filen for å kunne kjøre den. Men jeg kunne ikke bekrefte dette fordi på systemet mitt var det ikke noe cgi-bin mappe med «printevn» filen. Jeg søkte rundt på systemet og fant en fil med dette navnet, som jeg er nesten sikker på er den samme filen. Jeg kopierte den filen over i «www» mappen for å sjekke om jeg kunne kjøre den i en webleser. Men når jeg prøvde å kjøre denne filen ved å skrive http://host/printevn i webleseren, kom det bare opp melding om jeg ville laste den ned. Jeg sjekket da om alle rettighetene i filen var som de skulle, men alt var greit her. Jeg prøvde igjen, og fikk opp samme meldingen om jeg ville laste ned filen. Jeg tok da og konfigurerte webleseren til at den skulle lese denne filen. Når jeg prøvde igjen fikk jeg bare opp fem uleselige tegn. Jeg kunne derfor ikke bekrefte eller avkrefte noe her som hadde med rettighetene til script filen å gjøre.

Konfigurasjon av serveren

Hvis vi vil konfigurer serveren, kan vi gjøre dette gjennom konfigurasjonsfilen til webserveren. I denne filen finner vi root katalogen til serveren, og den heter httpd.conf. Åpner vi denne filen, vil vi få opp et stort tekstdokument med mye informasjon. Det kan være vanskelig å forstå så mye av det som står der, og finne ut hva du skal forandre. Måten denne filen fungerer er at alle linjer som starter med # vil ikke bli kjørt av programmet. Alle linjene i dokumentet som har # foran seg, er som regel

linjer som forklarer hva du skal gjøre for å forandre konfigurasjoner. Alle linjene som ikke har #, vil bli kjørt av programmet, og det er her du går inn og gjør forandringene.
Vi skal nå først prøve å finne linjer i konfigurasjonsfilen som sier noe om index.html filen og at denne blir vist bare ved å oppgi stien til filen. Til dette bruker vi programmet «grep» som gjør at vi kan søke etter ord i en tekst. Måten vi bruker kommandoen på er å skrive **grep** *-n index.html httpd.conf.* -n gir oss hvilken linje i dokumentet vi finner det vi søker på. Ved å bruke kommandoen, ser vi at vi finner det vi søker etter på linje 397.
Vi prøver så å finne hvor dynamiske sider skal legges, ved å søke etter «cgi». Vi finner en del linjer der dette står om cgi, men vi klarer å finne ut at vi skal legge dynamiske sider i følgende katalog
`/usr/lib/cgi-bin/`
Vi prøver og å finne ut hvor vi kan endre port nummeret som serveren skal lytte på. Vi søker etter «Port» og finner ut at vi kan forandre porten på linje 274.

```
Insurrectica2:/etc/apache# grep -n index.html httpd.conf
397:    DirectoryIndex index.html index.htm index.shtml index.cgi
Insurrectica2:/etc/apache#

Insurrectica2:/etc/apache# grep -n cgi httpd.conf
215:LoadModule cgi_module /usr/lib/apache/1.3/mod_cgi.so
397:    DirectoryIndex index.html index.htm index.shtml index.cgi
580:ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
583:# "/usr/lib/cgi-bin" could be changed to whatever your ScriptAliased
586:<Directory /usr/lib/cgi-bin/>
784:    #AddHandler cgi-script .cgi .sh .pl
819:# Format: Action media/type /cgi-script/location
820:# Format: Action handler-name /cgi-script/location
847:#ErrorDocument 404 /cgi-bin/missing_handler.pl
946:# support/phf_abuse_log.cgi.
948:#<Location /cgi-bin/phf*>
950:#    ErrorDocument 403 http://phf.apache.org/phf_abuse_log.cgi

Insurrectica2:/etc/apache# grep -n Port httpd.conf
271:# Port: The port to which the standalone server listens. For
274:Port 80
434:# Port to form a "canonical" name.  With this setting off, Apache will
718:    # Italian (it) - Portugese (pt) - Luxembourgeois (lb)
720:    # Polish (pl) - Brazilian Portuguese (pt-br) - Japanese (ja)
```

Åpner vi denne filen i en teksteditor kan vi endre innholdet. Etter å ha testet litt med rettigheter, er den minste rettigheten vi trenger for å kunne endre og lagre endringer, «w» rettighet. Vi testet og hvilke rettigheter httpd må ha for å lese httpd.conf, og vi fant til vår store forundring at den trengte ingen rettigheter. Vi stoppet serveren, fjernet alle rettigheter fra httpd.conf, og startet serveren igjen uten problem. For å være sikker på at serveren fungerte, prøvde vi å komme oss inn på http://host med webleseren, dette gikk uten problem.
Før vi fjernet alle rettighetene fra filen, tok vi og og endret porten som serveren skal lytte på. Vi endret porten fra 80 til 100. Etter vi hadde sjekket om httpd klarte å lese filen, tok vi og sjekket om porten hadde endret seg. Vi brukte kommandoen netstat -a|grep tcp for å få frem alle tcp porten. Vi kunne bekrefte at serveren nå lyttet på port 100.

```
Insurrectica2:/etc/apache# chmod -rwx httpd.conf
Insurrectica2:/etc/apache# ls -l httpd.conf
----------    1 root     root        35087 2004-10-06 19:52 httpd.conf
Insurrectica2:/etc/apache# apachectl start
[Wed Oct  6 20:05:45 2004] [alert] apache: Could not determine the server's fully
qualified domain name, using 127.0.0.1 for ServerName
/usr/sbin/apachectl start: httpd started

Insurrectica2:/etc/apache# netstat -a|grep tcp
tcp        0      0 *:100                   *:*                     LISTEN
tcp        0      0 localhost:427           *:*                     LISTEN
tcp        0      0 *:netbios-ssn           *:*                     LISTEN
tcp        0      0 *:sunrpc                *:*                     LISTEN
tcp        0      0 *:ftp                   *:*                     LISTEN
tcp        0      0 *:1013                  *:*                     LISTEN
tcp        0      0 *:ipp                   *:*                     LISTEN
tcp        0      0 *:microsoft-ds          *:*                     LISTEN
tcp        0      0 *:2654                  *:*                     LISTEN
```

Feilsøking

Når vi skal sette opp og drifte en webserver kan det ofte oppstå problemer som gjør at en webleser ikke vil få tilgang materialet som ligger på serveren. Somme ganger kan problemet ligge i konfigurasjonen av serveren, som kan være vanskelig å finne. Andre ganger kan problemet ligge i aksessrettighetene til filer, dette er som oftest lettere å finne siden webleseren gir feilmelding om han ikke får tilgang til en fil.
Jeg er ikke i Molde når jeg gjør denne oppgaven, så jeg kan ikke teste andre eller la andre teste meg ved å introdusere feil i hvordan serveren fungerer. Jeg kommer i stedet til å snakke litt om forskjellige ting en kan gjøre for å introdusere feil. Jeg tar og å snakker litt om hvordan en kan finne feil, og hvordan en kan gå frem ved feilsøking.
Som en start ville jeg ha introdusert enkle feil som ikke er så vanskelig å finne. Dette kan foreksempel være å endre aksessrettigheter på filer eller mapper som webserveren skal hente frem, eller å endre navn på filer.
Feil som kan være vanskeligere å finne er hvis du starter å forandre ting i konfigurasjonsfilen til serveren. Slike feil kan være å endre porten som serveren skal lytte på, eller endre hvilken fil som skal vises når bare stigen til filen blir oppgitt. Skal en gjøre det enda vanskeligere kan man starte å endre på konfigurasjoner som sier hvor serveren finner filer som gjør at den kan starte eller fungere skikkelig.

For å finne feil er det lurt å gå strukturert frem når en leter. Det kan være lurt å ha en sjekkliste hvor en merker av hva en har gjort. På denne måten er du sikker på at du ikke hopper over noe, eller gjør noe på nytt igjen.
Det første en må se etter når et problem oppstår, er å se om vi får en feilmelding. Feilmeldinger gir oss ofte verdifull informasjon om hva som kan være galt. Er for eksempel aksessrettighetene til en fil feil, vil som regel webleseren gi melding om at vi ikke har rettighet til å aksessere denne ressursen. Da kan vi starte med å sjekke aksessrettighetene til filer og mapper.
Endrer vi navn eller flytter på mapper og filer vil webleseren gi melding om at den ikke finner ressursene. Når vi får en slik melding kan feilen ligge mange plasser. Det er alltid lurt å sjekke om vi får tilgang på andre ressurser eller om vi ikke får tilgang til noe. Hvis for eksempel serveren ikke er startet vil vi få feilmelding om at webleseren ikke finner ressursen, og den vil så klart ikke finne noe

annet heller. Derfor kan det være lurt å sjekke enkle ting først, slik som om serveren er oppe og går. Videre kan du sjekke om filer og mapper ligger der de skal, og heter det de skal.
Endrer en portnummeret som serveren skal lytte på, vil vi ikke få opp noen feilmelding i webleseren. Vist vi derfor ikke får opp noen side eller feilmelding i webleseren, kan det være lurt å sjekke hvilken port serveren lytter på.
Gjør vi noe med filer som serveren trenger for å kjøre, vil vi ganske ofte få opp en feilmelding når vi skal starte serveren. Denne feilmeldingen sier som oftest hvilken fil som det er problemer med.
Det som kan være mest vanskelig å finne, er hvis en endrer en del verdier i konfigurasjonsfilen som serveren ikke oppdager som feil. Å forandre porten den skal lytte på er en slik ting. Er det avanserte funksjoner som blir endret, må en som regel ha litt kunnskap om hva disse funksjonene skal være for at en skal finne feilene.

Oppsummering

Vi har nå lært litt om å bruke apache webserver, og hvilke prosesser og filer den bruker for å fungere. Vi har og sett litt på hvordan serveren presenterer statisk materiale og feilmeldinger i en webleser. Vi har og sett litt på hva som skjer når en bruker dynamisk materiale på en webside.
Konfigurasjon av webserveren er en viktig del av å få serveren til å fungere som den skal. Dette har vi sett og testet litt ut. På slutten så vi litt på hvilke feil som kan oppstå med en webserver. Det viktigste vi så på i sammenheng med feil, er hvordan vi kan finne disse feilene og rette på dem.

AV STINE JOHNSEN STRANDE

FLYKONTROLLERING VS PROGRAMMERING

IN209 Fagdidaktikk Høsten 2004

Innholdsfortegnelse:

PROBLEMSTILLING

Jeg vil undersøke om vi kunne brukt det flygeledere lærer seg for å lære aspiranter å kontrollere fly, til å lære nybegynnere å programmere.
Jeg vil holde fokus rundt veiledningssituasjonen. Kan det de som skal veilede flygelederaspiranter være relevant for en som skal være veileder i programmering?

INNLEDNING

Et ordtak sier at "vi lærer så lenge vi lever". Vi lærer av egne erfaringer og av andre rundt oss hele tiden. Samtidig lærer andre av oss. Bevisst eller ubevisst.
Mye går av seg selv, men når det blir noe formelt rundt situasjonen, kreves det nøye gjennomtanke og gjerne skolering av de som skal lære oss noe. Men er det egentlig noen vesentlig forskjell i måten vi lærer noe bort på? Er måten flygeledere lærer seg å skolere sine aspiranter på like aktuell som måten vi kan lære en nybegynner å programmere? Kan vi bruke de samme læringsmetodene for begge læringssituasjonene? –Er det å undervise universelt?

Aller først, vil jeg fortelle litt grunnleggende om hva vi legger i begrepet veiledning.

Så vil jeg fortsette ved å fortelle litt om OJT-kurset til flygelederne, kurset de skoleres i for å lære seg å lære bort. Deretter vil jeg forsøke å fortelle litt om programmering, også her hovedsakelig knyttet til læring og undervisning.

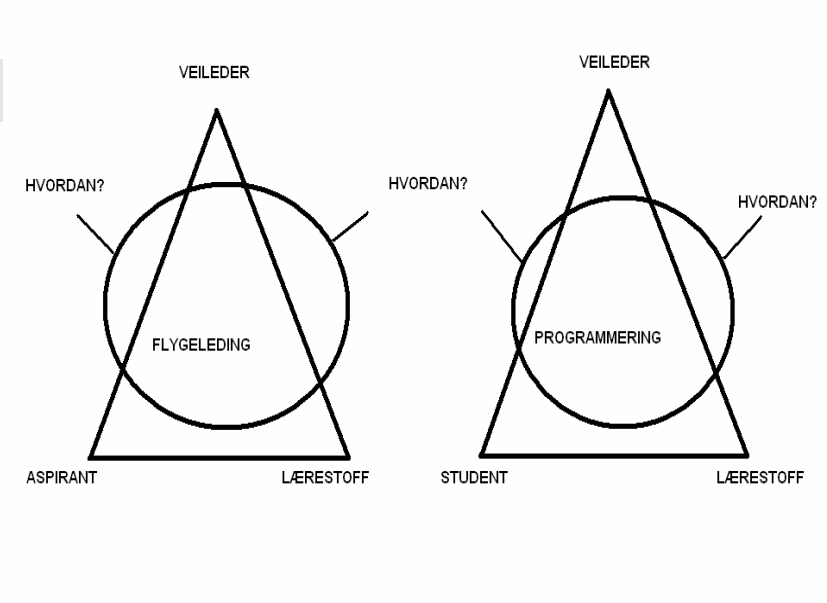
Jeg skal ikke lage noe konklusjon rundt problemstillingen om det å lære er universelt eller ikke, men jeg vil ta for meg et punkt i OJT-kurset for flygeledere, og sammenligne denne med undervisningen hjelpelærere har i programmeringskurs. Fokuset vil her være på læresituasjonen, ikke på *hva* som læres, men *hvordan*. Fokuset blir heller ikke lagt på hvordan man kan arbeide for å lære seg noe, mer på hva man skal gjøre for å lære bort. Er problemstillingene flygelederne møter like aktuelle for en veileder i programmering?

Etter dette vil jeg gi en generell sammenligning av OJT-kurset til flygelederne med in150 – introduksjon til programmering. Her vil jeg også forsøke å komme inn på en del grunnleggende likheter og forskjeller i fagområdene. Jeg vi hele veien ta utgangspunkt i flygelederopplæringen, og sammenligner denne med å lære bort programmering, ikke omvendt.

HOVEDDEL

LITT OM VEILEDNING GENERELT

Det hevdes at veiledning består av relasjonen mellom veilederen og den som veiledes, det de snakker om, og måten de snakker om det på. Dette forsøker jeg å vise under, med en relasjonstrekant.



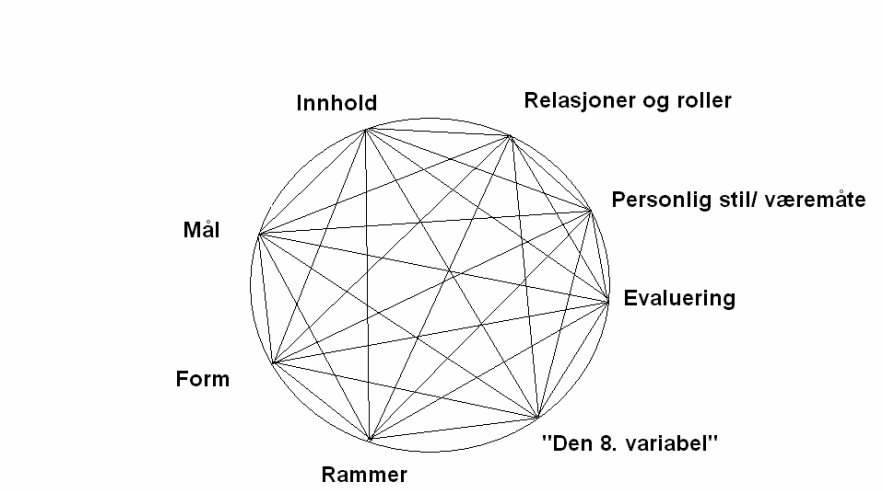
Figur 1. Måten jeg oppfatter samhandlingen mellom student, veileder og læremateriale

For at veiledningssituasjonen skal bli best mulig, er ulike kommunikasjonsferdigheter og metodiske fremgangsmåter nyttige. En definisjon jeg har funnet om veiledning er som følger:

> *"Veiledning er en utviklingsorientert samtale der formålet er å bidra til refleksjon og ettertanke og gi hjelp til å gjøre kvalifiserte valg som vil kunne føre til endring og forbedring.*
> *Det karakteristiske ved samtalen er at den gjennomføres i en atmosfære av gjensidig tillit og er preget av støtte og utfordring"*
>
> - Hentet fra boken Veiledningsmetodikk, P. Mathisen og R. Høigaard.

I den samme boken legges det vekt på at veiledning er svært kontekstavhengig. Det finnes ikke noe perfekt steg-for-steg-løsning på hvordan en perfekt veiledningssituasjon er, for en rekke forhold vil til enhver tid påvirke situasjonen på forskjellige måter. Konteksten endres raskt.

Veiledningen vil gjerne bestå av åtte deler. Disse er nært knyttet sammen, og vil være i kontinuerlig forandring. Disse åtte delene er vist i figuren på neste side:



Figur 2. De åtte komponentene i veiledningen henger sammen og står i et gjensidig avhengighetsforhold til hverandre. Tatt fra boken "Veiledningsmetodikk" av P. Mathisen og R. Høigaard

Forklaring til figuren:

Noe av forklaringen til denne figuren, er direkte hentet fra boken figuren er hentet fra.

Formen på veiledningen er de metoder og strategier en velger å benytte. Noen har strenge rutiner og bestemmelser for hvordan en situasjon skal være, mens andre "tar det som det kommer". Uansett form, vil kunnskapene til veileder være det som begrenser mest.

Målet for veiledningen kan være klart definert, eller utvikles som et resultat av prosessen. De fleste lærebøker jeg har sett tidligere, er svært fokusert på viktigheten av å ha ett mål for det man gjør. Jeg vil tro dette også er gjeldende i en veiledningssituasjon.

Innholdet kan være valgt av veileder, eller det kan være noe den som veiledes tar med seg i samtalen. Dette kan være et problem, en utfordring, noe som har skjedd, suksesser som skal videreføres, noe som skal skje eller som kan komme til å skje. Det kan handle om fag, saker, prosjekter, ideer, løsninger, karriere, personer eller relasjoner.

Evaluering av veiledning kan skje fortløpende underveis i en veiledningssituasjon, eller legges som oppsummering på slutten. Det er viktig å se evalueringen i forbindelse med målene en har satt for veiledningen. Det er også viktig at veilederen evaluerer seg selv, både underveis og etterpå.

Rammer. Veiledningen kan foregå innenfor indre og ytre rammer. De ytre rammene kan være ro, nok tid, økonomi, sted og lignende. De indre rammene vil være uttrykk for selve samtaleforløpet.

Relasjonen mellom veileder og den veiledede er viktig. Denne kan være preget av trygghet, tillit eller mistillit eller spenning. Holdningen til hverandre kan være likeverdige eller ovenfra-og-ned. Relasjonen kan være positiv eller negativ, man vil selvfølgelig strebe etter å ha en positiv relasjon for å ha en best mulig veiledningssituasjon. Likevel kan relasjonen være asymmetrisk, ved at veilederen er en overordnet, ved at veileder er eldre, har mer kompetanse osv. Dette vil imidlertid ikke bestemme om relasjonen blir positiv eller negativ, det har mer med *holdninger* å gjøre. Veileder vil gjerne ha en form for makt overfor den som veiledes, og det er viktig at veileder håndterer sin makt på en etisk forsvarlig måte.

Roller. Både den veiledede og veilederen vil ha roller i relasjonen. Rollen en tilegner en annen, trenger ikke være den samme som vedkommende karakteriserer seg selv med. En kan oppfattes som interessert, lærevillig, uinteressert m.m. En kan også spille roller i selve relasjonen med hverandre,

som voksen – barn, voksen – voksen og lignende. Dette karakteriserer måten vi behandler hverandre, og måten vi oppfører oss i relasjon med den andre.

Personlig stil og væremåte veileder har, vil være avgjørende for hvordan veiledningen faktisk blir. Personlige egenskaper vil til enhver tid være sammenvevd med måten veileder utfører veiledningsarbeidet på. Derfor er det viktig at veileder har en viss forståelse for hvem han eller hun er og hvordan denne væremåten vil virke på andre. Veileder bør også ha evne og vilje til å justere væremåten sin til det beste for veiledningssituasjonen og den som veiledes.

Den 8. variabel består av bestanddeler i veiledningen som ikke kan registreres, beskrives eller settes i en bestemt form. Dette vil stadig være faktorer som på ulik måte virker inn og er avgjørende for at veiledningen blir som den blir. Dette punktet vil derfor være "resten" av påvirkningsforhold som ikke er nevnt i de andre syv punktene.

HVORDAN SKOLERES FLYGELEDERNE FOR Å KUNNE TA IMOT ASPIRANTER?

Flygeledere i Norge gjennomgår en utdannelse på ca 3 skoleår. De har et visst pensum de skal gjennom, og testes med eksamener underveis. Dette blir en form for sertifisering. Noen greier å gjøre ferdig utdannelsen på to år, mens andre bruker fire. Det er hele tiden strenge krav og eksamener underveis, som må bestås for å kunne gå videre. Under utdannelsen har de lønn, og det er strenge opptakskrav for å i det hele tatt få lov til å begynne på studiet. Det siste året, er aspirantene (heretter brukt om flygelederstudenter) utplassert på en norsk flyplass, og får praktisk trening der. Dette året må de hele tiden overvåkes av flygeledere. De vil jo kontrollere fly med dyrebar last, og rommet for grove feil eksisterer ikke. Derfor må flygelederne som skal ha ansvar for aspirantene gå gjennom et kurs hvor de skal bli i stand til å lære fra seg flygelederyrket i praksis på en god måte, et OJT-kurs.

Dette kurset er felles for de aller fleste land i Europa, og kalles altså OJT-kurs. OJT står for On Job Training.

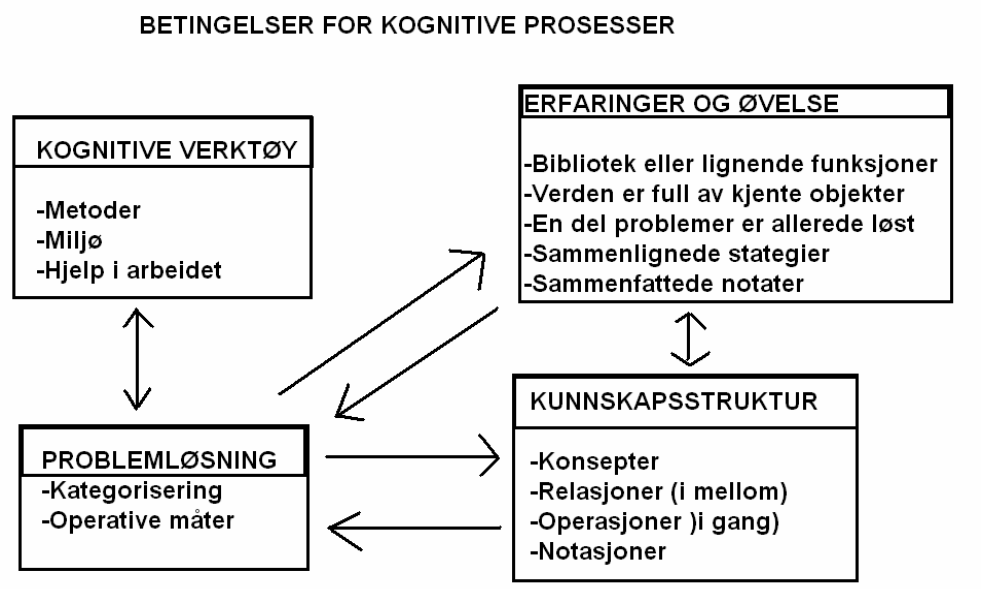
Alle land som tar del i OJT har samme pensum. Dette er en manual på ca 200 engelske sider. Hvert land står så fritt til å lage sitt eget undervisningsopplegg med utgangspunkt i dette.
I Norge har de som underviser i dette kurset sammen laget dette til et toukers kurs. Dette er delt opp i en teoretisk del, og en del der de simulerer forskjellige situasjoner som kan oppstå mellom flygeleder og aspirant. Jeg har her kun konsentrert meg om det teoretiske. I den teoretiske delen, har de samme underviserne laget elleve powerpointpresentasjoner som skal dekke det viktigste i den nevnte manualen. Jeg vil først kort presentere de elleve presentasjonene:

1. Introduksjon
2. Læring
3. Motivasjon
4. Menneskelig samhandling
5. Kommunikasjon
6. a) Kommunikasjonshinder
   b) Spørsmålsteknikk
7. Briefing
8. Monistorering og Intervenering
9. Debrifing
10. Stress
11. Evaluering og Rapportskriving

HVORDAN SKOLERES HJELPELÆRERE TIL Å LÆRE BORT PROGRAMMERING?

Dette faget – in209 Fagdidaktikk – er et hjelpelærerkurs. Her lærer vi mye relevant, både om retting av oppgaver, om å gi tilbakemeldinger, om hvordan man skal oppføre seg i forhold til en student, og hvordan man skal takle forskjellige problemstillinger, og forskjellige "typer" studenter. Dette er imidlertid et generelt kurs, som er aktuelt for alle hjelpelærere, ikke bare de som skal være hjelpelærere i begynnerkurset i programmering. For meg som ikke er flygeleder, er det vanskelig å fortelle nøyaktig hva en flygeleder gjør på jobb, men jeg vet jo litt mer om hva som skjer i den prosessen det er når man lærer å programmere. Dette vil jeg fortelle litt om, men forutsetter at leser har grunnkunnskaper, og vil derfor ikke forklare alle grunnleggende begreper og henvisninger måten en arbeider på i programmering.
Jeg har samtidig forsøkt å relatere dette til hjelpelæreres arbeid i programmeringskurset in150;

Det å programmere er en kompleks prosess. Når man skal lære å programmere for første gang, handler det på mange måter om å lære seg et helt nytt språk, med en helt ny logikk. Dette er selvsagt svært krevende – både for den som skal lære, og for den som skal lære bort. Prosessen dette er, vil gjerne bestå av mange situasjoner, heller enn undervisningstimer. Veiledere er nødt til å være klar over at hva studentene lærer, avhenger av måten de oppfatter ting på, og alternativer og beslutninger veileder tar under opplæring.
Under ser vi en figur som presenterer et generelt teoretisk rammeverk for å representere kunnskap i programmering. Den består av fire beslektede felt; kunnskapsstrukturen, problemløsning, øvelse og kognitive verktøy. Dette rammeverket representerer det faktum at kunnskap er konstruert og bestående av problemløsning:



Figur 3. Frame for task analysis, hentet fra Psychology of Programming, av J.Rogalski og R. Samurcai

Gjennom prosessen det er å lære seg å programmere, skjer det en utvikling mellom de fire feltene. Nye kunnskaper blir tilegnet, gjennom nye interaksjoner og problemløsninger.

En er nødt til å beherske det å tenke abstrakt for å kunne lage program. Når vi for eksempel tenker på et objekt i en klasse (for eksempel såpe generelt) tenker vi ikke på hvert eneste objekt (hver eneste såpe), men på typen objekt i en abstrakt betyding. Når vi forestiller oss objekt av en klasse tenker vi på det som alle objektene har til felles, ikke på det som er spesielt for hvert eneste objekt. Vi

tenker på egenskaper og ikke på verdiene disse egenskapene har. Dette er abstraksjon. Vi tenker generelt på hvilken type objekt vi har med å gjøre, hvilke egenskaper som er felles, og hva disse objektene kan utføre. Man vil hele tiden forestille seg en gitt situasjon når man programmerer. –En programmerer løser ofte problemer han eller hun selv aldri vil støte på i det "virkelige liv."

SAMMENLIGNING

MONITORERING OG INTERVENERING

En viktig oppgave for flygeledere som skal fungere som veiledere, er å vite når de skal intervenere, og måten de skal monitorere aspiranten på. De er som tidligere nevnt i en virkelig situasjon, med virkelige fly, og virkelige mennesker, og rommet for grove feil er fraværende. –Hvor mye skal veilederne la aspirantene få prøve seg uten at det blir en fare for sikkerheten? Når skal de blande seg inn og hvordan skal de blande seg inn for å både beholde sikkerhet og godt forhold til aspiranten? –Aspiranten vil jo ikke lære noe av å sitte og se på i ett år! Man lærer gjennom å få erfaring, og erfaring innebefatter også å gjøre feil, veileder må bare sørge for at feilene ikke er grove, og at man lærer av disse.

I en veiledningssituasjon i programmering, vil hjelpelærer ofte måtte hjelpe studenten frem til en løsning.. Det kan være vanskelig å senke seg ned nivåmessig. Følelsen av å ville sette seg ned og bare løse problemet mens studenten ser på, vil ofte være reell. –Men vi vil jo selvsagt at studenten skal kunne finne løsningen selv, vi vil bare "dytte" han/hun et stykke på vei Det er derfor viktig at hjelpelærer kan noen "triks" for å heller kunne stille de riktige spørsmålene, så studenten får en følelse av å ha løst problemet selv.

I OJT-kurset, lærer veilederne følgende om monitorering og intervensjon:

a) <u>Monitorering</u>

Det er viktig å være positiv. Det er lettere å være kritisk enn å være konstruktiv, men betydningen av å være konstruktiv er desto større. Man må hele tiden gi aspiranten tilbakemelding når noe gjøres riktig. Dette er viktig for å legge til rette for at aspiranten skal kunne vokse personlig. Om aspiranten etter endt dag ikke har fått tilbakemelding på om noe har gått bra eller mindre bra, vil det skape frustrasjon. Veilederne blir bedt om å stille seg selv følgende spørsmål; "Hva gjøre aspiranten bra?", "Hvorfor fungerer det så godt for henne/han?", "Hvordan skape ytterligere fremgang?", "Hva går ikke så bra, eller hva går galt?", "Hvordan kunne eventuelle feil vært unngått?", "Hva må gjøres annerledes i fremtiden for at dette ikke skal gjenta seg?"

Det er viktig at aspiranten ikke sitter og lurer på hva veilederen gjør og tenker. Det er en stor fordel om veilederen heller hele tiden gjør aspiranten oppmerksom på hva som skjer. Hvis veileder for eksempel tar notater, kan han/hun si at han/hun tar notater, og vil merke seg både positive og negative ting, og at de godt kan gå gjennom det sammen etterpå. Det kan være lurt å vente litt med å ta notater for disassosiering. I tillegg kan det hjelpe og sitte litt unna, så ikke aspiranten stresses opp. Notater anbefales, da dette hjelper til med å kunne gi en mer fokusert tilbakemelding i etterkant. Det har mye å si hvordan veileder plasserer seg i forhold til aspiranten. Det kan være lurt å variere hvor man plasserer stolen – innenfor eller utenfor aspirantens synsfelt. Ved å sitte for nært, økes gjerne aspirantens potensielle stress.

Det er imidlertid viktig å hele tiden være til stede både fysisk og mentalt, for å gi aspiranten trygghet, og selvfølgelig oppfylle sikkerhetskrav. Hvis veileder for eksempel leser avisen, vil aspiranten føle seg avskåret fra å be om råd, og vil gjerne trekke en konklusjon om at veilederen er uinteressert. Man bør også unngå å sitte med overlegen eksaminatormine og unødige og ødeleggende ikke-verbale uttrykk.

Man bør heller konsentrere seg om måter man kan gi anerkjennelse og støtte på, enn å riste på hodet, tappe utålmodig med fingre og lignende.

OJT-kurset gir en del retningslinjer under trening;

- Gi akt på aspirantens evner, både som flygeleder og som teammedlem
- Gi akt på hennes eller hans selvtillit
- Prøv å være rolig uansett hva som skjer.
- Bruk din spørsmålsteknikk til å hjelpe aspiranten å finne egne svar.
- Spør kun når det er lite eller enkel trafikk
- Et spørsmål er en distraksjon og belastning for aspiranten. Et spørsmål kan være ødeleggende og føre til større vanskeligheter.
- Skill mellom nødvendig og unødvendig snakk
- Følg aldri aspirantens tunneltenking slik at du mister bildet
- Husk at du har og skal ha kontrollen hele tiden, slik at du øyeblikkelig kan ta over hvis det blir nødvendig.

b) <u>Intervensjon:</u>

Dette handler om å blande seg inn i aspirantens tankeprosess; aspirantens usynlige virksomhet. Veileder er av og til nødt til å blande seg direkte inn i handlinger. Med dette frigjøres aspiranten for ansvar, og flygelederen tar selv over håndteringen av flytrafikken. Man må imidlertid være nøye med når man skal blande seg inn. Korrigerer man for ofte eller for tidlig, vil det kunne føre til usikkerhet hos aspiranten. Det kan også medføre at aspiranten ikke får prøve seg i vanskelige situasjoner, og på den måten ikke får bygget opp selvtilliten sin. Samtidig, kan aspirant bli *for* selvsikker hvis veileder griper for lite inn. Er aspiranten for selvsikker, kan det føre til at veileder ikke følger godt nok med, eller kanskje ikke følger med i det hele tatt. Dette er ikke bare ulovlig, men kan føre til alvorlige hendelser. Det er en kjensgjerning at en aspirant vil gjøre feil, uansett hvor godt forberedt man er. Veiledningen og instruksjonen er likevel ikke basert på det. –Men når feil gjøres, må man prøve å trekke mest mulig positiv erfaring ut av dem.

Når skal man gripe inn (intervenere)?

- Når aspiranten selv ber om det
- Når det er grunn til å tro at aspiranten er i ferd med å tape kontrollen, eller at sikkerheten er truet uten at han/hun synes å være oppmerksom på det.
- Det oppstår en situasjon en føler aspiranten ikke kan mestre.
- Når aspiranten er i ferd med å bli "overkjørt" og det er verdiløst å fortsette
- Når instruktøren føler at situasjonen utfordrer egne grenser.

Noen feil kan forhindres, noen kan overses, noen kan rettes senere (de-brief), mens noen må korrigeres øyeblikkelig.

I OJT-kurset, skal kursmedlemmene etter dette deles inn i grupper, og diskutere den siste setningen.

SAMMENLIGNING – MOTIVASJON OG INTERVENSJON – FLYGELEDING VS PROGRAMMERING

For flygelederne, er monitorering og intervenering et tema ikke bare for at læringsmiljøet skal være best mulig, men også fordi det er et sikkerhetskrav. Om en student gjør en ekstremt stor feil når han/hun programmerer, vil det verste som kan skje være at programmet studenten holder på å utvikle ødelegges. Det er selvfølgelig surt og synd, men ingen fatal krise. Om en flygeleder ikke følger skikkelig med når en aspirant får prøve seg på flykontrollering, og vedkommende gjør en betydelig feil, kan i verste fall liv gå tapt.

Det vil ikke være naturlig for en hjelpelærer i programmering å hele tiden monitorere arbeidet til en student; det vil heller være etter behov, gjerne etter forespørsel fra stdenten. -Samtidig er det viktig å følge med på om det er studenter som ser ut til å sitte fast med et problem, og tilby hjelp når det er nødvendig.

En problemstilling mange hjelpelærere møter, er at mange studenter kvier seg for å be om hjelp. Grunner til det kan være flere. Det kan for eksempel være at de føler seg dumme; at problemet de har er noe de kanskje burde klare selv, og derfor føler seg for dumme til å spørre om hjelp. Da er det viktig at veilederen greier å hele tiden stille seg disponibel, og sende ut signaler om at ingen spørsmål er for dumme. Å ha en holdning som sier at "det finnes ikke dumme spørsmål, bare dumme svar" er avgjørende. Å si ting som at "kan du ikke dette?" eller "dette burde du ha kunnet" burde være bannlyst. Studenten må selvfølgelig få beskjed om han/hun ligger langt etter det som er meningen, men da heller på en måte som at "Jeg ser du ligger litt etter, men vi kan sette oss ned å se på det, så er du sikkert à jour snart." En "dette skal vi klare" holdning, er i så fall avgjørende.

Ut av dette kan vi skille ut følgende likheter og forskjeller;

Både for veileder i flygeleding og veileder i programmering, er det viktig å sende ut signaler om at en er åpen for å gi hjelp. Det er viktig å se an problemstillingen, og gi "riktig" hjelp eller veiledning i den aktuelle problemstillingen. Imidlertid vil hjelpesituasjonene være forskjellige for flygeleding og programmering. Mens en flygeleder hele tiden vil overvåke aspiranten sin, vil hjelp i programmering bare gis ved behov. Dette vil skape hver sin utfordring, både for flygeleder og programmeringsveileder. Flygelederen vil hele tiden ha full oversikt over hva som skjer. Problemstillingen her, består jo i stor grad av hvordan man overvåker. –Om man griper ofte inn, når det er nødvendig, når man kan vente osv. For hjelpelæreren vil utfordringen gjerne heller bestå i både at en er nødt til å stille seg til disposisjon som hjelper, og prøve å se hvem som trenger det og ikke. I tillegg må en veileder i programmering være i stand til å på kort tid sette seg inn i gjeldende problemstilling. Studenten vil jo helst ønske tilbakemelding og hjelp der og da.
Når hjelpelærer og student har en veiledingssituasjon, vil det stort sett alltid være fordi studenten har et eller annet problem han eller hun trenger hjelp til å løse. Det vil at veiledningen stort sett alltid er knyttet til problemløsning. En aspirant og en flygeleder, vil ikke ha det på samme måte, da flygelederen hele tiden vil være til stede og forsøke å forhindre feil. Mens man ser på feil som en utfordring, og noe som kan løses og stadig dukker opp i programmering, ser man mer på å unngå feil i flygelederyrket. –Skjønt problemløsning er et aktuelt tema også der.

FORSKJELLER OG LIKHETER GENERELT

Nivå

Flygelederne testes hele tiden under utdannelsen, og alle som får bli aspiranter, skal være på omtrent samme nivå. Dette vil si at vi i stor grad kan si at "flygelederaspiranter kan ditt, og vet datt". Dette er de testet på, og *skal* kunne. På samme måte vil ferdig utdannede flygeledere kunne omtrent det samme. Når en skal veilede studenter i et introduksjonskurs i programmering, vil nivået være svært spredt. Noen har programmert i årevis og vil ganske enkelt ha kunnskapene sine "på papiret", mens andre aldri har forsøkt å programmere. Samtidig vil kunnskapene man har etter endt eksamen ha bred variasjon blant informatikkstudentene. At studentene har forskjellig nivå, vil by på utfordringer for veilederen. Han eller hun må hele tiden tilpasse seg ulike nivåer på en tilfredsstillende måte. Denne problemstillingen vil ikke flygeledere som veileder aspiranter møte på i betydelig grad.

Ulikhet i personlighet – oppfanging av signaler

Alle er forskjellige, både blant aspiranter og studenter. Enten man studerer flygeleding eller programmering, vil oppfattelsesevnen være ulik, engasjementet og motivasjonsfaktorer like så. Noen er flinke til å hele tiden sende ut signaler om eget ståsted, enten det er i form av snakk eller

kroppsspråk; mens andre har komplett mangel på sådan. Når kommunikasjonen er dårlig, vil det være vanskelig for veileder å legge opp veiledningen hensiktsmessig. Det kan også nevnes at mens en flygeleder gjerne har en, to eller kanskje tre aspiranter å forholde seg til, har hjelpelærer i programmering nærmere femti. Mens flygeleder og aspirant som regel vil bli godt kjent, og forhåpentligvis trygge på hverandre ganske fort, kan en student og en hjelpelærer gå gjennom et helt semester uten å veksle mer enn få ord.

Spørsmål og spørsmålstyper

Spørsmålsteknikk vil være svært viktig å kunne litt om, både for veiledere i programmering og for flygeledere. De finnes mange typer spørsmål, noen av dem jeg fant nevnt i læringsboka om veiledningsteknikk, er affektive spørsmål, avgrensede spørsmål, informasjonsspørsmål, kognitive spørsmål, ledende spørsmål, mestringsspørsmål, mirakelspørsmål, refleksjonsspørsmål, sirkulære spørsmål, tildekkende spørsmål, unntaksspørsmål, utfordrende spørsmål og utvidende spørsmål. Noen ganger stiller man ledende spørsmål, hvor svaret omtrent blir lagt i munnen på den som veiledes. Andre ganger stiller man spørsmål for å finne ut hva slags nivå vedkommende ligger på. – Kanskje stiller man spørsmål for å sette fast en student som trenger å bli mer klar over egne begrensninger? Uansett bør man tenke over hva slags svar en forventer når en stiller et spørsmål. For hjelpelærere i programmering vil det være svært viktig å stille spørsmål for å finne ut ståstedet til studentene. Flygelederen vil gjerne bruke spørsmålsteknikk i forkant av en situasjon, for å få vite hva aspiranten har tenkt å gjøre. Hvis aspiranten har tenkt til å gjøre noe som ikke er særlig lurt, kan flygelederen ved hjelp av spørsmålsteknikk hjelpe aspiranten til å se konsekvensen av valgene sine. ”Hva vil skje med flyet som skal lande hvis du lar et fly ta av nå?” Forskjellen vil altså være at man bruker mer spørsmål i forkant som flygeleder, mens hjelpelærer gjerne vil stille spørsmål i etterkant av en situasjon.

Kroppsspråk

Vi leser andres kroppsspråk, og snakker med vår egen kropp, stort sett uten å tenke over det. Som veileder er det viktig å tenke over hva slags signaler man sender og mottar. Her vil det som jeg har vært inne på før, være en elementær forskjell for flygeledere og programmerere. Flygelederen sitter hele tiden sammen med sin aspirant, og vil plukke opp feil underveis i prosessen. Veilederen i programmering, vil kun være til stede i enkelte situasjoner, og må da i mye større grad enn flygelederen, -snappe opp de studentene som sender ut signaler om hjelp. Her er det viktig at hjelpelærer er obs på signaler, både når det gjelder øyekontakt, ansiktsuttrykk, kroppsholdning og bevegelse. Dette gjelder både signaler en sender ut og signaler en mottar.
Det er særlig viktig at veileder i programmering, bruker signaler aktivt. Hvis en hjelpelærer sitter og jobber med ei av sine egne oppgaver, snakker med noen andre eller spiser mat, vil det være vanskelig for en litt beskjeden student å spørre om hjelp. Denne problemstillingen er også gjeldende for flygeledere, men ikke i like stor grad.

En stor forskjell er forholdet til abstraksjoner. Mens det å programmere, særlig i læringsprosessen, består av å løse problemer hvor en må tenke abstrakt, er flygelederaspirantens opplæring hele tiden knyttet tett opp mot virkelige situasjoner.

Den største forskjellen, opplever jeg at er fokus. Alle temaene flygelederne tar opp i OJT-kurset er for så vidt viktige å lære for en hjelpelærer i programmering, men viktighetsgraden i de enkelte temaene er vidt forskjellige. Mens forhindring av stress er svært viktig for flygelederne, vil dette være et mindre tema i programmering. Problemløsning ville tatt mye oppmerksomhet for en som skal veilede i programmering, mens flygelederne først og fremst fokuserer på å forhindre at feil i det hele tatt oppstår. Motivasjon av en som skal lære et helt nytt programmeringsspråk er essensielt, mens man regner med at en aspirant er rimelig motivert av å få prøve seg i praksis etter tre år med kun simulatortrening. –Det betyr selvsagt ikke at motivasjon ikke er viktig, bare at det ikke er fullt så avgjørende som for en novise i programmering.

KONKLUSJON

Jeg vil konkludere med at det å lære å lære bort, er svært generelt. Mye av undervisningsmaterialet til flygelederne går på lærdom om mellommenneskelige forhold, måten

mennesker reagerer på forskjellige ting, hvordan de kan motiveres, når læringssituasjonen er optimal og lignende. Dette kan benyttes i nær sagt en hver læringssituasjon. Jeg opplever at å studere dette OJT-kurset, har lært meg mye, som jeg er sikker på at jeg vil ha stor nytte av, både på høgskolen og ellers. Når det gjelder sammenligning mellom å lære noen å programmere og å lære noen å kontrollere fly, opplever jeg at kurset er mer generelt enn jeg hadde forutsett, og at det derfor absolutt kunne vært brukt til en som skal være hjelpelærer i programmering. Den største forskjellen er vel at en kanskje ville stokket om på viktighetsgraden til de forskjellige temaene.

KILDEBRUK

Materiell om flygeleding:

Alt materiell om OJT og flygelederopplæring, er skaffet til veie av OJT- instruktør Stein A.Johnsen

Programmering:

F. Dètienne (2002). *Software Design – Cognitive Aspects*. Springer Vorlag London

J. Rogalski, Renan Samurcai (1990). *Psychology of programming*. Academic press limited

Høgskolen til Sogn og Fjordane, kurs in650 i programmering. Kapittel 1. Her omhandles temaet abstraksjon.
http://www.hisf.no/aos/ibh/in650/webNotat/kapitel1/OOP2.htm

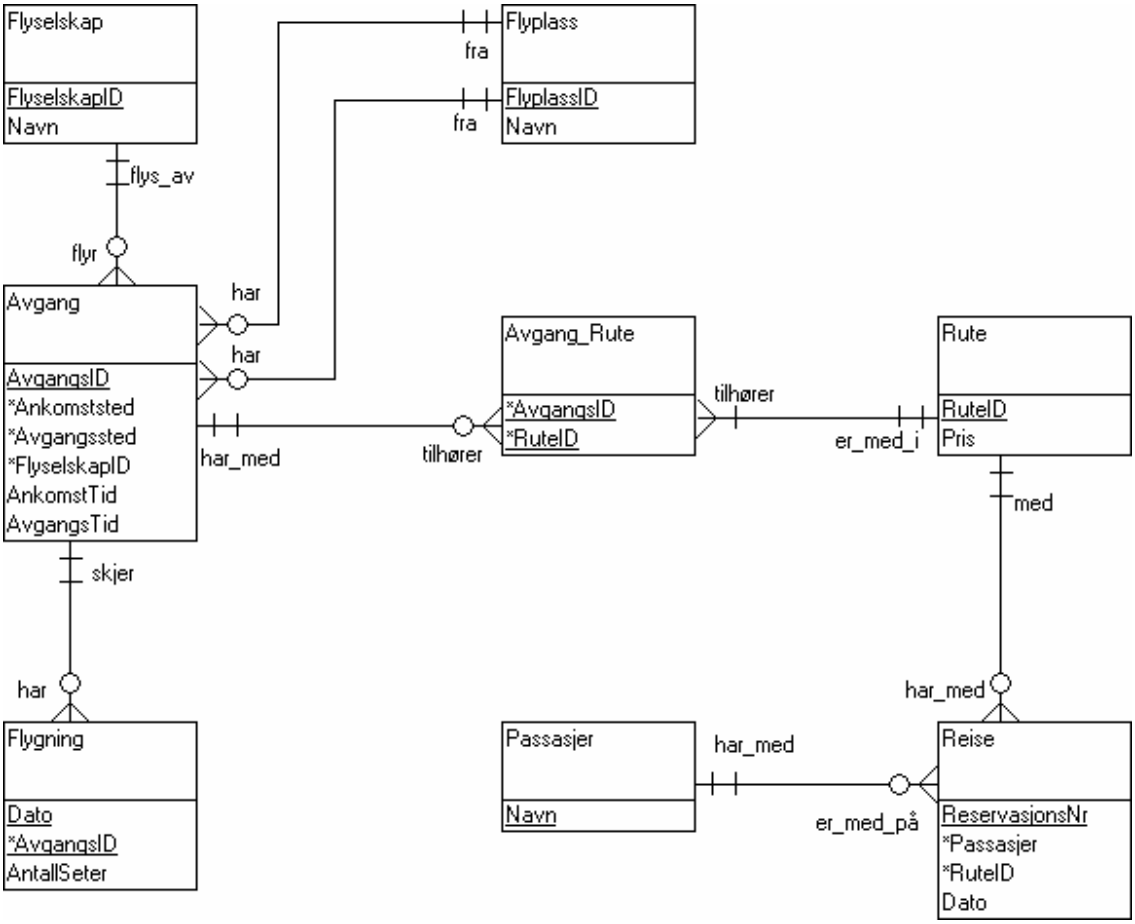
Erfaringer fra deltakelse programmeringskurset in150 og in151.

Ellers: P. Mathisen, R. Høigaard (2004). *Veiledningsmetodikk*. Høyskoleforlaget Undervisningen i in209 - fagdidaktikk

3. IN601: Databaseapplikasjoner av Anders Gjendem og Lene Østby

Innholdsfortegnelse

Oppgave 1 - Datamodell



All informasjon om en rute hentes fra Avgang_Rute- og Avgang-tabellene for å unngå redundans. Vi vurderte å legge til et rutenavn, evt start- og sluttsted, men henter i stedet dette fra ankomst- og avgangssted til første og siste avgang på ruten. For å bestemme rekkefølgen på avgangene, vurderte vi å legge til et "sekvensnr" i Avgang_Rute som forteller i hvilken rekkefølge avgangene går på en rute, men har i stedet valgt å løse dette ved å sortere på avgangstid – synes dette var en "penere" løsning.

I avgang-tabellen kunne vi ha tenkt oss en kontroll på avgangstid < ankomsttid, men har ikke fått til dette i Access (har heller ikke prøvd så hardt, ettersom registrering av nye avganger ikke var en del av funksjonaliteten). Dersom vi skulle ha registrert ruter i det ferdige programmet, måtte vi også ha lagt inn kontroll på hvilke avganger som kan benyttes ved opprettelse av nye ruter, og ved endringer i tidspunkt i Avgang-tabellen måtte alle ruter som benytter denne avgangen kontrolleres for å se om de fortsatt var gjennomførbare. Vi anser det som usannsynlig at avgangs- og ankomssted endres, da bør det heller opprettes en ny avgang.

Flyselskap og flyplass valgte vi å lage som egne entiteter i stedet for å ha dem som attributter i Avgang-tabellen – det gjør oppdatering enklere, og sikrer korrekt informasjon. Det blir også enklere dersom man ønsker å registrere andre opplysninger enn navn om både flyplass og flyselskap, noe som ikke er utenkelig.

I Reise-tabellen bruker vi et reservasjonsnr som primærnøkkel for enkelhets skyld, men det kreves at kombinasjonen passasjer, rute og dato skal være unik for å unngå dobbeltbestillinger. Det kan muligens hende at en passasjer vil reise Molde-Oslo to ganger på en dag, men ettersom en rute består av spesifikke avganger på bestemte tidspunkt, vil dette være to forskjellige ruter.

Ved å knytte en reise til bare én passasjer har vi gjort det på enkleste måte, men vi synes det ville vært mer realistisk at flere passasjerer kunne bestille reise med samme reservasjonsnr – for eksempel en familie. Det vil være uinteressant for en familie på fire å bestille en reise for én og én person, ettersom det må være plass til alle. Dersom en reise/reservasjon skal knyttes til flere passasjerer, får vi en mange til mange relasjon som må løses opp. Vi vurderte å implementere dette, men bestemte i samråd med Tarjei at det var litt utenfor oppgaveteksten.

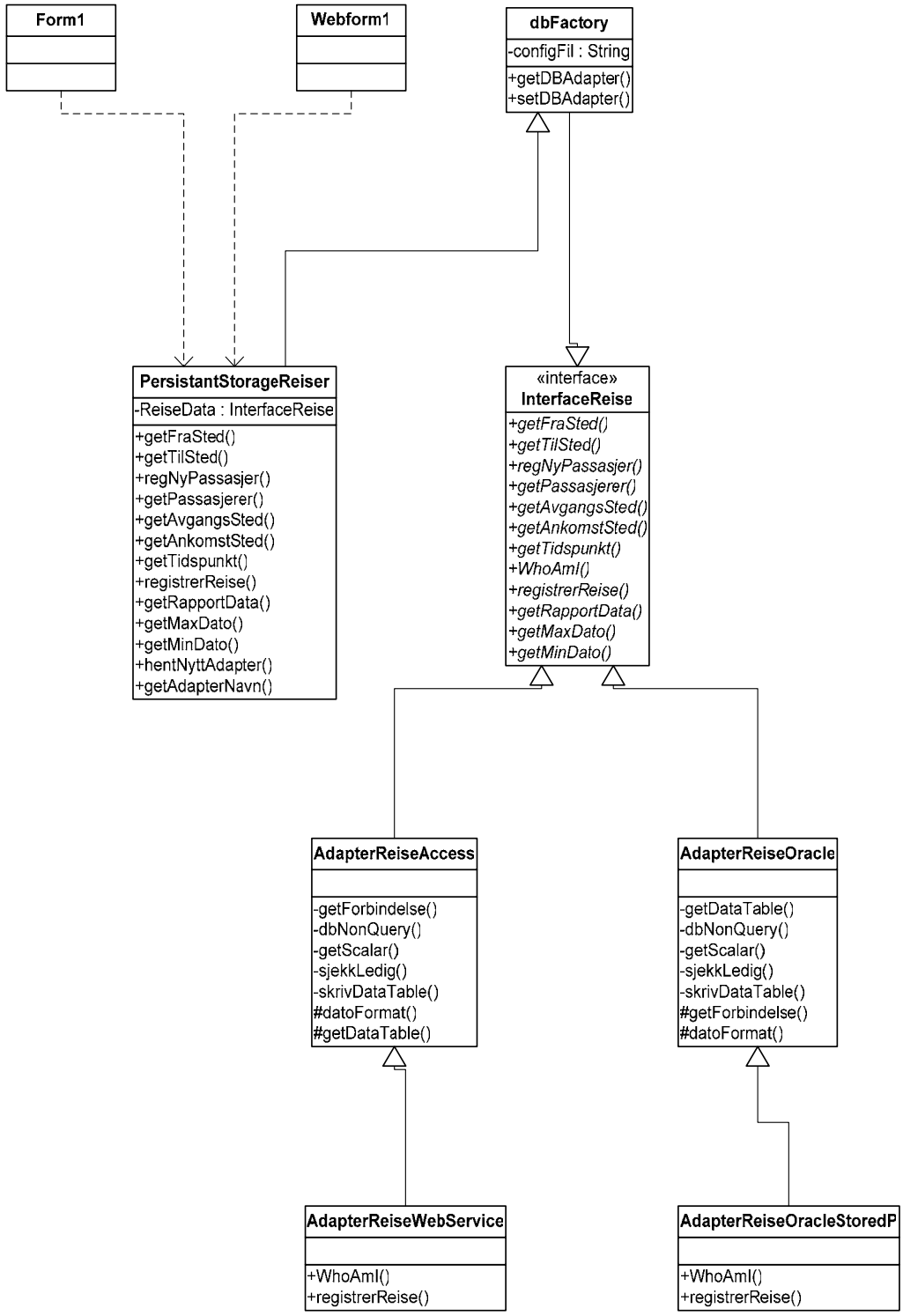
I den opprinnelige modellen fra øving 1 valgte vi å registrere kun antall seter på en flygning – antall ledige seter kan da beregnes ut fra antall reiser. Ulempen med dette er at vi må bruke en trigger til å kontrollere at antall reserverte seter på en flygning ikke er større enn antallSeter – samtidig unngår vi redundans og får noen fordeler programmeringsmessig. Vi slipper også oppdateringsproblemer dersom en reise slettes. Det viste seg allikevel å være vanskelig å gjennomføre i praksis; den samme beregningen måtte gjøres om og om igjen for hver gang vi registrerte en ny reise, og det ble programmeringsmessing mer komplisert enn vi hadde trodd.. Det endte med at vi, etter diskusjon med Tarjei, la til antallLedigeSeter i Flygning-tabellen.

Dersom en passasjer slettes, slettes også eventuelle reservasjoner (reiser) knyttet til passasjeren – skal dette implementeres, må også antall ledige seter på de forskjellige flygningene oppdateres når en reise slettes. Aktuelle rader i Avgang_Rute tabellen slettes også dersom en rute slettes. Ellers har vi brukt restrict. Ved update bruker vi cascade, med forbehold om at det mangler noe kontroll – dersom tidspunkt for en avgang endres, må eksisterende ruter kontrolleres.

For å slette (egentlig kansellere) en flygning, må også alle reiser som benytter flygningen slettes/oppdateres. I stedet burde vi ha et status-felt for en flygning, slik at man har mulighet til å informere passasjerene, evt gjøre en ombooking eller avbestilling. Men dette er utenfor oppgaven.

Oppgave 2 – Implementasjon

Klassediagram



Kort om den grove informasjonsflyten

Klassediagrammet viser hvordan vi har bygd opp klassene våre i prosjektet med arv, bruk av interface og mønster som Adapter, Factory, Low Coupling og High Cohesion, og bruk av et felles grensesnitt mellom dataene og grensesnittet brukeren ser. Pilene mellom InterfaceReise og adapterne er sannsynligvis feil, siden de implementerer interfacet, ikke arver det.

Når vi snakker om adapter i denne teksten, mener vi vår egne klasser, som for eksempel AdapterReiseAccess – og ikke adapterne som er innebygd i vb.net og baserer seg på slik som Jet4.0/Oracle Orahome9.0 providerne. Disse brukes igjen av våre adaptere. Grunnen til at de heter adapter, er fordi det er det mønsteret kalles, og at jobben deres er å oversette fra en type data, til noe annet (lagret form) – akkurat slik som adapterne i vb.net bruker sine providere.

Vi valgte å følge de metodene vi etterhvert har lært oss i fag som in212 og in295 – i stedet for den, ihvertfall ved første øyekast, mer lettvinte metoden. Vi så tidlig at det ville være en del å tjene på de senere oppgavene ved å investere litt ekstra tid i oppgave 2. På denne måten blir systemet veldig oversiktlig å kode, det blir minimalt med kode å vedlikeholde, og vi får ikke minst en mye større fleksibilitet for utvidelser enn den alternative alt-i-ett metoden som vanligvis blir benyttet. Sånn sett var oppgaven ideell for vår løsning – ved at vi skulle utføre de samme oppgavene i programmet, men mot flere forskjellige databaser.

Klassen PersistantStorageReiser fungerer her som den bindingen mellom de klassene som tar seg av input og output i systemet fra brukeren – og de klassene som står for lagring/skriving og bestilling. PersistantStorageReiser sine metoder returnerer grunndata, mye i form av DataTable, Date og lignende, slik at grensesnitt-klassen får ansvaret for å utforme/vise dataene på en måte som egner seg for det aktuelle grensesnittet. For eksempel er det i windows-grensesnittet en comboboks hvor vi viser aktuelle avgangs/ankomsttidspunkt mellom to valgte flyplasser. Der omformes tidspunktene til korrekt format, for eksempel ”09:10 – 10:15”. Dette er da avgangs- og ankomsttidspunktet til en gitt rute, og vi har her brukt tilhørende ruteid som Valuemember i comboboksen.

Det aktuelle grensesnittet oppretter en instans av klassen PersistantStorageReiser. Denne klassen oppretter en instans av dbFactory klassen, som så finner sin XML-konfigurasjonsfil som brukes til å avgjøre hvilken lagrings/bestillingsløsning som benyttes til enhver tid. Ved eventuelle lesefeil eller manglende konfigurasjonsfil, vil den som standard velge Access-løsningen. Den løsningen som er valgt blir det så opprettet en ny instans av dynamisk:

```
dsConfig.ReadXml(configFil)

' Henter ut verdien fra <configuration> <StorageAdapter> i xml fila
Dim ValgtAdapter As String = CStr(dsConfig.Tables(0).Rows(0).Item(0))

If ValgtAdapter <> "" Then
    adapter = CType(Activator.CreateInstance(Type.GetType(ValgtAdapter)), InterfaceReise)
Else
    adapter = New AdapterReiseAccess
End If
```

Klassen dbFactory returnerer så det valgte adapteret til PersistantStorageReiser som videre tar vare på instansen. Klassen dbFactory tar seg også av lagring av endringer i valg av aktivert adapter. Adapter kan byttes on-the-fly både i web og det windows-baserte grensesnittet; det er tatt hensyn til at dette skal kunne skje uten å få en ugyldig registrering når database byttes, ved at data leses inn på nytt ved bytte av adapter.

InterfaceReise spesifiserer et felles grensesnitt som alle data/bestillings-klassene må implementere for å kunne brukes på denne modul-baserte måten. Klassene kan så i tillegg ha sine egne spesialiseringer som private funksjoner, alt etter hvilke lagrings/bestillingsmetode som benyttes. I teorien burde vi relativt lett kunne skrive ett adapter som kan bruke tekstfiler i stedet for databaser eller en webservice, og det var også et mål å legge til rette for en slik fleksibilitet.

Som sagt er klassen PersistantStorageReiser en binding mellom grensesnitt og data/bestilling. Metodene i klassen tilsvarer de som vi finner i InterfaceReise, i tillegg til de aktuelle i dbFactory. Når grensesnittet vil registrere en ny passasjer, tar PersistantStorageReiser imot ordren, og utfører den tilsvarende metoden på det valgte adapteret. Adapteret utfører da ordren på sin spesifiserte måte.

Alle adapter-klassene har selvfølgelig også alle metodene i InterfaceReise, selv om de ikke synes i klassediagrammet direkte. De metodene som synes, er i AdapterReiseAccess og AdapterReiseOracle, de Private-metodene (-) disse to klassene benyttet seg av, i tillegg til de metodene som er deklarert som Protected (#). Vi ser at de begge har en metode som oppretter en connection til databasen – nemlig getForbindelse(). Den metoden er igjen brukt av dbNonQuery(), getScalar() og getDataTable(). Disse metodene i database-adapterne benyttes av Public-metoder for å utføre sql-kommandoer og skaffe grunndata fra databasen, noe som gjør koden mindre, og vedlikeholdet lettere. Hvis vi mot formodning skulle ha gjort en gjennomgående feil i en metode, ville dette tidligere ført til at vi måtte gå over alle metodene og rette feilen – nå trenger vi sannsynligvis bare å endre det ett sted for hvert adapter.

Klassen AdapterReiseOracleStoredP er en spesialisering av Oracle-adapteret vårt, som vi ser har den kun to metoder som er forskjellige fra den første AdapterReiseOracle-klassen vår. Metoden WhoAmI() returnerer en string som identifiserer klassen med klassenavn, hovedsaklig for debugging, og for at vi skal være sikre på at rett klasse er i bruk, samt at det kan brukes i grensesnittet til å spesifisere hvilket adapter som er i bruk på en enkel måte - for eksempel ved å markere det aktive adapteret som valgt i gruppen av radio-bokser i Konfigurasjons-delen av windows-programmet. Metoden registrerReise() er den eneste større endringen her, ettersom metoden kaller opp en Stored Procedure på Oracle-serveren i stedet for å utføre kommandoene/feilsjekking selv.

Klassen AdapterReiseWebService har tilsvarende endringer, den arver fra AdapterReiseAccess siden vi bestemte oss for å benytte den lokale Access-databasen videre som klient-side-database i denne oppgaven. Den identifiserer seg korrekt, og metoden registrerReise() er endret ved at den i tillegg til den vanlige registreringen bestiller reisene på webservicen vår, og passer på at en eventuell feil håndteres korrekt med en form for rollback. Transaksjonshåndteringen må kunne betegnes som optimistisk. Et array samler opp reservasjonsnummer etterhvert, og det blir kontrollert for hver registrering om vi har fått en feil/negativ verdi tilbake. Ved feil kjøres avbestillings-metoden for alle registreringene. Også her har vi lagt inn sjekker – hvis avbestilling ikke lykkes – blir referansenummerene vist på skjermen slik at de kan utføres manuelt eller ved en senere anledning når webservicen er tilgjengelig igjen.

Akkurat på denne biten er det forbedringspotensiale, men på grunn av generelt mye arbeid har vi valgt å forhold oss til at det er en prototype vi skulle lage, og har dermed ikke utvidet databasen med en nødvendige entitet for registrering av reservasjonsnummer for en gitt reise, samt hvilken webservice/leverandør reservasjonen er hos. Dette burde være en relativt smal sak å utvide med i videre faser, siden det bare er snakk om noen ekstra kall til dbNonQuery() metoden for å registrere reservasjoner knyttet til en reise. Registrering av reise utføres, og disse dataene er i en prototyp-fase strengt tatt unødvendig – helt til noen eventuelt må avbestille en reise på et senere tidspunkt. Når en

registrering er utført, vises også her en rapport med priser, hvilken rute som skal flys osv, samme som før.

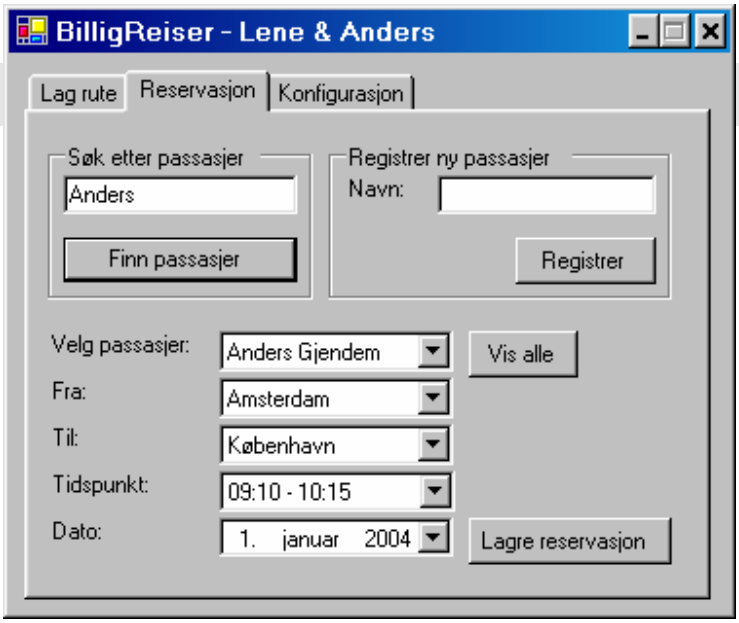
Løsningen er ikke helt perfekt slik den ble, siden klassene i vb.net kompileres inn i exe-filen uansett. Det fører til at når man legger til for eksempel en web-reference – så får alle klassene i utgangspunktet tilgang til den namespacen. Det gjør også at vi ikke mens programmet kjører, kan legge til nye adapter, selv om det strengt tatt er helt unødvendig å dra det så langt. Med litt bedre tid ville vi nok implementert adapterne våre som egne dll-filer – slik at dette kunne gjøres på samme måte som i java, hvor klassene ikke kompileres inn i en felles fil, på den måten ville for eksempel kun AdapterReiseWebService ha tilgang til web-service namespacen. Slik ville vi fått en plug-in løsning i programmet. Vi tror uansett at vi sannsynligvis har gjort mer objektorientert vb-programmering enn de fleste andre til sammen, og det har vært veldig interessant og lærerikt å se hvor likt det kan gjøres i forhold til Java.

Brukergrensesnitt

Ruteregistrering

Skisse av brukergrensesnitt for ruteregistrering – man velger blant registrerte avganger, legger til og bestemmer pris. Hvilke avganger som kan velges, bestemmes av sist valgte avgangs ankomststed og ankomsttid. Skulle vi ha brukt dette, burde vi ha gjort en del forandringer – kunne kanskje vært lurt å ha mulighet til å søke etter avganger på frasted, burde også kunne se klokkeslett på avgangen man velger. Vi var også inne på tanken om å bruke en algoritme for å generere mulige ruter mellom to steder, listet på enten pris eller tid, eventuelt en kombinasjon.

Reservasjon



Når man skal registrere en reise kan man enten starte med passasjeren, eller å finne riktig rute. Passasjeren kan enten velges i comboboksen hvis han har reist med oss før, eventuelt kan man registrere en ny passasjer i tekstboksen over. Ved registrering av ny passasjer, blir den passasjeren automatisk valgt i comboboksen. Vi har også lagt til en liten søkemulighet, slik at man kan søke på deler av navn, og få resultatet opp i comboboksen. Siden vi i begge disse tilfellene viser et utvalg av kundene våre, har vi lagt med en knapp "Vis alle" som igjen fyller comboboksen med alle våre registrerte kunder.

Det neste som må gjøres er å velge avgangssted. Vår løsning baserer seg på at kunden i hvert fall vet hvor han vil reise fra, og det må velges først. Tilgjengelige ankomststeder listes så automatisk opp i comboboksen Til, og tilhørende tidspunkt tilsvarende. På denne måten sørger vi for at kun gyldige ruter blir mulig å velge blant. Så snart ankomststed er valgt blir også informasjon om tidspunkt for ruten vist i en egen comboboks. Denne viser avgangstidspunkt, og forventet ankomsttidspunkt. Dette siste valget tilsvarer en rute i databasen, og identifiserer den entydig, og det er faktisk denne informasjonen vi baserer resten på. Comboboksen tidspunkt har ruteiden satt som Valuemember (valgt verdi).

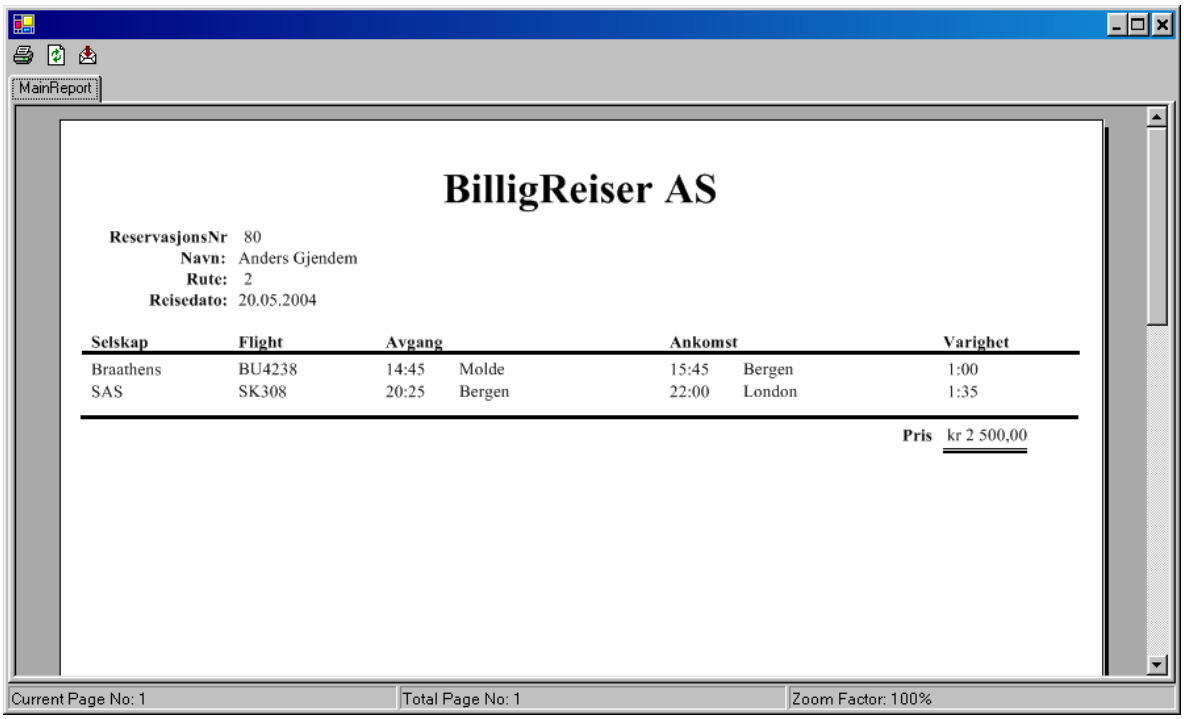
Neste trinn er så å velge en dato for reisen. Det er sperret for i kalenderen slik at man ikke kan velge tidligere datoer enn dagens, eventuelt tidligere enn første gang ruten går. Tilsvarende for siste gang ruten er oppført. Hvis det ikke er noen tilgjengelige datoer får kunden beskjed om det allerede her. Dessverre er det ikke mulig med dette datetimepicker-objektet å kun få vist de faktiske flygningsdatoene, uten veldig mye ekstra arbeid. Et alternativ ville vært å liste opp datoene nok en gang i en comboboks.

Siste trinn er bekreftelsen, og endelig bestilling i databasen, her kontrolleres selvfølgelig at en flygning på ruten skjer den aktuelle dagen, og at det er ledig plass.

Vi sjekker først om det er ledige plasser igjen på den aktuelle flygningen – hvis ja registrerer vi reisen. Deretter sjekker vi at antallet plasser ikke er overskredet for å være helt sikre på at ikke noen andre

også har forsøkt å booke på samme flygning samtidig, slik at vi får inkonsistente data. I så fall kjøres en rollback.

Bekreftelse



Bekreftelse på registrert reise, med reiseplan, pris og tidspunkt.

Når en reise er registrert korrekt i databasen, vil funksjonen hente ut avgangsinformasjon basert på den valgte ruten, i tillegg til informasjon om den valgte reisen. Dette kombineres og vises oversiktlig i rapporten over, med avgangstid og sted, ankomsttid og sted, beregnet varighet, selskap, og flightnummer. I tillegg vises prisen for ruten.

Adapterne som jobber mot database henter i svært stor grad data fra de 11 viewene vi har laget i både Access- og Oracle-databasene. Det er flere grunner til dette, det er for det første nok en måte å spesifisere et abstraksjonslag mellom programmet og databasen – for fremtidige endringer, og ikke minst så gjør det selve select-setningene i våre adapter enklere og kortere. Så lenge kolonnenavnene stemmer overens så kan vi fritt endre på de underliggende kommandoene og tabellene i databasen uten å tenke for mye på hvilke andre plasser i programmet vi må endre. Flere av viewene er igjen basert på andre view for oversiktens skyld mens vi programmerte, ytelsesmessig ville det sikkert vært lurt å slå disse sammen.

Et av disse viewene er det følgende, som viser en liste over alle ruter, og alle tilhørende flygninger, tidspunkt og flyplasser. Viewet er sortert på rute og avgangstid, og brukes som grunnlag for rapporten over, slik at all listing i rapporten baserer seg på rekkefølgen angitt i viewet.

viewRuterDetaljert: (grunnlag for rapportdata)

```
SELECT ar.AvgangsID, ar.RuteID, s.Navn AS Selskap, fra.Sted AS Avgangssted, a.AvgangsTid,
til.Sted AS Ankomststed, a.AnkomstTid
FROM Avgang_Rute AS ar, Avgang AS a, Flyplass AS fra, Flyplass AS til, Flyselskap AS s
WHERE a.AvgangsID=ar.AvgangsID And fra.FlyplassID=a.Avgangssted And
til.FlyplassID=a.Ankomststed And s.FlyselskapID=a.FlyselskapID
ORDER BY ar.RuteID, a.avgangsTid;
```

viewRuterDetaljert : Select Query

AvgangsID	RuteID	Selskap	Avgangssted	AvgangsTid	Ankomststed	AnkomstTid
SK123	1	SAS	Stavanger	06:25:00	Oslo	07:15:00
KL140	1	KLM	Oslo	11:25:00	Amsterdam	13:30:00
KL890	1	KLM	Amsterdam	16:00:00	Paris	17:00:00
BU4238	2	Braathens	Molde	14:45:00	Bergen	15:45:00
SK308	2	SAS	Bergen	20:25:00	London	22:00:00
BU4238	3	Braathens	Molde	14:45:00	Bergen	15:45:00
BU346	3	Braathens	Bergen	16:10:00	Stavanger	16:45:00
BU161	4	Braathens	Molde	14:00:00	Oslo	14:55:00
KL725	5	KLM	London	09:10:00	Oslo	11:30:00
SK456	5	SAS	Oslo	13:05:00	Molde	14:00:00
LH212	6	Lufthansa	Amsterdam	09:10:00	København	10:15:00
SK123	7	SAS	Stavanger	06:25:00	Oslo	07:15:00
KL140	7	KLM	Oslo	11:25:00	Amsterdam	13:30:00
KL890	7	KLM	Amsterdam	16:00:00	Paris	17:00:00

Record: 1 of 14

Et annet eksempel er viewRuter som viser avgangssted, ankomststed og tidspunkt for alle rutene. Dette baserer seg på to andre view som lister opp avgangssteder og ankomststeder.

```
SELECT fra.Rute, fra.Sted AS Avgangssted, fra.AvgangsTid AS Avgangstid, til.Sted AS
Ankomststed, til.Ankomsttid AS Ankomsttid
FROM viewRuterAvgangsSted AS fra, viewRuterAnkomstSted AS til
WHERE fra.Rute=til.Rute;
```

viewRuter : Select Query

Rute	Avgangssted	Avgangstid	Ankomststed	Ankomsttid
1	Stavanger	06:25:00	Paris	17:00:00
2	Molde	14:45:00	London	22:00:00
3	Molde	14:45:00	Stavanger	16:45:00
4	Molde	14:00:00	Oslo	14:55:00
5	London	09:10:00	Molde	14:00:00
6	Amsterdam	09:10:00	København	10:15:00
7	Stavanger	06:25:00	Paris	17:00:00

Record: 1 of 7

Konfigurasjon

Vinduet viser tilgjengelige adapter mot forskjellige datakilder. Når man kommer til vinduet så er det aktiverte adapteret markert, man kan velge hvilket som helst annet, og trykke bruk, deretter oppretter programmet en ny instans av den aktuelle klassen, og tar denne i bruk, slik at når man går tilbake til reservasjon, så vil denne datakilden sine ruter vises.

Stored Procedure

```
CREATE OR REPLACE PROCEDURE OppdaterFlygning(
 p_dato IN flygning.dato%TYPE,
 p_navn IN passasjer.navn%TYPE,
 p_rute IN viewRuter_Dato.rute%TYPE)
IS
 i_minimum NUMBER;
 i_registrert NUMBER;
BEGIN
 SELECT MIN(ANTALLLEDIGESETER) INTO i_minimum
 FROM viewRuter_Dato
 WHERE Rute = p_rute
 AND Dato = p_dato;

 SELECT COUNT(*) INTO i_registrert
 FROM reise
 WHERE passasjer = p_navn
 AND ruteid = p_rute
 AND dato = p_dato;

 IF i_minimum = 0 THEN
  RAISE_APPLICATION_ERROR(-20001, 'Ikke ledig på alle flygninger');
 ELSIF i_registrert >= 1 THEN
  RAISE_APPLICATION_ERROR(-20002, 'Allerede registrert');
 ELSIF i_minimum > 0 THEN
  UPDATE FLYGNING SET AntallLedigeSeter = (AntallLedigeSeter-1)
  WHERE Dato = p_dato
  AND AvgangsID IN (SELECT AVGANGSID
    FROM viewRuter_Dato
    WHERE Rute = p_rute
    AND Dato = p_dato);
  INSERT INTO reise (Reservasjonsnr, Passasjer, RuteID, Dato)
  VALUES (reise_teller.nextval, p_navn, p_rute, p_dato);
 ELSE
  RAISE_APPLICATION_ERROR(-20003, 'I_minimum ikke positiv');
 END IF;
END;
```

Det sendes inn 3 parameter, dato, navn og rute. Det deklareres to interne variabler, minimum og registrert. Neste trinn er å finne ut om det er ledig på alle flygningene. Det gjør vi ved å hente ut ledige seter fra alle flygningene, og finne minste antall ledige. Hvis det antallet er 0, kan ikke bestillingen godtas. Neste som kontrolleres er om reisen allerede er registrert på samme dato, for samme person. Det gjør vi ved å telle antallet registrerte reiser med samme input-informasjon som nå. Hvis vi får et positivt tall, er reisen allerede registrert.
Disse verdiene blir så kontrollert i if-setningene mot ovennevnte kriterier, hvis noen ikke er oppfyllt, vil en feilmelding kastes, og vi får en rollback automatisk. Denne feilmeldingen plukker så klient-programmet opp, og bruker videre i sin tilbakemelding til brukeren. Hvis alt er ok, oppdaterer vi databasen med reisen, og senker antall ledige seter for den aktuelle flygningen. Den siste else-setningen skal i teorien aldri kunne skje, og burde kanskje vært droppet, eventuelt slått sammen til if i_minimum <=0 then … i første sjekken.

Access vs Oracle

Det var spesielt to større forskjeller mellom disse systemene. Det første og viktigste er datoformat. Det håndterte vi ved å lage en egen funksjon som konverterer mellom Date datotypen i VB.net, og datoformatet databasen bruker. Dette returnerte vi som en string. Et alternativ ville være å bruke parameter-felt til kommandoobjektet, men siden vi ikke har gjort det noen andre plasser i koden, var det mest konsist å bruke funksjonen i stedet. Den andre større forskjellen var egentlig ikke selve databasen sin skyld, men vårt ukloke valg av provider til adapteret. Vi valgte Oracle sin egen, Orahome9.0, i stedet for Microsoft sin. Forskjellen mellom disse er at Orahome9.0, i motsetning til Microsoft provideren, er case-sensitiv når det kommer til kolonnenavn. I og med at vi brukte våre adaptere var det ikke noe stort problem, og vi valgte å holde oss til Oracle sin fordi det var mer interessant å prøve litt forskjellige løsninger.

Oppgave 3 - Transaksjonssikkerhet

Isoleringsnivå

Vi har valgt å bruke isoleringsnivå Read Committed. Her settes det X-lås ved skriving, men leseoperasjonen setter ingen S-lås, noe som gjør transaksjonen svært effektiv. Selv om Serializable er det eneste isoleringsnivået som garanterer en konsistent database etter avsluttede transaksjoner, mener vi at dette er et forsvarlig valg – vi har lagt vekt på effektivitet og små forsinkelser ettersom dette er et billettbestillingssystem, og selv noen få sekunder virker svært lenge for brukeren. Det er også få oppdateringer som skal gjøres i databasen – det er kun feltet antallLedigeSeter i Flygning-tabellen som oppdateres, Reise-tabellen får en ekstra rad. Registrering av en ny reise byr på få problemer dersom reservasjon av seter på de ulike flygningene går bra – ettersom kombinasjonen kunde, rute og dato må være unik vil man få feilmelding og rollback dersom samme reise forsøkes registrert to ganger. Read Committed anbefales brukt på applikasjoner som ikke har behov for å ha tilgang til flere rader i en sammenheng. Vår applikasjon skal oppdatere flere rader i flygning-tabellen og passer derfor ikke helt inn i denne kategorien, men med så begrenset funksjonalitet som dette programmet har, mener vi det er forsvarlig. Skulle vi ha utvidet funksjonaliteten, burde vi muligens vurdert å ”øke” isoleringsnivået, men foreløpig mener vi at vi har greid å fange det meste som kan gå galt med Read Committed. Hovedgrunnen til at dette isoleringsnivået ble valgt, var altså effektivitet og ”smidighet” i forhold til Serializable, og ikke minst større grad av samtidighet. I Access-databasen er også Read Committed det høyest tilgjengelige nivå.

Vi har til en viss grad også benyttet oss av en slags optimistic concurrency controll – vi later som om konflikter ikke inntreffer, og kontrollerer i etterkant at alt er ok. Men: optimistic concurrency controll krever tillatelse til commit, men kan lese og skrive uten vurdering – ved bruk av read committed benytter vi oss av skrivelås, og kan alltid gjøre commit. Vi gjennomfører også en sjekk før skriving til databasen, som kontrollerer om det er noen ”vits” i å forsøke å oppdatere i det hele tatt. Denne sjekken kunne for så vidt ha vært droppet, uten andre konsekvenser enn at det måtte foretas rollback oftere – vi har allikevel valgt å beholde den, ettersom den sparer en del tid.

Access vs Oracle

Oracle opererer med immediate update, som gjør alle skriveoperasjoner tilgjengelige med mindre de ikke er låst. Ettersom Read Committed setter skrivelås, vil alle berørte rader i en transaksjon være låst

til commit ved bruk av Oracle – og andre transaksjoner som skal oppdatere de samme radene må vente til den aktive transaksjonen har kjørt commit. Access benytter derimot deffered update – transaksjonen skriver til et privat område, og verdiene herfra skrives først til fellesområdet ved commit. Dette medfører at den siste sjekken vi har brukt for å utelukke overbooking egentlig er overflødig i Access – transaksjonen opererer på en privat kopi av de aktuelle rader i databasen, og sjekken vil derfor ikke utføres på "selve" databasen. Den slår dermed aldri inn. Men i motsetning til Oracle setter ikke Access skrivelås kun på aktuelle rader – den låser hele tabellen ved update. Ved isoleringsnivå Read Committed vil altså ingen andre transaksjoner få lese fra Flygning-tabellen mens en annen transaksjon skriver, noe som gjør Access-databasen sikret mot en del feil som kan inntreffe i Oracle (se under) ved Read Committed. På den annen side gjør dette også Access-databasen svært mye tregere og gir liten grad av samtidighet.

Kilde: http://www.microsoft.com/accessdev/articles/jetlund.htm

I følge denne diskusjonsgruppen er det kun Read Committed som er støttet i jet4.0 mot Access, foruten dirty read. Det er heller ingen muligheter med ADO.net som gjør at vi kan tvinge en låsing før vi leser, og være 100% sikker på at kun vi har tilgang. En mulighet er å fremtvinge en skrivelås som det første man gjør, ved å for eksempel oppdatere en rad. Denne låsen vil da holdes til transaksjonen er avsluttet.

Så vidt vi har greid å finne ut, bruker ikke Access timestamping som default. Tar forbehold om denne påstanden, men det vil i tilfelle bety at dersom to transaksjoner leser data fra Flygning-tabellen samtidig, blir den ene stående og vente mens den første transaksjonen kjører sine oppdateringer og gjør commit – deretter vil den utføre oppdateringer i Flygning-tabellen på feil grunnlag. Dersom det er riktig at den siste testen vår ikke slår til, vil vi få en overbooking. Brukes derimot timestamp, vil den andre transaksjonen kunne "se" at noen har lest/oppdatert dataene siden den leste dem inn, og en exception vil kastes.

Lost update problem

Om to stykker oppdaterer antallLedigeSeter-feltet i flygning-tabellen rett etter hverandre, vil det ikke ha noe å si med mindre verdien blir mindre enn 0 – feltet gis aldri en spesifikk verdi, det trekkes bare fra én. Problemet løses ved at vi gjennomfører en sjekk på verdien også etter at transaksjonen er gjennomført og kjører evt rollback – overbooking blir dermed fanget opp. En svakhet ved denne måten å gjøre det på, er at to stykker som forsøker å booke den siste billetten på en flygning begge kan få beskjed om at det ikke er plass – i så fall kan en heldig tredjemann få plass på et tilsynelatende "fullt" fly. Vi vurderer ikke dette som en stor svakhet i forhold til spart tid – det er også lite sannsynlig at to kunder prøver å gjennomføre denne sjekken nøyaktig samtidig særlig ofte. I alle tilfelle mener vi at å gi en tredjemann sjansen til å "snike" i køen er et bedre alternativ enn å overbooke!

Uncomitted data problem

Når alle transaksjoner har Read Committed som isoleringsnivå, vil ikke dette være et problem – Read Committed beskytter mot inkonsistens pga en oppdatering som senere blir kansellert med rollback.

Inconsistent data problem

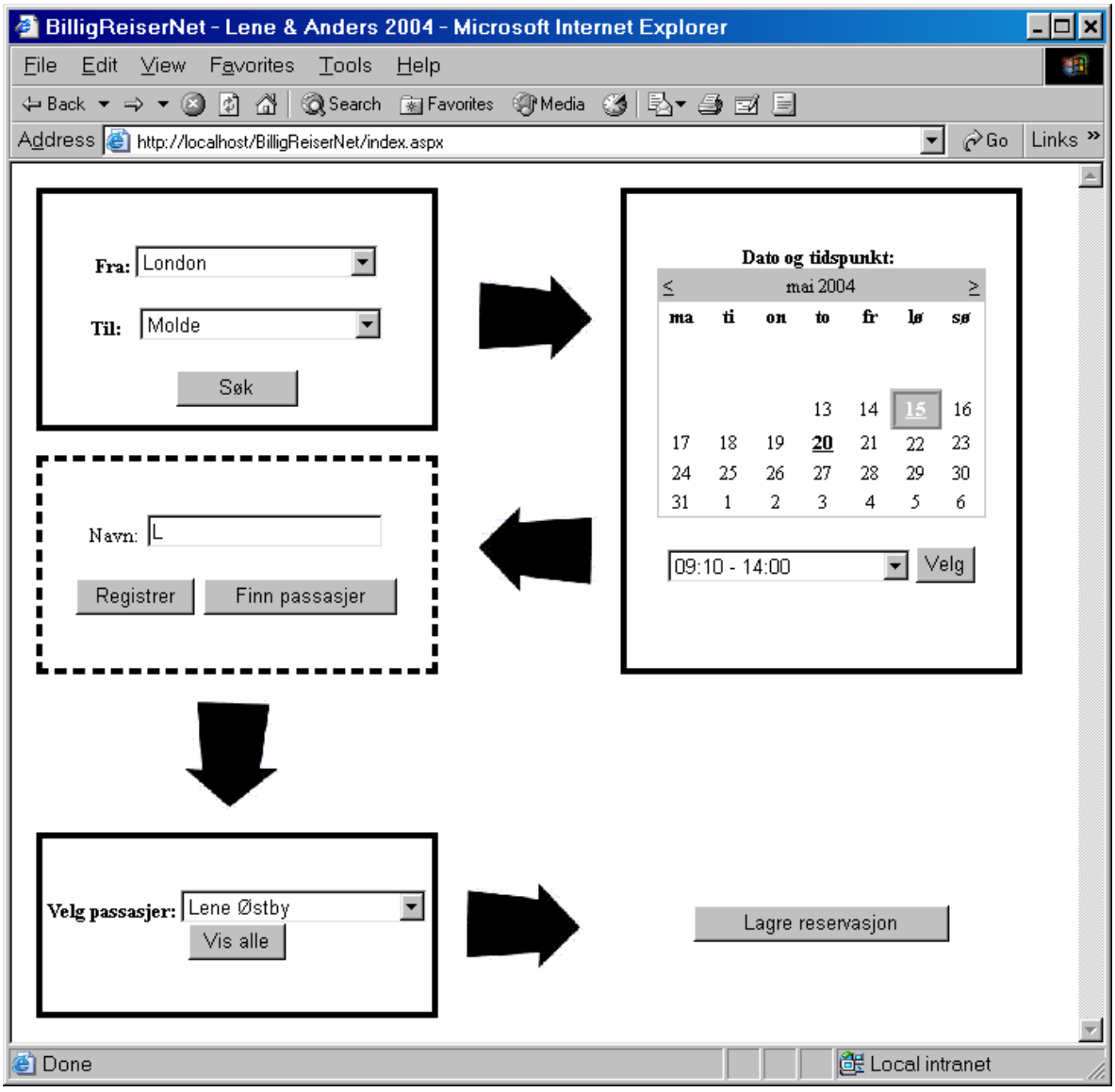
Dersom to transaksjoner leser antallLedigeSeter på en flygning samtidig, er det mulig for transaksjon nr 2 å oppdatere det samme feltet etter at transaksjon nr 1 har "tatt" den siste ledige billetten. Antallet ledige plasser transaksjon nr 2 forholder seg til vil dermed ikke stemme overens med virkeligheten, og vi får overbooking. Også dette problemet fanges opp ved å gjennomføre den samme sjekken på ledige seter både før og etter oppdatering på de aktuelle radene, slik at evt rollback kan utføres. Dette har den samme svakheten som ved lost update problem: dersom to kunder utfører denne siste sjekken samtidig, vil begge kunne få beskjed om at flygningen er full, men vi vurderer altså denne risikoen som relativt liten og uten alvorlige konsekvenser.

Phantom insert problem

Read committed beskytter ikke mot dette problemet, men i vår applikasjon vil det ikke ha noen betydning. Det er kun reise- og evt passasjertabellen det utføres insert-setninger på, og dette er det eneste som utføres på disse tabellene. Med den begrensede funksjonaliteten denne applikasjonen har, vil ikke fantomdata være et problem.

Oppgave 4 – ASP

Webløsningen



Webløsningen vår bruker de samme metodene som windowsversjonen, men vi har valgt å endre litt på den logiske rekkefølgen, og innføre noe mer tvang i brukergrensesnittet. Pilene har vi brukt i stedet for nummerering, eventuelt å dele det opp på flere sider. Man kunne også i den gamle løsnigen velge å gjøre det i samme rekkefølge som her. Vi synes løsningen gir god oversikt over flyten i bestillingen, hvor mange trinn som er igjen osv.

Første trinn er å velge fra- og tilsted. I windowsversjonen fyllte vi automatisk til-feltet med mulige byer, noe vi ikke har gjort her siden det ikke var ønskelig med "unødvendige" reloads av siden. Vi er ikke helt enig i det, i hvert fall ikke med den datamengden vi har. I stedet for en ekstra reload av

siden for å hente tilstedene, er det ikke uvanlig med 10-20 søk før man nå finner en rute som er mulig å gjennomføre, altså betydelig større ressursbruk enn den ene ekstra reloaden. Det er dog enkelt å endre mellom disse to teknikkene.

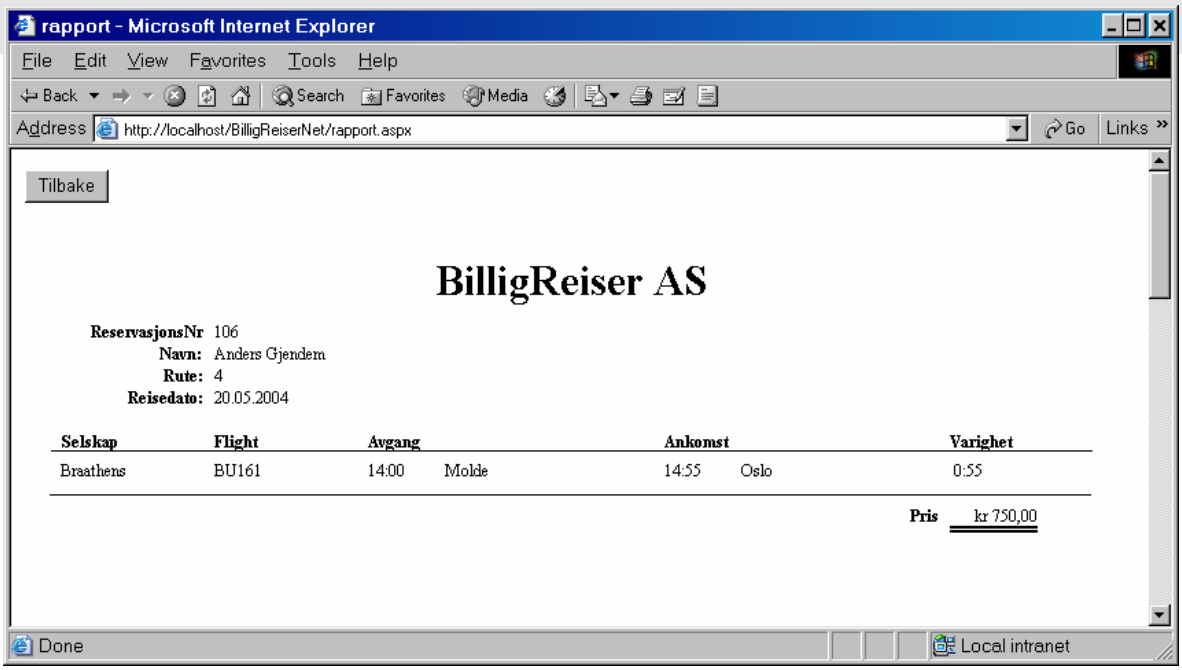
Så snart vi har identifisert en rute, blir kalenderen tilgjengelig. Datetimepickeren i ASP.net er betydelig enklere å tilpasse enn i windows, og vi har her valgt å fjerne alle datoer før i dag, og det er heller ikke mulig å velge andre datoer enn de datoene vi har en flygning på. Disse markeres bold og som en link, så her er webløsningen betydelig bedre enn den windowsbaserte. Her velger man også tidspunkt, og vi har dermed identifisert hvilken rute det er snakk om, nok en gang er ruteid det underliggende valget man gjør der.

Det neste trinnet er valgfritt. Det har vi forsøkt å vise ved at vi her bruker en stiplet linje rundt mulighetene, i stedet for den vanlige sammenhengende linjen. Valgene her er å enten registrere seg som en ny passasjer, eller å søke blant våre registrerte passasjerer.

I det neste punktet kan man da enten velge passasjer eller velge blant resultatet av søket. Hvis man har en nyregistrering, vil denne være automatisk valgt. Også her har vi en knapp som sørger for at hele listen over alle registrerte passasjerer vises igjen ved behov. Etter at dette er gjort, er det bare å trykke Lagre reservasjon for å gjennomføre den siste delen av registreringen. Den vanlige sjekken av ledige plasser utføres, og man får enten en feilmelding som under, eller man får se en bekreftelsesrapport. Feilmeldingen er minst like synlig som de vanlige meldingsboksene vi brukte i windowsversjonen, og er det eneste elementet som bryter med gråtone-stilen vi brukte. Rapporten er for øvrig akkurat den samme som sist, uten noen form for endring. Crystal Reports er utrolig fleksibelt på den måten.

Feilmelding på web

Rapporten på web



Eneste endring som er gjort er å legge til en tilbake-knapp i stedet for at man måtte lukke vinduet som i windowsversjonen. Det er under 100 linjer som er endret mellom windows- og webversjonen, og alt hadde med endring av søket mellom to steder, og noen små endringer i metodene til webkontrollene å gjøre, akkurat som forventet. Det bekreftet egentlig bare at vi har lykkes med bruken av datalag.

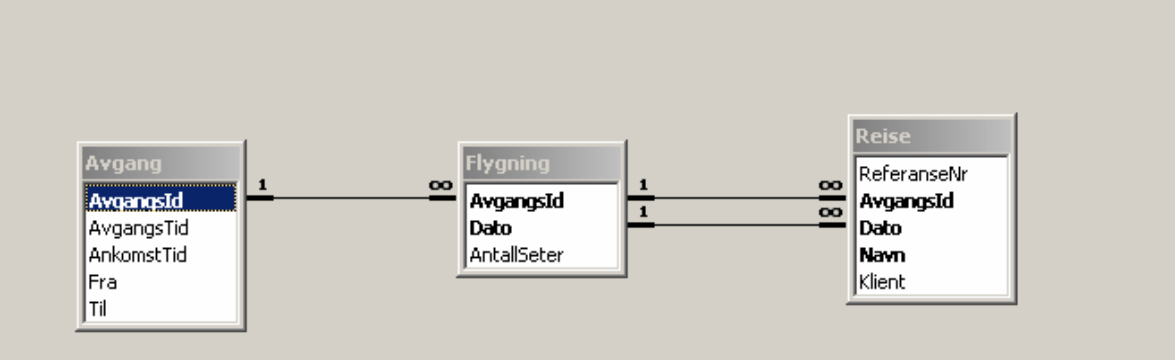
Oppgave 5 - Webservice

Webservicen har vi naturligvis ikke implementert med bruk av våre egne adaptere, siden disse er ment å være statiske hos en leverandør – i hvert fall fra vårt synspunkt. I praksis ville vi helt sikkert gjort en lignende løsning her som i resten av oppgaven. Vi valgte å basere den på en Access-database uten at det var noen spesiell grunn til det. Vi tok med oss de private standard-metodene våre fra Access-adapteret og brukte de om igjen her: getForbindelse(), dbNonQuery() og datoFormat(). Disse blir nok en gang brukt av WebMethodene i klassen til å utføre de aktuelle sql-setningene. Bestillingsfunksjonen er en lettere omskrevet variant av den originale i Access-adapteret, tilpasset de aktuelle data og tabeller. Disse kommandoene kjøres i en transaksjon. Både bestill og avbestill-metodene følger spesifikasjonen fra oppgaven.

Det første som skjer er at det kontrolleres at den valgte flygningen faktisk finnes på den aktuelle datoen, og at det faktisk finnes ledige seter, før insert-setningen kjøres og referansenummer hentes ut. Transaksjonsnivået som brukes er ReadCommitted, som er det høyeste tilgjengelige i Access – dette er lett å endre ved behov. Det ideelle ville nok vært å bruke en Oracle-database her også, gjerne med en stored procedure eller trigger som tok seg av alt.

Avbestilling er en ren sletting av bestillings-raden i databasen, etter valgt referansenummer, og returnerer en boolsk verdi. Alle hendelser i systemet skrives til debug-vinduet, og kan også lett logges, dette gjør feilsøking veldig enkelt. Flere creeping features vi gjerne skulle implementert er at ved benyttelse av webservice ville vi kontaktet alle serverne, lastet ned deres tilgjengelige avganger, og fått kombinert dette til vårt datagrunnlag. Slik det er nå, har vi vår database hvor vi manuelt har lagt inn tilsvarende avganger som de forskjellige flyselskapene og laget ruter basert på det, men denne utvidelsen kan fort bli mye jobb og hører ikke hjemme i denne prototypen.

Datamodellen til webservicen



Datamodellen i prototypen er minimal. Avgangstabellen ville naturligvis være knyttet til flyplass, og en klient-tabell ville vært aktuell. I reise har vi to fremmednøkler, som blir markert med to streker i Access, disse i kombinasjon med navn fungerer som primærnøkkel, slik at vi også i Access får en unik kombinasjon av avgang, dato og navn. Dette fordi det ikke skal være mulig å fly samme avgang mer enn en gang per dag, noe som naturligvis ville være umulig siden en avgang går på et bestemt tidspunkt på en gitt dag. Hver slik unik kombinasjon blir identifisert med et referansenummer, som er av typen autonummer. I tillegg registreres hvilken klient som har bestilt reisen.

Konklusjon

Det har vært interessant å jobbe med prosjektoppgaven, spennende å sette teori ut i praksis. Det skal sies at det ikke alltid har vært like lett å "finne igjen" prosjektoppgaven i forelesningene – vi synes prosjektet blir litt for fokusert på VB i stedet for databaser. Vi har allikevel lært mye, og synes det har vært et interessant prosjekt.

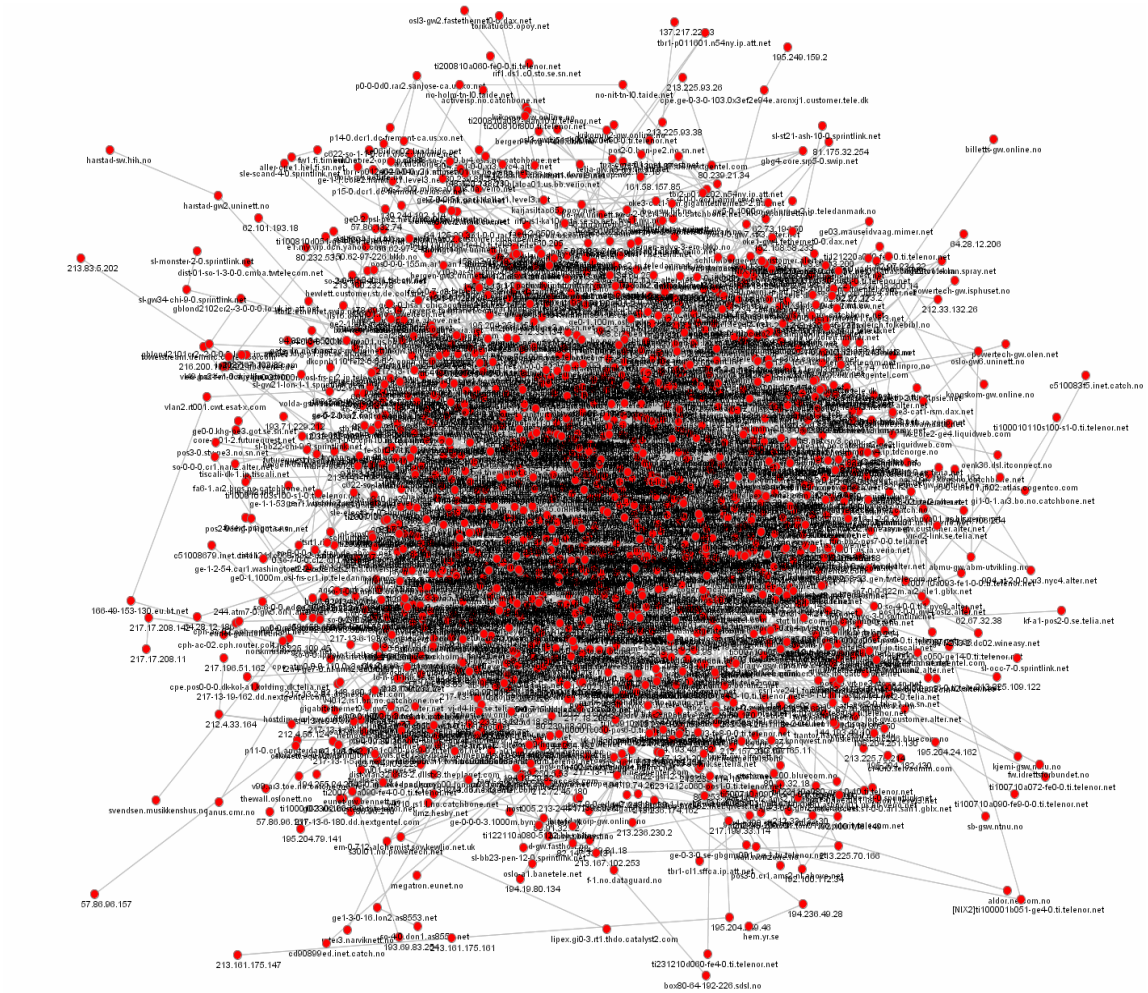
PS: Flink hjelpelærer!

4. IN765: Tree-Like Exploration of IP-Networks av Tommy Kvalvik

TREE-LIKE EXPLORATION OF IP- NETWORKS

MAKING A MAP OF THE INTERNET

By Tommy Kvalvik



INTRODUCTION

The Internet has grown to become the largest computer network in existence today, millions and millions of nodes like personal workstations, laptops, web servers, mobile devices interconnect together through a very complex infrastructure of high-speed links and routers. This infrastructure consists of smaller sub-networks governed by a large amount of independent Internet service providers that use the IP protocols maintained by IETF (Internet Engineering Task Force) to connect their networks together to form a single world-wide IP network.

The simplest way to map such a network would be to select a random number of nodes and follow the path to each node and record the jumps in between in a quantitative manner. Due to the huge size of the Internet it will be impossible to map everything, some kind of filtering is necessary. The result of this trace on a static network will be a tree or snowflake topology. However, the Internet is not static, the routers are constantly calculating the fastest, not the shortest, path between sub-networks and we can exploit this when we do our traces.

The creation and growth of a large network like the Internet will probably follow an exponential characteristic obeying a power law, making it a scale-free network. Finding the exact power law by tree-like explorations will not be 100% accurate due to missing links in the trace, according to [2] the degree distribution, λ, found will be significantly lower than the actual distribution. Others [3] have proven that even random or regular networks will exhibit a power law when parsed as a static tree. Previous attempts have usually estimated the exponent $\lambda = 2.2 \sim 2.5$

The IP protocols are engineered with redundancy in mind, but the general perception is that this has not been exploited in the Norwegian networks, instead of several networks connecting to each other in a 'scale-free' way has been replaced by a single hub, the NIX, Norwegian Internet Exchange, maintained by Uninett. The result is a 'snowflake' with the NIX routers in the middle. A map of the network should support this perception, but may also reveal alternate paths between ISPs and be helpful when we try to assess the importance of the NIX routers.

Once a map has been made it will be possible to get an overview of the topology and see how the main ISPs are connected, identify the most important nodes and locate vulnerabilities.

METHODOLOGY

First of all I need something to trace: A large list of IP-addresses. This can be obtained in many ways; I wrote a simple web-spider that collects URLs in a text file.

The second step was to translate the URLs into traceable IP-addresses.

Then each of the IP addresses where passed to a trace program and the output saved to a file.

The trace output is parsed into nodes and links and a virtual representation of the network can be built. When the network has been built we can start analyzing it using simple mathematics and graph parsing algorithms.

EXECUTION

SPIDER

The web spider was implemented as a Java program. It starts by downloading a predefined page and parses for links, for each link it finds it will extract and store the host address from the URL and enqueue the page for parsing. Pages that reside on hosts that have not yet been visited are placed in the front of the queue while the rest is placed in the rear. The queue is of finite size and pages will be removed from the rear when it grows out of bounds. This is an easy way to ensure that it doesn't get stuck parsing pages at the same host, popular sites like discussion forums and news sites can easily contain more than a million pages with news stories and discussion threads, parsing all of those is not very effective. Finally all the host addresses are written to a text file.

One issue I had to resolve was how to limit the search, the spider was already programmed to parse a maximum number of pages (limited by available memory), but letting it just wander off anywhere might result in a very sparsely connected network. One solution was to limit it to Norwegian or Nordic hosts only, but since IP addresses does not contain any geographical information and there is no correlation between the ranges assigned to different ISPs this was not an option. I choose to limit the search to hosts with a .no suffix. These hosts are not necessarily located in Norway, but their URL implies that they are interesting for Norwegian Internet users.

The spider was run on a virtual machine with 192MB of initially assigned memory. Java is not a very memory efficient programming language, but it was enough to parse 100 000 web pages residing on 1258 unique web servers.

PARSING AND MAPPING

The actual mapping of the network was done using Microsoft's tracert program included with all Windows versions. The tracert program is a pure command line tool, thus all input an output are in a pure text format. Although many traces are successful, a relatively large amount of them will always fail in reaching its target within the time available; there are a number of reasons for this:

- The host is not available at the time of the trace.
- Badly configured routers do not send correct ICMP responses.
- Bugs in well known operating systems send responses with too short lifetime. (namely the BSD 4.3 kernel)
- Strict firewall settings that do not allow ICMP traffic.

Regardless of these problems the output will always contain some information unless the trace ended at the first gateway (rare). Tracert was run once for each host identified by the spider. Since the routers update themselves constantly to find the fastest route to each of its neighbors the trace was run several times at different times of the day hoping that changing traffic patterns on the network would result in different paths.

The output from all the traces were written to a single text file and parsed by a second Java program that identified lines in the trace as nodes and consecutive lines as links. The program builds a object oriented representation of the network as it is parsed.

GRAPHICAL REPRESENTATION AND CALCULATIONS

The Java Universal Network/Graph Framework (JUNG) is, as its name implies, a powerful framework for representing and manipulating graphs using the Java programming language. It contains functionality for object representation of graphs, graphical visualization, many simple and some not so simple algorithms for calculating useful properties and a couple of samples of how to perform some useful tasks. The mapped network was translated into a JUNG graph before doing anything further. More information on JUNG can be found at http://jung.sf.net

Visual representation of the graph was done by rewriting one of the sample applications from JUNG into reading my network instead of the sample data. A few other things such as setting a smaller font for labels were also done. Drawing all the data from the network on a panel would not be very interesting since there is far too much data to be represented on any sensible sized display, however, by applying a filter to remove all the uninteresting/unimportant nodes, one can get an idea of how things are connected.

Which nodes are important and which are not? Albert-László Barabási has defined what we can call hubs as nodes with a central role in the network, connecting many others together. Part of the definition was that hubs have a high degree, that is, they are directly connected to a lot of other nodes. In the scale free Barabasi-Albert model where the only factor determining linkage is a power law, this will be true, but consider this; A computer lab or any other local network may connect a large amount of nodes to a single gateway, but this gateway is has only one connection to an ISP. Now this gateway has a very high degree, maybe 100 or even several hundred workstations connected, but it does not connect the rest of the internet to anything other than office PCs. We will, correctly, consider this gateway as being on the 'outskirts' of the network since it is of no significance to the rest of the network, even though it has a very high degree.

Measuring importance requires another approach, and that may be "betweeness centrality", this property detonates how many shortest paths in the network passes through this particular node, removing a node with a high betweeness centrality will extend or cut off many paths through the network. This is exactly the measure we need, and the necessary algorithms and filters are already implemented by JUNG. As the Internet is thought to be far more complex than the BA model I expect the between degree and centrality rankings to be significantly different from each other.

Other calculations like cluster coefficients, filter by domain-name and degree ranking are all trivial tasks that don't need an explanation.

RESULTS

SPIDER AND TRACE OUTPUT

The spider returned 1258 unique hosts that could be mapped after parsing 100 000 web-pages, the spider was only run once as I considered this to be enough.

With the addresses in hand, the trace was run 4 times on different times of the day, early morning, around noon, in the afternoon and late evening. The first trace identified 2215 nodes and 2343 links between them, after the second run the numbers are 2442 nodes and 2785 links, the numbers did increase, but not by much during the two last runs. The clustering factor of the entire network increased a little after every run indicating that alternate paths were found. The final result was 2492 nodes connected by 2917 links.

LOCATING HUBS

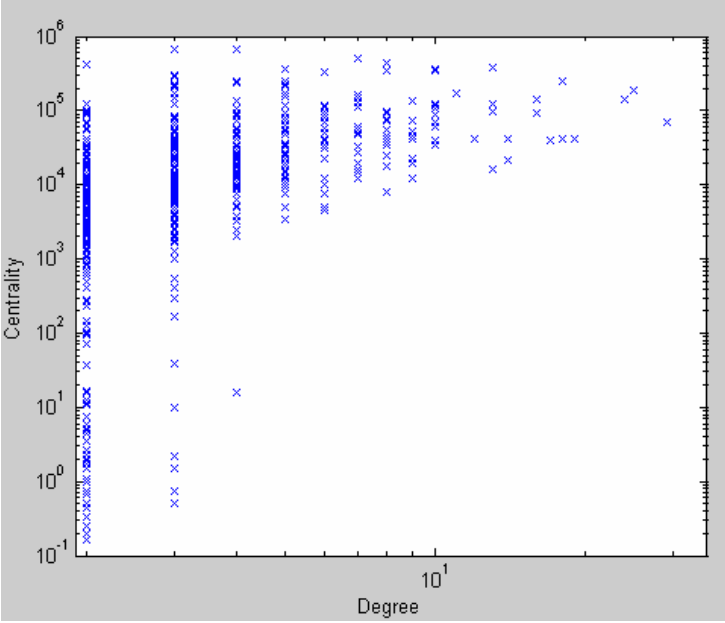
Having completed the trace it is now possible to calculate the centrality of the nodes. As stated earlier, there is a problem in how we define a node as a hub. I have ranked the nodes according to degree and betweenness centrality.

To make any sense of the list we will need to know how to read the DNS names, most of them will tell us which ISP or network operator maintain them, here is a short list of the most important ones:

- Uninett is the Norwegain educational and reaserch network and has grown to become the largest ISP in the country.
- NORDU is a joint network between the Nordic research and education networks (uninett, sunet, finet, RHnet and a few others) connecting these to the rest of the world. NORDU is also maintaining one of the DNS root servers.
- Level 3 is a commercial company maintaining one of the largest physical IP networks on the Internet, spanning from the US across at least two transatlantic links into Europe.
- Telenor is the largest commercial ISP in Norway and the second largest overall.
- [NIX1] Various routers connected to the Norwegian Internet Exchange.

Betweenness centrality		**Degree**	
Node	**Score**	**Node**	**Score**
Oslo-gw1.uninett.no	2553119	oslo-gw1.uninett.no	41
se-kth.nordu.net	1661438	vlan101.msfc0.ds1.osl2.alter.net	39
no-gw.nordu.net	1377487	activeisp.no.catchbone.net	29
so-1-0.hsa2.stockholm1.level3.net	673157	[NIX1]s01i01.no.powertech.net	25
s-b4-pos2-3.telia.net	672617	194.19.81.46	24
[NIX1].gaus.no.catchbone.net	495183	213.236.212.110	19
ti200002c050-ge14-0.ti.telenor.net	435782	gw2-0.webhuset.no	18
[NIX1].ti.telenor.net	426553	as-0-0.bbr1.newyork1.level3.net	18
[NIX1]oslo-nix1.banetele.net	372492	s10i01.no.powertech.net	17
[NIX1]nyd-cr1.ip.teledanmark.no	359255	bix.eltelevest.no	16

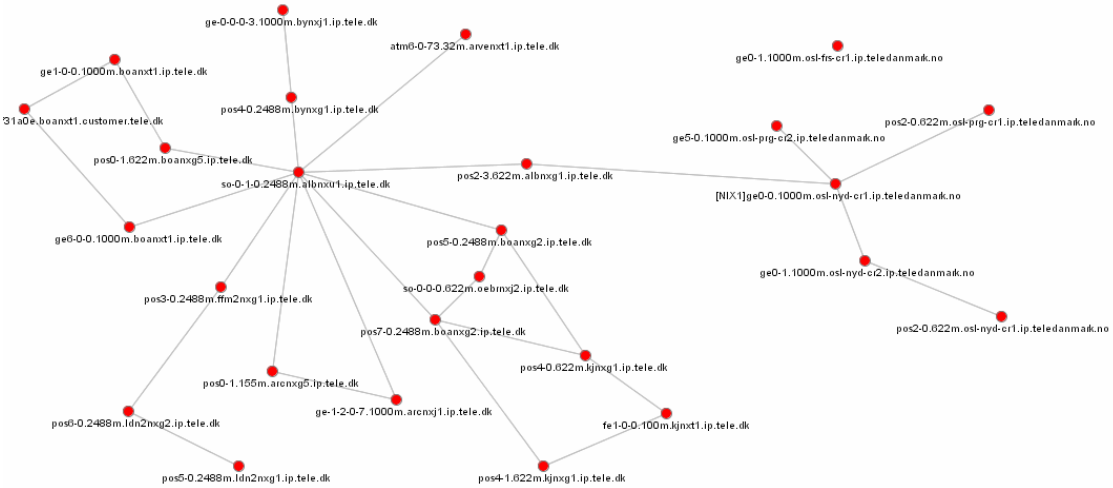
We can se that there is only one node represented on both rankings; this would imply that one of the measures does not represent the importance of the nodes very well. We can identify all the major networks in the centrality list, and four NIX entries support the theory of its importance. Ranking of degree does not reveal any pattern, only three nodes belong to networks that appear on the centrality ranking. To get a better view of the situation consider the following chart.



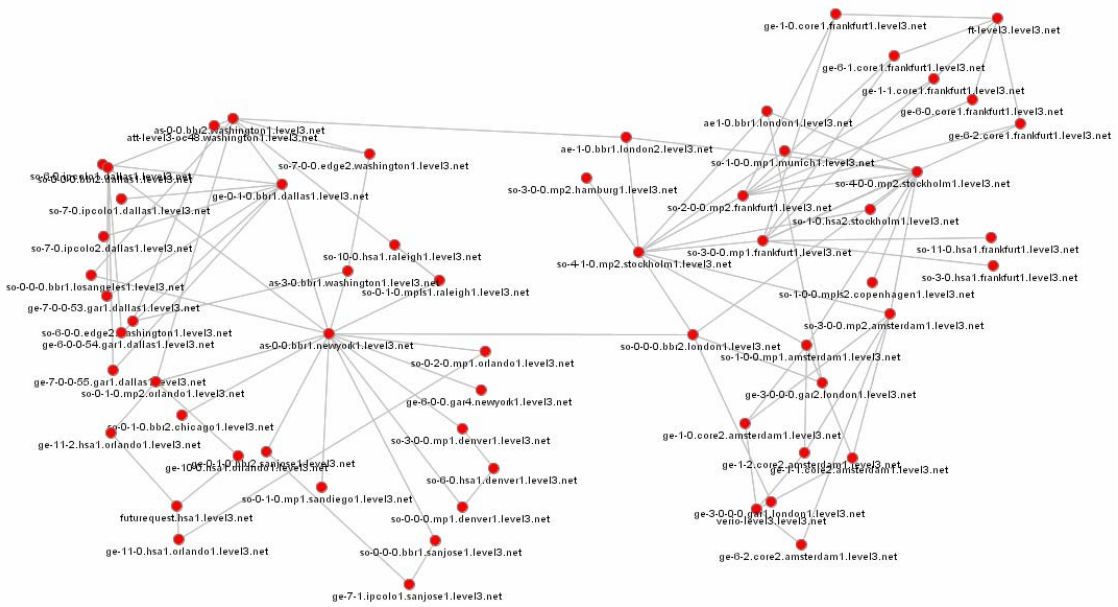
While there seem to be central nodes all over the scale, high degree nodes tend to have a higher probability of being central than nodes with a lower degree. This is perfectly sensible, but considering the fact that a lot more traces have passed through the center of the network then the edges, there is a higher probability of detecting a high degree among the center nodes. The tendency has been amplified significantly by this fact.

LOCATING AND EXPLORING THE INDIVIDUAL NETWORKS

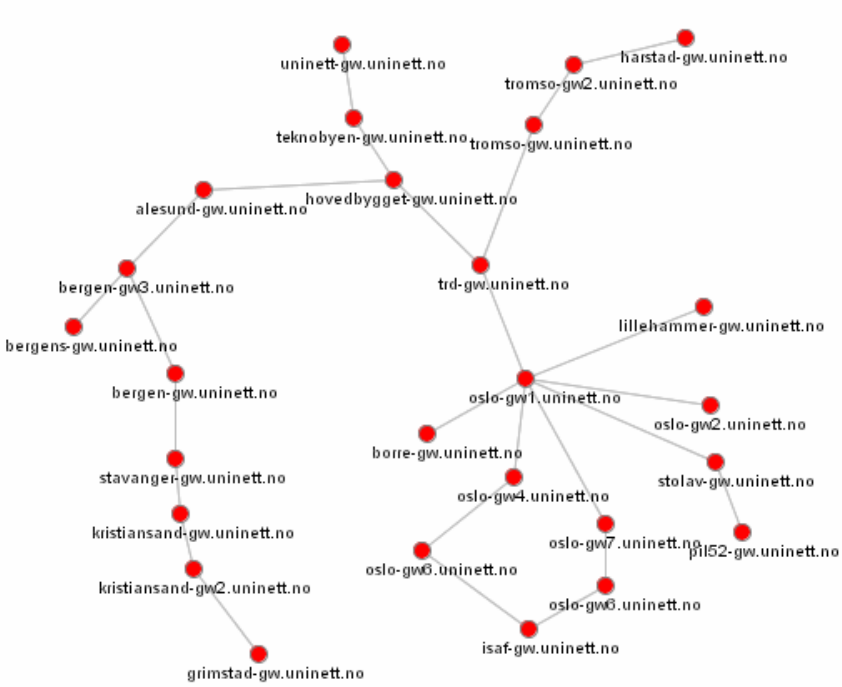
Using the DNS suffix of the nodes it is simple to filter out nodes belonging to a specific ISP or network. By using the filtered data, a map can be drawn. Most of the networks contain too many nodes to visualize, so all the networks drawn are filtered by centrality and the nodes that does not belong to paths between other nodes are removed since they contain the least information on the topology. "Tele Danmark" has an interesting network:



Even though this is a very small part of the total Internet, engineered and governed by a single ISP we can see that, without doing any calculations, it looks like a scale free network, there are two clusters, one located in Norway (to the right) and the other is the home network in Denmark. Level 3 is another network that exhibits almost exactly the same properties in a bit larger scale, the cluster on the left is their US network and the one on the right is located in Europe:



This is common to nearly all the ISPs, some consist of only one highly connected cluster, others of several, but the characteristics of the clusters are the same with few exceptions. Uninett (Figure 3) is one such exception:

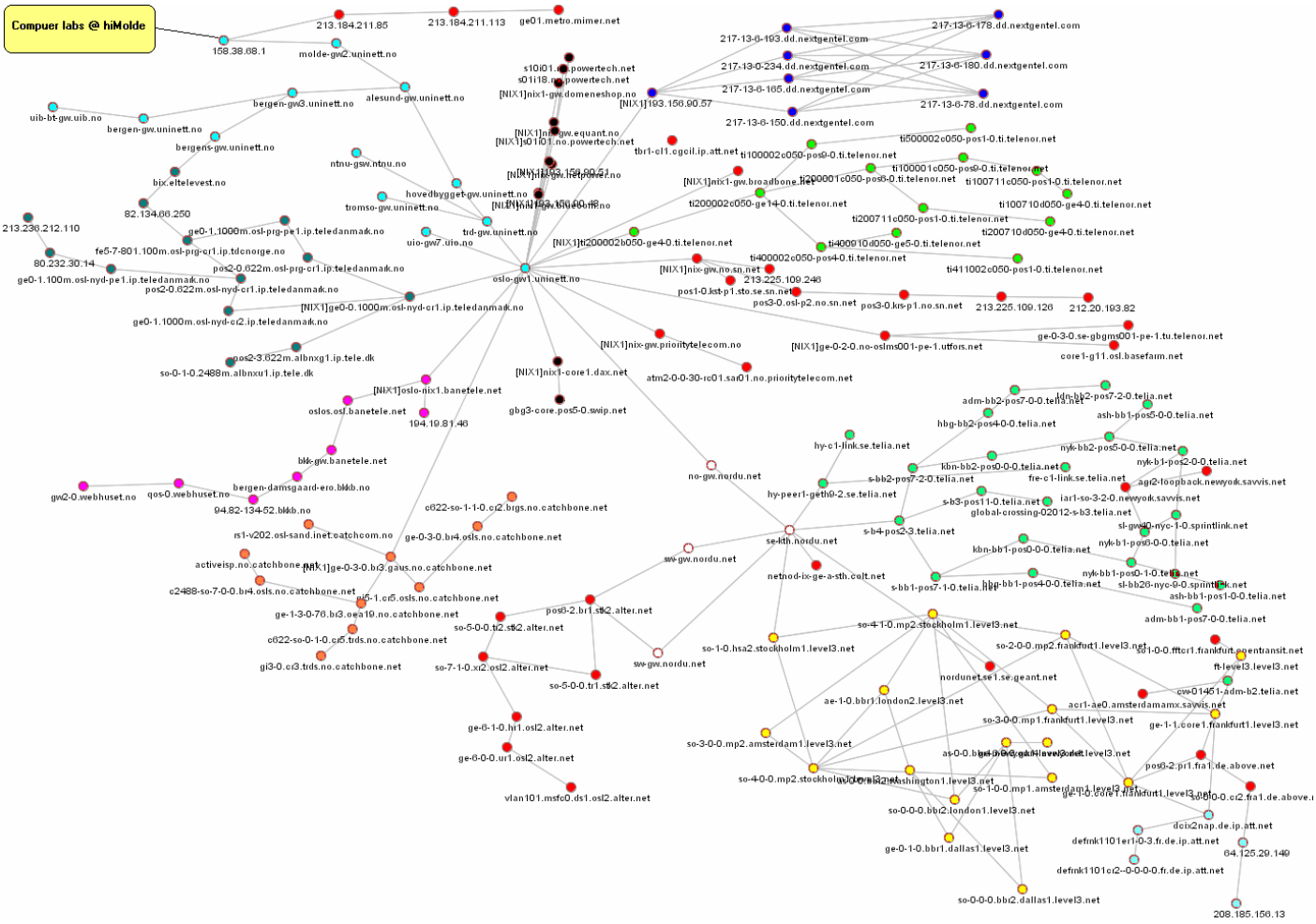


The core of the mapped part of Uninett consists of only one node connected to more than two others suggesting that the oslo-gw1.uninett.no is absolutely vital for holding the network together.

THE FULL PICTURE

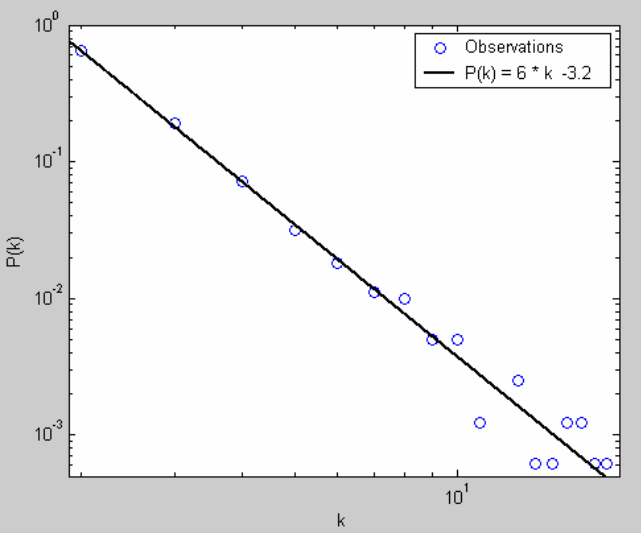
Using the same kind of filter on the entire network keeping only the most central nodes (the ones that make up the most 'shortest paths') produced a surprisingly well connected, even the ten most central nodes would form a completely connected network of their own. It is also clear that the NIX routers are essential to keep the Norwegian ISPs together; the only alternate path is between Uninett and TeleDanmark through ElTele Vest, a small ISP that only operates in western Norway and thus does not have direct NIX access.

Any correlation between the centrality and degree of the nodes cannot be found, several important nodes are not connected to more than two or three others. This confirms the theory that the network follows additional rules to the BA model.



LINK DISTRIBUTION

Link distribution in scale free networks follow a power law, that is, the probability of a node being connected to k other nodes can be expressed as $P(k) = C * k^{-\lambda}$ where λ denotes the degree exponent. The exploring algorithm used considers the web to be a spanning tree; it will detect a relatively higher portion of the links of central nodes than links of nodes around the edges, thus boosting the exponent found. Previous research has estimated the real exponent $\lambda = 2.5$, I expected to find a degree somewhat higher than this.



The degree found was 3.2. Estimating using the expression
$1 - \log(3.2) / \log(2.5) = 0.27$
the minimum amount of links needed to decrease the exponent to 2.5 is 3704 while the number of links found was 2917. The real exponent of the Internet is probably closer 2.2 than 2.5 making the actual number of links as high as 4300 for the tiny part mapped, the number we found is more than 65% of this.

CONCLUSION

Mapping a network using purely tree like methods is possible, and due to the dynamic changes in routing tables and shifting traffic we can get a more diverse view of the network by performing multiple traces to each node. This has been proven by the fact that the links found divided by possible links increased every time the trace was repeated.

There is a problem that many links don't get detected, especially 'horizontal' links and the further away from the center of the network, the less probability there is for detecting links. The center of the network is much more complete. Hubs can not be located by looking at degree, but more complex algorithms like betweeness centrality has proven to rank the nodes in a very useful way.

Even though this was a small experiment conducted over a short period of time and all traces conducted from the same location the method was able to detect more than 60% of the estimated number of links. The fact that it is possible to use this map to confirm facts we already know about Internet topology would suggest that the method could be useful.

REFERENCES

[1] T. Peterman, Paolo De Los Rios, Exploration of Scale-Free Networks (2004)

[2] A. Clauset, C. Moore, Traceroute sampling makes random graphs appear to have power law degree distributions (2003)

[3] Barabási, Albert-László, Linked (2003)

[4] Traceroute Unix manual

[5] Network operator websites (www.uninett.no, www.nix.no, www.nordu.net, www.level3.net)

[6] Java Universal Network/Graph Framework website (http://jung.sf.net)