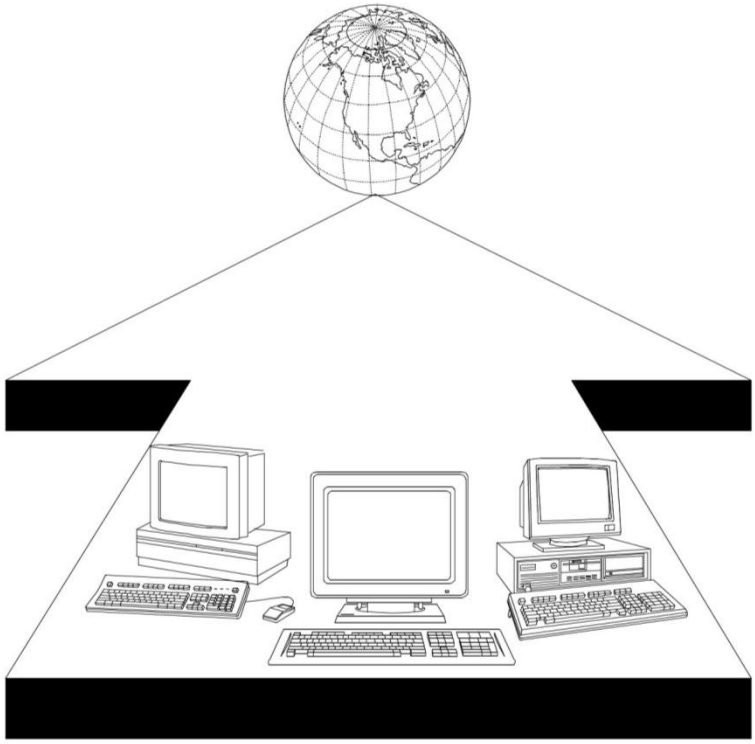
DISTRIBUERTE TJENESTER

En modell for tilgang til tjenester i et datanett med mange typer arbeidsplassmaskiner

Hovedfagsoppgave



Peter Hausken

Våren 1992
Institutt for Informatikk
Universitetet i Oslo

Innhold

1. INNLEDNING

Elektronisk databehandling har gjennomgått en rivende utvikling de siste 30 årene. Bruken av datamaskiner har endret seg, både med hensyn på hva vi bruker dem til, og hvordan vi bruker dem. I mange tilfeller er ikke databehandlingen lenger konsentrert om en sentral stormaskin. I stedet brukes mange maskiner til forskjellige oppgaver. Disse har etter hvert blitt knyttet sammen i nettverk og kan dermed dele oppgaver og ressurser seg i mellom.

Problemet er at disse maskinene ofte er svært forskjellige. Noen maskiner er velegnet til enkelt kontorarbeid, andre til større matematiske beregninger og atter andre til administrative oppgaver. Til tross for ulike operativsystemer og maskinarkitekturer, har man behov for å utveksle informasjon mellom ulike systemer. For å kunne utnytte de mulighetene som finnes for distribuerte tjenester, må maskinene integreres i et helhetlig databehandlingsmiljø, som gjør at oppgavene og ressursene kan deles mellom alle maskintyper. Det er gjort mye i retning av distribuert databehandling. Jeg skal i denne oppgaven beskrive noen av de løsningene som finnes og peke på problemer som gjenstår å løse.

1.1. Bakgrunn for oppgaven

I mer enn to år har jeg jobbet for Universitetets Senter for Informasjonsteknologi (USIT tidligere USE) ved Universitetet i Oslo. Arbeidsområdene har vært datanettverk generelt, og integrasjon av Pc-er spesielt. I arbeidet med å bringe tjenester ut til Pc-er har jeg erfart noen av de problemene vi står ovenfor innen databehandling i et distribuert maskinmiljø.

Jeg har vært så heldig å få jobbe med problemstillingen på to plan. Som informasjonsmedarbeider for det norske datanettet for forskning og utdanning - UNINETT, og som prosjektleder og medarbeider i flere prosjekter for distribusjon av tjenester til Pc-er og Macintosher. Denne dobbeltrollen har i stor grad bidratt til problemstillingene i oppgaven.

1.2. Mål for oppgaven

Oppgaven tar sitt i utgangspunkt i distribuerte systemer; hvilke tjenester som lar seg distribuere og hvordan. Med mange ulike maskintyper er det viktig at de kan integreres i et felles data-behandlingsmiljø. Det finnes en lang rekke løsninger for dette. Jeg vil beskrive hvilke løsninger USIT og andre forsknings- og utdanningsmiljøer har funnet hensiktsmessige og gi referanser til hvor disse er beskrevet mer utfyllende.

Universitetets Senter for informasjonsteknologi har kommet langt med å integrere ulike maskinplattformer i et distribuert system. Brukerne ved universitetet har fått helt nye muligheter for bruk av sine personlige datamaskiner til. Mange oppgaver er blitt betraktelig raskere å utføre. De kan i dag fra en hvilken som helst UNIX-maskin, VAX/VMS-maskin, Macintosh eller MS-DOS PC tilknyttet nettet:

- ha tilgang til de samme filområdene på lokale og felles filtjenere ved UiO og dermed slippe å flytte filer fram og tilbake.

- skrive ut til nesten alle skrivere tilknyttet datanettet ved UiO, samt enkelte skrivere ved de andre universitetene i Norge.

- sende og motta elektronisk post over hele verden.

- sende telefaks til en hvilken som helst telefaksmottager over hele verden.

- foreta filoverføringer til maskiner verden over.

- benytte sin arbeidsplassmaskin som terminal mot maskiner verden over.

- lese og skrive artikler i nyhets- og konferansesystemer

Stadig flere tjenester vil bli distribuert til brukerne. Blant de ting det jobbes med, er å lage et felles hjelpe- og informasjonssystem for Universitetet. Her skal brukeren kunne finne informasjon om hva som skjer hvor, få elektronisk hjelp på ting som f.eks. tilgjengelig programvare og tjenester i nettet, hvem man kan kontakte for mer informasjon og andre ting som måtte være av interesse. Med dette systemet håper man at informasjonen kan oppdateres ett sted og samtidig være tilgjengelig for alle maskinplattformer.

Som i mange andre sammenhenger har vi sett at løsningen av ett problem har avdekket nye. Mye arbeid gjenstår før vi kan gi brukerne et fullgodt databehandlingssystem som redskap i forskning, administrasjon og andre oppgaver. Brukerne har tilgang til et verdensomspennende nettverk, men hvordan man finner informasjon, personer eller tjenester i nettverket er langt fra klart. For eksempel:

- Hvordan skal brukeren finne fram til informasjonen og tilgjengelige applikasjoner for å få tilgang til informasjonen i et verdensomspennende datanett?

- Hvordan skal forskere (og andre) i forskjellige land på en enkel måte kunne samarbeide og dele resultater ved hjelp av datamaskiner, uten å være data- og nettverkseksperter?

- Hvordan skal brukeren benytte sitt arbeidsplassutstyr med maksimal brukervennlighet mot systemer på andre maskiner?

- Hvordan skal vi kunne dra nytte av den kunnskap brukeren har tilegnet seg ved bruk av sine lokale systemer, slik at de også kommer til nytte ved bruk av andre systemer?

- Hvordan kan vi spare brukere og systemadministratorer for utgifter og arbeid på halvgode, temporære løsninger for å få tilgang til et eller annet system ute i verden?

- Etter at man har lokalisert det man er ute etter; hvordan få tilgang til systemet? Hva hvis eieren av systemet krever betaling for bruken?

- Er det mulig å bygge opp felles arkiver innen en organisasjon, på et fagfelt, eller innen et geografisk område som er tilgjengelig fra mange type utstyr? Uten at dette krever tilpasninger for den enkelte maskin?

- Kort sagt; hvordan skal vi dele all den informasjon som finnes, på en enkel og brukervennlig måte?

Utviklingen av brukervennlige systemer som benytter vindusteknikker og avansert grafikk er kostbart og tidkrevende. Bill Joy hos Sun Microsystems hevder at 75 % av all kode i et moderne

program er brukes til å styre brukergrensesnittet. Ting som å manipulering av vinduer, rullegardinmenyer, ikoner, flytting av mus, etc. [COAD 91]

Det sier seg selv at hvis kostnadene og tidsforbruket ikke skal bli for stort, må det gjøres valg med hensyn på hvilket utstyr man skal utvikle systemet på og hvilke typer arbeidsplassutstyr man skal kunne benytte mot systemet. Med dagens teknologiske utviklingstakt er det utstyret man har mulighet til å velge nærmest foreldet, eller i hvert fall ikke i henhold til tidens krav, når systemet er ferdig utviklet. Det er et ordtak innen amerikansk dataindustri som sier: "*When it finally works, it is obsolete*". Å utvikle nye brukergrensesnitt for flere maskinplattformer er en så formidabel og fordyrende oppgave at få vil ta seg råd til å gjøre det. Dessuten krever det at systemet lages så modulært at det i det hele tatt lar seg gjøre uten en komplett redesign. For eksempel kan ikke en IBM stormaskin, en CYBER eller en annen stor flerbruker datamaskin, tilby brukergrensesnitt som kan konkurrere med Macintosh eller en X-terminal i brukervennlighet. Men disse systemene har til gjengjeld ikke tilstrekkelig datakraft til å kjøre de store systemene.

Oppgaven tar sikte på å skissere en modell av et system for hvordan ulike datamaskinplattformer kan benyttes til de samme oppgavene, mot de samme dataene, uavhengig av geografisk plassering, og uten å måtte programmere spesielt for hver maskinplattform. For hver applikasjon som lages, skal det holde å lage brukergrensesnittet én gang. Uansett hvilken maskinplattform applikasjonen benyttes mot skal den kunne fungere så godt det er mulig gitt de teknologiske begrensningene på hver maskinplattform.

Å gi den komplette løsningen vil overskride grensene for en hovedfagsoppgave. Det blir derfor en skisse av hvordan det kan gjøres, ikke en fullstendig, implementerbar beskrivelse av systemet.

1.3. Målgruppe for oppgaven

Oppgaven retter seg primært mot dem som jobber med integrasjon av ulike maskinplattformer i et distribuert datanett. En annen viktig gruppe oppgaven retter seg mot er informasjonsleverandører og organisasjoner som driver databasesystemer for salg i et stort marked. Den kan kanskje også ha interesse for dem som er opptatt av brukergrensesnitt i heterogene datamaskinmiljøer.

1.4. Strukturen i oppgaven

I kapittel 2 vil jeg definere viktige begreper, og gi motivasjonen for hvorfor distribuerte systemer ofte kan være en bedre løsning enn store maskiner.

Kapittel 3 tar for seg dagens muligheter for distribuerte tjenester i datanett. Distribuerte systemer har fått vind i seilene de siste årene, da særlig innen forsknings- og utdanningsmiljøer i den vestlige verden. Dette gjør at utviklingen går fort framover, og deler av det som beskrives er tatt med for å gi et bilde av dagens muligheter og begrensninger. Disse kan fort forandres ved at nye løsninger og systemer tas i bruk.

I kapittel 4 ser jeg på hvilke typer brukergrensesnitt som finnes, hva som skiller dem og hva de har felles. Dette for at leseren skal kunne følge med i den videre behandlingen av det jeg har kalt ***den distribuerte applikasjonsmodellen***. Videre tar kapittel 4 for seg de mest brukte datamaskintypene som er i utbredt bruk. Hensikten er å vise hvilke egenskaper de ulike maskinplattformene har og hvilke muligheter de har til å integreres i et distribuert system.

Kapittel 5 beskriver ulike modeller for hvordan brukeren kan benytte applikasjoner og hvilke fordeler og ulemper som er knyttet til hver av modellene. Gjennom eksempler vil jeg forsøke å vise at selv den populære klient-tjener modellen ikke ubetinget holder mål i lengden.

Kapittel 6 tar for seg de ideelle målsettingene for ***den distribuerte applikasjonsmodellen***. Hva bør den kunne håndtere og hvordan kan dette løses med de teknologiske løsningene vi har.

Kapittel 7 gir det teoretiske fundamentet for objektorientert tankegang, og en motivasjon for hvorfor jeg har valgt en slik løsning. Videre vil jeg skissere hvilke komponenter en distribuert applikasjonsagent må bestå av, og hvordan disse kan implementeres i en objektorientert løsning. Dette kapitelet viser hvilke hovedklasser en applikasjonsagent bør/må ha og hvilke mekanismer som bør være tilgjengelige.

Kapittel 8 vil belyse ulike anvendelser av den distribuerte applikasjonsmodellen gjennom eksempler, og peke på viktige aspekter ved den.

2. Distribuerte systemer

Distribuerte systemer er et forholdsvis nytt område og jeg har ikke vært i stand til å finne en definisjon som dekker det jeg mener inngår i begrepet. [SLOMAN 87] gir følgende definisjon:

> *A distributed processing system is one in which several autonomous processors and data stores supporting processes and/or databases interact in order to co-operate to achieve an overall goal. The processes coordinate their activities and exchange information by means of information transferred over a communication network.*

P. H. Enslow gir i sin artikkel "What is a distributed system" [SLOMAN 87] definisjoner av distribuerte systemer, alt etter i hvor stor grad de er desentralisert. Kravene til systemet er at det består av heterogene generelle prosessorer som er autonome og samarbeidende, og at ingen data dupliseres, men hentes fra kilden. I tillegg forutsetter Enslow at distribusjonen er transparent, slik at brukerne ikke skal behøve å merke at flere prosessorer er involvert.

Jeg vil i de kommende avsnitt vise til noe av det som er skrevet om distribuerte systemer, og peke på hvilke krav som stilles til dem, samt hva som anføres som fordeler og ulemper med distribuerte systemer.

For at distribuerte systemer skal kunne bli effektive og brukervennlige må de konstrueres på en måte som gjør at tilgangen til systemene blir transparent. Det vil si at selve nettverket og maskinene er skjult. Hvilke maskiner som gjør hva, og hvordan de er koblet sammen, skal ikke ha innflytelse på bruken av de tjenestene som finnes i det distribuerte systemet. En oppdeling i former for transparent tilgang er beskrevet i [ANSA 87]. Man opererer med følgende åtte former:

- **Tilgangstransparent.** Muliggjør tilgang til filer og andre tjenester på andre maskiner ved hjelp av samme operasjoner som for tilgang til de som befinner seg på den lokale. Operasjonene er uavhengig av hvilken maskin de utføres fra.

- **Lokasjonstransparent.** Muliggjør tilgang til tjenester uten å ha kjennskap til hvor de fysisk befinner seg.

- **Samtidighetstransparent.** Muliggjør at flere brukere eller programmer kan aksessere felles data uten at de kommer i konflikt med hverandre.

- **Kopitransparent.** Muliggjør flere instanser av filer eller andre data for å øke ytelse og pålitelighet uten at brukerne eller applikasjonsprogrammene har kjennskap til disse kopiene.

- **Feiltransparent.** Muliggjør tildekking av feil. Tillater brukere og applikasjons-programmer å fullføre sine oppgaver til tross for feil på maskinvare eller programvare komponenter.

- **Migrasjonstransparent.** Tillater flytting av tjenester innen systemet, uten at dette merkes av eller påvirker brukeren og utførelsen av applikasjonsprogrammene.

- **Ytelsestransparent.** Tillater systemet å bli rekonfigurert for bedre ytelsen ettersom bruken endres.

- **Skaleringstransparent.** Tillater systemet og applikasjonene å utvides uten endring av systemets struktur eller applikasjonsalgoritmer.

[COULOURIS 88] deler opp lokasjonstransparent tilgang i tre nivåer.

- På nivå 0 må brukeren vite på hvilken maskin ting befinner seg. Det finnes mulighet for overføring av filer og eksekvering av programmer på andre maskiner. Terminalaksess, filoverføring og UNIX programmer som rsh (*remote shell*), rexec (*remote execute*) og rcp (*remote copy*) er eksempler på nivå 0.

- På nivå 1 finnes visse applikasjonsprogrammer som dekker over det faktum at tjenestene befinner seg på forskjellige maskiner. Elektronisk post er et eksempel på dette. På nivå 1 er det stort sett overlatt til programmereren å påse at det fungerer transparent i et flermaskinsmiljø.

- På nivå 2 finnes støtte i tjenermaskinene for å tilby generell tilgang og deling av data og ressurser i hele nettet. Dette skjuler flermaskinsmiljøet for applikasjonsprogrammene. Med mindre programmereren bevisst velger å gå utenom de tjenestene systemet tilbyr, vil applikasjonen ikke være klar over hvordan maskinvaren er konfigurert, fordi tjenestene er felles for hele systemet. Suns Network File System (NFS) er et eksempel på dette (Se avsnitt 3.3.1).

2.1. Hvorfor ha distribuerte systemer

Det er en rekke faktorer som gjør distribuerte systemer stadig mer fordelaktige i forhold til konvensjonelle tidsdelte stormaskiner. [SLOMAN 87] [COULOURIS 88]

- **Pris/Ytelse.** Prisen på prosessorer og lager har sunket drastisk de siste årene og gjør det fortsatt. Forholdet mellom pris og ytelse er endret dramatisk i løpet av de siste 5-10 årene. Dette har gjort det fordelaktig med mange mindre datamaskiner framfor en stor. Prisen på periferiutstyr som laserskrivere, datalager osv. har ikke sunket like drastisk. Med distribuerte systemer, har man muligheten til å dele dyrt utstyr fra mange maskiner.

- **Modularitet.** Distribuerte systemer må konstrueres svært modulært. Hver komponent må ha et grensesnitt og sett med tjenester som er veldefinert ovenfor resten av systemet, for at det skal kunne fungere. Dette fører til enklere design, installasjon og vedlikehold. Dessuten blir det lettere å verifisere at hver komponent fungerer som den skal.

- **Skalerbarhet.** Distribuerte systemer fordrer modularitet både i program- og maskinvare. Dette gjør det lett å legge til eller fjerne komponenter, uten at det trenger å påvirke resten av systemet. En ny maskin kan føyes til når ny prosessorkraft trengs. Det er derfor enkelt å starte med en liten konfigurasjon, og så utvide etter behov. Ved å benytte standard kommunikasjonsprotokoller og distribuerte tjenester er det også mulig å koble sammen maskiner fra flere leverandører til et system. Bindingen til en leverandør blir dermed mindre. Dette er ting som gir distribuerte systemer god fleksibilitet og store utvidelses-muligheter.

- **Teknologi.** Nettverksteknologi er blitt hyllevare og til en overkommelig pris. Dette muliggjør sammenkobling av mange små datamaskiner i et nettverk.

- **Tilgjengelighet** er i mange sammenhenger svært viktig. Når en stormaskin ikke er i normal drift gir det ringvirkninger for alle. Hvis deler av et distribuert system faller ut, vil det som er igjen fortsatt kunne fungere. Konsekvensene av at en maskin i et distribuert system faller ut er som oftest er langt mindre.

- **Fleksibilitet.** Tjenestene fra de tradisjonelle stormaskinene med mange samtidige brukere gir ofte dårlig og uforutsigbar responstid. Det er vanskelig å konfigurere maskin- og programvare etter brukernes ønsker, og oppgradering av maskinvare krever gjerne større investeringer.

- **Desentralisering.** Det kan være gunstig å plassere datakraften der den skal brukes. I stedet for en eller flere sentrale stormaskiner, har det vært sterkt press mot å desentralisere datakraften ut til brukerne. Desentraliseres datakraften ut til brukerne gir det gjerne bedre fleksibilitet i bruken og mer optimal utnyttelse av kommunikasjonsressursene. Geografisk spredning gjør også at man er bedre sikret mot konsekvensene av brann, sabotasje, lokale strømbrudd og andre katastrofer.

- **Bruker krav.** Økende spredning i ønskene fra brukerne om applikasjoner og muligheter. For eksempel til grafiske grensesnitt, bruk av mange skrifttyper, interaktiv grafikk, avansert informasjonssøking osv. De tradisjonelle stormaskinene har som oftest begrensninger i type brukergrensesnitt som kan lages (se Kapittel 6).

Det finnes også argumenter som fortsatt taler for sentraliserte løsninger:

- **Samlet ytelse.** Store sentraliserte datamaskiner har mer datakraft samlet som kan utnyttes til en enkelt operasjon. Ting som avanserte beregninger, eller håndtering av store databaser kan være enklere å gjennomføre på en stormaskin enn på mange små. Noen programmer er så omfattende at de ikke kan kjøre på annet enn en super datamaskin. Dette gjelder særlig en del vitenskapelige beregninger.

- **Driftskostnader.** Kostnadene ved sentralisert drift holdes lettere nede, enn ved en desentral drift. Det vil ikke være mulig å ha alle typer spesialisert personell ved hver maskin. En løsning på dette kan være sentral drift ved hjelp av nettverk. Å opprettholde en nødvendig fysisk sikkerhet rundt hver enkelt maskin blir også dyrere hvis maskinene står spredd på mange ulike steder.

- **Faglig miljø.** De som jobber med drift, utvikling og vedlikehold av maskinene, vil ha en større faglig tilfredsstillelse ved å være i et stort fagmiljø, enn spredd ut på hvert sitt sted. Igjen, distribusjon av maskinene, trenger ikke nødvendigvis å implisere distribusjon av personellet. Det er mange ting som taler for å holde dem samlet. Faren er stor for at den enkelte sitter og utvikler egne løsninger. Det kreves sterk sentral styring med hensyn på hvilke standarder som skal følges. Dette for at det totale systemet ikke skal bli fullt av lokale tilpasninger som ikke fungerer sammen i den helheten det er ment å være.

- **Kompleksiteten i datanettet** er blitt lett undervurdert. De besparelser man gjør ved enklere og mer modulært opplegg på maskiner og program komponenter, kan fort bli brukt opp av de økte kostnadene som følger av nettverket. Selv om nettverkskomponentene som regel vil være standardisert og gjennomprøvd før de settes i drift, bør ikke undervurdere kostnadene forbundet med drift av datanett.

- **Ny teknologi.** Hovedproblemet med distribuerte systemer, er at det er forholdsvis nytt og at man ennå ikke har full forståelse av hva det innebærer. Det totale systemet har ingen determinerbar tilstand, og det er derfor vanskelig å konstruere og verifisere at alt fungerer som forutsatt. Mange, særlig innen næringslivet, har nok betraktet dette som noe nytt og ikke ønsket å ta det i bruk før det er mer utprøvd.

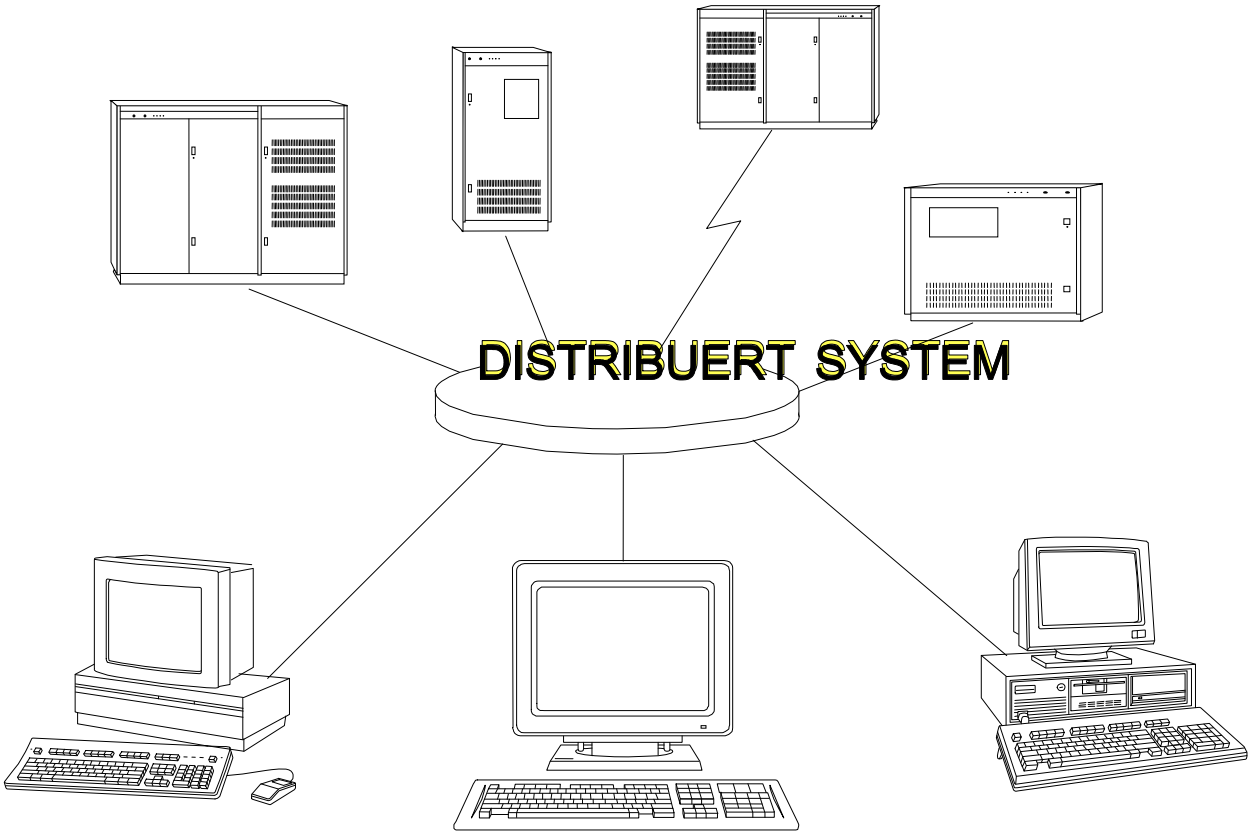
2.2. Basisforutsetninger for distribusjon

For å kunne distribuere tjenester rundt på ulike maskiner må en del grunn forutsetninger oppfylles.

- **Kabling.** Det må finnes en eller annen form for fysisk sammenkobling mellom maskinene. Dette kan være tilkobling til samme lokalnett, eller via et større datanett.

- **Protokoller.** Når to maskiner skal kommunisere må det finnes et felles sett med protokoller som muliggjør dette. To maskiner kan kommunisere via en tredje maskin som fungerer som protokollkonverterer.

Maskinene trenger ikke å være av samme type eller ha samme operativsystem, selv om distribusjonen av tjenester forenkles betraktelig hvis dette er tilfellet. Det kan være gunstig å se på distribusjon som bestående av to deler; brukernes arbeidsplassutstyr og tjenere. Brukernes arbeidsplassutstyr er de datamaskinene man bruker for å få tilgang til tjenestene som finnes på tjenermaskinene. Tjenermaskinene inneholder de programsystemene som brukerne har tilgang til. Dette kan være programmer som kjører på den lokale arbeidsplassmaskinen fra et distribuert filsystem, programsystemer som er oppdelt etter klient/tjener modellen, eller programsystemer som kjøres fra en terminal på maskinen.



Figur 1 - Det distribuerte systemet bør kunne opptre som ett system

3. Distribuerte tjenester

Hvilke tjenester eller datamaskinoppgaver er det snakk om å distribuere? Hvordan kan dette gjøres? Jeg vil her peke på noen av de tjenestene som kan eller bør distribueres. Senere i kapittelet kommer jeg tilbake til hvilke muligheter det finnes i dagens systemer for å tilfredsstille alle eller noen av de kravene jeg setter opp.

De grupper av tjenester som klart peker seg ut for distribuering er:

- **Filsystem.** Et distribuert filsystem er en av nøkkelfunksjonene i et distribuert system. En bruker bør kunne logge inn på en av de maskiner han har behov for og finne sine egne oppstartsfiler, datafiler, dokumenter ol., uavhengig av hvilken maskin han benytter. Tilsvarende må applikasjonene kunne finne sine filer uavhengig av hvilken maskin i lokalnettet de startes fra. Ideelt sett bør data bare oppdateres ett sted, men samtidig være tilgjengelig fra alle datamaskiner som trenger disse dataene, som om de var lokale.

- **Tilgang til felles periférienheter.** Det er ikke bare av kostnadsmessige hensyn at deling av skrivere er interessant. Brukeren må kunne ha en skriver i sin fysiske nærhet som alle maskiner kan skrive ut på, uavhengig av maskintype og operativsystem. Likeledes er det nyttig å kunne dele mer kostbart utstyr som fotosetter, fargeskrivere, høyhastighetslasere, plottere, utstyr for sikkerhetskopiering mm.

- **Prosessering.** For å kunne skalere bruken av systemet, og for å kunne utføre oppgaver som går for sent eller ikke kan kjøres på en gitt datamaskin, må man ha muligheten for å sende en jobb for eksekvering på en annen maskin. Systemet må selv kunne finne ut hvilke prosesser som overhodet ikke kan kjøre, eller som det ikke er hensiktsmessig å kjøre. Isteden overføres oppgaven til en annen maskin som systemet selv velger. Brukeren skal ikke merke noe av dette. Resultatene av jobben skal kunne presenteres som om de var utført på den lokale maskinen bare at det går raskere. Prosesseringen bør kunne distribueres uavhengig av prosessortype og operativsystem.

- **Brukergrensesnitt.** Ved å dele applikasjonene i en prosesseringsdel og en presentasjonsdel, kan man fra en og samme arbeidsstasjon jobbe med flere applikasjoner hvor prosseringsdelene kjører på ulike maskiner.

- **Kommunikasjon.** For å kunne få tilgang til ressurser utenfor lokalnettet trengs portnere eller broer. Dette er som oftest høyt spesialiserte maskiner som kun har til oppgave å håndtere kommunikasjonen med omverdenen. Sett fra brukerens side bør det kunne fortone seg som om hele verden er samme lokalnett.

- **Applikasjoner.** Sist, men ikke minst, er det nødvendig å organisere applikasjoner slik at det er transparent for brukeren hvor de befinner seg og eventuelt hvor de henter sine data fra. Brukeren skal tilbys ett brukergrensesnitt som er likt brukergrensesnittene i de andre programmene på arbeidsplassutstyret vedkommende bruker. Det må også finnes en applikasjon som forteller hvilke applikasjoner som finnes i systemet. Denne må oppdateres ettersom nye applikasjoner blir tilgjengelige. På dette området er det gjort lite. Foreløpig er det opp til brukeren å sette seg i forbindelse med den maskinen som har den

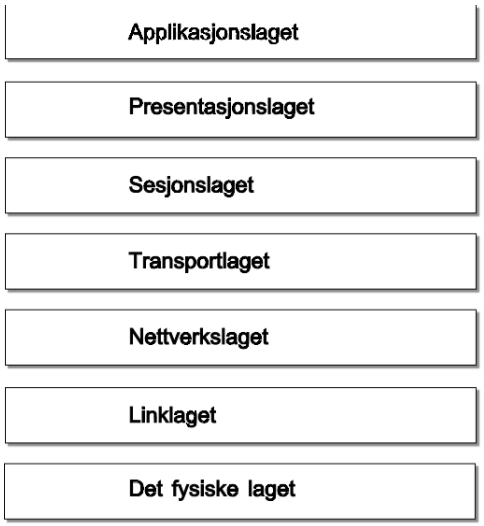
applikasjonen man ønsker å benytte. Dette vil jeg komme nærmere inn på i siste del av oppgaven.

3.1. Dagens muligheter for distribusjon

I dette kapittelet vil jeg ta for meg en del distribuerte tjenester og beskrive kort de protokollene/systemene som i dag er i utbredt bruk. Jeg vil konsentrere meg om noen av de systemene som har, eller er i ferd med å få status som de facto standarder innen forsknings- og utviklingsmiljøer, samt begynner å få anerkjennelse innen industrien. Utvalget er langt fra komplett. Jeg har forsøkt å begrense meg til de distribuerte tjenester som er mye brukt ved Universitetet i Oslo. Jeg vet av erfaring at dette er temmelig representativt for andre akademiske institusjoner i den vestlige verden. Hensikten er å peke på hvilke tjenester som i dag lar seg distribuere ved hjelp av velkjente standarder og produkter, for å kunne trekke paralleller til disse senere i oppgaven.

3.2. Kommunikasjon

Forutsetningen for at maskiner skal kunne kommunisere er at begge typer maskiner benytter samme protokoll. Med andre ord, at de snakker samme språk over nettverket. ISO (*International Standard Organization*) har med sin OSI (*Open Systems Interconnection*) laget referanse modellen de fleste benytter for å beskrive protokoller som benyttes i datanett. Modellen er delt i 7 lag. Hvert lag har sitt sett med oppgaver det utfører ovenfor det ovenforliggende laget, slik at kompleksiteten på det enkelte lag avgrenses. Tjenestene denne oppgaven omhandler, tilhører øverste lag, applikasjonslaget [TANENBAUM 89].



Figur 2 - ISO/OSI referanse modell

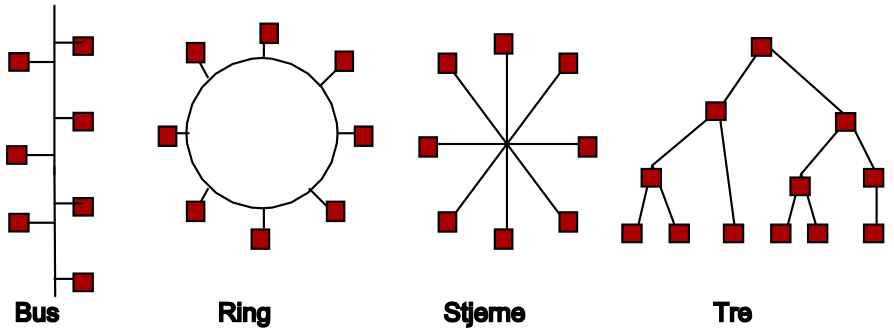
3.2.1. Kabling

Nettverket er på mange måter selve hjertet i distribuerte systemer. Det finnes flere standarder for kabling. De mest vanlige er fiberoptiske kabler, koaksialkabel eller tvunnet trådparskabel. De fysiske karakteristika varierer etter hvilke nettverksprotokoller de er ment å støtte. Fiberoptikk er på vei inn i en rekke sammenhenger. Spesielt på lengre strekk og hvor høye overføringshastigheter er nødvendig. Trådparskabel er i ferd med å overta internt i bygninger på grunn av fleksibiliteten et slikt opplegg gir sammenlignet med koaksialkabel. Det finnes flere metoder for å koble sammen kablene til et nettverk. Det vanligste er enten bus, ring, stjerne eller trestruktur som vist i Figur 2. [TANENBAUM 89]

3.2.2. Lokalnett

Lokalnettet er det nettverk som knytter de lokale datamaskinene sammen. Som regel er det fysiske begrensninger på hvor stort et lokalnett kan være med hensyn på antall noder og utstrekning. Det er laget mange løsninger for lokalnett. Jeg skal her kort beskrive de som er mest i bruk. [TANENBAUM 89]

Det framherskende i lokalnett er det som til daglig omtales som Ethernet. Et mer presist navn er CSMA/CD (*Carrier Sense Multiple Access / Collition Detect*) definert i ISO standard 8802.3 og i IEEE 802.3. Ethernet benytter koaksialkabel eller tvunnet trådpar. Nodene i nettet tillates å sende hvis det ikke er andre noder som sender. Overføringshastigheten er teoretisk 10 megabit/sekund. Det finnes også Ethernetbroer som gjør det mulig å koble sammen to eller flere Ethernet over telelinjer med varierende hastigheter.



Figur 3 - Kablingstopologier

IBM har laget sitt alternative lokalnett, kalt Token-Ring. Dette er definert i ISO standard 8802.5 og i IEEE 802.5. Som navnet antyder er nettet organisert i en ring hvor hver node i nettet må vente på et token før den har tillatelse til å sende. Dette sikrer at alle noder med høyeste prioritet får en garantert mulighet for å sende. Overføringshastigheten er teoretisk 4 eller 16 megabit/sekund. Det finnes broer som kobler sammen Token-Ring med Ethernet.

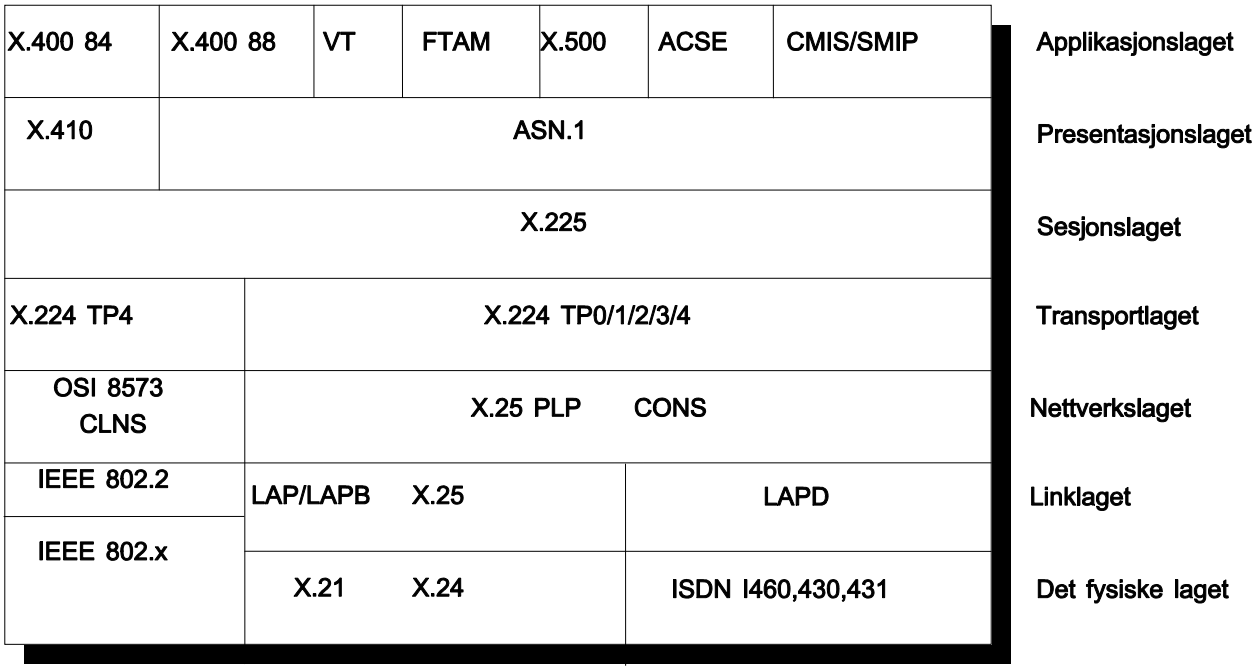
Token-Bus (ISO 8802.4 / IEEE 802.4) har ikke fått noen stor utbredelse, men er tatt inn i samme standardsett som Token-Ring og Ethernet. Token-Bus er koblet som Ethernet, men fungerer som en logisk ring, hvor nodene sender et token á la Token-Ring. Det ble laget for å garantere at prioriterte noder fikk mulighet til å sende, noe som er viktig ved styring av maskiner i industri-produksjon.

FDDI er på vei inn, men utbredelsen går foreløpig langsomt. Dette skyldes høye priser på blant annet adaptere. FDDI er basert på to fiberoptiske ringer med automatisk rekonfigurering i tilfelle en av dem faller ut. På samme måte som Token-Ring benytter FDDI seg av token prinsippet, men det kan til enhver tid være flere token på samme ring. Teoretisk overføringshastighet er 100 megabit/sekund. En ring kan være 100 kilometer i omkrets. Det finnes broer for å koble FDDI til lokale Ethernet eller Token-Ring nett.

3.2.3. Fjernnett

Ved bruk av lokalnett får man mulighet for å kommunisere mellom maskiner som er fysisk plassert i nærheten av hverandre. Ved hjelp av fjernnett kan enkeltstående maskiner, eller hele lokalnett knyttes sammen uavhengig av avstanden mellom dem. Det finnes allerede akademiske forskning og undervisningsnett som dekker store deler av den industrialiserte verden. UNINETT står for drift og utvikling av et slikt nett i Norge. Dette nettet er koblet sammen med de andre akademiske nettverkene i Europa og USA, og gjennom disse videre til land som Australia, New Zealand, Japan, Korea, Mexico, Canada med flere. Hastigheten i fjernnett varierer fra 9.6 kilobit/sekund til flere megabit/sekund. Det pågår forskning med sikte på å få hastighetene opp i gigabit/sekund. [UNINETT 90]

OSI PROTOKOLLER



Figur 4 - En del av OSI protokollene i ISO/OSI referanse modell

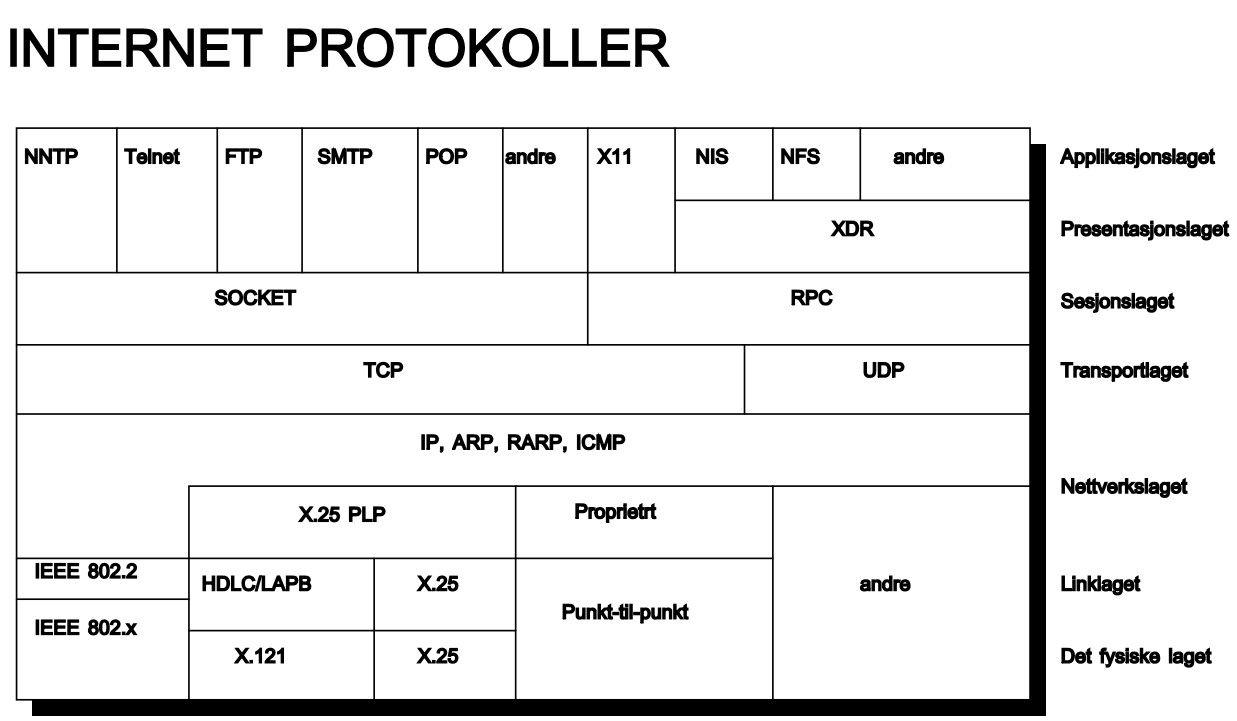
De fleste televerk tilbyr en X.25 tjeneste etter anbefalingene gitt av den internasjonale konsultative komité for telegrafi og telefoni (*CCITT*). X.25 er preget av at den er utviklet av televerkene. Den er basert på å sette opp og koble ned forbindelsen mellom nodene. Dette gir mulighet for å benytte samme X.25 abonnement mot mange noder. Protokollen inneholder også mekanismer for lukkede brukergrupper, noteringsoverføring og lignende. Betalingen for X.25 er for det meste basert på antall oppkoblinger og overført datamengde, samt abonnements avgift. X.25 splitter meldingene, opp i mindre enheter enn det som er vanlig på et lokalnett. Dette fører til en del ekstra overhead ved at pakkene å reassembleres på mottagersiden ved sammenknytning av to lokalnett. Mot noder som ikke er fast tilknyttet et datanett er ofte X.25 eneste mulighet. X.25 består av tre lag med protokoller fra lag 1 til 3.

Alternativet til X.25 er stort sett å leie linjer av televerket og kjøre de protokoller man har behov for direkte på disse. Ulempen er at dette bare er en punkt-til-punkt forbindelse, og man trenger en flere leide linjer for å nå dem man ønsker. Fordelene er at ved høye trafikkvolum blir det langt billigere enn X.25, samt at det gir høyere overføringshastighet for de fleste protokoller fordi man slipper å gå veien om X.25. Ved bruk av leide linjer er protokoller som HDLC eller LAPB vanlig, men også leverandørspesifike protokoller er i bruk. Disse gir mulighet for bedre utnyttelse av linjen.

3.2.4. Kommunikasjonsprotokoller på høyere nivå

Mange protokoller kan benyttes på høyere nivå, men det er to som klart peker seg ut i store datanett. Det ene er protokollsettet ISO har spesifisert i OSI. Det andre er protokollene som er utviklet i forbindelse med det amerikanske forsvarsdepartementets utviklingsprosjekt for datanett. Dette gikk opprinnelig under navnet DARPA, men omtales nå mer og mer som TCP/IP eller Internet protokollene. Andre protokoller eller protokollfamilier som IBMs SNA (*Structured Network Architecture*) og Digitals DECnet er i bruk, men de fleste leverandører har signalisert at de i framtiden vil basere seg på OSI-protokollene.

De kommunikasjonsprotokollene som går under OSI etiketten inneholder mange interessante muligheter. Særlig de høyere lagene åpner for et sterkt forbedret tjenestetilbud sammenlignet med andre og eldre protokoller. Internt i datanett- og telematikkmiljøene har det i flere år pågått heftige diskusjoner om hvorvidt man skal basere seg på forbindelsesorientert eller forbindelsesfri kommunikasjon, spesielt på nettverkslaget. Dette har endt ut i et kompromiss, hvor OSI åpner for begge typer. [QUATERMAN 90]. En rekke regjeringer i de industrialiserte land har laget egne OSI profiler (*GOSIP - Government OSI Profile*). Blant disse er Storbritannia, USA og de nordiske landene. Dette setter krav til at alt nytt datautstyr som kjøpes skal støtte OSI protokollene. Dette er



Figur 5 - Internettprotokoller satt inn i ISO/OSI referansemodell

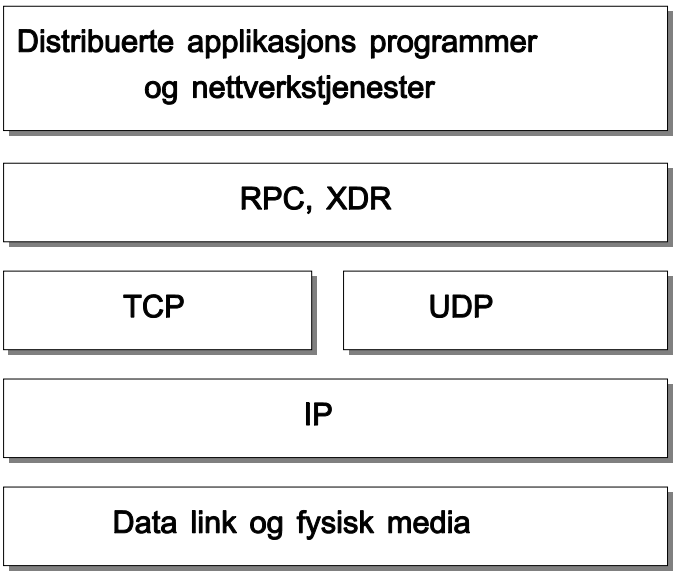
en viktig pådriver for alle større datamaskinleverandører til å utvikle sine OSI produkter.

I de akademiske datanett er Internet protokollene foreløpig de mest utbredte. Disse omtales ofte som TCP/IP (*Transmission Control Protocol / Internet Protocol*). De er basert på protokollene utviklet for ARPA (*Advanced Research Project Agency*) i USA. Det finnes mange produkter og for de fleste maskintyper basert på dette produktsettet. Det er i praksis bare 5 lag i TCP/IP. Lag 5 (sesjon) og 6 (presentasjon) mangler, eller rettere de er dekket opp av laget over [TANENBAUM 89]. RPC (*Remote Procedure Call*) og Socket mekanismen i BSD UNIX dekker noe av det et

sesjonslag skal dekke. XDR (*eXternal Data Representation*) dekker deler av det presentasjonslaget i OSI dekker [SUN 89]. For en gjennomgang av TCP/IP, se [DAVIDSON 88], [QUATERMAN 90], [HENDRIK 88] og [TANENBAUM 89]

3.2.5. Tjenestene i datanett

Hvilke tjenester det er mulig å tilby over et datanett er det vel bare fantasien som setter begrensninger for. Med stadig økende båndbredde i datanettene, blir også nye tjenester mulig. Det jobbes mye med å integrere data, video og tale i samme nett. Med mega- eller gigabit/sekund overføringshastighet vil vi i framtiden kunne se tjenester som multimedia fjernundervisning, konferansesystemer og andre ting som i dag knapt er teknisk mulig og i hvert fall økonomisk utenkelig. I det følgende skal jeg gå gjennom en del av de distribuerte tjenestene beskrevet i starten av dette kapittelet, og trekke fram de mest aktuelle mulighetene.



Figur 6 - Tjenester basert på TC¬P/IP

Mange av tjenestene er i dag bygget på TCP/IP. Fordi dette er et utbredt sett med protokoller, med gode muligheter for å benytte det eksisterende Internettet, blir også stadig nye tjenester basert på TCP/IP. Det finnes en Internet RFC 1006 (*Request For Comment*) som beskriver hvordan OSI transportlag klasse 4 kan benyttes over TCP. Det er en trend i USA i retning av å ta i bruk OSI tjenestene, men basere dem på TCP/IP på de lavere lagene. Jeg vil peke på tjenester i begge protokollsett under hver type tjeneste. Tjenestene er gruppert etter hvilket nivå de er lokasjonstransparent etter [COULOURIS 89] gruppering beskrevet på side **Error! Bookmark not defined.**2.2.

3.3. Fullstendig lokasjonstransparente tjenester (nivå 2)

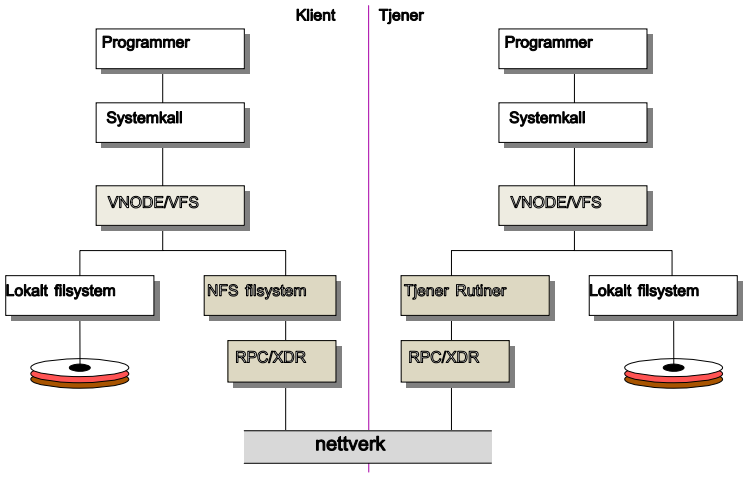
De distribuerte tjenestene på nivå 2 som er helt like uansett hvilken node i systemet man benytter dem fra. Uten å måtte foreta seg noe spesielt kan programmerer eller systemadministrator lage det transparent for brukeren hvor tjenesten befinner seg. Systemet støtter selv distribusjonen.

3.3.1. Distribuert filsystem

Et distribuert filsystem er muligheten for en maskin til å bruke filsystemet på en annen maskin som om det var en lokal disk. Uavhengig av operativsystemet bør alle maskiner kunne benytte felles disker. Dette betyr at man kan dele data direkte mellom ulike maskinertyper. Mellom maskiner med samme prosessor og operativsystem kan man også dele programmer.

Det finnes ulike løsninger for deling av diskplass mellom maskiner. Dette har vært den viktigste funksjonen for nettverk mellom mikromaskiner. De største på dette området er NOVELL for

MS-DOS baserte Pc-er, og Appleshare for Macintosh. De fleste av disse mikromaskinløsningene bygger på andre protokoller enn OSI eller TCP/IP på de lavere lagene. Dette gjør dem lite egnet i store nasjonale og regionale nettverk, på grunn av problemer med ruting av trafikken, begrenset adresseringsfelt og så videre. Slike mikromaskinnettverk er løsninger for små, isolerte grupper i en bygning.



Figur 7 - NFS oppdeling i klient/tjener

Mellom UNIX-maskiner er Network File System (NFS) det mest utbredte. NFS er utviklet av Sun Microsystems Inc., og mesteparten av koden er lagt ut som Public Domain [RFC 1048]. Dette har gjort at NFS er implementert på de fleste UNIX systemet. NFS er en de facto standard for fildeling mellom UNIX-maskiner. I tillegg er det også versjoner av NFS for operativsystem som MS-DOS, SINTRAN, NOS/VE, VAX/VMS, MVS, VM med flere [SUN 89].

NFS er implementert etter klient-tjener modellen. På klienten blir det lagt inn et ekstra lag mellom systemkallene og filsystemet. Dette detekterer om det er lokale eller eksterne filer og oversetter operasjoner på eksterne filer til nettverkskall til tjeneren [SANDBERG] [ARNOLD 88]

3.3.2. Brukerautentisering

Med et større antall maskiner i nettet vil behovet for brukerkontoer på ulike maskiner og maskingrupper være tilnærmet uhåndterlig uten at man har en felles database som inneholder informasjon om alle brukerkontoene. En ting er å opprette brukerkonti, noe annet er det å vedlikeholde dem og sørge for at de alltid er like på alle maskiner. For brukeren vil det bli vanskelig å skifte passord uten et distribuert system for brukerautentisering. Det er ikke særlig gunstig å måtte logge inn på flere titalls maskiner for å skifte passord på hver enkelt, når man isteden kan endre passordet på en maskin og dermed ha endret det for alle andre også.

Network Information System (NIS) fra Sun Microsystems Inc. (Tidligere kalt YP - *Yellow Pages*) tar seg av mer enn bare brukerautentisering. Det er et datahånterings- og oppslagssystem for hele nettverket, som tillater global administrering av endringer i nettverksomgivelsene innen et NIS domene. I praksis er det en database med systeminformasjon som definerer mekanismer for å vedlikeholde kobling mellom par av nøkkelverdier. For eksempel koblingen mellom brukernavn og passord, mellom maskinnavn og Internettadresse, og tjenester etter navn. Dette er oppgaver OSI katalogen kan komme til å håndtere i framtiden, men NIS er i bruk i dag. Det er visse tvilsomme sikkerhetsaspekter ved NIS, men ikke alvorlige nok til at Universitetet i Oslo forhindres i å bruke det.

Kerberos er et annet system som stammer fra Athena prosjektet ved MIT. Kerberos tilbyr langt bedre sikkerhetsmekanismer enn NIS, men foreløpig finnes ikke implementasjoner på tilstrekkelig antall maskinplattformer til at det har fått noen stor utbredelse.

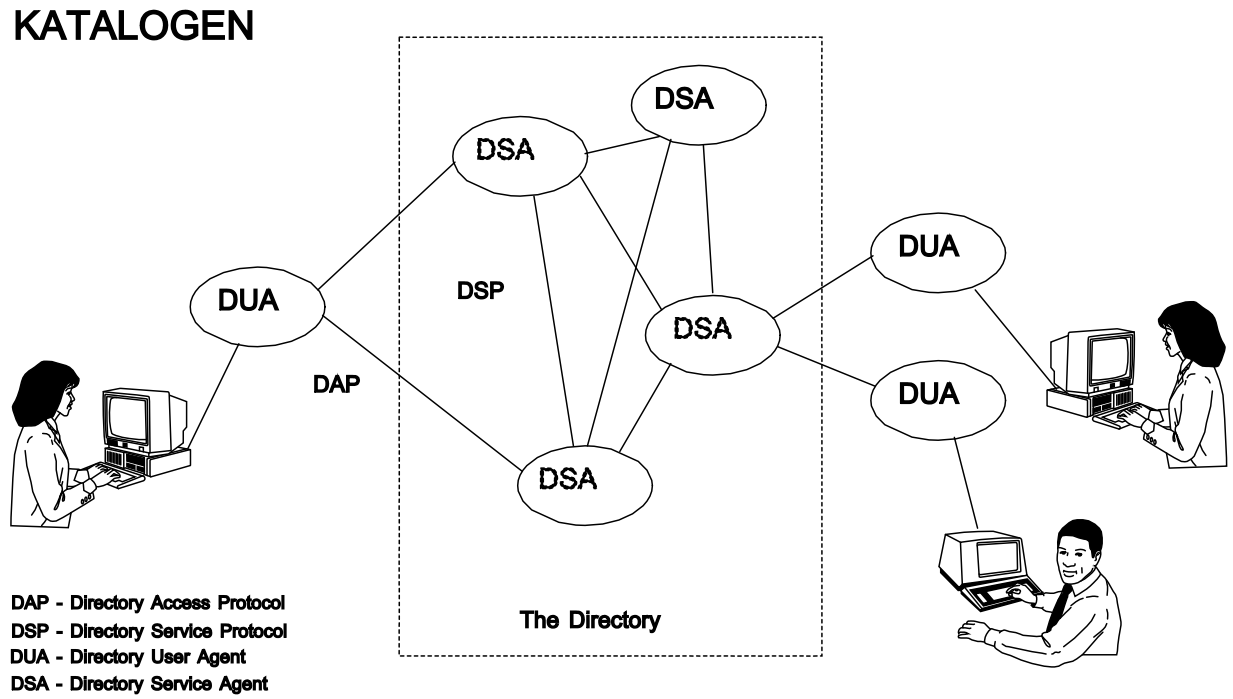
3.4. Delvis lokasjonstransparente tjenester (nivå 1)

I denne gruppen med distribuerte tjenester er det opp til programmerer og/eller systemadministrator å gjøre dem lokasjonstransparente. Det er flere teknikker som kan benyttes til dette. Elektronisk post kan gjøres slik at alle maskiner tar i mot post og sender den videre til en maskin som fungerer som postkontor. Denne sørger så for å fordele den lokale posten til den maskinen den skal, og sende ikke-lokal post videre ut i verden. Hvordan dette gjøres, bør brukerne behøve å vite så lite som mulig om. De trenger bare å vite hvilken/hvilke maskiner de skal benytte for å lese og skrive elektroniske meldinger. Tilsvarende kan utskriftstjenesten legges opp. Alle utskriftskøer tilordnes globale navn innen nettverket og oppsettet sørger for at utskriften sendes til den tjenermaskin som håndterer skriveren rent fysisk.

3.4.1. Katalog

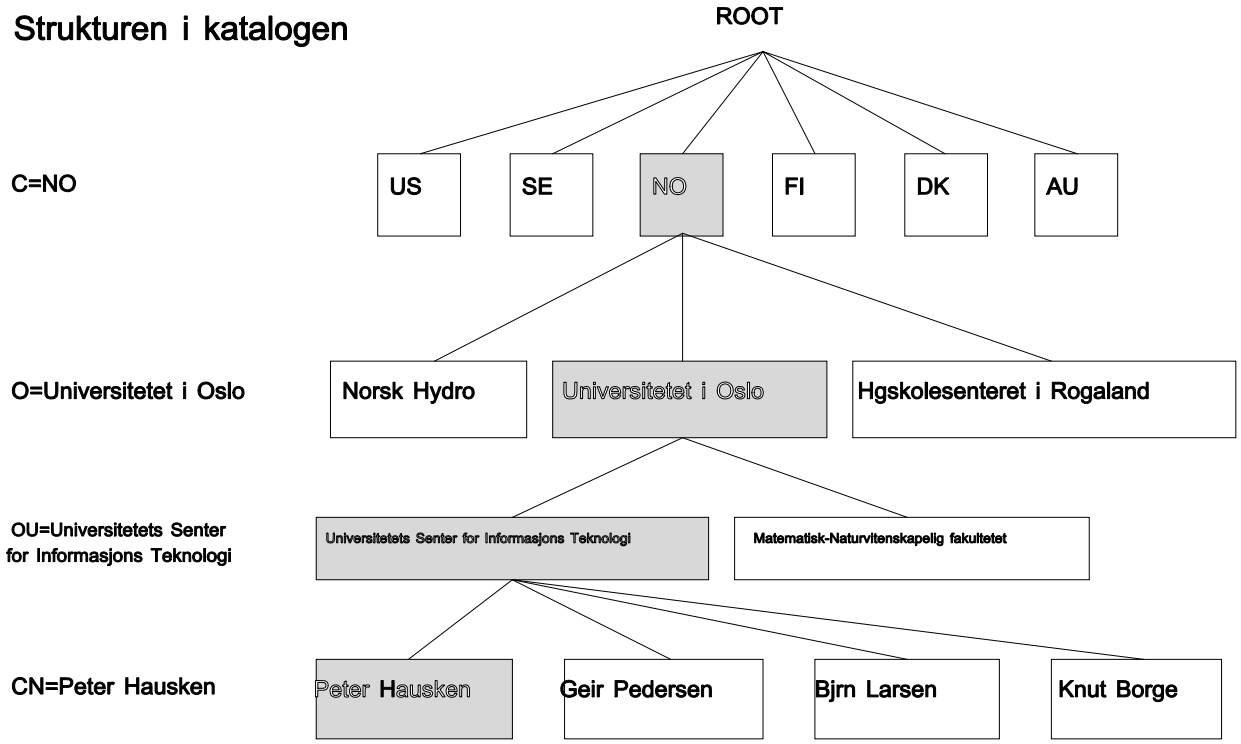
OSI katalogen (*Eng. Directory*) [ISO 9594], oftest omtalt som X.500, ble i første rekke designet for å dekke televerkets brukeres behov for å finne fram til telefonnummer, telefaksnummer, elektronisk postadresse og lignende. Den kan best sammenlignes med en elektronisk utgave av telefonkatalogen, men uten de begrensninger en papirbasert informasjonskilde har med hensyn på spredning og oppdatering. Katalogen er organisert etter en objektorientert tankegang, hvor hvert objekt i katalogen representerer et objekt i den virkelige verden. Objektene er hierarkisk ordnet i nivåer etter land, organisasjon, avdeling og personer. Katalogen kan også inneholde annen informasjon, så som nettverkstjenester, referanser til prosjekter og nettverksapplikasjoner [HEINÄNEN 89].

På samme måte som telefonkatalogen er splittet i mange bind etter geografisk eller bransjemessige oppdelinger, er OSI katalogen delt opp i mange *Directory Service Agents* (DSA). Den har likhetstrekk med en distribuert database, men på grunn av dens natur unngår katalogen de problemer man står ovenfor med distribuerte databaser. Det er tatt utgangspunkt i en hierarkisk struktur med langt flere forespørsler enn oppdatering, og uten behov for øyeblikkelig oppdatering



Figur 8 - Katalogen består av mange DSAer

internasjonalt. Ideen er at informasjonen skal vedlikeholdes så nær kilden som mulig og med erkjennelse om at ingen datamaskin vil være stor nok til å kunne romme en komplett *Directory Information Base* (DIB) for hele verden. En *Directory User* (DU) kan være en bruker eller en applikasjon. Disse aksesserer katalogen gjennom en DUA (*Directory User Agent*). Hvis DSAen ikke har informasjonen kan den enten henvise DUA til en annen DSA (*Referral*), selv hente informasjonen fra en annen DSA (*Chaining*), eller spørre flere DSAer om hvor informasjonen finnes (Multicasting) [HEINÄNEN 89]



Figur 1 - Katalogen er hierarkisk oppbygget

Driften av kataloger av denne type er foreløpig på forsøksstadiet, spesielt innen de akademiske datanett i Europa og USA. Målet er å bygge det ut til en verdensomspennende katalogtjeneste. Den mest brukte implementasjonen er kalt QUIPU og er utviklet ved University College London (UCL). På de lavere lagene benytter den ISODE (*ISO Development Environment*), som kan benytte både OSI TP0 over X.25 og TCP/IP som nettverksprotokoller. Flere datamaskin leverandører har annonsert at de vil komme med X.500 produkter.

Figur 9 viser hvordan katalogen kan være strukturert. Den har en toppnode som er roten i hele katalogtreet med alle land organisert under denne. Innen hvert land er det subnoder i katalogen for organisasjoner og avdelinger innen disse. Nederst er det noder for enkelt personer. Ettersom bruken blir mer utbredt vil det naturlige være at man har en maskin for hver organisasjon som forvalter sin del av katalogtreet. Hva slags informasjon som ligger i de enkelte nivå av treet er gjenstand for en viss standardisering. Det er likevel fritt opp til det enkelte land og/eller organisasjon å legge inn informasjonen slik man ønsker. Hierarkiet kan også organiseres annerledes, bare man ivaretar det som trengs for at andre katalogtjenere kan være i stand til å hente ut den informasjon de trenger fra katalogen.

Det er ikke bare personer og organisasjoner det er interessant å legge inn i katalogen, men også informasjon om selve nettverkene, maskiner, skrivere ol. Sist, men ikke minst, er det av interesse å vite hvilke tjenester som er tilgjengelig.

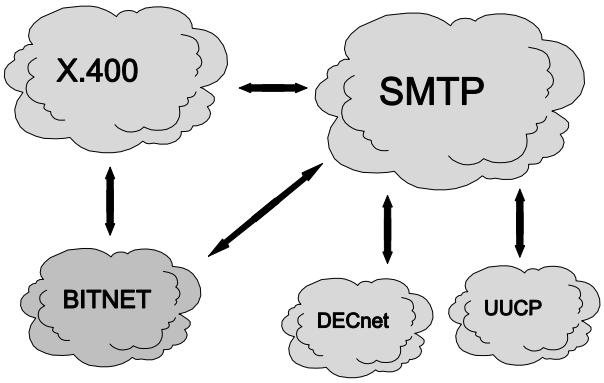
3.4.2. Elektronisk post

Elektronisk post gir mulighet for å utveksle meldinger mellom brukere. I første rekke tekst, men etter hvert også data, programmer, grafikk, bilder og lyd. Dette er en tjeneste hvor nettverket for det meste er usynlig for brukeren. For å få en melding fram, oppgis adressen dit meldingen skal, resten besørges av systemet. Arbeidet ligger på systemadministratorsiden. Tabeller over hvordan meldingene skal sendes, må til enhver tid holdes oppdatert, og dette krever en god del arbeidsinnsats av de involverte. Dette er et typisk eksempel på distribuerte tjenester på nivå 1 [COULOURIS 89].

OSI standarden for elektronisk post er kalt MOTIS, men CCITTs variant X.400 er mer kjent. 1988 utgaven av denne standarden innholder blant annet mulighet for kryptering av meldingen, overføring av grafikk og ulike tegnsett. X.400 er blant annet valgt som basis for Televerkets TelemaX.400 tjeneste [TANENBAUM 89].

En annen populær standard er den som i Internet er beskrevet i [RFC822], vanligvis omtalt som SMTP (*Simple Mail Transfer Protocol*) [TANENBAUM 89]

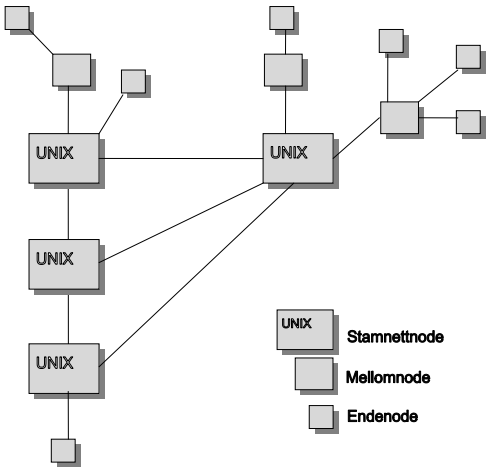
Flere andre systemer er i bruk. UUCP (*UNIX-to-UNIX CoPy*) er et enkelt system for automatisk overføring av elektronisk post over vanlige telefonlinjer ved hjelp av modem. Adresseringen er nå for det meste tilpasset RFC822. BITNET/EARN er et system blant annet for overføring av elektronisk post mellom maskiner. Digital har også et elektronisk postsystem for DECnet. Det er laget portnere mellom SMTP og disse systemene, slik at man fra et system kan sende elektronisk post til brukere i et annet system. Det er også laget portnere fra andre postsystemer. All informasjon i en elektronisk melding vil ikke kunne oversettes fra et system til et annet fordi ulike systemer har ulike muligheter [TANNENBAUM] [SAMS 87]



Figur 10 - Portnere mellom ulike postsystemer

3.4.3. Distribuerte konferansesystemer

Konferansesystemer er ikke en fullstendig enhetlig gruppe av tjenester. Tradisjonelt har de vært knyttet til en maskin. Alle har måttet logge inn på den maskinen, og kjøre konferansesystemet som en applikasjon. Det foregår fortsatt mye forskning på dette feltet, og til tider også heftige diskusjoner om hva som er det beste.



Figur 11 - Logisk kobling mellom NetNews koder

NetNews har et annet utgangspunkt enn de tradisjonelle konferansesystemene. NetNews har en anarkistisk fortid og utviklingen og spredningen er fortsatt ikke kontrollerbar. Det startet med et sett UNIX skript filer laget av Tom Truscott og James Ellis ved Duke University i 1979. Det ble starten til USENET i USA, og til det senere verdensomspennende konferansesystemet kjent under navnet USENET NetWork News, for kort omtalt som NetNews. Innleggene er delt i grupper etter emne. Noen grupper er modererte, mens andre er såkalt åpne grupper hvor hvem som helst kan skrive inn artikler. Systemet er bygget på at NetNews noder sprer artiklene videre til maskiner de har avtale med som et slags edderkoppnett. Transportprotokollene er enten UUCP (*Unix-to-Unix-CoPy*) eller TCP/IP. NNTP (*Network News Transport Protocol*) brukes både for å overføre artikler mellom NetNews tjenere og for å distribuere artiklene ut til NetNews lese programmer [RFC 977].

I Norge har UNINETT valgt NetNews som sitt konferansesystem. Det er i utgangspunktet en distribuert løsning, noe som gjør at det passer godt inn i den akademiske nettverden. Jeg vil komme tilbake til NetNews i kapittel 3 som et eksempel på problemer som oppstår når man skal distribuere slike tjenester [SAMS 87] [HAUSKEN 90].

Det finnes også muligheter for det som kalles distribusjonslister. Distribusjonslister skiller seg lite fra elektronisk post i teknisk forstand. Man sender meldingen til en adresse og så spres meldingen til et gitt sett mottagere. Meldingene sendes og mottas vha de samme programmer som meldinger fra person til person. Problemer knyttet til distribusjon av slike tjenester blir den samme som for elektronisk post.

3.4.4. Skriverdeling

Skriverdeling er muligheten for å dele skrivere mellom mange maskiner og brukere. Selv om prisen på datamaskiner har gått ned, har prisen på skrivere ligget forholdsvis høyt. Særlig laserskrivere har det vært stor interesse for å kunne dele mellom mange maskiner i et nettverk. På samme måte som for filsystem er det laget lokale løsninger for å tilfredsstille dette behovet, som regel i samme system som for fildeling. Disse løsningene fungerer greit innenfor avgrensede områder, men har problemer med å distribueres over et større nettverk.

I UNIX finnes en protokoll kalt LP. Denne gjør det mulig å sette opp skriverkøer på en maskin slik at utskriftsjobben sendes videre til en annen maskin. Den benytter TCP/IP, og kan derfor sendes til en hvilken som helst maskin tilkoblet Internet med en skriver som man har lov å skrive til. Det betyr at en bruker i Norge kan sende sitt dokument ut på en skriver i USA eller Australia, hvis begge er tilknyttet Internettet.

Ved USIT er det utviklet et system kalt PRISS (*The Print Spooling System*). Her er flere protokoller for utskrift kombinert. Det kan ta i mot utskrift fra maskiner og sende det videre til skrivere på forskjellige protokoller og formater. PRISS inneholder moduler for å oversette og formatere utskriften basert på hva den får inn og hvilken type skriver utskriften skal ut på. Systemet er modulært bygget opp, slik at nye moduler fort kan legges inn etter som behovet oppstår.

Både LPD og PRISS er nivå 1 løsninger. En systemadministrator må sette opp og konfigurere skriverkøer for at det skal fungere, men brukeren merker ikke dette. For dem ser det ut som alle skrivere er tilknyttet alle maskiner og kan ta imot utskrift i de fleste formater.

3.5. Tjenester på nivå 0

Dette er tjenester hvor brukeren rent faktisk må vite hvilken maskin tjenesten befinner seg på. En måte å skjule hvor tjenestene ligger, er å benytte logiske navn på tjenermaskinene etter hvilken tjeneste de utfører. Dermed har systemadministrator mulighet til å flytte en tjeneste fra en maskin til en annen, uten at brukerne berøres av det. Hvis en maskin har et statistikkprogram installert, kan maskinen tilordnes navnet *"stat"*. Brukerne trenger bare å forholde seg til at statistikkprogrammet ligger på maskinen *"stat"* og hvis det flyttes til en annen maskin kan system-administrator gjøre det uten at brukerne merker det. Det er en langt enklere operasjon enn å fortelle flere tusen mennesker at nå ligger ikke tjenesten på maskin A lenger, men er flyttet til maskin B. Jeg tror ikke man skal undervurdere tiden det tar å informere folk om endringer i systemet, og heller ikke den frustrasjon de kan føle ved at deres verden hele tiden ikke bare er i utvikling, men også i forandring. Mennesker er stort sett vanedyr i og de som har mest motstand mot forandring er ofte vanskeligst å informere. Hvis man ikke har et slikt opplegg, kan det være en idé å vurdere om man skal flytte maskinnavnet med tjenesten og heller gi et nytt navn til den gamle maskinen hvis det er en mye brukt tjeneste. Kanskje litt unaturlig for systemadministratorene, men sett fra brukernes side en enklere løsning.

3.6. Filoverføring

Filoverføring er muligheten til raskt å flytte dokumenter, data og programmer fra en maskin til en annen over en eksisterende forbindelse.

Filoverføring tilhører en av basistjenestene i Internet. FTP er programmet og protokollen som benyttes for filoverføring for maskiner med TCP/IP. Det krever at brukeren vet navnet eller nummeret på maskinen og enten har et gyldig brukernavn og passord på maskinen, eller at den er åpen for såkalt *"anonymous*" innlogging.

OSI protokollen for filoverføring går under navnet FTAM (*File Transfer, Access and Management*) og er beskrevet i ISO standard 8571. Også her er brukeren avhengig av å vite navnet på maskinen. FTAM har en rekke avanserte muligheter for å aksessere deler av en fil og mulighet for rollback hvis noe går galt [TANENBAUM 89]

3.6.1. Terminalaksess

Terminalaksess er muligheten for å opprette forbindelse fra en datamaskin til en annen og utføre programmer på den andre maskinen, som om den var den primære maskinen brukeren er innlogget på.

Innen Internet er Telnet den framherskende protokollen, gjerne også med en terminalemulator for å etterligne ulike terminaler. For UNIX finnes også *rlogin* (*Remote Login*) som gjør det samme som *telnet*, men med mulighet for å hoppe over selve innloggingssekvensen. *rsh (Remote Shell*) og *rexec* (*Remote Execute*) kan benyttes for å starte programmer på en annen maskin uten at man

logger inn. De kan nærmest sammenlignes med å benytte *rlogin*, utføre en kommando og logge ut, i én operasjon.

Innen OSI har det pågått en del arbeid med å definere det man har kalt en Virtuell Terminal (VT). Foreløpig er det bare definert det som kalles "*Basic Class*". Standarden ISO9040 beskriver de primitiver en slik virtuell terminal skal kunne håndtere. ISO9041 beskriver hvordan den skal håndteres. Funksjonaliteten er på linje med en vanlig "dum" terminal.

3.6.2. "Ikke standarder"

Felles for mange av disse *"standardene"* er at de ikke er anerkjent som internasjonale standarder. De er basert på UNIX operativsystem og brukes for å få forskjellige UNIX systemer til å samarbeide. De er også delvis portert til mikromaskiner med operativsystemer som Finder og MS-DOS, samt til andre stor/minimaskin operativsystemer, slik som NOS/VE, VAX/VMS, SINTRAN og andre. Standardene er utviklet ved universiteter eller av datamaskinprodusenter og så blitt akseptert av andre produsenter som en standard, og implementert i deres systemer av markedsmessige hensyn.

4. Arbeidsplassmaskinen og brukeromgivelsene

Jeg vil i dette kapittelet kort beskrive de arbeidsomgivelsene den enkelte datamaskinbruker har i sin hverdag. Det vil si hvilke brukergrensesnitt som er vanlig og på hvilke maskinplattformer. Dette for å gi forutsetningene som ligger til grunn for den distribuerte applikasjonsmodellen.

4.1. Brukergrensesnitt

Brukerens kontaktflate mot datasystemet. Den delen av et system som brukeren kan se på skjermen og benytte til å gi kommandoer og lese informasjon fra. Til å begynne med var det primære å få systemet til å fungere og hvordan data og kommandoer ble tastet inn var av sekundær betydning. I tillegg har tilgjengelighet og pris på maskinkraft begrenset mulighetene for å lage annet enn svært enkle brukergrensesnitt. Med dagens kraftige og relativt billige datamaskiner er disse hindringene langt på vei borte.

Kravet til funksjonalitet og brukervennlighet har økt i takt med tilgjengelig datakraft. Innføringen og populariseringen av vindusbaserte brukergrensesnitt gjennom arbeidene ved Xerox PARC og Apples lansering av Macintosh, har på mange måter satt nye mål og standarder på hvordan brukergrensesnitt skal lages. Produkter som følger opp disse ideene, har blitt lansert fra Microsoft i form av Presentation Manager for OS/2 og MS-Windows for MS-DOS og det distribuerte vindussystemet X fra MIT. En rekke andre maskinvareleverandører har laget egne grafiske brukergrensesnitt for sin maskinvare [KUTAL 90].

Det har vært mange diskusjoner om hvordan et godt brukergrensesnitt skal se ut. Jeg vil her bare ta for meg de ulike typer brukergrensesnitt som tilhører interaktiv databehandling, eller ON-LINE kjøring som det het for snart 10 år siden og peke på styrke og svakheter ved de ulike typene.

4.2. Linjeorientert

De først interaktive terminaler var omtent som en skrivemaskin. Eller kanskje rettere en slags teleks maskin. Brukeren kunne skrive en kommando og resultatet ble skrevet direkte ut på papir. Det er ikke mange slike teletypeterminaler igjen, men den type brukergrensesnitt de benyttet, er fortsatt i bruk. De fleste operativsystem har denne type grensesnitt og en rekke programmer som benyttes for å kontrollere maskiner og nettverk er fortsatt linjeorientert. Fordelen er at det er svært enkelt å programmere, krever lite maskinkraft og det kan benyttes fra de fleste terminaler. Ulempen er at brukeren enten må svare på en masse spørsmål, eller huske kommandoer med innviklet syntaks, samt holde oversikten over hvilke kommandoer som er mulige på hvilket nivå og i hvilken modus programmet nå befinner seg i [TESLER 81].

Det linjeorienterte brukergrensesnittet har fortsatt sin aktualitet. Særlig i forbindelse med operativsystem kommandoer, som kan være vanskelige å lage gode brukergrensesnitt for, på grunn av de mange valgmulighetene. Det er også det minste fellesmultiplum i mange sammenhenger og det eneste man kan forutsette at alltid lar seg benytte. Mengden arbeid som kreves for å lage mer avanserte brukergrensesnitt fører nok også til at mange systemer fortsatt bare har et linjeorientert brukergrensesnitt.

4.3. Skjermorientert

Med innføringen av skjermterminaler kom også mulighetene for å lage mer avanserte brukergrensesnitt i og med at man tok hele skjermen i bruk. Man kunne presentere brukeren for mer informasjon samtidig og gi mulighet for å styre markøren. En viktig gevinst var mulighetene for å lage editorer hvor brukeren kunne manøvrere og redigere enklere, noe som gav langt bedre oversikt over teksten man arbeidet med. Også applikasjoner for administrativ databehandling fikk økt brukervennlighet ved innføringen av skjermorienterte brukergrensesnitt. Menysystemer som hjalp brukeren i å finne fram i systemene, hjalp til at det skjermorienterte brukergrensesnittet fikk økt utbredelse. Som oftest er slike brukergrensesnitt begrenset til 24 linjer i 80 kolonner og kun bokstaver og tegn som finnes i terminalen. Fordelen er at brukeren får mer brukervennlige systemer, enn ved rene linje orienterte systemer. Ulempen er at det krever mer maskinkraft og mer av programmereren, samt at det kan bare brukes fra terminaler som kan skjermstyring som støttes av systemet. Jeg vil i denne oppgaven referere til det som skjermorienterte systemer, men navn som alfanumeriske, fullskjerm eller karakterbaserte er andre navn som ofte benyttes om disse systemene.

For mange systemer er et skjermorientert brukergrensesnitt tilstrekkelig. For eksempel hvor brukeren ikke har behov for å skifte mellom applikasjoner og bare trenger tekst ut på skjermen. Hotellresepsjoner, billettbestilling, bank, postkontor og lignende bruksområder kan ofte være like tjent med et hurtig og effektivt skjermbasert brukergrensesnitt, som et mer langsomt og muskrevende grafisk brukergrensesnitt. De fleste systemer som baserer seg på større fellesmaskiner er fortsatt skjermorienterte og ikke grafiske.

4.4. Grafiske brukergrensesnitt

XEROX PARC (*Palo Alto Research Center*) var tidlig ute med å eksperimentere med hvordan det var mest intuitivt for brukerne å kommunisere med en datamaskin. En grafisk skjerm med vinduer og styring av funksjonene ved hjelp av et pekeredskap (mus) var det man fant gav brukeren det beste brukergrensesnittet. De ideene som kom fra XEROX PARC har brukt lang tid på å få gjennomslag, men de siste årene har prisene på maskinvare sunket så mye at populariteten har økt betraktelig. Fordelene er at det gir brukeren et langt bedre grensesnitt mot systemet. Teknikken med å benytte vinduer, gjør det mulig å benytte applikasjoner med linjeorientert brukergrensesnitt i et vindu, skjermorientert i et annet og grafisk i et tredje. Grafiske brukergrensesnitt gir muligheter som ikke finnes i de to andre. For eksempel muligheten for grafisk presentasjon av tallmateriale, bilder, ikoner og annet som kan gi brukeren en langt bedre visualisering av det man jobber med. En annen og en ikke uvesentlig fordel, er redusert behov for opplæring av brukerne. De fleste av disse systemene har standardiserte måter for hvordan et program skal oppføre seg, noe som gjør at brukeren kjenner seg igjen fra program til program. Ulempen er at det krever mye av maskinressursene og ikke minst er det mye systemering og programmering involvert i å lage et godt grafisk brukergrensesnitt. Man skal ikke undervurdere kostnadene ved å utvikle applikasjoner for slike brukergrensesnitt, selv om det i dag er vindussystemer som tar seg av det meste av interaksjonen med tastatur, mus og skjerm, samt styrer vinduenes plassering og lignende.

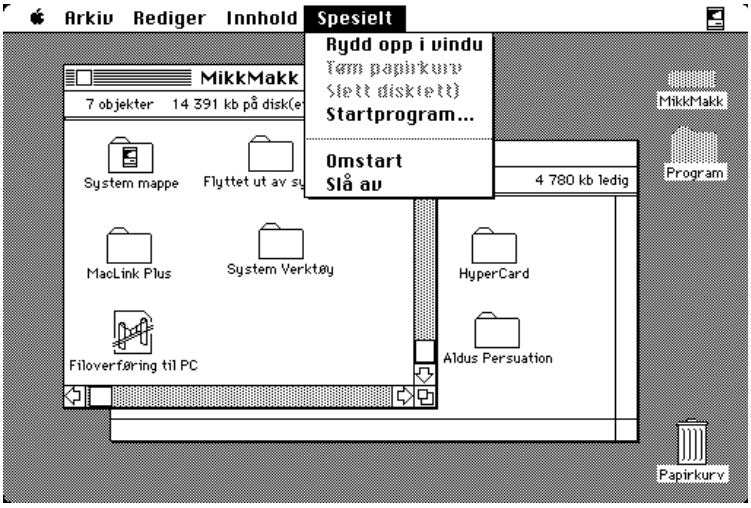
De grafiske vindusorienterte brukergrensesnitten er kommet for å bli og vi vil trolig se mer og mer av det i årene som kommer. Både fordi folk arbeider mer effektivt og fordi brukerne selv krever det. Jeg skal i det følgende kort beskrive på de mest vanlige brukergrensesnittene og peke på forskjeller og likheter i dem.

For en mer utførlig gjennomgang av ulike vindusmiljøer, se [KUTAL 90]

Macintosh

Macintosh var den første kommersielle maskinen med grafisk brukergrensesnitt som ble solgt i stort antall. Macintosh har et komplett grafisk brukergrensesnitt, med ting som vinduer, menyer, ikoner, dialogbokser og alarmbokser. Et program må benytte dem for å kunne kommunisere med brukeren, siden det ikke finnes noen mulighet for å gå rundt dem. Apple har skrevet en bok kalt *Human Interface Guidelines*. Denne oppfordrer alle programmerere til å følge et sett med metaregler for hvordan et brukergrensesnitt skal se ut og oppføre seg for at brukerne skal kjenne seg igjen fra applikasjon til applikasjon.

Brukeren kan styre applikasjonene ved hjelp av mus og tastatur. Normalt vil en rekke av de mest brukte operasjonene kunne utføres ved tastekombinasjoner som "Command"-*bokstav*, "Alt"-*bokstav*. Alene, sammen med "Shift"-tasten, eller i flere på en gang. Men den normale måten for en ny bruker er å benytte menyene som alltid befinner seg i øvre kant av skjermen. Den aktive applikasjonen viser sin meny. Menyer som tilhører andre applikasjoner kan ikke vises samtidig. For å skifte aktiv applikasjon peker man på en del av det vinduet man ønsker å gjøre aktivt. Det er ingen ekte muligheter for flere programmer å være aktive samtidig. Det er bare det programmet som er aktivt og fremst på skjermen til enhver tid som kan utføre noe.

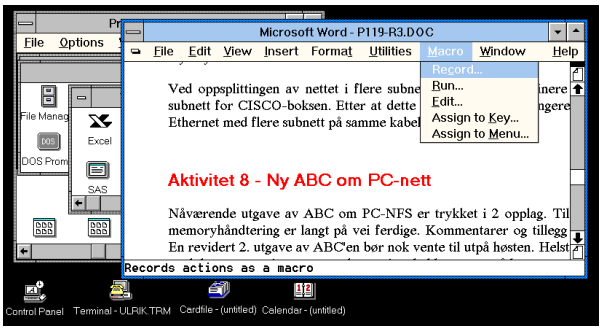


Figur 12 - Brukergrensesnitt på en Macintosh

Macintosh benytter overlappende vinduer og programmet må tegne opp igjen de delene av vinduet som har vært overlappet av andre vinduer når det aktiviseres og dermed også blir gjort til det øverste vinduet på skjermen. Alt som vises på skjermen, blir håndtert som rastergrafikk. Grafikk-mulighetene i QuickDraw inkluderer primitiver for å tegne tekst, linjer, rektangler, rektangler med runde hjørner, ovaler, buer og polygoner. Operasjonene på disse basiselementene inneholder muligheter for å fylle en flate med forskjellige mønstre. Skrifttyper er lagret i form av bitmaps og når QuickDraw skal tegne opp bokstaver eller tegn i en gitt type, størrelse og snitt, blir det enten tegnet fra bitmap, hvis den finnes, eller generert fra andre eksisterende skrifttyper.

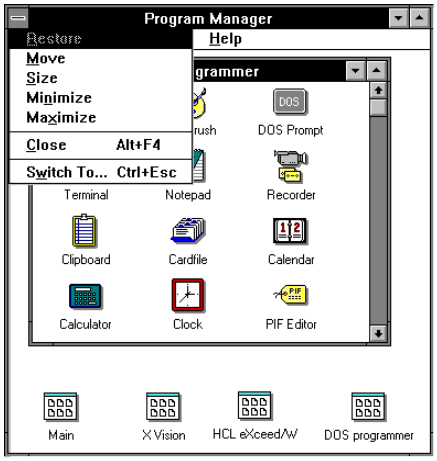
MS-Windows 3.0 og Presentation Manager

Til tross for at MS-Windows benytter MS-DOS og Presentation Manager benytter OS/2 som operativsystem, er likhetene i måten de arter seg for brukeren så ensartet, at jeg velger å beskrive dem sammen. PM er utviklet av IBM og Microsoft i samarbeid, mens MS-Windows er Microsofts eget produkt. Begge er grafiske vindusorienterte brukergrensesnitt som går som et skall over operativsystemet. Det er altså egentlig en selvstendig applikasjon og ikke en integrert del av operativsystemet. Programmer som ikke er skrevet mot dem kan fortsatt benyttes, enten i et eget vindu, eller ved at hele skjermen overlates til programmet. Dette er forskjellig fra Macintosh (hvor alle programmer må være skrevet for og kjøre under vindussystemet).



Figur 14 - Eksempel på Microsoft Windows versjon 3.0

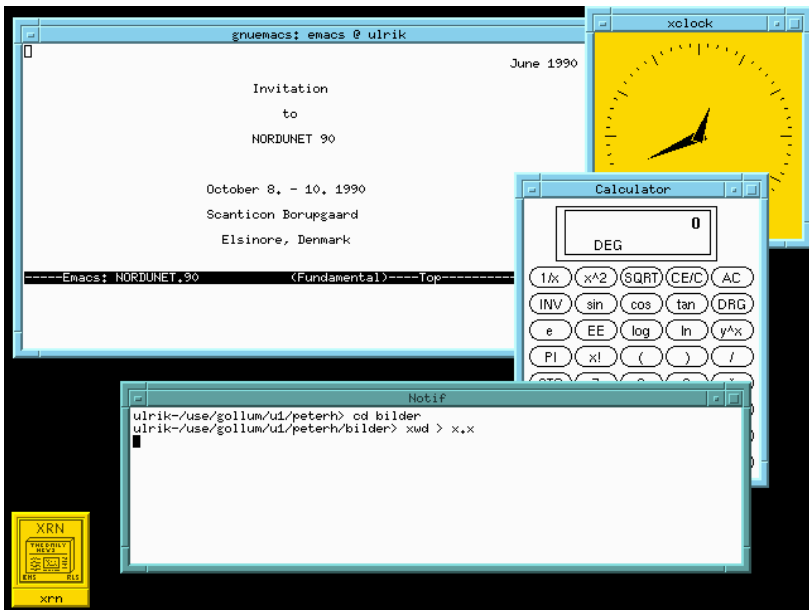
Brukeren kan styre applikasjonene ved hjelp av mus og tastatur omtent på samme måte som på en Macintosh, men det mulig å styre alle funksjoner bare med tastaturet. Ved hjelp av tastkombinasjoner kan brukeren aktivisere menyer, velge fra menyer, aktivisere vinduer og ikoner. Det normale for en ny bruker vil nok være å benytte menyene som finnes i overkant av vinduet for hver applikasjon ved hjelp av mus. For å skifte aktiv applikasjon peker man på en del av det vinduet man ønsker å gjøre aktivt. Det er også mulig å benytte kommandotaster eller en egen *"Task Manager"* for å rotere aktivt vindu. OS/2 er i stand til å utføre flere oppgaver samtidig og under Presentation Manager vil et program være aktivt og kunne utføre oppgaver, selv om det ligger i bakgrunnen, eller er ikonisert. I MS-Windows 3.0 finnes tre modi den kan kjøres i. I praksis er det bare maskiner med Intel 80386 prosessor som kan utføre flere samtidige oppgaver.



Figur 13 - Program Manager i Windows 3.0

Både PM og Windows benytter overlappende vinduer. Et Dialogbokser er begrenset til størrelsen av hovedvinduet. Hvis man forsøker å flytte det utenfor vil det bli klippet av mot kanten av hovedvinduet. Et program kan eie flere uavhengige vinduer. Begge systemer er uavhengig av hva slags grafisk skjerm som benyttes.

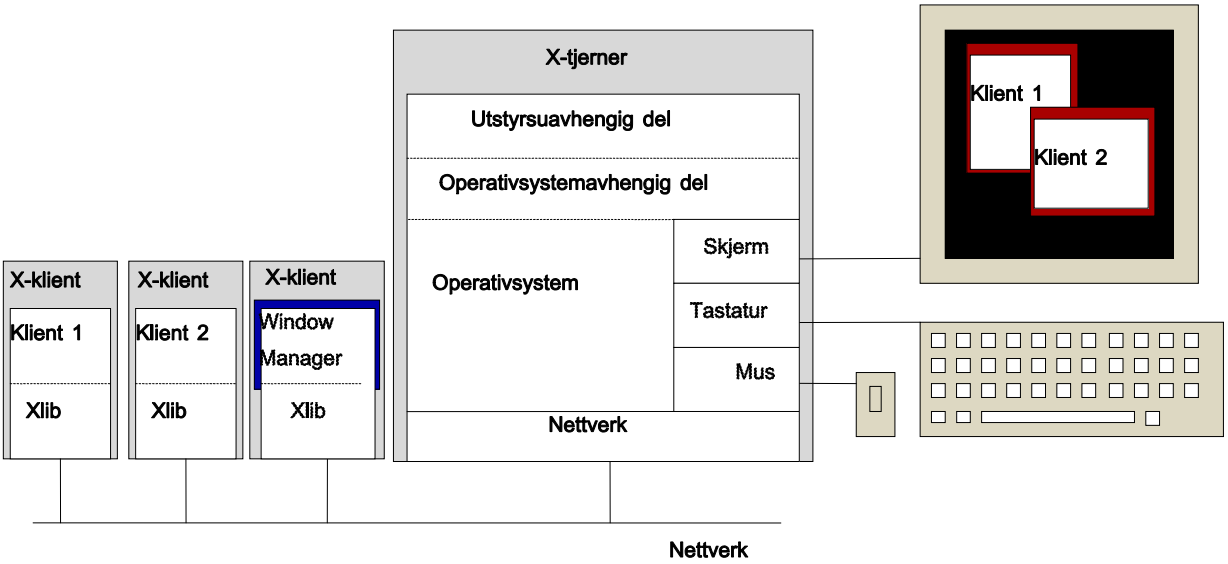
X-Windows



Figur 15 - Eksempel på X-Windows med OSF/Motif

X Windows Systems (eller bare X) er utviklet ved MIT (*Massachusetts Institute of Technology*) i forbindelse med det såkalte Athena prosjektet. Utgangspunktet var et ønske om å kunne distribuere bilder og annen grafikk rundt i nettet til grafiske arbeidsstasjoner fra ulike leverandører. Selv om det opprinnelige målet var begrenset til å kunne flytte grafikk rundt, har resultatet blitt et fullt grafisk vindus-orientert brukergrensesnitt. X er, eller er i hvert fall i ferd med å bli grafisk standard for UNIX-arbeidsstasjoner. Kildekoden er tilgjengelig fra MIT for en billig penge og X er implementert på en lang rekke maskinplattformer og for en lang rekke operativsystemer. De fleste større maskinleverandører tilbyr i dag X for sine grafiske arbeidsstasjoner, samt at det finnes egne terminaler for X og

programvare for mikromaskiner som MS-DOS Pc-er og Macintosher som gjør at de kan benyttes som X-tjenere.



Figur 16 - Klient-Tjener modellen i X Windows Systems

X er basert på tjener-klient modellen. Tjeneren (Eng. server) går på den maskinen som har skjermen, mens oppdateringene til skjermen kommer fra klienten. Klienten kan enten være et program på samme maskin som tjeneren, eller en annen maskin i nettverket. Dette gjør det mulig å kombinere arbeidsplassmaskiner fra ulike leverandører i nettverket og la dem benytte de samme applikasjonene [LARSEN 89] [LEE].

Det er viktig å merke seg at i X-Windows brukes uttrykkene klient og tjener omvendt av det som er vanlig i andre sammenhenger. X-tjeneren er det som driver skjermen, mens X-klienten er applikasjonen som benytter X-tjeneren for å kommunisere med brukeren.

Den nettverksorienterte arkitekturen i X Windows system, gjør den velegnet i et distribuert system. Den gjør applikasjonene uavhengig av hvilken maskinplattform som benyttes i klienten og tjeneren. Så lenge de begge benytter samme nettverksprotokoller, samt X-protokollen, er X-maskinvare uavhengig. Selve nettverket er transparent for X. Klient og tjener kan godt befinne seg i samme maskin. Maskinene trenger ikke engang være tilknyttet noe nettverk for at X skal kunne fungere, selv om man da selvsagt går glipp av den utmerkede muligheten til å benytte kollegaenes maskiner til tunge oppgaver. X-tjeneren, som er assosiert med en skjerm, har til oppgave å tegne opp det klienten ber om og returnere input fra tastatur og mus tilbake til klienten. Dette gjør at X-applikasjoner med mye grafikk, kan legge beslag på en stor del av kapasiteten i nettverket. Hver gang brukeren trykker på en knapp, flytter litt på musen, velger en meny, eller lignende sendes det pakker fram og tilbake mellom tjeneren og klienten. Hvis man for eksempel har en klokke på skjermen, vil denne kreve konstant oppdatering. Det er heller ingen muligheter i X for å finne ut hvilke applikasjoner man kan benytte, fra hvilke maskiner i nettverket.

Hvordan X arter seg for brukeren kan variere. Dette fordi det ikke er definert noen standard vindushåndtering, eller utgitt noen absolutte retningslinjer for hvordan applikasjoner under X skal arte seg. X som sådan er bare protokollen og arkitekturen. Vindushåndteringen (Eng. *Windows Manager*) er en X-klient som alle andre applikasjoner. Det finnes en rekke slike Window-Managere og disse oppfører seg tildels svært forskjellig. I tillegg er det mulig å

konfigurere en lang rekke parametere for å tilpasse systemet slik den enkelte foretrekker det. For en vanlig bruker vil det å konfigurere X og X-applikasjoner kunne arte seg som et sant mareritt i kryptiske filer, koder og oppsett. De siste par årene har OSF (*Open Software Foundation*) arbeidet med sine anbefalinger for MOTIF, som er ment å bli en slags standard vindushåndtering og retningslinjer for utseende og oppførsel til X-applikasjoner. UNIX International har jobbet hardt for å få sitt Open Look som standard vindushåndtering. Fordelen med OSF/MOTIF er at det ligner svært på Presentation Manager og Microsoft Windows, noe som gjør det enklere for brukerne å bytte mellom maskinplattformene. Til tross for manglende standardisering er det mulig å benytte de fleste X-applikasjoner uavhengig av hvilken vindushåndtering som benyttes i X-tjeneren. Og funksjonene som finnes er stort sett de samme. Det er overlappende vinduer, menyer og submenyer, muligheter for å endre størrelse og posisjon på vinduene, ikoner, klipp-og-lim og lignende. Problemet er at brukerne ikke har noe standard grensesnitt å forholde seg til. Inntil enten OSF/MOTIF, Open Look eller andre går ut som den absolutt dominerende standarden, vil brukerne bli skadelidende.

Andre

Det finnes en rekke andre grafiske brukergrensesnitt enn de som er nevnt her. Blant annet kan nevnes NextStep som ble introdusert med NeXT maskinen høsten 1988. Sun Microsystems Inc. grafiske vindussystem SunView for UNIX-arbeidsstasjoner og NeWS (*Network Extensible Windows System*), som er basert på PostScript rutiner. PostScript er egentlig et sidebeskrivelses språk for vanlige papirskrivere, men med visse utvidelser har det også fått en anvendelse i NeWS. For MS-DOS har det vært flere forsøk på grafiske brukergrensesnitt, med GEM som et at de mest kjente.

4.5. Konklusjon

Brukergrensesnitt kan deles i tre hovedgrupper: linjeorientert, skjermorientert og grafiske. Disse tre har ulike fordeler og ulemper. Sett fra brukerens synspunkt vil nok de grafiske vindusorienterte brukergrensesnittene virke mest tiltrekkende og gi det beste omgivelsene. Sett fra programutviklerens synspunkt, vil kanskje skjermorienterte brukergrensesnitt gi et resultat som er tilfredsstillende med langt mindre programmeringsinnsats enn tilfellet er for grafiske vindusorienterte grensesnitt. Selv linjeorienterte brukergrensesnitt kan være tilfredsstillende og/eller optimalt i en del sammenhenger. Mangfoldet av ulike vindussystemer gjør at det er vanskelig å tilby brukerne de samme distribuerte tjenester i form av applikasjoner på alle de aktuelle maskinplattformer. X-Windows skiller seg ut som en mulighet for å integrere flere maskinplattformer i et distribuert system, men det stiller store krav til kapasiteten i nettverket.

Det interessante sett fra denne oppgavens synspunkt, er at til tross for at fellestrekkene mellom brukergrensesnittene tross alt er ganske store (kanskje bortsett fra linjeorienterte brukergrense-snitt). De har felles ting som menyer, teksteditering, endring av skjermstørrelsen, gjen-oppfriskning av innholdet i skjermbildet osv. Jeg vil komme tilbake til dette i Kapittel 6.

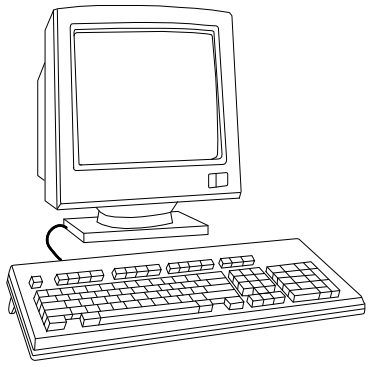
4.6. Arbeidsplassmaskiner

For å kunne beskrive hvordan distribusjon av tjenester kan og blir gjort, er det nyttig å se litt på de maskinplattformer som er i bruk. Dette bildet forandres fort og de tall som angis vil fort være foreldet, men det gir de teknologiske forutsetninger dette er skrevet under. Hvis man har enhetlig datamaskinutstyr vil integrasjonen mellom ulike anvendelser av informasjonsteknologi være enklere. Situasjonen er imidlertid ikke så enkel. Hvis man skal kunne ha tilgang til systemer på tvers av organisatoriske grenser, vil man uansett komme opp i problemer med integrasjon av mange ulike maskinplattformer. Det finnes en rekke arbeidsplassmaskiner, med en nesten like

lang rekke med ulike operativsystemer. For å begrense antallet, vil jeg her bare ta for meg dem som er i vanlig bruk ved Universitetet i Oslo. Dette er et utvalg som er temmelig vanlig og jeg mener det er tilstrekkelig for å illustrere problemområdet.

4.6.1. Terminaler

Det er lite behov for å ta med kortlesere, hullbånd ol., men det som nå omtales som "dumme terminaler" er det fortsatt mange som bruker. Det finnes et utall av ulike fabrikatet og modeller for terminaler, men Digitals VT serie dominerer for asynkrone terminaler og IBMs for synkrone. Hver leverandør har ofte hatt sin egen "standard", men støtte for VT100 har de fleste i tillegg til sine egne terminaler. Det har utviklet seg en slags de facto standard rundt terminalene fra Digital Equipment Corporation (Digital eller DEC). Det finnes en rekke programmer for de fleste arbeids-plassmaskiner som etterligner terminalene fra VT52 til VT340. VT100 har blitt stående som et slags minste felles multiplum som nesten alle har en løsning for. Denne terminaltypen har først og fremst sin begrensning i at den er 7-bit. De norske tegnene æøåÆØÅ blir lagret i som {|}[\] i 7-bit ASCII. VT220 og oppover løser dette med å tilby det som kalles DEC-multinational character set. De fleste applikasjoner og operativsystem er fortsatt skrevet mot VT100 og utnytter ikke VT220's muligheter.



Figur 17 - Terminal

Konfigurasjon

Terminalene har ingen generell prosessor som kan utføre noen av de oppgavene som kommer fra den datamaskinen de er oppkoblet mot. De har intern logikk som kan være implementert i form av en mikroprosessor, men denne brukes da kun til å vise output fra maskinen og ta input fra tastaturet. All interaksjon med maskinene styres av maskinene selv.

Nettilkobling

Terminaler har normalt en asynkron seriell forbindelse til en datamaskin. Ved hjelp av tilleggsutstyr kan de kobles til lokalnettet ved hjelp av en terminalkonsentrator. IBM har sitt eget opplegg for terminaler som er lite anvendbart for annet enn tilgang til IBM stormaskiner.

Brukergrensesnitt

De aller fleste terminaler er begrenset til linjeorienterte eller skjermorienterte brukergrensesnitt mot applikasjonene. Manglende standardisering på skjermhåndteringen i ulike terminaler skaper ofte problemer når det gjelder å benytte skjermorienterte brukergrensesnitt. Da særlig i et distribuert system, når man for eksempel skal benytte applikasjoner ved en annen institusjon ved hjelp av terminalaksess. Dette gjør at man i mange tilfeller blir henvist til dårlige linjeorienterte brukergrensesnitt fra terminaler.

4.6.2. X-terminaler

Det siste året har det dukket opp en ny type terminaler, kalt X-terminaler, som kan komme til å få en interessant plass i distribuerte systemer. X-terminalene er bygget for å være en X-tjener. Det vil si at den ikke kjører applikasjoner selv, men er avhengig av maskiner i nettet for å benytte X-klienter, men kan gi brukeren samme brukergrensesnitt som en datamaskin med X-tjener. Prisen ligger foreløpig i underkant av 30000, men den er forventet å synke drastisk hvis volumet tar seg opp. Fordelen med denne er blant annet støy, enklere sentral drift av ressursene, samt pris.

Den gjør det også mulig å la flere brukere dele maskinkraften i en UNIX-arbeidsstasjon og la alle ha samme brukergrensesnitt mot applikasjonene.

Konfigurasjon

En X-terminal er normalt utstyrt med fra 1.5 til 8 Mb hukommelse, en egen prosessor, samt høyoppløselig skjerm, tastatur og mus. Hvor mange applikasjoner som kan benyttes samtidig er avhengig av hukommelsen.

Nettilkobling

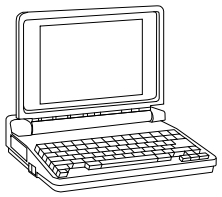
TCP/IP over Ethernet finnes for de fleste X-terminaler. I tillegg leveres tilkobling til Token-Ring og mulighet for seriell tilkobling for bruk som en vanlig terminal. Det er X-terminaler for DECnet. X-terminalene er avhengig av laste inn en del programvare, font definisjoner og annet fra en tjenermaskin i nettet.

Brukergrensesnitt

En X-terminal gir et X-Windows brukergrensesnitt, samt mulighet for å benytte den som vanlig terminal over en seriell forbindelse.

4.6.3. IBM kompatible Pc-er

Dette er den maskinfamilien som vokste fram som følge av IBMs lansering av IBM Personal Computer i 1981. Det finnes mange leverandører for denne type maskiner og prisen er etter hvert blitt meget lav. Fra 10-50000Kr for en normalt konfigurert maskin er vanlig.



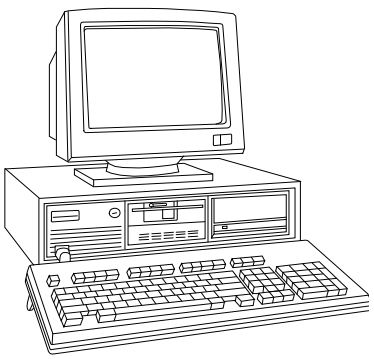
Figur 18
Laptop

Konfigurasjon.

IBM PC, IBM PS/2 eller kompatible maskiner er svært utbredt. Disse har en prosessor fra Intel (80x86 familien). Normalt vil de ha 640KB intern minne for DOS bruk, samt at en del vil ha tilleggshukommelse på normalt 2-8MB.
Normalt vil de også ha en innebygget harddisk på mellom 20 og 100MB. Det finnes en rekke ulike grafikkstandarder for disse maskinene, men EGA og VGA er det mest utbredte. Det kan kobles til mus og en rekke andre pekeredskaper, men det like vanlig ikke å ha dette. Bærbare Pc-er eller såkalte laptoper skiller seg etter hvert lite fra de stasjonære maskinene med hensyn på ytelse og muligheter. De har derimot fordelen at man kan ha tilgang til datakraft hvor som helst.

Operativsystem

MS-DOS er det alt framherskende operativsystemet til disse maskinene. Det er riktig nok laget både UNIX-varianter, CP/M kloner og andre operativsystemer for denne maskinfamilien men ingen har enda kunnet true MS-DOS som operativsystem selv om IBM og Microsoft satser mye på at OS/2 med tiden skal kunne erstatte MS-DOS.



Figur 19 - Bordmodell

MS-DOS

MS-DOS er et svært enkelt operativsystem som kun kan kjøre en oppgave a gangen og det har ingen tilgangs- eller sikkerhetsbegrensninger i systemet. Operativsystemet er begrenset til 640KB og spesielle drivere og omprogrammering er påkrevd for å kunne komme rundt denne grensen. Det er mulig å lage programmer som er interruptdrevet og går

i en slags semiparallellitet, men MS-DOS gir ingen direkte støtte for dette. For å bevare kompatibiliteten med gammelt utstyr er MS-DOS fortsatt basert på instruksjonssettet i 8088 prosessoren og utnytter ikke de nyere prosessorenes muligheter. På grunn av den lave hastigheten på 8088 prosessoren er en rekke programmer skrevet for å gå utenom operativsystemet og direkte mot maskinvaren.

MS-DOS Brukergrensesnitt

Interaksjon med brukeren er i utgangspunktet kommandobasert, men det finnes flere tilleggs-produkter som tilbyr grafiske brukergrensesnitt basert på vindusteknikk og musstyring. Mest oppmerksomhet har Microsoft Windows fått. Dette løser delvis noen av problemene med MS-DOS, samt at det utnytter de nyere prosessorenes muligheter. Skjermorienterte brukergrensesnitt har hatt stor utbredelse på denne maskinplattformen. En stor overvekt av programmene er enda skjermorienterte med mulighet for å gå over i grafisk modus når det trengs. Det er utviklet mye og avansert skjermorientert programvare for MS-DOS. Det finnes programvare som gjør at en MS-DOS maskin kan benyttes som X-terminal.

OS/2

OS/2 ble lansert 1.april 1987 og er nå tilgjengelig i versjon 1.2 med et grafisk brukergrensesnitt a la MS-Windows (eller omvendt) kalt Presentation Manager. Dette er et langt mer avansert operativsystem enn MS-DOS og inneholder de funksjonene man trenger for å bygge en avansert arbeidsplassmaskin. Foreløpig har ikke OS/2 "tatt av" som operativsystem. Mye på grunn av mangelen på programvare og fordi det i praksis krever en kraftig 80386 basert maskin. En ny versjon av OS/2 er annonsert fra IBM for levering høsten 1991. Denne skal gjøre det mulig å benytte både MS-DOS og MS-Windows programvare under OS/2 bedre enn i MS-DOS. Hvis denne versjonen inneholder det ryktene forteller, vil det kunne bli et gjennombrudd for OS/2 som operativsystem.

OS/2 Brukergrensesnitt

OS/2 er utstyrt med de samme muligheter for brukergrensesnitt som MS-DOS. Det er en linjeorientert kommandomodus mot selve operativsystemet og en del programmer er utviklet som skjermorienterte applikasjoner. Presentation Manager ble lansert samtidig med OS/2, men var ikke ferdig utviklet før OS/2 versjon 1.2. Dette har nok sinket noe av utbredelsen av OS/2, da markedet ser ut til å kreve Presentation Manager versjoner av programmene. For brukeren er det minimale forskjeller mellom brukergrensesnittet i MS-Windows og Presentation Manager. IBM har annonsert at de planlegger å lage et produkt som gjør det mulig å benytte en maskin med OS/2 som X-terminal.

MS-DOS Nettilkobling

Det er laget en rekke nettverkssystemer for MS-DOS maskiner. Disse finnes for omtrent det som er av tilgjengelige nettverksløsninger for datamaskiner. De fleste av dem er rettet mot å koble flere MS-DOS maskiner sammen slik at de kan dele fillagre og skrivere, samt annet periferiutstyr. Dette er for det meste svært MS-DOS orientert. De fleste av systemene er svært lukkede og ikke stort annet enn forvaltere av felles disker og skrivere. Det vil si at det ikke er noen triviell oppgave å knytte andre maskinplattformer inn i nettet og få dem til å kommunisere. Slike løsninger, der de finnes, er stort sett dyre og med en rekke mangler. I de senere årene har det også blitt laget programmer som bruker de samme kommunikasjonsprotokoller og nettverkstopologi som UNIX-maskiner, nemlig Ethernet, TCP/IP med NFS, NIS, RPC mm. Ved å bruke Network File System (NFS) kan MS-DOS maskiner dele fillagre med UNIX-maskiner. I motsetning til de

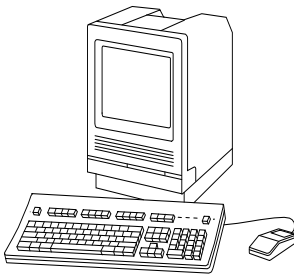
spesialiserte og lukkede løsningene, gir slike systemer brukerne av MS-DOS maskiner tilgang til de samme distribuerte tjenestene som finnes på UNIX-arbeidsstasjoner.

OS/2 Nettilkobling

For OS/2 finnes så langt kun selve TCP/IP delen. IBM har annonsert NFS, NIS og andre produkter for distribuerte tjenester.

4.6.4. Macintosh

Denne maskinen bygger på ideene fra Alto maskinen ved Xerox PARC. Den er først og fremst kjennetegnet med sitt grafiske brukergrensesnitt og mus som pekeredskap. Maskinene ligger typisk fra 10-90000Kr alt avhengig av konfigurasjon.



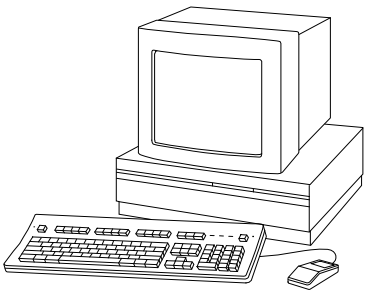
Figur 20
Klassisk Macintosh

Konfigurasjon.

Macintosh er basert på prosessorer fra Motorola i 680x0 serien. Normalt har de fra 512KB til 8MB hukommelse og fra 20-160MB disk. Grafikken er den samme på alle maskintypene. De har alltid en mus som pekeredskap. Maskinen er ubrukelig uten.

Operativsystem/brukergrensesnitt

Finder (og MultiFinder) er det vanligste operativsystemet. Det er også utviklet en UNIX-variant, men denne har foreløpig en beskjeden utbredelse. All interaksjon med brukeren skjer ved hjelp av ikoner og rullegardinmenyer. Finder er i utgangspunktet begrenset til en oppgave av gangen og er enbrukerorientert. Mulitifinder gir muligheten for å hoppe mellom applikasjoner uten å måtte avslutte hver av applikasjonene. Det finnes muligheter for en slags semiparallellitet i bakgrunnen. Det finnes produkter som gjør at man kan benytte en Macintosh som X-terminal.



Figur 21 - Mac II

Nettilkobling

Macintosh var i starten kritisert for sine manglende muligheter for å kommunisere med omverdenen. Dette har Apple rettet opp ved å lage sin egen nettverksprotokoll AppleTalk som kan gå over enten tvunnet parkabel (LocalTalk/Phonenet) eller over Ethernet (EtherTalk). Tredjeparts leverandører har kommet med broer mellom LocalTalk og Ethernet, samt at det er laget TCP/IP for Macintosh som muliggjør kommunikasjon mot UNIX-maskiner. Den lenge etterlengtede NFS for Mac har enda ikke materialisert seg. Derimot er det utviklet et eget UNIX-program kalt EtherShare som gjør at Macintosh kan få tilgang til filer på en UNIX-maskin og fra den maskinen igjen bruke NFS for å nå filer på andre maskiner som har NFS.

4.6.5. UNIX-arbeidsstasjoner

Dette er maskiner som stort sett har noen få ting felles, nemlig at de har UNIX som operativsystem og at de har en høyoppløselig skjerm og mus. Ellers kan de variere sterkt i hastighet og muligheter. Mye av grunnfilosofien for distribuert databehandling har sprunget ut fra typiske UNIX-miljøer. Det er derfor naturlig at det er på denne plattformen det er enklest å få til et distribuert system.

Konfigurasjon

UNIX-arbeidsstasjoner er bygget rundt flere prosessortyper. Digital benytter MIPS og VAX prosessorer, Sun har brukt Motorola 680x0, men satser nå på sin egenutviklede RISC *(Reduced Instruction Set Computer*). IBM har sin egen RISC prosessor. Det finnes også en del andre prosessorer i bruk på UNIX-arbeidsstasjoner. De er typisk utstyrt med minimum 4MB intern hukommelse, normalt 8-16MB. Skjermen er som regel høyoppløselig med 1024-768 punkter eller mer. Som oftest vil det også være en innebygget Ethernet-tilkobling. De bruker mus som pekeredskap. En god del av UNIX-arbeidsstasjonene har ikke intern disk, men bruker en annen maskin i nettet for å starte operativsystemet og som fillager. Disse såkalte diskløse arbeidsstasjonene er som oftest lydløse.

Operativsystem

UNIX er et gammelt operativsystem og opprinnelig laget for å brukes som et tidsdelings operativsystem på flerbruker datamaskiner. Det har virtuell minnehåndtering. En av årsakene til at UNIX har vokst fram som et mye brukt operativsystem er at de mest brukte nettverksprotokollene er utviklet for UNIX og dermed også godt tilpasset. Det er også et operativsystem som er utviklet for og av programmerere og har derfor god støtte for programutvikling.

Brukergrensesnitt

All interaksjon med brukeren i UNIX er i utgangspunktet kommandobasert, men X-Windows er mye brukt. En UNIX-arbeidsstasjon vil normalt ha både klient og tjener funksjonen i X-Windows. Det er også mulig å lage skjermorienterte brukergrensesnitt i UNIX, eller terminalemulatorer mot andre stormaskiner som har slike programmer.

Nettilkobling

De aller fleste UNIX-arbeidsstasjoner kommer med innebygget Ethernetkort og har TCP/IP med NFS, NIS og de andre applikasjonsprotokollene for distribuerte tjenester i Internett. Det begynner også å komme en del produkter basert på OSI-protokoller. Det er forventet at Berkeleys variant av UNIX (BSD) i versjon 4.4 vil inneholde både OSI og TCP/IP som standard.

4.6.6. Andre

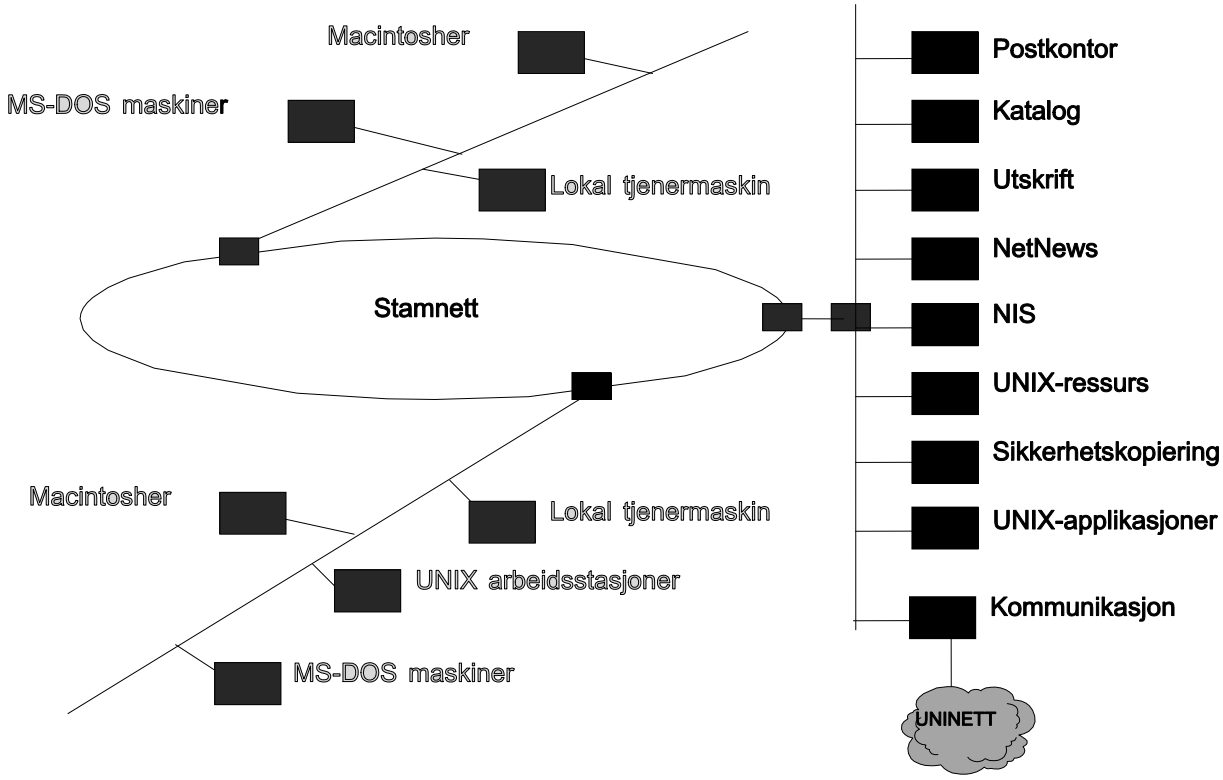
Det finnes en rekke ulike arbeidsplassmaskiner. For at disse skal kunne benyttes i et distribuert system, kreves det at de har tilgang til nettverket og benytter de protokollene som trengs for at de skal få tilgang til de distribuerte tjenestene brukeren har behov for. Det er en temmelig sterk trend i retning av standardisering rundt de tre maskinplattformene som her er nevnt. Nettopp manglende muligheter for nettilknytning er ting som gjør at andre typer arbeidsplassutstyr ikke får innpass i markedet. For at en "dark horse" skal kunne komme inn på banen er nettopp slike tinge ting etter hvert blitt avgjørende. Mye tyder vel på at de store revolusjoners tid er forbi og vi i årene framover mer vil få se evolusjon av de maskinplattformer som alt er i bruk.

4.7. Tjenermaskiner

For å kunne fordele belastningen og sikre en mer optimal utnyttelse av ressursene er det nok fornuftig å dele på funksjonene maskinene i et distribuert system skal fylle. Hvilke maskiner og operativsystem som benyttes for tjenermaskiner vil kunne variere, alt etter hvilke oppgaver de er ment å ha. Det vanligste innen forskning og utdanning er å benytte standard UNIX-maskiner med full tilgang til Internet og/eller OSI. En og samme UNIX-maskin kan i mange tilfeller benyttes som arbeidsstasjon og tjenermaskin.

4.7.1. UNIX tjenere

Skille mellom en UNIX-arbeidsstasjon og en UNIX-tjenermaskin går stort sett på hvilke oppgaver maskinen har som sin hovedoppgave. Ved Universitetet i Oslo er det satt opp spesialiserte tjenermaskiner for oppgaver som elektronisk post, X.500 katalog, utskriftstjenester, brukerautentisering, applikasjonstjener for UNIX programvare, filtjener for Pc-er, Macintosher og diskløse arbeidsstasjoner, nettverksovervåkning, maskinovervåkning og fjerndrift, sikkerhetskopiering, samt egne tjenermaskiner for fakulteter, institutter eller avdelinger. Etter hvert vil trolig også ulike administrative systemer bli lagt til UNIX-maskiner. De maskinene som benyttes varierer mye i maskinkraft, intern hukommelse og diskplass og man forsøker å tilpasse maskinene best mulig til de oppgavene de er tildelt. Etter som kapasitetsbehovet, tilgjengelig teknologi og pris forandres, kan maskiner omdisponeres for å få best mulig ytelse ut av den tilgjengelige maskinparken.



Figur 22 - Eksempler på tjenere i UiOs nett

4.7.2. VAX/VMS tjenere

Operativsystemet VAX/VMS fra Digital Equipment Corporation har lenge hatt en høy stjerne innen den akademiske verden. Dette gjør at en lang rekke forskningsrelaterte programsystemer er utviklet for denne maskinplattformen. Det er den såkalte VAX-arkitekturen som benyttes i disse maskinene og Digital og tredjeparts leverandører har sørget for Internet og OSInet tilknytningsmuligheter for disse maskinene i tillegg til Digitals eget DECnet. Det finnes både X tjener og klient for VAX/VMS arkitektur. Dette gjør at de kan benyttes som arbeidsstasjoner og tjenermaskiner omtrent på lik linje med UNIX-maskiner. Det er allikevel en klar dreining vekk fra VAX/VMS, som er et proprietært operativsystem, til fordel for UNIX. Digital har også tatt konsekvensen av dette og leverer sin UNIX-variant Ultrix for VAX-arkitektur.

4.7.3. Andre

I praksis kreves det av en maskin som skal fungere som tjener at den er tilknyttet et datanett og at den kan understøtte de protokoller som er nødvendig for å fylle sin oppgave som tjenermaskin for distribuerte tjenester. For eksempel kan en SINTRAN maskin fra Norsk Data med TCP/IP og Ethernet godt fungere som tjenermaskin. For eksempel for databaser basert på SIFT (*Søking i fri tekst*). Tilsvarende kan en IBM stormaskin med nødvendig programvare være en god databasemaskin. Etter hvert finnes det OSI og/eller Internet muligheter for de fleste maskinarkitekturer, slik at de kan benyttes som tjenermaskiner i distribuerte systemer.

4.7.4. Valg av tjenermaskin

Momenter man bør ta i betraktning ved valg av tjenermaskin er:

- **Tilpassning til oppgaven.** Ulike maskinarkitekturer kan passe til ulike oppgaver. Viktige faktorer er ting som prosessorhastighet, I/O hastighet til hukommelse og disk, overføringshastighet på intern bus og sist, men ikke minst det totale pris/ytelsesforholdet. Men kanskje viktigst er at maskinen har den programvare som trengs og muligheter for å benytte de kommunikasjonsprotokoller som er aktuelle.

- **Teknologisk framtid.** Foregår det videreutvikling av den arkitekturen maskinene benytter? Satser leverandøren på den plattformen? Det er svært kjedelig å oppdage at leverandøren dropper en arkitektur etter at et stort system er utviklet på den. Hvis den blir skjøvet i bakgrunnen som en mindre interessant plattform, vil gjerne vedlikehold og oppgraderinger bli dyrt. Med andre ord en slags melking av kunder med gammelt utstyr. Ofte er alder en indikasjon. Svært gamle systemer er gjerne på vei ut, mens helt nye systemer derimot, har ingen garanti for at de finner sin plass i markedet.

- **Utbredelse.** Hvor mange installasjoner finnes rundt om i verden? Og hvor mange finnes i egen region? Dette sier som regel noe om hvilket støtteapparat leverandøren har råd til å holde oppe rundt en maskinplattform. Det gir også økte muligheter for at det utvikles gratis programvare og annet man kan dra nytte av. Det vil også gi økte muligheter for at tredjeparts leverandører kommer med gode produkter.

- **Intern kompetanse.** Å opprettholde kompetanse på et vidt spekter av maskiner er svært kostbart. Dette er en faktor som tilsier at man så langt det er praktisk mulig bør begrense antallet ulike typer maskinplattformer man benytter som tjenermaskiner.

4.8. Integrering av arbeidsplassmaskiner og tjenermaskiner

Jeg har i de foregående kapitler pekt på hvorfor distribuerte systemer kan være gunstig og hvilke tjenester som lar seg distribuere, samt pekt på noen av de internasjonalt anerkjente standardene og noen av de facto standardene som finnes. Videre har jeg pekt på hvilke brukergrensesnitt som finnes og hvilke maskinplattformer som er utstrakt bruk.

Den store oppgaven ved nyutvikling av slike systemer, er ikke å lage tjenerdelen. Det er en forholdsvis triviell sak slik det lages i dag. I og med at man ikke har noe felles grensesnitt å forholde seg til, må det ofte lages linjeorientert. Noe som ikke gir det optimale grensesnitt for brukeren. Alt som skal til for at brukergrensenittet skal bli presentabelt, må gjøres på hver enkelt maskinplattform. Det må lages et grensesnitt for Macintosh, et for MS-DOS, et for UNIX og muligens et for VAX/VMS og OS/2 også. Hvis man virkelig skal dra full nytte av brukergrensesnittene, må det faktisk lages to eller tre på alle plattformer, unntatt Macintosh. For

MS-DOS maskiner trenger man i hvert fall et skjermorientert og fortrinnsvis også et MS-Windows grensesnitt, samt muligens et linjeorientert i tillegg. Samme gjelder for UNIX. Der trengs et for X-Windows, et skjermorientert for VT100 og et linjeorientert. VAX/VMS og OS/2 kommer langt bak i rekken og vil nok bli avspist med et linjeorientert brukergrensesnitt. Skal man altså lage et optimalt system med maksimal brukervennlighet ved Universitetet i Oslo, må man lage 12 brukergrensesnitt mot 5 operativsystemer.

Det er denne problematikken jeg vil konsentrere meg om i resten av oppgaven

5. Modeller for applikasjonsaksess

Som beskrevet i kapittel 3, er det utviklet en rekke systemer som muliggjør en tettere integrasjon av maskiner med forskjellig operativsystem og maskinvare. Det er imidlertid ett problemområde som i liten grad har vært gjenstand for oppmerksomhet, nemlig brukernes muligheter for å få applikasjoner mot distribuerte tjenester til sitt arbeidsplassutstyr.

For en bruker som til daglig benytter en Macintosh, kan det være en frustrerende opplevelse å ta i bruk for eksempel elektroniske postsystemer som kun finnes på en felles tjenermaskin. Brukeren må starte et terminalaksessprogram, logge inn og deretter bli presentert for et linjeorientert brukergrensesnitt med masse kompliserte kommandoer som langt fra ligner på det brukergrensesnittet Macintosh applikasjoner benytter. Dagens trend mot åpne og distribuerte systemer krever nye metoder for applikasjoners håndtering av ulike typer arbeidsplassmaskiner. Jeg skal her gå gjennom de ulike måtene dette blir gjort på nå og se om det kan gjøres på en mer effektiv måte. Jeg vil forsøke å vise at vi med dagens modeller har valget mellom et minste felles multiplum mot et felles brukergrensesnitt, eller å lage nye brukergrensesnitt for hver maskinplatt-form for hver eneste applikasjon.

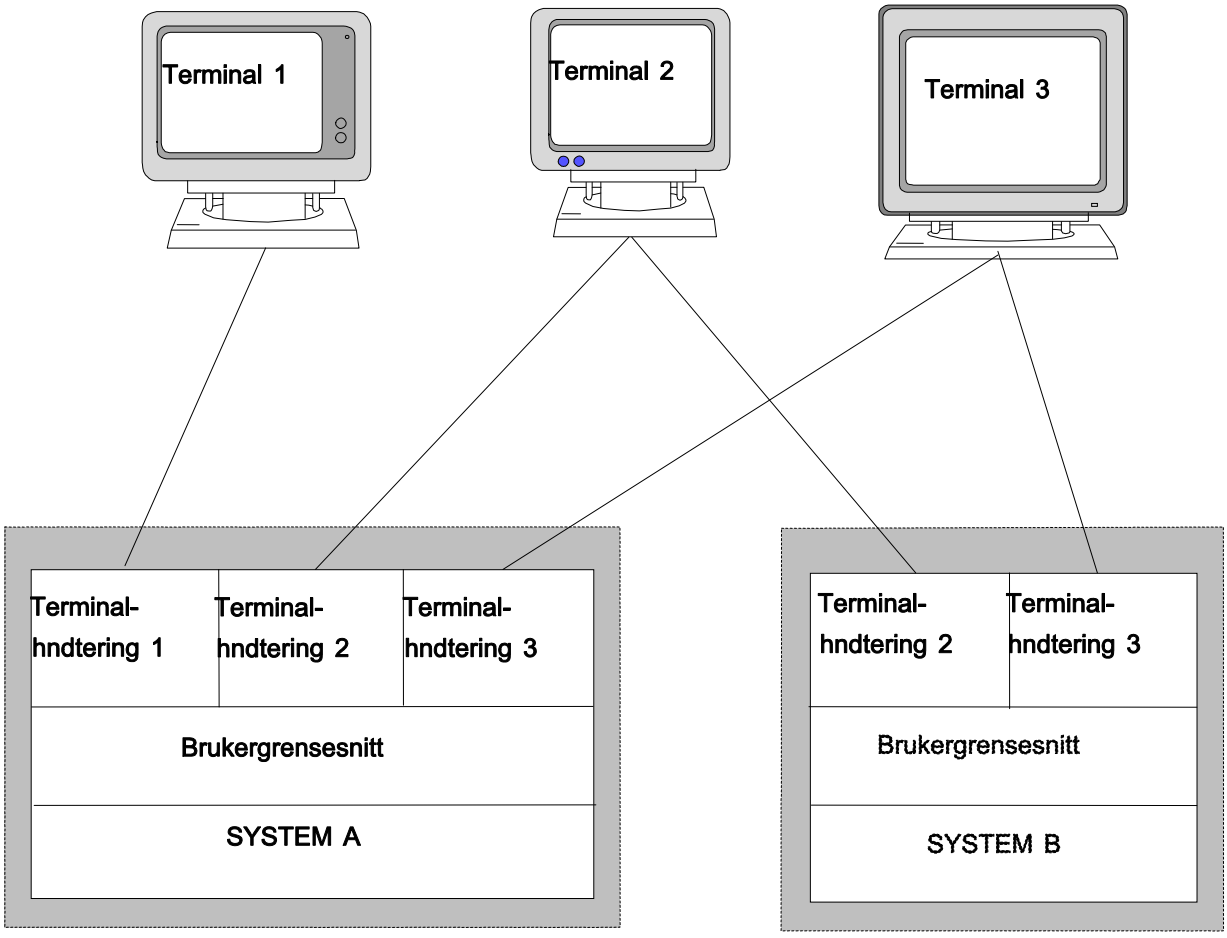
Jeg har identifisert to modeller for hvordan tjenester distribueres mellom to eller flere datamaskiner. Disse vil jeg omtale hver for seg og peke på styrke og svakheter ved hver av dem. Til slutt vil jeg ta for meg mitt forslag til modell for distribusjon av tjenester. De tre modellene jeg vil behandle i dette kapittelet er:

- Tradisjonell terminalmodell
- Klient-Tjener modellen
- Den distribuerte applikasjonsmodellen

Disse tre modellene er ikke motsetninger. Dagens situasjon er en blanding av de to første modellene. Hvis vi holder terminaler utenfor kan det aller meste av arbeidsplassutstyr fungere etter alle modellene og ofte også på samme tid. En Macintosh med Multifinder kan f.eks. kjøre et terminalaksessprogram som Kermit eller Telnet, og samtidig kjøre Microsoft Word som en lokal applikasjon. I tillegg kan den benytte et elektronisk post program som kommuniserer med postsystemet på en UNIX-maskin etter klient-tjener modellen. Med den distribuerte applikasjonsmodellen får brukeren i tillegg tilgang til applikasjoner som følger denne modellen. Alle tre modellene kan altså benyttes samtidig.

5.1. Den tradisjonelle terminalmodellen

Den tradisjonelle måten å håndtere applikasjoners brukergrensesnitt på har vært å la den styre terminalen direkte. Applikasjonen har selv bestemt at nå skal skjermen blankes, markøren flyttes i øvre venstre hjørne og så skrive ut et skjermbildet. Det er applikasjonen som har holdt kontroll med hvor markøren til en hver tid befinner seg og hvilke taster brukeren har trykket. Det er en rekke ulike "standarder" for hvordan slik terminalstyring blir gjort. For å forenkle jobben med å utvikle nye applikasjoner har det gjerne vært laget moduler for terminalhåndtering som applikasjonsutvikleren har kunnet benytte. De fleste datamaskinleverandører har laget muligheter for å håndtere ulike terminaler i sine operativsystemer. I de tilfeller terminalhåndteringen har ligget innbakt i selve applikasjonen, har man måttet skrive om applikasjonen for å få den til å fungere mot en ny terminaltype. Kundene har tilpasset seg dette og stort sett kjøpt de terminaler som leverandøren har gitt støtte for i sitt system. Terminalmodellen er i praksis alltid begrenset til linje- eller skjermorienterte brukergrensesnitt.

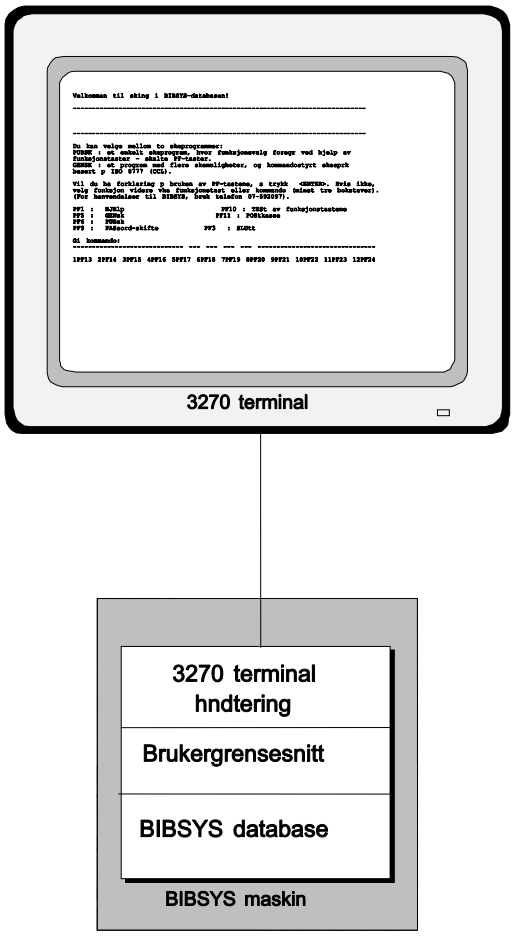


Figur 23 - Tradisjonell terminalmodell

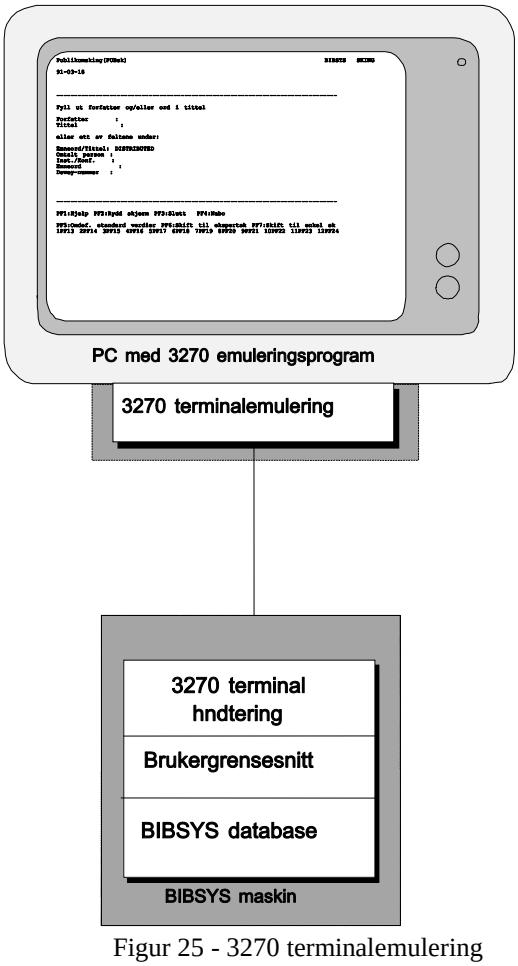
I Figur 23 er det tre ulike terminaltyper. Dette kan enten være virkelige terminaler av forskjellig type, eller andre datamaskiner som er i stand til å emulere en gitt terminaltype. Maskinene er to maskiner av ulikt fabrikat. Terminal 1 støttes bare av system A, mens terminal 2 og 3 støttes av begge. Brukerne av terminal 2 og 3 har altså tilgang til begge systemer, mens brukerne ved terminal 1 ikke har aksess til system B. Det som mangler for terminal 1, skal få tilgang er en terminalemulering for denne terminaltypen. Hvis dette er innbakt i programvaren kan det være en stor jobb. Mens hvis selve terminalhåndteringen ligger i forbindelse med operativsystemet er det mulig leverandøren kan bidra med terminalhåndteringen. Hvis for eksempel terminal 1 er en IBM

3270 terminal vil den ikke kunne brukes mot for eksempel en Norsk Data maskin, fordi måten denne forventer at terminaler skal oppføre seg på, ikke oppfylles av en 3270 terminal. Normalt vil det være samme brukergrensesnitt som benyttes mot alle typer terminaler. Det har ikke vært vanlig å lage forskjellige versjoner av applikasjoner for forskjellige terminaltyper.

Før mikromaskinene og arbeidsstasjonene fikk utbredt popularitet i løpet av 80-årene, var terminalmodellen den eneste muligheten for brukerne å komme i kontakt med datamaskiner. Fortsatt er det eneste mulighet for å aksessere en del større informasjonssystemer. Men terminaler er ikke lenger det brukeren primært ønsker som sitt arbeidsplassutstyr. En rekke oppgaver, som for eksempel tekstbehandling, utføres langt raskere og bedre på en personlig Macintosh, UNIX-arbeidsstasjon eller MS-DOS PC' enn på en felles stormaskin. Når arbeidsplassutstyret skal brukes på samme måte som en tradisjonell terminal, har de samme begrensede funksjonalitet som en tradisjonell terminal. Dette skyldes det linje- eller skjermorienterte brukergrensesnittet terminalorienterte systemer normalt er begrenset til.



Figur 24 - Eksempel på BIBSYS fra terminal



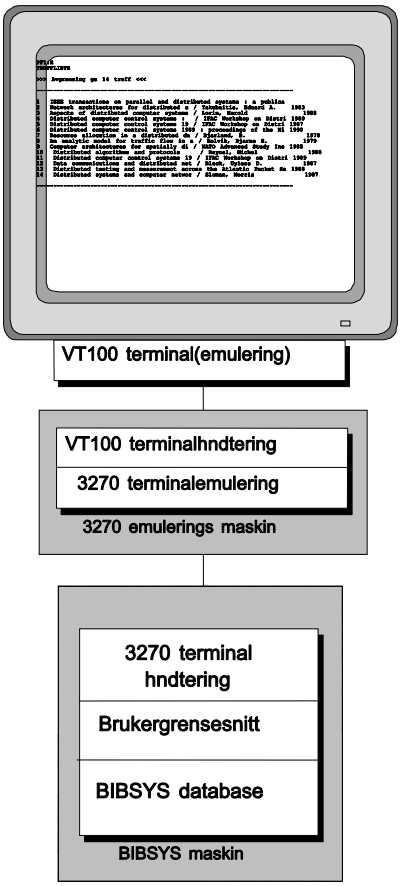
Figur 25 - 3270 terminalemulering

Et typisk eksempel på terminalmodellen er Universitetsbibliotekenes database BIBSYS. Denne er lagt til en IBM-stormaskin i Trondheim og er basert på IBM 3270 terminal eller terminalemulering mot denne. Valget av en sentralisert løsning med terminalaksess har selvsagt forenklet utviklingen av systemet for dem som gjorde selve utviklingsjobben, men har medført problemer for brukerne. Ettersom universitetsbiblioteket er spredd på en lang rekke steder, var det første problemet å få dem fysisk tilknyttet universitetenes lokale nettverk for å kunne kommunisere over UNINETT mot Trondheim. Deretter var det en god del arbeid med å finne terminalemuleringsprogrammer for

ulike typer arbeidsplassutstyr. Løsningen har i mange tilfeller blitt å gå via en egen terminalemuleringsboks som er satt foran selve maskinen. Her blir VT100 emulering oversatt til IBM 3270 og tilbake igjen. Løsningen fungerer, men har ikke gitt brukerne noe fullgodt brukergrensesnitt, i hvert fall ikke hvis man skal sammenligne det med hva som er mulig. Det krever også en god del av den som skal bruke systemet, fordi oppkoblingen i en rekke tilfeller går via en annen maskin. Dette innebærer at man først må logge seg inn på en lokal datamaskin, deretter videre til BIBSYS og så gjennom flere innlogginger der.

BIBSYS er neppe noe ekstremt system i denne sammenhengen. Universitetet i Oslo benytter terminalmodellen mot sine administrative systemer, som er plassert på en CYBER. Tilsvarende problemstillinger gjelder for en rekke større informasjonssystemer. Mulighetene for å komme i kontakt med disse er som regel begrenset til bestemte typer terminaler, og via en bestemt type nettverk, som systemet er skrevet for. Dette er selvsagt ikke noen kritikk av systemene som sådan, men av måten brukeren kommer i kontakt med og kommuniserer med systemet.

Terminalorienterte systemer utnytter langt fra de muligheter som finnes i dagens arbeidsplassutstyr. Riktignok kan ett terminalemulatorprogram gå i et vindu på en X-terminal, på en Macintosh, MS-Windows eller Presentation Manager, men det gir i seg selv ingen økt funksjonalitet for applikasjonen som går i vinduet. En fordel er det likevel at man kan klippe informasjon ut av skjermbildet og lime den inn en tekstbehandler. Brukerdialogen i BIBSYS og UiOs administrative systemer er svært forskjellige, og brukeren har liten glede av å kunne det ene, når man skal ta i bruk det andre. De rullegardinmenyene som kommer opp har med terminalemuleringen å gjøre, og ikke med selve applikasjonen, slik de burde. Det er altså en rekke ankepunkter mot denne modellen i et distribuert system.



Figur 26 - 3270 emulering via en maskin som oversetter VT100 til 3270.

5.1.1. Fordeler og ulemper med terminalmodellen

Fordeler

- Ett brukergrensesnitt for alle, uansett arbeidsplassutstyr. Dette gjør at dokumentasjon og brukeropplæring for en gitt applikasjon blir den samme uansett hvilken type arbeidsplassutstyr brukeren har.

- Billigere å implementere fordi det bare er et brukergrensesnitt som skal lages for hver applikasjon.

Ulemper

- Begrensede muligheter for å lage brukervennlige grensesnitt fordi man er begrenset til linje-/skjermorienterte brukergrensesnitt

- Andre brukergrensesnitt enn for lokale applikasjoner på arbeidsplassutstyret.

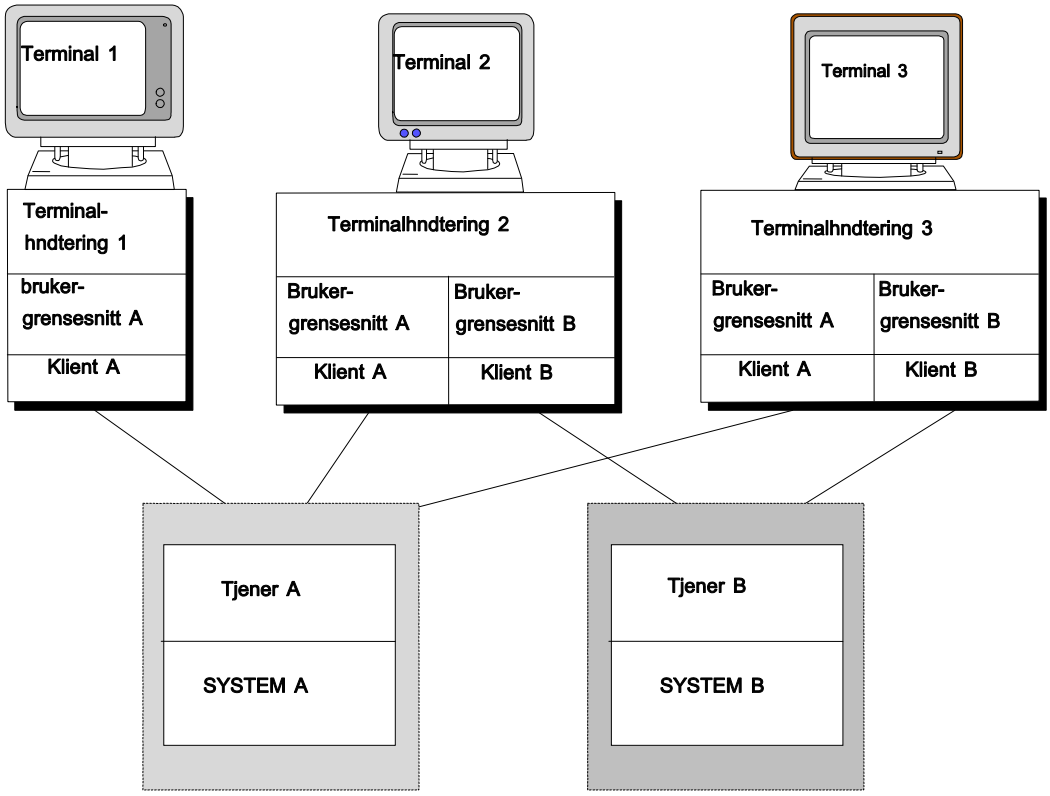
- Ofte forskjellige brukergrensesnitt i de ulike applikasjonene. Selv på samme maskin kan det gjerne være flere typer brukergrensesnitt. Brukernes kunnskap fra en applikasjon kommer derfor ikke til nytte i andre.

- Må lage terminalhåndtering for hver terminaltype systemet skal støtte. Noe som ofte resulterer i at applikasjonene skrives mot et minste felles multiplum.

5.2. Klient-tjener modellen

Klient-tjener modellen deler applikasjonene opp i to. En presentasjonsdel som går på den enkelte arbeidsplassmaskin og en tjenerdel som går på en større flerbrukermaskin. Denne metoden utnytter arbeidsplassmaskinenes muligheter til å lage et brukervennlig applikasjonsgrensesnitt samtidig som den gir flere tilgang til samme system.

I stedet for å styre hva som skjer på skjermen, slik applikasjoner etter terminalmodellen gjør, definerer klient-tjenermodellen hva som skal skje av datautveksling mellom klient og tjener. For å lage en tjeneste/applikasjon etter klient-tjener modellen, må man utvikle en protokoll de to kan benytte for kommunikasjon mellom klienten og tjeneren.



Figur 27 - Klient-tjener modellen

For hver ny tjeneste må man lage en ny tjenerdel som skal kommunisere over nettet, og på hver maskinplattform som skal benytte det må man lage nye klienter og nye brukergrensesnitt. For en del maskinplattformer kan det til og med være nødvendig med flere brukergrensesnitt for å kunne tilby brukerne et fullverdig alternativ. Et vindusorientert brukergrensesnitt for dem som har kraftige arbeidsplassmaskiner, et skjermorientert for dem med mindre kraftig utstyr og et linjeorientert for dem som benytter utstyr som ikke kan benytte et av de to andre. Dette gjør gjerne at brukergrensesnittene blir forskjellige og krever betydelige ressurser, både i utvikling, opplæring og bruk. Eventuelt vil det resultere i at brukergrensesnittene blir linjeorienterte. Selv om man som oftest benytter standard datanettprotokoller til transport av data mellom klient og tjener, må man ikke undervurdere arbeidet med å lage nye brukergrensesnitt til hver eneste maskinplattform.

I 4 kan det som er kalt Terminal 1 til 3, være en MS-DOS PC, en Macintosh, en UNIX-arbeidsstasjon, eller en annen arbeidsplassmaskin. Det kan også være en annen datamaskin som igjen har en terminal tilknyttet seg. Kompleksiteten med brukergrensesnittet er flyttet vekk

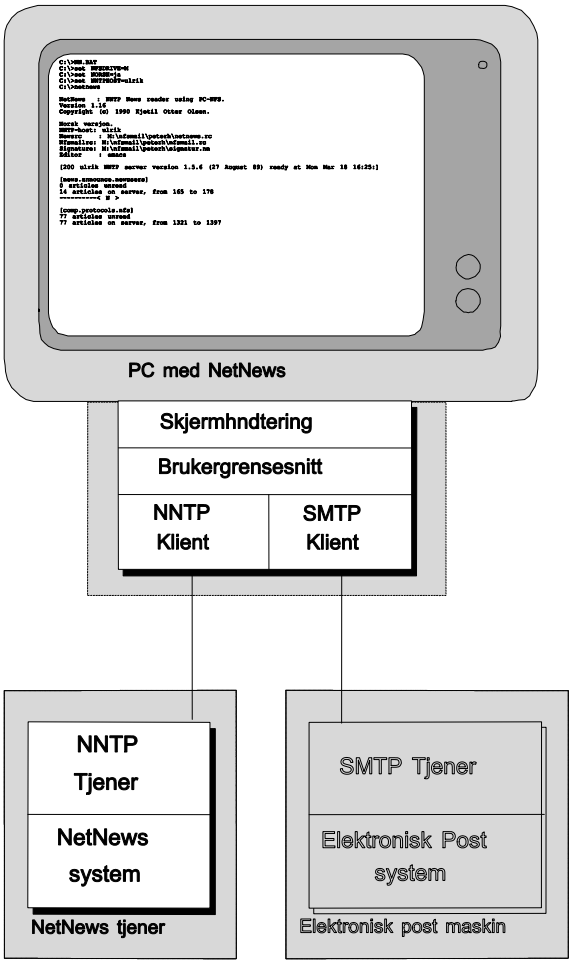
fra tjeneren og håndteres av hver av klientene. For at terminal 1 skal kunne benytte system B må det utvikles en klient B og et brukergrensesnitt B for terminal 1.

Et godt eksempel på tjenester som benytter klient-tjener modellen er de protokollene som benyttes for elektronisk post (SMTP og POP) og NetNews (NNTP). Disse benytter TCP/IP som transportprotokoll (se kapittel 3).

Elektronisk post er en av de mest sentrale tjenestene i akademiske nettverk. Det er lagt ned mye arbeid i å formidle elektronisk post mellom ulike datamaskiner verden over. Mye er også gjort for å få tjenesten ut på arbeidspulten til den enkelte. Klientprogrammer benytter for det meste SMTP for å utveksle post, men det begynner også å komme løsninger for X.400 Brukeragenter (Eng. *User-Agent, UA*). Enbrukermaskiner som bare kan utføre en oppgave av gangen, som for eksempel MS-DOS maskiner og Macintosh, benytter en av POP protokollene (*Post Office Protocol*). Dette gjør at posten oppbevares på en flerbrukermaskin inntil brukeren ønsker å hente posten, på samme måte som i en postboks på postkontoret. Dette gjør det forholdsvis enkelt å lage protokollene for å hente og sende elektronisk post også fra maskiner som ofte er slått av. Problemet er å lage gode brukergrensesnitt til de maskintypene som er aktuelle.

NetNews, sammen med elektronisk post, er vel den tjenesten det er laget flest brukergrensesnitt for. Dette fordi den er totalt avhengig av å fungere etter klient-tjener modellen. Mengden data i form av artikler er så stor at det vil være en utrolig sløsing med plass å ta inn alt på alle maskiner i lokalnettet. De fleste mindre maskiner vil ikke engang ha diskkapasitet til å ta det inn. Klient-tjener modellen er derfor i praksis den beste løsningen for en tjeneste som NetNews.

NNTP er protokollen som benyttes (Se kapittel 3) for å overføre artiklene fra den sentrale NetNews tjeneren til arbeidsplassmaskinen. Denne protokollen er enkel og inneholder de kommandoene som skal til for å overføre artikler til og fra et brukergrensesnitt program på en klient. Det blir bare overført en kopi som man kan lese. Artiklene blir ikke lagret på den lokale maskinen uten at brukeren ber om det.



Figur 28 - Enkel NetNews leser for MS-DOS

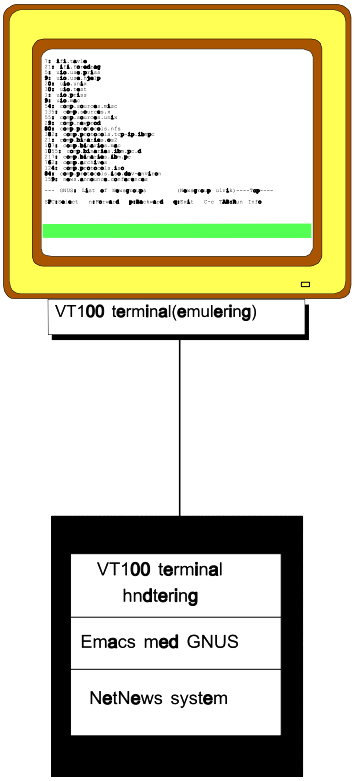
En NetNews leser setter opp forbindelsen til sin NetNews-tjener og så gi kommandoer til tjeneren. For eksempel kan den gi kommandoen *"GROUP"* etterfulgt av navnet på gruppen. Tjeneren svarer med hvor mange artikler den har i den aktuelle gruppen, samt laveste og høyeste nummer på innleggene. Klienten kan så gi kommandoen *"ARTICLE*", evt. etterfulgt av nummeret på en gitt artikkel. Tjener sender så hele artikkelen til klienten. For å hente neste artikkel, kan klienten sende en *"NEXT"* kommando og få tilbake en kort linje om den artikkelen. Det er mulig å lese NetNews ved å koble opp med Telnet mot NNTP porten på en NetNews tjener, men det vil bli en svært tungvint måte å gjøre det på. For å forenkle dette, lages det NetNews lesere som presenterer brukeren for hvilke diskusjons- og nyhetsgrupper som finnes, holder rede på hvilke brukeren er interessert i å følge med i og merker seg hvilke artikler brukeren har lest og varsler når nye grupper oppstår. Sist, men ikke minst, vil NetNews-leseren presentere artiklene på skjermen for brukeren.

Eksemplet i 4 er fra en enkel versjon laget ved USIT. Dette programmet gjør ikke stort annet enn å kommunisere med NNTP, pluss å sende elektronisk post ut ved hjelp av SMTP. Det ble utviklet som en prøve på hvordan det gikk å lage denne type programmer og hvor mye det krevde av utviklingstid, maskinkraft og nettverkskapasitet. Det enkle linjeorienterte brukergrensesnittet ble valgt fordi det var enkelt å lage og fordi det ikke var kapasitet til å lage noe mer avansert.

Det var under utviklingen av dette at jeg innså hvor lang tid det ville ta å bringe tjenester basert på klient-tjener modellen ut til mer enn 3000 brukerne av MS-DOS Pc-er ved Universitetet i Oslo. Og da kanskje bare i form av enkle, linjeorienterte brukergrensesnitt. Det er ingen grunn til å tro at det er en endelig mengde tjenester som er kandidat for distribusjon.

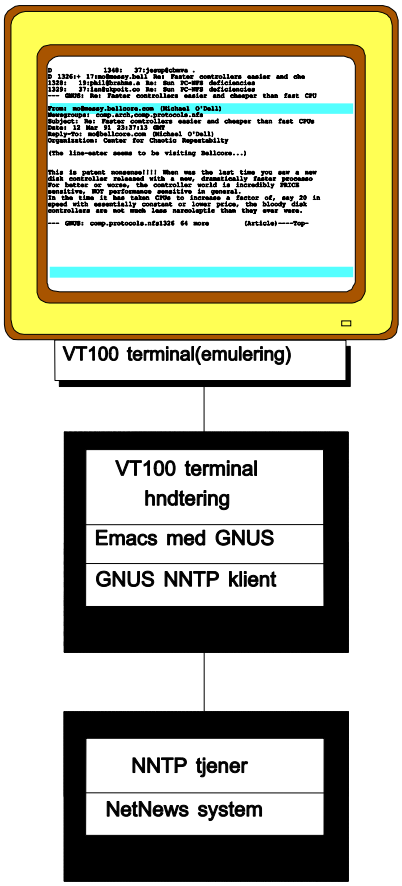
5.2.1. Kombinasjoner av Terminalaksess og klient-tjener modellen

Et eksempel på et brukergrensesnitt som benytter både terminalaksess og klient-tjener modellen, er NetNews leseren GNUS i Emacs på UNIX-maskiner. Dette er skrevet som en applikasjon til editoren Emacs, i dennes innbygde Lisp. Brukeren kan benytte det fra en hvilken som helst terminal som kan emulere en terminaltype som Emacs støtter.

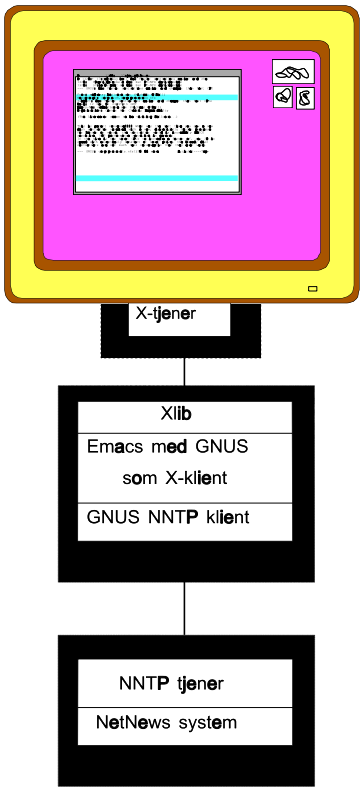


Figur 30 - Terminal mot maskin med Emacs

Emacs selv fungerer etter terminal-modellen. Gnus er skrevet etter klient-tjener modellen, men det er noe brukeren ikke merker. Emacs er også tilpasset slik at den kan benytte X-Windows. Det vil si at på en X-tjener, kan den ha sitt eget vindu og trenger ikke kjøre i et eget terminal-vindu. Vi kan altså godt kombinere disse to modellene i en applikasjon og dermed gi flere muligheter med hensyn på hvilket utstyr som trengs for å benytte en applikasjon.



Figur 29 - Terminalmodellen mot klient-tjener applikasjon.



Figur 31- Emacs som X-applikasjon med GNUS etter klient-tjener modellen

Fordeler og ulemper med Klient-Tjener modellen

Fordeler

- Utnytter arbeidsplassutstyrets muligheter for brukervennlige applikasjonsgrensesnitt.

- Ikke så ressurskrevende for tjenermaskinen som en applikasjon etter terminal modellen, siden den ikke trenger å håndtere brukergrensesnittet.

- Brukergrensesnittet kan lages slik at det fungerer på samme måte som i lokale applikasjoner.

Ulemper

- Må lage tjener protokoll for hver applikasjon.

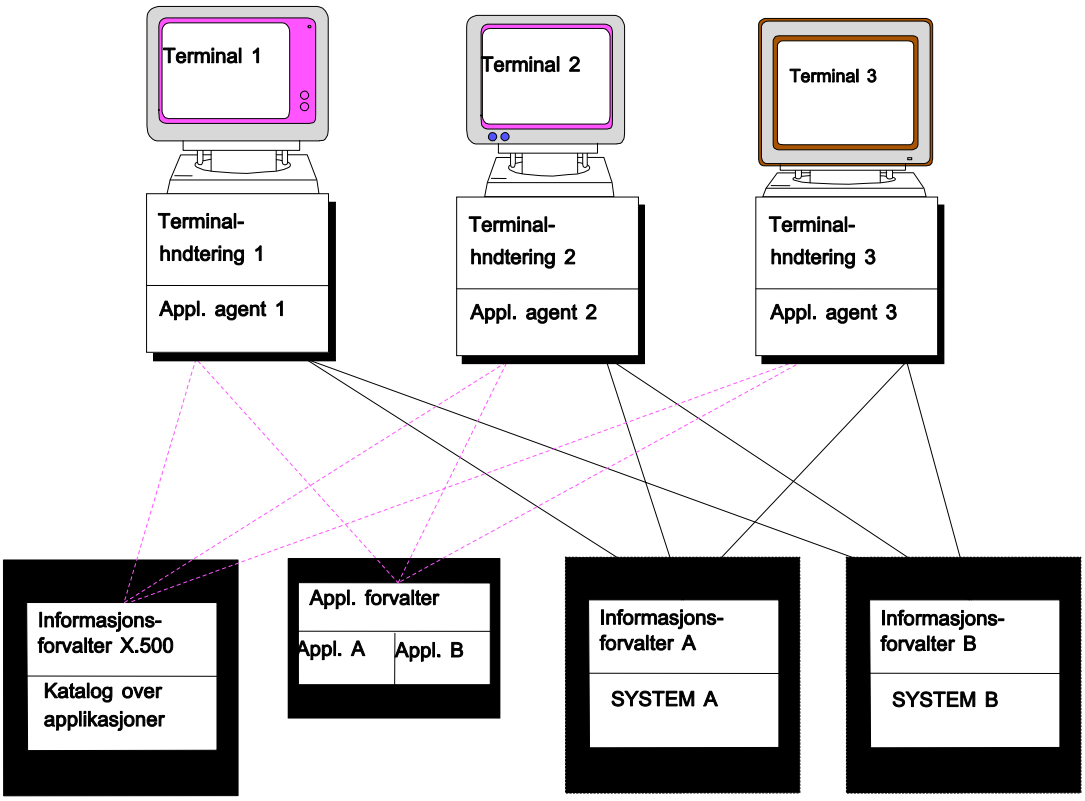
- Må lage klient og brukergrensesnitt for hver applikasjon på hver maskinplattform. Noe som kan kreve mye ressurser i utvikling.

- Ulike brukergrensesnitt på de forskjellige plattformene mot samme distribuerte system. Noe som gjør at brukeren ikke umiddelbart kan dra nytte av kunnskap fra tidligere bruk av samme tjeneste.

5.3. Den distribuerte applikasjonsmodellen

Denne modellen tar utgangspunkt i de foregående modellene og forsøker å kombinere de beste egenskapene i en ny modell. Målet er å kunne:

- Lage ett brukergrensesnitt for alle maskinplattformer.
- La hver maskinplattform presentere applikasjonens brukergrensesnitt slik andre, lokale applikasjoner fungerer på den spesifikke maskinplattformen.
- Muliggjøre individuelle tilpasninger av applikasjoner på arbeidsplassutstyret med hensyn på hvordan det presenteres.
- Gjøre det enkelt for applikasjonsutviklerne å lage gode brukergrensesnitt for ulike maskinplattformer uten å måtte ta høyde for hvordan dette presenteres for brukeren.



Figur 32 - Den distribuerte applikasjonsmodellen

Katalogen benyttes for å finne applikasjoner i det distribuerte systemet. Denne inneholder referanser til hvilken applikasjonsforvalter som har den symbolske koden. Agenten laster ned koden fra applikasjonsforvalteren og eksekverer denne. Applikasjonen selv har informasjon om hvilken maskin/maskiner den kan benytte som informasjonsforvalter.

Applikasjonene lastes over til applikasjonsagentene over datanettet fra applikasjonsforvaltere. Agenter på ulike maskinplattformer tolker applikasjonskoden og generer nødvendige interne strukturer for å kunne håndtere den. Dette er selve koden applikasjonsagenten trenger når den eksekverer applikasjonen. Kommunikasjonen mellom applikasjonsagenten og informasjons-forvalteren foregår ved at agenten utveksler data med applikasjonens informasjonsforvalter etter en protokoll som er definert for den applikasjonen. En rekke ting av det brukeren foretar seg, slik

som å bla i menyer, editere tekst, bla i tekst utføres i av agenten lokalt. Agenten har selv de nødvendige metodene for hva den skal gjøre i forskjellige tilfeller innbakt i klassene. Først når brukeren ber om data som er lagret på tjeneren og som agenten ikke har lokalt, sender agenten en forespørsel til informasjonsforvalteren om å få disse dataene. Tilsvarende, hvis brukeren ønsker å lagre eller oppdatere data på tjeneren blir disse overført til tjeneren og lagret der i henhold til de definisjoner som er gitt i objektene for den applikasjonen.

Det kan være nyttig å se dette som en forlengelse av klient-tjener modellen. Brukergrensesnittet er lagret på en tjenermaskin (applikasjonsforvalter), men overføres i form av symbolsk kode som eksekveres på klientmaskinen. Under eksekvering av applikasjonen på det jeg har valgt å kalle applikasjonsagenten fungerer den på samme måte som en applikasjon etter klient-tjener modellen. Applikasjonen som agenten eksekverer benytter sin informasjonsforvalter på tjenermaskinen for å utveksle data på samme måte som andre klient-tjener applikasjoner. Det vil la seg gjøre å benytte eksisterende tjenere for ting som Elektronisk post, NetNews og lignende, eller man kan lage nye. Det som skiller den fra dagens klient-tjener applikasjoner er at den delen som håndterer brukergrensesnittet er felles for alle applikasjoner og koden som avgjør hvordan brukergrensesnittet fungerer og hvordan kommunikasjonen mellom klient og tjener foregår, er felles for alle maskinplattformer.

I 4 kan Terminal 1 til 3 være en hvilken som helst arbeidsplassmaskin som har en applikasjonsagent. For å få tilgang til applikasjonen, må de først finne informasjon om tilgjengelige applikasjoner i katalogen. Deretter må de laste ned den symbolske koden fra applikasjonsforvalteren. Til slutt kan de starte eksekveringen av applikasjon A for å kommunisere med System A, eller applikasjon B for å komme i kontakt med System B. Etter at koden for en applikasjon er lastet over, trenger agenten ikke kontakte applikasjonsforvalter eller katalog for å bruke den flere ganger. Dette er bare nødvendig første gang en applikasjon benyttes.

5.4. Konklusjon

De tre modellene jeg har sett på i dette kapittelet er metoder for å gi brukere tilgang til tjenester som befinner seg på større fellesmaskiner i et distribuert system. Terminalmodellen er et resultat av tidligere satsing på stor- og minimaskiner. Den benyttes fortsatt, fordi den er velprøvd og velkjent, samt at det er billigere å utvikle ett brukergrensesnitt enn mange, slik klient-tjener modellen krever. I store miljøer kan den imidlertid forårsake en god del problemer med tilpassing av ulike typer utstyr mot terminalbaserte applikasjoner. Klient-tjener modellen er et resultat av tanken om distribuerte systemer, men i miljøer med mange typer arbeidsplassutstyr, kan det fort bli en uoverkommelig jobb å utvikle brukergrensesnitt mot de tjenestene man ønsker å distribuere.

Den distribuerte applikasjonsmodellen er et forsøk på å kombinere disse to modellene, slik at det blir like enkelt for brukeren å benytte applikasjoner for distribuerte tjenester, som lokale applikasjoner. Samtidig som utvikleren klarer seg med å utvikle et brukergrensesnitt som er felles for alle maskinplattformer. Det er denne modellen jeg vil beskrive nærmere i de følgende kapitler.

6. Den distribuerte applikasjonsmodellen

Den distribuerte applikasjonsmodellen tar sitt utgangspunkt i at det finnes et datanett og at det i dette nettet finnes mange typer arbeidsplassutstyr og mange type tjener maskiner. Med andre ord et heterogent datamaskinmiljø. Målsetningen er at alle typer arbeidsplassutstyr skal kunne ha tilgang til de distribuerte tjenester/applikasjoner som finnes.

Hypotesen bak modellen er at et brukergrensesnitt kan splittes i en syntaktisk del og en semantisk del, slik at det mulig å lage en syntaks for funksjonaliteten i en applikasjons brukergrensesnitt, mens semantikken kan være forskjellig alt etter hva som er mulig, vanlig eller ønskelig å lage på den enkelte type maskinvare.

6.1. Målsetning

Målsetningen for modellen er altså å beskrive et verktøy for å kunne utvikle en applikasjon en gang og så kunne bruke samme applikasjon fra mange maskinplattformer uavhengig av:

- Operativsystem
- Maskinarkitektur
- Vindussystem
- Skjerm og grafikksystem

Andre målsetninger er:

- Skille mellom brukergrensesnittet og selve oppgaven applikasjonen utfører.
- Lokale operasjoner så langt det er mulig, for å avlaste nettverket og tjeneren
- Følge standarder som finnes for datanett, grafikk, lydrepresentasjon osv.
- La applikasjonene oppføre seg på samme måte som lokale applikasjoner
- Håndtere mer avanserte muligheter ved datamaskiner, slik som lyd og animasjon.
- Ha utvidelsesmuligheter for ny, forbedret teknologi og åpne for andre anvendelser.
- Kunne benytte applikasjoner på mer eller mindre avansert utstyr med ulik funksjonalitet.

I utgangspunktet høres det kanskje ut som en umulighet å klare å lage et system der samme kode kan benyttes på alle typer maskinutstyr. Det er da store forskjeller i brukergrensesnittet på en Macintosh og en MS-DOS PC? For ikke å snakke om en VT100 terminal! Mitt argument er at visst er det forskjeller, men det går på utseende og hvordan man utfører ting, mens det man gjør er det samme. Jeg vil i de følgende avsnitt se på hvordan en applikasjon er bygget opp og prøve å splitte applikasjoner i funksjoner som bygger på hverandre. Deretter vil jeg se på hvilke deler en applikasjonsagent må bestå av for å kunne utføre det den skal.

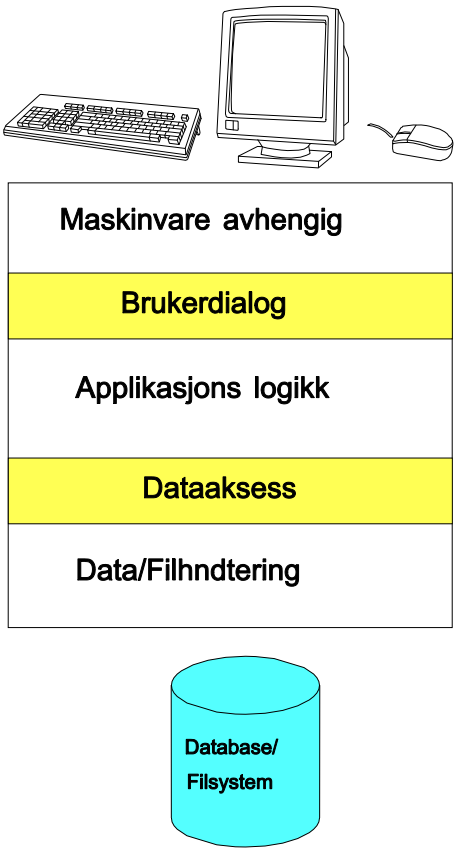
6.2. Komponenter i en applikasjon

En applikasjon er primært et verktøy for brukeren for å søke etter, hente, redigerer og lagre data. Hvor og hvordan dataene er lagret er i denne sammenhengen ikke vesentlig. Det er de manipulasjonene brukeren kan gjøre på disse dataene og hvordan det blir gjort som er det interessante. Jeg vil her forsøke å splitte opp og identifisere fem hovedfunksjoner i en applikasjon som har med bruken av en applikasjon å gjøre.

Sett fra brukerens synspunkt er det tastatur, mus, skjerm og annet som benyttes som for å kommunisere med applikasjonen, som er interessante. Disse komponentene er avhengig av maskinvaren. Applikasjonen kan bare tolke input og gi output, hvis den kjenner til hvordan denne maskinvaren kommuniserer. Første funksjon i applikasjonen er altså å kommunisere via de maskinvare avhengige komponentene på et nivå som består i å tolke elektriske signaler og omforme dem til noe som neste lag kan benytte. Registrere hvilke taster brukeren har trykket, hvordan musa manøvreres og hvilke mustaster som trykkes ned hvor. Tilsvarende blir det applikasjonen ønsker å skrive ut på skjermen i form av bokstaver, linjer, bokser og annet, gjort om til signaler skjermen forstår.

Neste funksjon har jeg kalt brukerdialog. Det er denne som håndterer dialogen med brukeren på et høyere nivå. Den knytter sammen selve logikken i applikasjonen og den maskinvare avhengige kommunikasjonen med brukeren. Det at en tast er trykket er en ting, men logikken i applikasjonen må også vite hvor en tast er trykket og i hvilken sammenheng.

Applikasjonslogikken er den delen som kobler brukerdialogen sammen med dataene og utfører de operasjoner som brukeren ber om i brukerdialogen. Det er den som sørger for at brukeren blir presentert for de data som blir forespurt og som sjekker at data eventuelt blir oppdatert etter de reglene som gjelder for oppdatering i applikasjonen. Dette kan være data i en database, eller det kan være data som er lagret i form av filer eller på andre måter. Applikasjonslogikken styrer også de interne manipulasjonene av data i applikasjonen.



Figur 33 - Komponenter for kommunikasjon mellom bruker og de data denne ønsker å manipulere

Det kan også være en del som står i kontakt med en annen datamaskin etter klient-tjener modellen. Dataaksess delen er der for å synliggjøre denne oppsplittingen. Hvis vi har en applikasjon som benytter klient-tjener modellen, vil dataaksess være klientdelen og Data/filhåndtering være tjenerdelen. Hvis en applikasjon benytter *Dynamic Data Exchange* (*DDE*) som finnes i MS-Windows og OS/2 eller tilsvarende mekanismer i andre operativsystem, vil dataaksess være den delen av applikasjonen som står i kontakt med de som styrer den dynamiske utvekslingen av data. I en rekke lokale applikasjoner vil dataaksess ikke ha betydning.

Det jeg her har kalt data/fil håndtering er den delen av applikasjonen som sørger for at data blir hentet og lagret slik logikken i applikasjonen ber om via dataaksess. Dette for å skille selve logikken i applikasjonen fra det maskinvare avhengige som måtte være involvert i å lese og skrive data til en fil på disk, hente data fra en tjenerapplikasjon, oppdatere data i en database eller andre metoder for å manipulere på data som kan være lagret på en datamaskin.

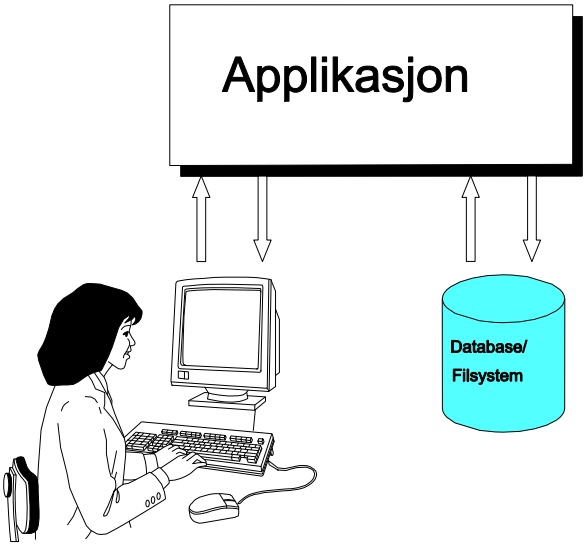
Ikke alle applikasjoner er avhengig av data. Et program for enkle matematiske beregninger trenger ikke nødvendigvis å kunne lagre data. Tilsvarende kan det være applikasjoner som ikke trenger

brukerdialog. For eksempel applikasjoner for å ta automatiske sikkerhetskopier av lokale disker. Om en av funksjonene er tomme spiller ingen rolle for oppsplittingen.

Denne måten å splitte opp en applikasjon, minner for så vidt mye om de metoder som blir benyttet innen datakommunikasjon og som gjenspeiler seg i OSI referanse modellen. Hensikten er å skille de tre komponentene maskinvare, applikasjonslogikk og data/filhåndtering fra hverandre ved å legge noe mellom som gjør at de blir uavhengige av hverandre. Brukerdialog og dataaksess er de delene som kobler sammen disse tre bitene.

6.3. Enkeltstående applikasjoner

Lenge har applikasjoner vært skrevet som enkeltstående programmer. Det vil si at de har hatt alle de delene de trenger for å gi brukeren tilgang til dataene inkorporert i selve applikasjonen. De har ikke vært avhengige av annet enn eventuelt operativsystemet for å utføre sin oppgave.



Figur 34 - Selvstendig applikasjon

Hvordan selve programmet er skrevet, spiller ingen rolle her. For eksempel kan programmereren ha brukt eksterne biblioteker under utviklingen av applikasjonen. Om det har vært brukt aldri så mange metoder for nettopp å modularisere og strukturere applikasjonen under utviklingen, er den ikke avhengig av andre systemer når den eksekverer.

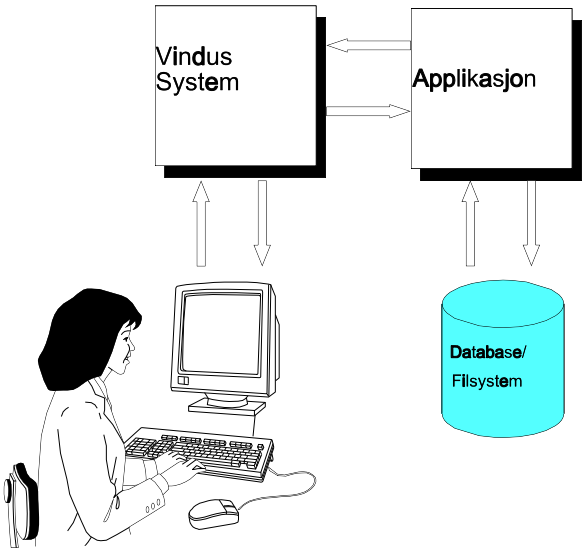
Applikasjonen må inneholde håndtering for alle typer maskinvare den skal kunne benytte. Ønsker programmereren å skrive for flere typer grafisk maskinvare, må applikasjonen inneholde rutiner for de aktuelle skjermtyper. Tilsvarende blir det hvis applikasjonen skal ha støtte for mus, så må det tas med under programmeringen. Applikasjonen vil gjerne forutsette at den får hele skjermen til disposisjon og skal den fungere i et annet miljø, som for eksempel under et vindussystem, må den gjerne lures til å tro at den har maskinvaren for seg selv.

Slike selvstendige applikasjoner kan like gjerne være laget etter den tradisjonelle terminal-modellen, som etter klient-tjener modellen. Det kan også være et program på en mikromaskin. Hvis programmet er modulært skrevet, kan det hende at det er enkelt å flytte den til en annen maskinplattform, men den vil ikke være kandidat for å tilfredsstille de krav den distribuerte applikasjonsmodellen forsøker setter opp.

En applikasjon av denne typen inneholder altså i hvert fall de fire komponentene: maskinvare avhengig, brukerdialog, applikasjons logikk og dataaksess. Hvis det er en lokal applikasjon vil den som regel også inneholde data/filhåndtering i tillegg til eller i stedet for dataaksess. Applikasjonen er totalt bundet opp til den maskinplattformen den er skrevet for.

6.4. Vindussystemer

Vindussystemer har som nevnt fått en sterkt økende popularitet de siste årene. Disse bygger på ideen om at man må skille mellom selve brukergrensesnittet og bare la hver applikasjon styre en del av skjermen, nøye avgrenset og kontrollert av vindussystemet. Det er vindussystemet som tar seg av de maskinvare avhengige delene og styrer applikasjonenes bruk av disse. Skjerm- og grafikksystemet i den enkelte arbeidsplassmaskin blir skjult for applikasjonene som dermed blir uavhengige av hva slags grafisk maskinvare de benyttes på og hvordan tastatur, mus og lignende er satt opp. Dette gjør at flere applikasjoner kan dele samme skjerm og ta input fra samme tastatur, mus ol. Programmereren trenger ikke ta hensyn til dette under utviklingen fordi det blir styrt av selve vindussystemet,



Figur 35 - Applikasjon i vindussystem

Brukerdialogen, applikasjonslogikken, data-aksess og/eller data/fil håndteringen er det applikasjonen selv som styrer. Utvikleren har gjerne veldefinerte biblioteker til disposisjon for enklere å kunne lage gode brukergrensesnitt, men applikasjonen er avhengig av det vindussystemet den er skrevet for. En applikasjon skrevet for X-Windows, vil ikke kunne gå under Microsoft Windows, uten omfattende omprogrammering. Ved å lage grensesnittene mellom applikasjonene og disse bibliotekene like, vil det være enkelt å flytte en applikasjon fra et vindussystem til et annet, men det vil bli to applikasjoner.

6.4.1. Elementene i et vindussystem

Jeg vil i det følgende kort skissere hovedelementene i et grafisk vindussystem. Hensikten er å peke på de funksjonene som er tillagt vindussystemet og hvilke som er tillagt applikasjonene for å klargjøre en videre diskusjon om hvordan man kan implementere den distribuerte applikasjonsmodellen.

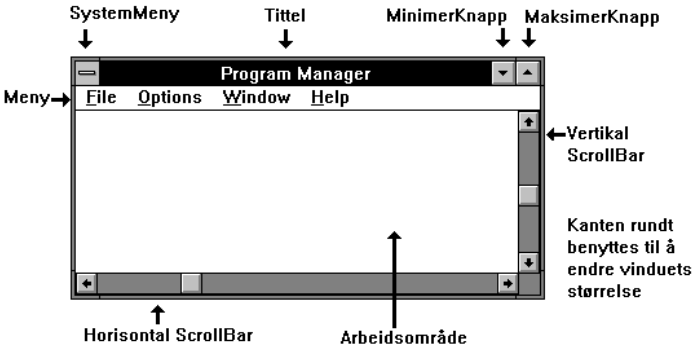
Et vindu er en del av skjermen innenfor angitte grenser, med egne attributter som størrelse, plassering på skjermen, bredde på bord, og eventuelt relativ plassering til andre vinduer. Vindussystemene gir muligheter for å opprette vinduer, endre størrelsen, flytte dem rundt på skjermen, fjerne dem eller lage dem til små ikoner. Ved at vinduer flyttes og kan overlappe hverandre vil innholdet i hele, eller deler av vinduer bli overskrevet. Det finnes muligheter for å gjenopprette innholdet i et vindu når et vindu igjen blir eksponert. Disse operasjonene er forskjellig implementert i de ulike vindussystemene, men de logiske operasjonene er stort sett de samme. [KUTAL 90]

En grafisk vindussystem har som oppgave å:

- Organisere vinduene på skjermen
- Vite hvor vinduene er
- Sørge for at overskrevne vinduer blir reparert
- Endre størrelsen på vinduer
- Flytte vinduer framover og bakover på skjermen (Z-planet)

- Tegne vinduene på skjermen
- Oversette koordinater på skjermen

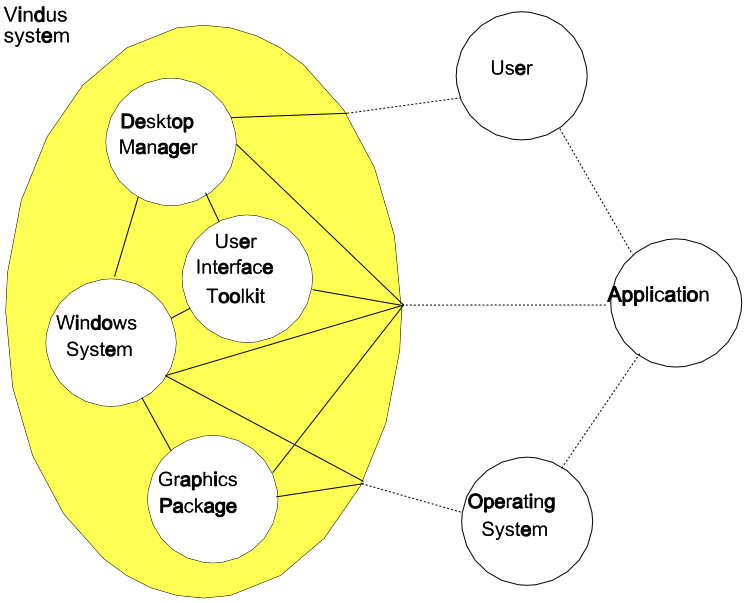
I tillegg er det også vindussystemet som tar i mot input fra brukeren og det må derfor kunne gi beskjed til den aktive applikasjonen hvilke taster eller "mus"-manøvrere brukerne gjør. De fleste systemer styrer dette ved hjelp av såkalte *events* (No: hendelser). Alt som skjer blir lagt i køer til den aktive applikasjonen. Alt som skrives ut til et vindu på skjermen, går gjennom vindussystemet for å sikre at det havner innenfor vinduet og for at applikasjonens lokale koordinater kan bli oversatt til skjermkoordinater. De grafiske rutinene som benyttes for å tegne på skjermen kan være generelle grafiske rutinebiblioteker, men er som oftest innebygget i selve vindussystemet. Disse rutinene forsyner vindussystemet med de rutinene som trengs for å tegne, klippe, finne koordinater og så videre. I tillegg er det ofte med et bibliotek med rutiner for applikasjonsutvikleren for å lage brukergrensesnitt. Gjerne omtalt som Toolkit eller API (*Application Program Interface*).



Figur 36 - Elementene i et applikasjonsvindu

Dette kombinerer lavere nivås funksjoner til mer avanserte, for å sikre at applikasjonene blir seende rimelig like ut og for å lette jobben for programmererne. Slike biblioteker inneholder gjerne rutiner for å håndtere ting som scrollbarer, titler, sjekkbokser, menyer og andre elementer i et vindu. Noen av dem kombinerer gjerne flere slike til halvferdige komponenter med standard elementer for vanlige vinduer. En siste komponent i vindussystemet er den delen som styrer hvordan ting ser ut og oppfører seg på skjermen, det som på engelsk kalles *"look-and-feel"*. Ulike vindussystemer har ulike navn, men på engelsk er navn som *"Desktop Manager"* eller *"Window Manager"* brukt. Det er dette som for eksempel bestemmer hvordan aktivt vindu velges. I noen systemer, som på Macintosh og MS-Windows må brukeren peke på vinduet med musa og trykke på knappen, mens i X kan det være nok å føre musa over i et annet vindu og det nye vinduet er aktivisert. Hvilken av disse metodene som benyttes har normalt lite å si for selve applikasjonen, men det vil føles veldig forskjellig for brukeren. For å kopiere ting mellom applikasjoner med så kalt klipp-og-lim, er det også ulike måter som er valgt i de ulike vindussystemene, avhengig av hvilken *"look-and-feel"* utviklerne har foretrukket.

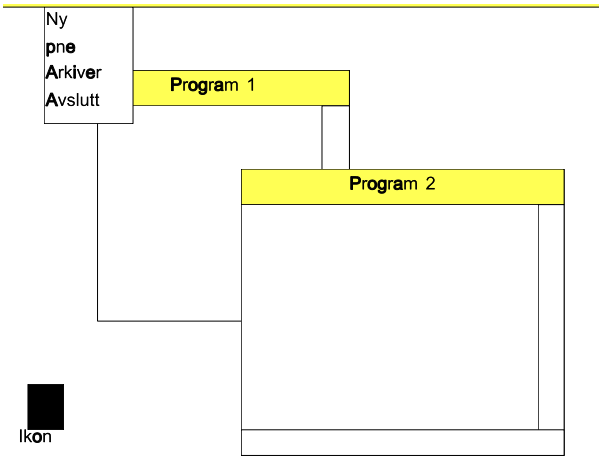
Figur 1 viser hvordan de forskjellige komponentene i et vindussystem henger sammen. Brukeren (*User*) står i kontakt med *Desktop manager* og *Application*. Applikasjonen (*Application*) har kontakt med alle fire elementene i vindussystemet, samt med bruker og operativ systemet. Strekene mellom er kall på rutiner i de ulike komponentene og meldinger om hva som skjer. Hvis vinduet til en applikasjon er blitt overskrevet av et annet vindu og brukeren ber om å få det på toppen igjen, er det *Desktop Manager* som sender en *event* (hendelse) til applikasjonen og ber den tegne vinduet om igjen. *User Interface Toolkit* er den samlingen rutiner som gjør at applikasjonsutvikleren får tilgang til mer eller mindre ferdige vinduskomponenter i applikasjonen uten å måtte sette dem sammen selv. *Graphics Package* er rutinene for å håndtere skjermens grafiske muligheter. En slik oppstilling vil aldri bli helt komplett, eller riktig, fordi komponentene tildels overlapper hverandre og grensesnittene mellom dem ikke alltid er helt veldefinerte. Men en oppstilling som dette er nyttig for å skille mellom de ulike komponentene i et vindussystem. Det riktigste hadde nok vært å sette navn på grensesnittene mellom de ulike komponentene også, men disse har ofte en forvirrende navning.



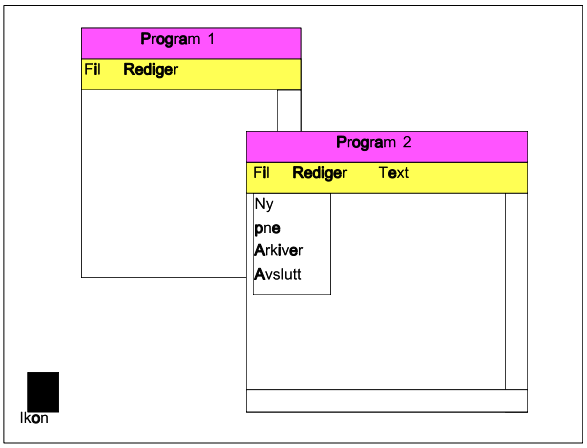
Figur 37 - Komponentene i et vindussystem og deres relasjon til resten av det som utgjør en vindusapplikasjon [KUTAL 90]

Hvis vi sammenligner ulike vindussystemer med hensyn på funksjonalitet og ikke tar hensyn til selve utformingen og funksjonene ovenfor brukerne, ser vi at det er mange felles trekk. Skjermen er oppdelt i vinduer med tittel, scrollbarer og andre elementer. Det finnes hovedmenyer og undermenyer, dialogbokser, meldingsbokser og listbokser, samt muligheter for å vise tekst, og grafikk, basert på vektorer eller bitmaps. Disse elementene er tilstrekkelig for de aller fleste applikasjoner. La meg ta et eksempel:

Sammenligner vi to av de mest brukte vindussystemene for mikromaskiner, nemlig Microsoft Windows 3.0 og Macintosh er det forskjeller med hensyn på hvor brukeren må peke og klikke for å få utført forskjellige oppgaver, men innholdet og de mulighetene en applikasjon har ovenfor brukeren er temmelig like. Menyene på en Macintosh er plassert øverst på skjer-



Figur 38 - Elementene på Macintosh



Figur 39 - Elementene i Windows/PM

men, mens de i Windows er en meny for hvert vindu. Windows kan styres utelukkende ved hjelp av tastaturet, mens Macintosh krever mus. I Windows blir menyene stående når de er aktivisert, mens de på Macintosh forsvinner straks brukeren slipper musknappen opp. Slike ting er forskjeller i *"look-and-feel"*, men har ingen innflytelse på selve applikasjonen og de mulighetene som finnes der. Det er ting som har betydning for brukeren, men i mindre grad for hvilke applikasjoner det går an å lage.

Hovedelementene som inngår i et moderne vindussystem er så like at det må la seg gjøre å generalisere applikasjoner til å fungere i ulike vindussystemer.

Litt avhengig av hvilke funksjoner en applikasjon krever, vil det også la seg gjøre å benytte den samme generalisering til også å fungere i et skjermorientert system. De samme elementene som inngår i et vindu i et vindussystem, vil også kunne implementeres på en maskinplattform som ikke benytter noe vindussystem, men da begrenset til et "vindu" som dekker hele skjermen. Dette vil sette begrensninger for hva applikasjonen kan utføre. Hvis det er en ikke-grafisk skjerm, vil den ikke kunne bruke grafikk og gjerne være begrenset til 80 tegn på 24 linjer. Dette burde allikevel være tilstrekkelig til å kunne lage temmelig brukervennlige applikasjoner for mange formål.

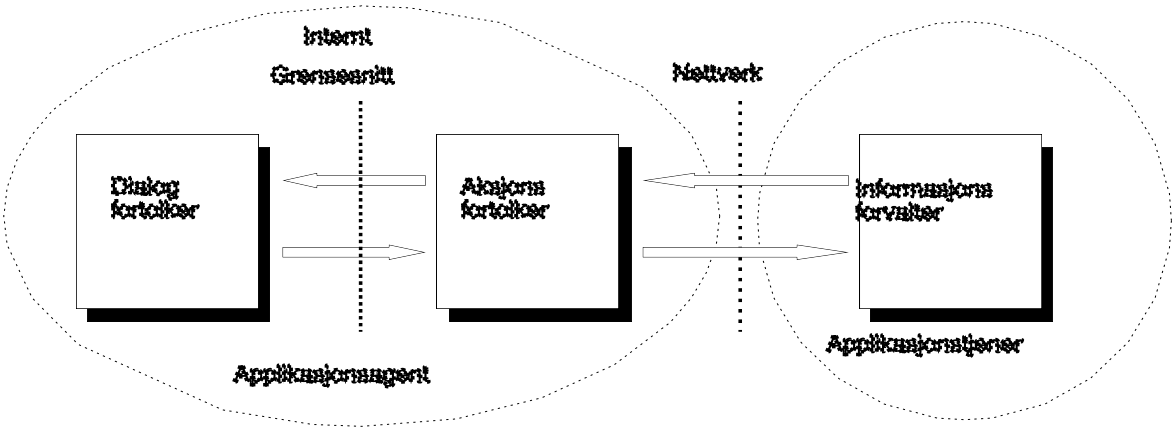
Enkelte applikasjoner, som for eksempel elektronisk post, konferansesystemer og sentrale hjelpesystemer har lenge vært basert på linjeorienterte brukergrensesnitt. Det gir tilfredsstillende brukergrensesnitt for en del formål, selv om det langt fra er optimalt.

Det er disse likhetene som sammen med ønske om å kunne distribuere tjenester i et heterogent datamaskinmiljø som har ført meg fram til den distribuerte applikasjonsmodellen.

6.5. Den distribuerte applikasjonsmodellen

Den distribuerte applikasjonsmodellen har som utgangspunkt at det finnes et datanett og det i dette nettet finnes mange typer arbeidsplassutstyr og mange typer tjenermaskiner. Med en enhetlig maskinpark vil de komponentene i applikasjonsagenten ikke ha noen direkte betydning. Det er i et heterogent miljø at dette gir mening. Poenget er at straks man er tilknyttet et datanett som omfatter mer enn en organisasjon, vil man som oftest fort komme inn i et heterogent datamaskinmiljø.

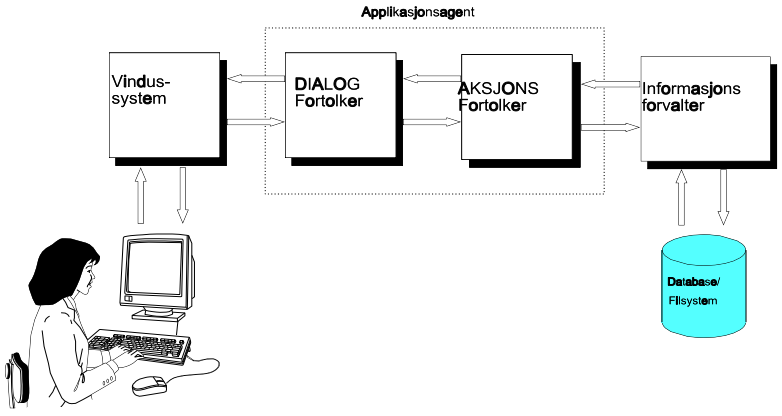
Det primære den distribuerte applikasjonsmodellen er ment å håndtere er kommunikasjon mellom brukeren og applikasjonen på en slik måte at selve applikasjonen blir uavhengig av hva slags maskinutstyr brukeren benytter. Som vi har sett, tilfredsstiller verken den tradisjonelle måten å lage selvstendige applikasjoner på, eller moderne vindussystemer disse kravene direkte. For å få til en modell som fungerer, må vi skille mellom de fem elementene i en applikasjon angitt i 4, slik at en applikasjon blir uavhengig både av skjerm og grafikksystem, vindussystem, maskinarkitektur og operativsystem.



Figur 40 - Hovedkomponentene

I 4 er bare de komponentene som inngår i den distribuerte applikasjonsmodellen tatt med. Applikasjonsagenten er i seg selv et program på arbeidsplassmaskinen. Den kan nærmest sammenlignes med en virtuell prosessor som kan eksekvere kode tilhørende en applikasjon som er skrevet for den distribuerte applikasjonsmodellen. Selve applikasjonen er den koden som applikasjonsagenten eksekverer. Applikasjonene selv kan eksekvere i applikasjonsagenter på alle typer arbeidsplassutstyr, men selve applikasjonsagenten må skrives for den maskinvaren den skal benytte. Informasjonsforvalteren er et program som kan svare på forespørsler fra applikasjonen over et datanett.

Når applikasjonen eksekverer på applikasjonsagenten er det tre komponenter som kommuniserer. Dialogfortolkeren håndterer grensesnittet med brukeren og gjør den jobben som i 4 er kalt brukerdialogen. Aksjonsfortolkeren gjør jobben til dataaksess og Informasjonsforvalteren tar seg av Data/filhåndtering. Applikasjonslogikken håndteres i applikasjonsagenten. De maskinvareavhengige tingene kan enten ivaretas av et vindussystem, eller det kan håndteres i dialogfortolkeren selv.
Å splitte applikasjonsagenten slik, er til dels motivert ut fra ønske om å skille logikken fra dialogen og fordi aksjonsfortolkeren og dens mekanismer kan generaliseres til å gjelde alle maskinplattformer, mens dialogfortolkeren må skreddersys for den enkelt maskinplattform. Man vil også kunne lage applikasjonsagenter som ikke krever noe vindussystem. Hvor altså dialogfortolkeren styrer de maskinvare avhengige delene direkte. Dette kan for eksempel være aktuelt for MS-DOS maskiner som ikke benytter Windows, eller for maskiner der applikasjonsagenten skal betjene dumme terminaler.



Figur 41 - Applikasjon i den distribuerte applikasjonsmodellen

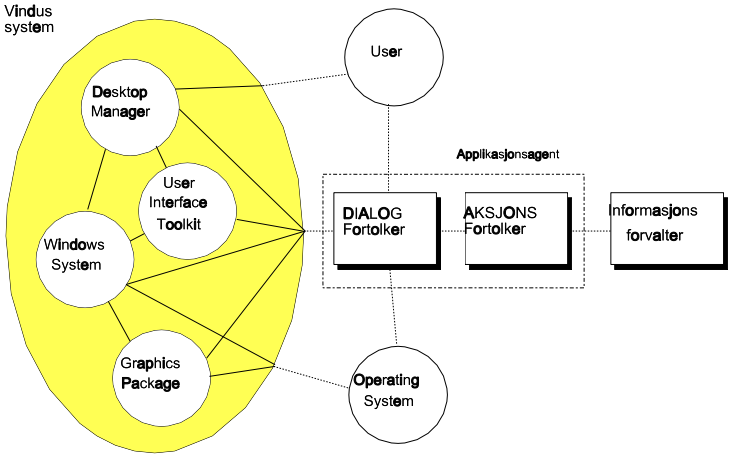
Modellen legger altså opp til to klart definerte grensesnitt. Ett mot det maskinvareavhengige og ett mot data/filhåndtering. Applikasjonsagenten trenger altså mekanismer for å eksekvere

applikasjonens kode og sørge for å oversette dennes instruksjoner som utgjør grensesnittet mot brukeren og mot dataene. Dette i motsetning til selvstendige applikasjoner som håndterer alt selv og til vindusapplikasjoner som bare har grensesnitt mot skjermhåndteringen.

Den koden applikasjonsagenten eksekverer og som utgjør selve applikasjonen er lik for alle typer arbeidsplassutstyr. En applikasjon etter denne modellen kan ikke benytte binær kode i tradisjonell forstand. Det vil si programkode som kan eksekvere direkte på ulike prosessorer. Både forskjellen i dagens prosessorer og utviklingstakten på nye prosessorarkitekturer, taler mot en slik løsning. Koden må derfor være i en slags symbolsk form som enkelt kan prosesseres i applikasjonsagenten. Det at den er symbolsk, behøver ikke å bety at den er lett leselig for mennesker, men den bør være slik at den enkelt lar seg oversette til noe som er lett å forstå for et menneske. Hva som gjøres med koden i applikasjonsagenten bør kunne variere. Den kan interpreteres som den er, eller kompileres til en binærkode som er direkte eksekverbar, alt etter hva som er gunstigst. Jeg vil komme tilbake til koden i kapittel 8.

Applikasjonene må selv inneholde de mekanismene som trengs for å finne ut om de har de nødvendige verktøy, eller metoder de behøver for å fungere på den aktuelle arbeidsplassmaskinen. For eksempel må de kunne finne ut om det finnes muligheter for lyd på maskinen, eller om applikasjonsagenten har tilstrekkelige grafiske muligheter til at det skal være meningsfylt å kjøre applikasjonen på den maskinen.

Figur 42 viser hvordan applikasjonsagenten, med dialog- og aksjonsfortolker passer sammen inn i den oppstilling som er gitt i Figur 33. Det er altså dialogfortolkeren som kommuniserer med vindussystemet og brukeren, mens aksjonsfortolkeren utveksler data med informasjonsforvaltereren på applikasjonstjeneren.



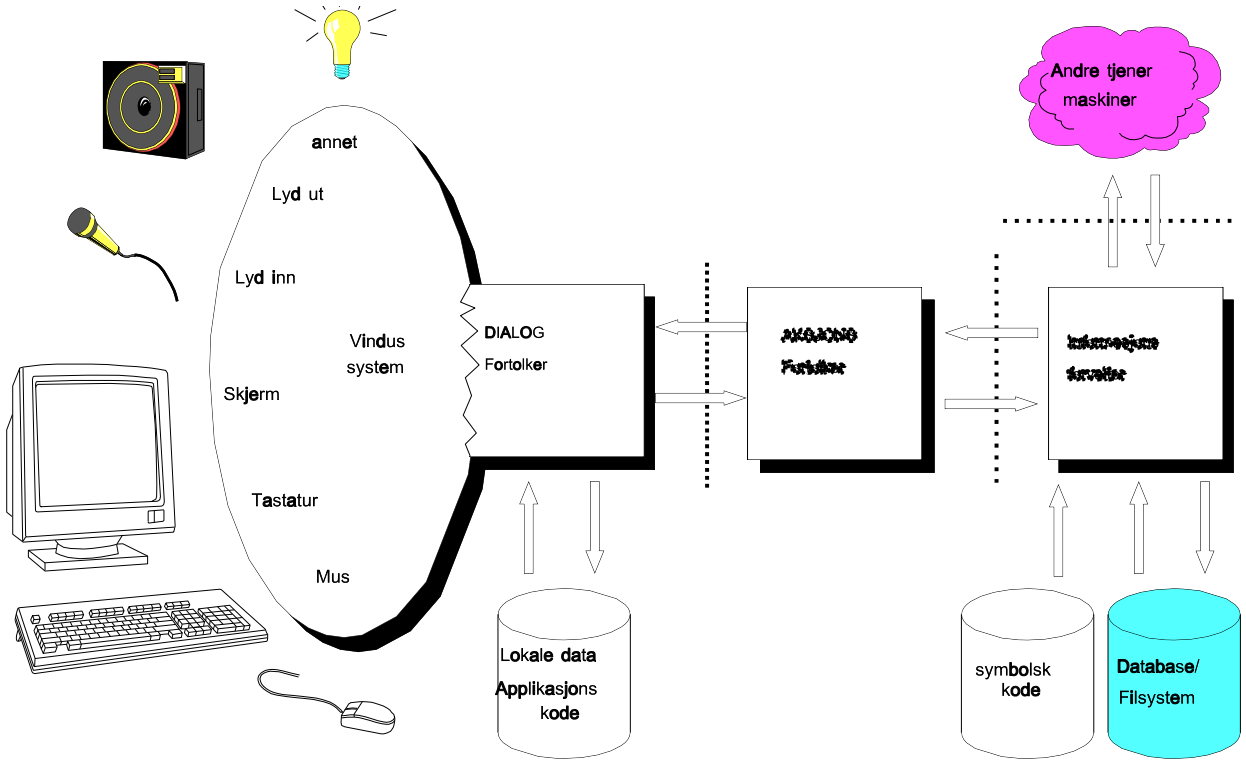
Figur 42- Samspillet mellom applikasjonsagenten og et vindussystem

Mellom aksjonsfortolker og informasjonsforvalter er det forutsatt å være en datanettforbindelse av et eller annet slag. Så lenge begge parter benytter samme protokoll er det uvesentlig for modellen hvilken som benyttes. Data utveksles i form av meldinger over nettet og kodingen av disse meldingene trenger ikke være standardisert. En applikasjon kan benytte TCP/IP mot en NNTP-tjener. En annen kan benytte OSI protokollen FTAM mot et fillager, mens en tredje kan være en applikasjon for å få tilgang til dataene i BIBSYS databasen over enten OSI eller TCP/IP. Applikasjonen er skreddersydd for den informasjonsforvalteren den benytter og har all koding av meldingene innbakt i sin egen kode. Det applikasjonsagenten må gi er muligheten for å sende meldinger over nettet, altså tilgang til rutiner for kommunikasjon. Applikasjonsagenten kan ha disse innbakt som en del av sin egen kode, men bør så langt det er mulig benytte den nettverksprogramvare som finnes på arbeidsplassmaskinen (Se kapittel 4)

En applikasjon kan ha behov for data fra mer enn en kilde. For eksempel vil en applikasjon som gir en totaloversikt over den økonomiske situasjonen i en bedrift, kunne ha behov for data fra regnskapssystemet, budsjettsystemet, lønnssystemet, materialstyringssystemet, samt fra en eller flere bankers kontosystem. I et distribuert system, vil det ikke være urimelig å anta at disse

befinner seg på en rekke forskjellige tjenermaskiner og således ikke kan sammenstilles automatisk. Dette kan enten løses ved at en tjenermaskin henter inn data og sammenstiller dem for applikasjonen, eller ved at applikasjonen selv kommuniserer med de tjenermaskinene som er aktuelle. Hvilken løsning som er den beste vil kunne variere fra applikasjon til applikasjon. Hvis det allerede finnes en applikasjonstjener for hvert av systemene kan muligens disse benyttes til å sette sammen dataene i applikasjonen i agenten. Særlig hvis det er sjelden at en slik sammenstilling er nødvendig, vil det kunne være fordelaktig. Hvis det derimot er en mye brukt applikasjon, som krever en del datakraft, kan det lønne seg å la en maskin sammenstille dataene og presentere dem for applikasjonen som om alle data fantes et sted. Applikasjonstjeneren kan enten selv fungere som en applikasjonsagent overfor de andre datakildene, eller benytte andre måter å hente data på. Hvordan en applikasjonstjener fungerer internt og hvordan den henter sine data er uvesentlig for applikasjonsagenten. Det er grensesnittet mellom dem som er av betydning.

Modellen bør kunne være anvendbar også for andre oppgaver enn bare vanlige applikasjoner. Det kan tenkes at det ikke er brukeren som benytter applikasjonen, men et program på maskinen som benytter aksjonsfortolkeren til å hente over data som benyttes i andre lokale applikasjoner. Et eksempel her vil kunne være å sørge for dynamisk oppdatering av prislister i et dokument fra en leverandørs prisregister, eller overføring av lokale dokumenter til et felles arkiv system. Slike applikasjoner vil kunne gå i bakgrunnen på flerprossessmaskiner og gjøre sin jobb uten at brukeren trenger tenke på det. De vil heller ikke benytte dialogfortolkeren annet enn til eventuelt å kunne gi feilmeldinger eller beskjed om at tingene er utført, samt lese og skrive data til lokal disk.



Figur 43 - Komponentene i modellen og deres omgivelser

4 forsøker å illustrere sammenhengen mellom alle delene i et system som benytter den distribuerte applikasjonsmodellen. På arbeidsplassmaskinen ligger operativsystem og maskinvare som prosessor og intern hukommelse også bak og styrer Dialog- og Aksjonsfortolkeren, selv om det ikke er tegnet inn. Applikasjonsagenten vil lagre koden for applikasjonene på lokal disk for å slippe gjentatte overføringer. Applikasjoner kan også ha tilgang til lokal disk for å kunne utføre operasjoner på lokale data hvis ønskelig.

6.6. Oppsummering

En applikasjonsagent må altså:

- Benytte nettverket over standard protokoller og kommunisere med andre maskiner.
- Kunne kommunisere med katalogen over applikasjoner
- Kunne laste over symbolsk kode for ulike applikasjoner fra tjenermaskiner
- Oversette den symbolske koden til noe maskinvaren kan håndtere.
- Kommunisere med brukeren enten direkte, eller via et vindussystem.
- Utveksle data med en eller flere tjenermaskiner i nettet alt etter hva brukeren ber om i applikasjonen

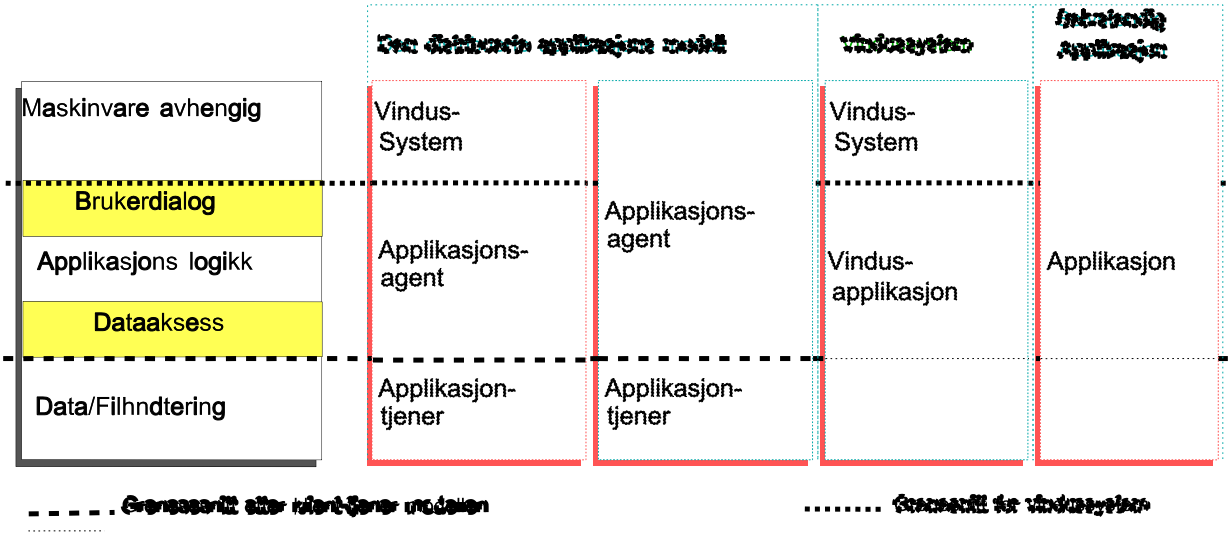
6.7. Oppstart av applikasjoner

De metodene som normalt benyttes når en datamaskin skal starte en ny applikasjon er å laste inn binærkode for applikasjonen i intern hukommelsen og så få prosessoren til å starte eksekveringen. Normalt lastes applikasjonene fra disk. Dette kan være en disk som maskinen selv har, eller det kan være en disk som er montert over et distribuert filsystem som for eksempel NFS. Et problem med denne metoden er at det kan være komplisert for brukeren å finne fram til hvor programmene befinner seg på nettverksdiskene, eller det krever at brukeren får programmet installert på sin maskin lokalt. Ved bruk av nettverksdisker har man redusert noe av arbeidet, men det kan gjerne oppstå problemer med tilpassing av programvaren til ulike type maskiner i nettet. Ved lokale applikasjoner tar det gjerne lang tid å oppdatere alle brukerne med nye versjoner av programvare, samt at samme arbeidet må gjøres mange ganger.

I den distribuerte applikasjonsmodellene er koden forutsatt å befinne seg på en eller flere applikasjonstjenere og alle applikasjonsagenter vil i utgangspunktet bare trenge en eneste applikasjon installert, nemlig den som trengs for å lete opp og installere nye applikasjoner over nettet. Jeg tenker meg X.500 katalogen brukt til å søke etter de ønskede applikasjonene og en egen protokoll for å laste dem over. Denne applikasjonen bør også inneholde mekanismer for oppdatering av applikasjoner. Når nye versjoner av en applikasjon legges opp, bør brukeren automatisk kunne få overført den nye koden.

Lasting av nye applikasjoner er egentlig ikke noe annet en spesiell applikasjon som benytter en fast protokoll for å laste over koden. Den informasjonsforvalteren som har koden, trenger ikke være den samme som den som benyttes for å utveksle data.

6.8. Sammenligning mellom de tre typene applikasjoner



Figur 44 - Ulike måter å splitte en applikasjon på

Figur 44 viser hvordan de tre typene applikasjoner skiller mellom komponentene i en applikasjon. Den distribuerte applikasjonsmodellen deler det i to eller tre biter med applikasjonsagenten og applikasjonen som bindeleddet mellom det maskinvare avhengige som brukeren benytter og data/filhåndteringen som er det brukeren skal manipulere. I en vindusapplikasjon vil alt unntatt de maskinvare avhengige delene være håndtert av applikasjonen, mens i en selvstendig applikasjon vil alt være håndtert av applikasjonen. Både vindus- og selvstendige applikasjoner kan benytte klient-tjener modellen. De er da splittet mellom dataaksess og data/filhåndtering, men som vist i kapittel 5 vil man måtte lage egne applikasjoner for hver maskinplattform.

X-Windows er et spesial tilfelle. Den splitter klient og tjener i grensesnittet mellom det maskinvareavhengige og brukerdialogen. Slik at en X-applikasjon vil kommunikasjonen mellom det maskinvare avhengige og brukerdialogen benytte nettverket. Selv om dette gir en god fleksibilitet i lokale nettverk, vil det i store internasjonale nettverk ikke være tilstrekkelig fordi all respons til brukeren genererer nettverkstrafikk og brukeren vil bli prisgitt belastningen i datanettet. Skal en bruker i Norge benytte en applikasjon i Australia over satellittsamband, vil dessuten forsinkelsen bli for stor til at det blir videre nyttig.

For Vindus- og selvstendige applikasjoner som benytter klient-tjener modellen, vil applikasjonen måtte installeres og konfigureres manuelt på den enkelte maskin. Noe som krever mye vedlikehold og gir liten fleksibilitet i et stort verdensomspennende nettverk. Endringer og forbedringer vil bli for kostbart og komplisert når tusener av maskiner i alle verdensdeler benytter en applikasjon.

Den distribuerte applikasjonsmodellen er tenkt å selv kunne finne fram til applikasjoner i nettverket. Disse applikasjonene vet om hvilke informasjonsforvaltere de kan benytte. Ved å legge inn mekanismer for versjonskontroll, kan de som utviklet applikasjonen automatisk oppdatere applikasjonsagentene med siste versjon. Den distribuerte applikasjonsmodellene stiller heller ikke større krav til datanettet, enn de data som skal overføres. Ved store overføringer vil det selvsagt

ikke gå fortere enn nettverket tillater. For en rekke mindre oppgaver, slik som forespørsler i databaser, vil oppkobling over telefonnettet være tilstrekkelig.

7. En objektorientert tilnærming til den symbolske koden

Å velge en objektorientert tilnærming til den symbolske koden for den distribuerte applikasjons modell, er ikke bare et resultat av mine studier ved Universitetet i Oslo. Det viste seg også å være den eneste tilnærmingen som kunne fungere. Applikasjonens kode som agenten eksekverer er generell og den utvikleren vet absolutt ingenting om hva slags type maskinvare applikasjonen vil bli benyttet på. Det eneste han kan forutsette er at agenten kan utføre det den blir bedt om. Hvordan maskinvare avhengige ting gjøres verken kan, eller skal han behøve å vite. Applikasjonskoden må beskrive hva som skal vises til brukeren, ikke hvordan. Den må si hvilke av brukerens handlinger den vil ha rede på og hvilke deler av koden som da skal utføres. Det er applikasjonsagenten som sørger for at bindingen mellom applikasjonen og de maskinvareavhengige tingene er tilstede.

[KOSHAFIAN 91] skriver: "*Object orientation provides better paradigms and tools for:*

- *Modelling the real world as close to the user's perspective as possible*
- *Interacting easily with a computational environment, using familiar metaphors*
- *Construction reusable software components and easily extensible libraries of software modules*
- *Easily modifying and extending implementations of components without having to recode everything from scratch*"

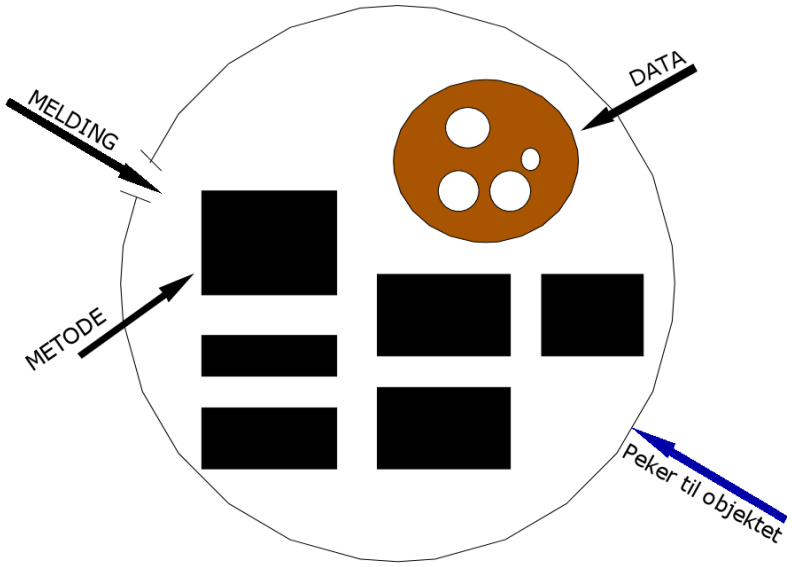
7.1. Objektorientert modell

Objektorientering har nærmest blitt et moteord innen databransjen de siste 2-3 årene. Mer enn 20 år etter at begrepet objekt ble innført med programmeringsspråket SIMULA, har disse idéene for alvor slått gjennom. Det er en mange ting som gjør det attraktivt å benytte en objektorientert tilnærming til den symbolske koden. Jeg skal i de følgende avsnitt peke på en del av fordelene. Det forutsettes at leseren har en viss kjennskap til objektorientert programmering og/eller objektorientert systembeskrivelse.

Innkapsling

Et objekt er en innkapsling av data og metodene for å manipulere disse dataene [BOGACKI 91]. Metoder vil i denne sammenhengen si operasjoner eller funksjoner og ordet er brukt i den engelske betydningen av ordet slik det er vanlig i engelsk språklig faglitteratur. Den eneste måten å manipulere disse dataene på bør være gjennom meldinger til objektet.

> "The principle of minimum visibility makes it possible to design objects that may be embedded in a wide variety of environments without the need of reprogramming. This principle is achieved through the clear separation of the outside and the inside of an object. The only characteristics visible on the outside are the operations that the object offers its neighbours, and the minimum requirements that the object puts on them. All details about data representation and program implementation are hidden within the object."
> [REENSKAUG 89]



Figur 45 - Et objekt er i følge Brad Cox: "en pakke informasjon og beskrivelsen av dets manipulasjoner"

De tingene applikasjonene trenger å forholde seg til, slik som vinduer, menyer, brukerinput ol., pakkes inn i objekter. Uansett hvilken maskinplattform og type agent applikasjonen eksekverer i, har disse objektene samme grensesnitt sett fra applikasjonens side. Applikasjonen sender en melding til objektet for et vindu om å vise en tekst og metodene i vindusobjektet avgjør hvordan teksten skrives ut.

Instansiering

Hvis vi trenger flere objekter av samme type, det vil si at objektene har samme data og metoder, kan vi lage en generell beskrivelse, kalt en klasse, for hvordan objekter av denne type skal se ut.

> "*Når vi oppretter objekter av en bestemt type, sier man at man instansierer klassen. Vi oppretter en instans av klassen. En instans av klassen A er det konkrete objektet som opprettes i henhold til beskrivelsen av klassen A*"
>
> [BOGACKI 91]

Typiske klasser i den distribuerte applikasjonsmodellen vil være klasser for tekstvinduer, editorer, scrollbarer og annet som trengs for å lage et brukervennlig grensesnitt. En applikasjon kan altså benytte de klassene applikasjonsagenten har og dermed selv bare bestå av den koden som trengs for å instansiere objektene den trenger med sine egne parametere, samt selve logikken for hva applikasjonen vil disse objektene skal gjøre. I tillegg kan den laste over klassedefinisjoner for egen bruk, som da igjen kan benyttes av andre applikasjoner, forutsatt at de er veldefinerte og kjent av dem som utvikler applikasjoner.

Spesialisering/generalisering

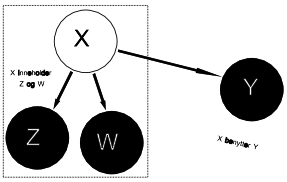
I stedet for å beskrive hver eneste klasse ned til minste detalj, kan vi beskrive en generell klasse som dekker et vidt felt. Ved å definere en annen klasse som subklasse til denne, arver den nye klassen alle egenskapene i sin superklasse. Subklassen kan redefinere metoder og datatyper som finnes i superklassen, samt legg til nye egenskaper. Hvis vi tenker oss noen vindusklasser med en generell vindusklasse på toppen som inneholder alle metoder og data som er felles for alle typer vinduer og så lager subklasser for spesielle type vinduer, får vi et hierarki av vindusklasser.

```
├─ Vindu ──┬─ Skjermvindu ─── ArbeidsVindu ──┬─ EditTekstVindu
          │                                  └─ HyperTekstVindu
          ├─ Dialog ──── "dialog subklasser"
          │
          └─ Kontroll ─── "kontroll subklasser"
```

Alle metoder i *Vindu* vil være tilgjengelig i *Skjermvindu*, *Dialog*, *Kontroll* og alle deres subklasser. Nye metoder for ting som scrollbarer, tittel ol. kan introduseres i klassen *Skjermvindu* og vil da være tilgjengelig i *ArbeidsVindu*, *EditTekstVindu* og *HyperTekstVindu*, men ikke i *Vindu*, eller noen av de andre klassene som er subklasser av *Vindu*. Hvis en metode er definert i både superklassen og subklassen, er det metoden i den laveste subklassen som benyttes. Det gir mulighet for å endre superklassens metoder. Der en applikasjon trenger ytterligere egenskaper i for eksempel et skjermbilde, kan programmereren lage egne subklasser med mulighet for å redefinere funksjonene i superklassen, samt føye til egne funksjoner og dataelementer til klassen. Alle standardobjektene og klassene som er definert i agenten er tilpasset en bestemt måte å presentere ting på for brukeren, men programmereren vil allikevel ha muligheten til å overstyre dette for sin applikasjon.

Pekere

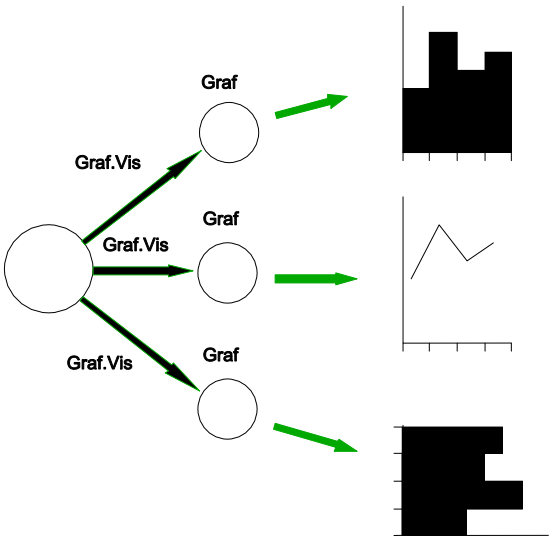
Pekere, eller referanser er mekanismen som tillater navngivning av objektene og som gjør det mulig å sende dem meldinger ved å bruke navnet. Pekere brukes også til å angi at et objekt benytter, eller består av andre objekter. De angir at *Objekt X benytter Objekt Y*, eller at *Objekt X inneholder objekt Z og objekt W*. For eksempel kan en applikasjon bestå av flere skjermbilder, som igjen består av flere menyer, felter, tegninger og annet.



Figur 46 - Pekere

Dynamisk binding

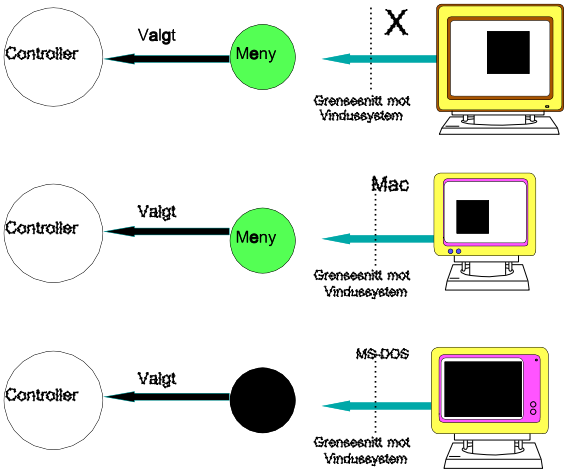
En melding til et objekt angir hva som skal gjøres, ikke hvordan. Dette gir mulighet for å endre ting i applikasjonen ved sende samme melding til et annet objekt som kan håndtere denne meldingen. For eksempel kan pekeren til et sammensatt grafisk objekt for å vise et stående søylediagram, endres til i stedet å peke på andre objekter for å vise en kurve, eller et liggende søylediagram ut fra den samme meldingen som sendes. I følge [RENSKAUG 89] gjør dette det enkelt å konfigurere vidt forskjellige systemet uten å reprogrammere, bare ved å velge og å koble objekter på ulike måter.



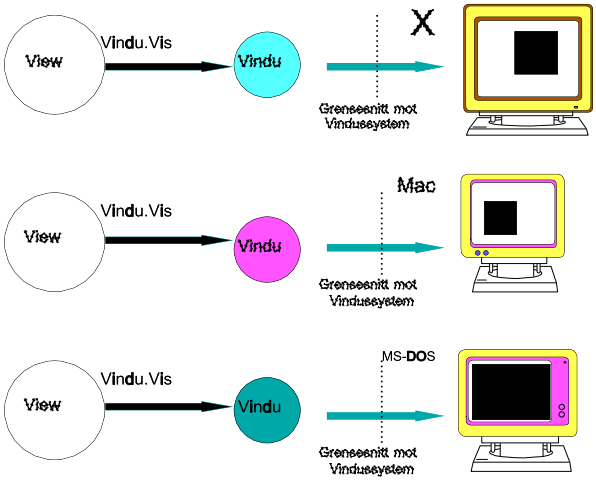
Figur 47 - Dynamisk binding

Ulik implementasjon på ulike maskinplattformer

En melding til for eksempel et skjermbildeobjekt som sier vis-skjerm, vil kunne utføres forskjellig på en PC og en Macintosh. Dermed er aksjonsfortolkerens oppgave den samme uansett hvilken type arbeidsplassutstyr det er snakk om.



Figur 49 - Eksempel på hvordan Valg fra en meny i et objekt av klassen Vindy kan gi samme melding i ulike vindussystemer.



Figur 48 - Eksempel på hvordan meldingen Vis til objekt av klassen Vindu kan være forskjellig i ulike vindussystemer

Tilsvarende kan samme teknikk benyttes for å ta imot det brukeren gjør. Uansett om applikasjonen benytter grafisk grensesnitt, funksjonstaster, eller Ctrl-Alt-ESC sekvenser for den del, tar den imot og sender samme melding tilbake til aksjonsfortolkeren. Det er metoden i objektet for eksempel i valgt meny opsjon, som er endret. Denne kan godt være knyttet til en funksjonstast, eller en kombinasjon av funksjonstaster, piltaster og retur.

7.2. Smalltalk

Smalltalk-80 ble utviklet på 70-tallet ved Xerox PARC. I første omgang som programvare delen av Dynabook, men siden også for andre maskinplattformer [KUTAL 90]. Smalltalk er både et programmeringsspråk og et interaktivt programmeringsmiljø [GOLDBERG]. I denne sammenhengen er det mekanismene i språket som er interessante.

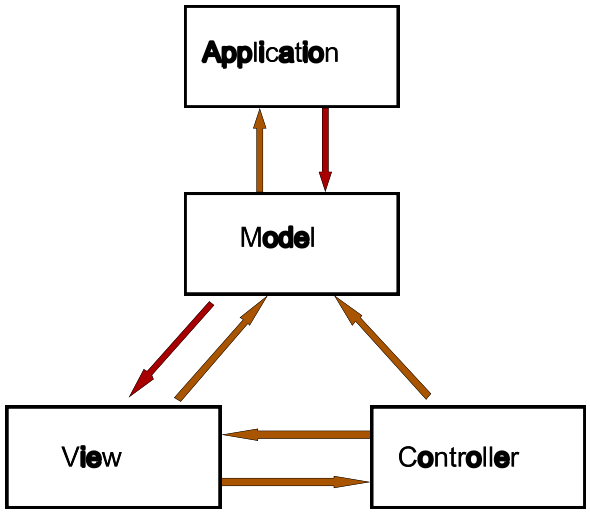
Smalltalk er objektorientert og basert på at objektene sender meldinger seg imellom. En klasse i Smalltalk beskriver implementasjonen for et sett med objekter som alle representerer samme type systemkomponent. De individuelle objektene beskrevet av en klasse kalles instanser. Instansene av en klasse er identiske i både eksterne og interne egenskaper. Et objekts eksterne egenskaper er de meldingene som utgjør grensesnittet mot andre objekter og de interne er privat hukommelse, samt et sett med metoder som beskriver hvordan ulike operasjoner skal utføres.

Objekter kommuniserer gjennom de meldingene de sender til hverandre. En melding er en forespørsel til et objekt om å utføre en av sine operasjoner. En melding forteller hvilken handling den ønsker utført, men vet ingenting om hvordan denne handlingen blir utført. Mottakeren, objektet meldingen blir sendt til, avgjør hvordan operasjonen skal bli utført. Hvilke meldinger et objekt kan svare på utgjør dets grensesnitt mot resten av systemet.

Klasser er relatert til sine superklasser og metaklasser. Disse klassene deler ekstern og intern beskrivelse og er derfor avhengig av hverandre. En ekstra type avhengighet er støttet i Smalltalk gjennom en avhengighetsliste (*Dependency list*) i klassen *Object*, som er superklassen for alle

klasser. For eksempel hvis objekt A er i avhengighetslisten for objekt B, vil objekt B bli informert om endringer i objekt A og kan avgjøre om det skal ta noen aksjon.

Smalltalk deler ansvaret for applikasjonenes vindushåndtering mellom objekter av typen model, view og control. Som hver har sine oppgaver for at brukeren skal kunne kommunisere med applikasjonen gjennom vindussystemet.



Figur 50 - Smalltalk deler håndteringen av vindussystemet mellom tre typer objekter.

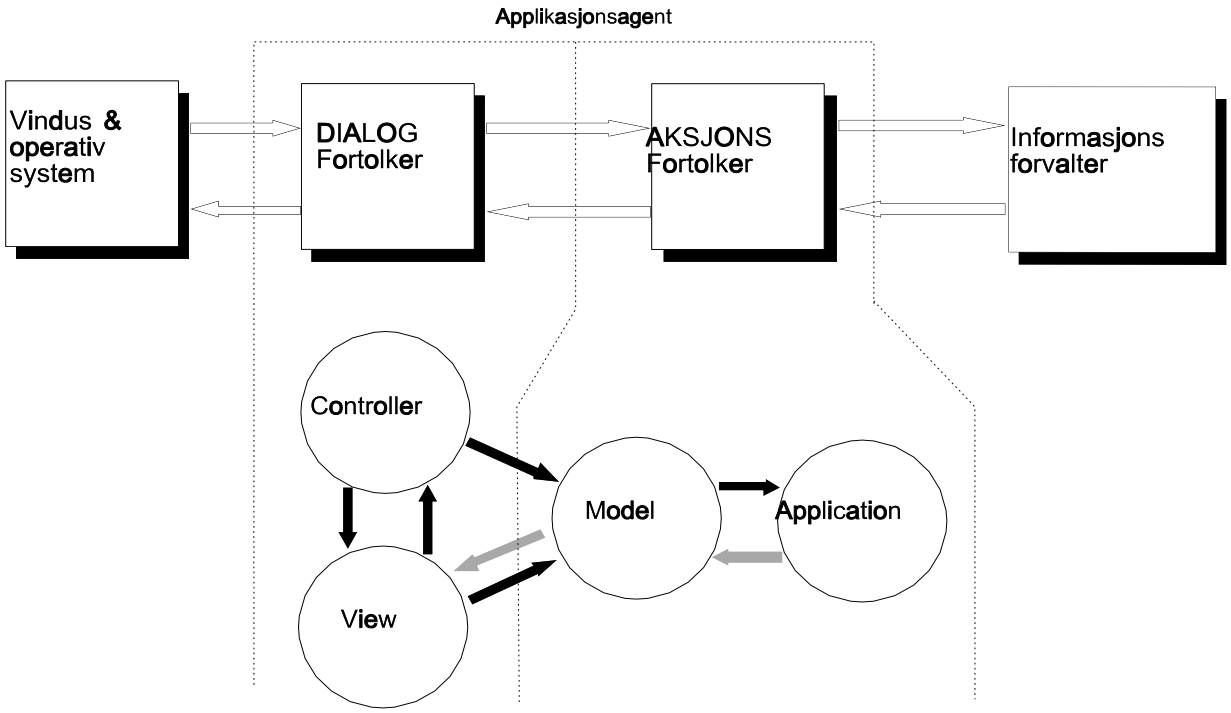
Model: Et objekt som representerer datastrukturen for det applikasjonen viser.

View: Et objekt som kontrollerer de visuelle aspektene ved vinduet. Det holder rede på grensene for vinduet og metodene for å oppdatere innholdet i vinduet. Objektet kan være sammensatt av flere subviews.

Control: Et objekt som kontrollerer brukerens interaksjon med vinduet. Det har informasjon om hvorvidt markøren er innenfor vinduet og eventuelt hvor, hva skal gjøres når vinduet blir aktivt, samt om brukeren har trykket på tastatur eller museknapper.

Generelt benyttes View og Control parvis. Det vil si at hvert View objekt har et Control objekt assosiert med seg. Applikasjonen har grensesnitt mot vindussystemet gjennom sine Model objekter [KUTAL 90] [LALONDE 91].

Forskjellen mellom det Smalltalk gjør og det som kreves for den distribuerte applikasjonsmodellen er at Smalltalk forutsetter et grafisk brukergrensesnitt og legger svært stor vekt på det, mens det ikke er noe absolutt krav i den distribuerte applikasjons modellen. Etterhvert som maskinkraften stadig blir bedre og billigere, vil nok grafiske brukergrensesnitt bli det framherskende. Modellen retter seg derfor mye mot denne type utstyr for å støtte deres funksjonalitet, samtidig som det gis en mulighet for å lage enklere applikasjoner for maskinplattformer som ikke har avanserte grafiske muligheter.



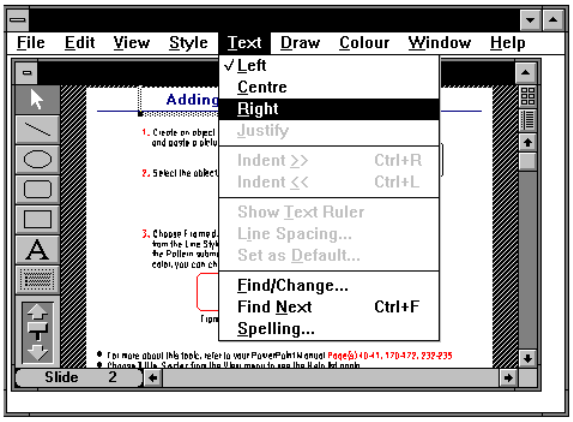
Figur 51 - Model-View-Control i den distribuerte applikasjons modellen

Figur 51 viser hvordan Smalltalks Model-View-Control tankegang passer inn i den distribuerte applikasjonsmodellen. Dialogfortolkeren er ment å ta seg av omtrent det samme som objektene for View og Control i Smalltalk, mens aksjonsfortolkeren skal ta seg av Model og applikasjonen. Applikasjonen på applikasjonsagenten må ha sitt grensesnitt mot informasjonsforvalteren for å kunne utveksle data.

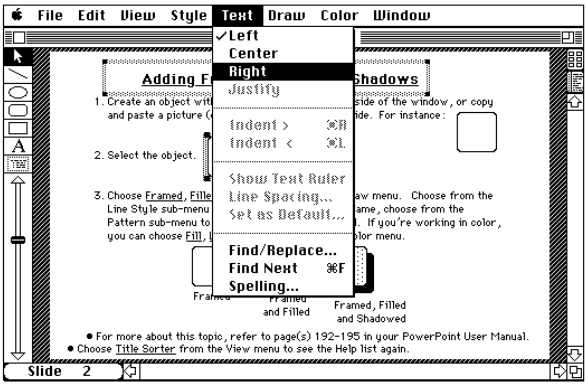
7.3. Klasser og objekter i den distribuerte applikasjonsmodellen

Klassedefinisjonene må deles opp i grupper og en del basis grupper av klasser må implementeres sammen med applikasjonsagenten. Hvilke av disse gruppene som implementeres, avhenger av maskinvaren den skal benyttes på. For eksempel er det liten vits å implementere lydklasser i en agent som går på arbeidsplassutstyr uten høyttaler. Mens klasser for tekstvindu, listbokser, scrollbarer ol. bør finnes i alle agenter. I en applikasjon kan man definere nye klasser som subklasser av standard basisklasser. Begrensningen ligger i hvilke aksjoner applikasjonsagenten kan håndtere. Det er derfor viktig å definere tilstrekkelig med standard basisklasser som en del av applikasjonsagenten. Det er disse som avgjør hva applikasjonsagenten er i stand til å utføre.

Det er viktig å understreke at applikasjonsutvikleren ikke har kontroll over hvordan objektene blir seende ut på den enkelte maskinplattform. Det er applikasjonsagenten, som altså er det programmet som får koden til å fungere, som bestemmer utseende. Applikasjonen ber om et vindu av en gitt type og ber om å få plassert tekst, knapper, grafikk og lignende plassert i det vinduet. Hvordan dette tar seg ut rent visuelt er det implementasjonen av applikasjonsagenten og dens utnyttelse av mulighetene i et eventuelt vindussystem som avgjør. En OK Knapp i X-Windows med OSF/Motif vil få den vanlige OK knappen der med tredimensjonal effekt, mens tilsvarende OK Knapp på en Macintosh vil ha doble linjer rundt seg. Menyer på en Macintosh vil oppføre seg som alle andre menyer der, mens de samme menyene i Windows vil oppføre seg og se ut som andre Windows applikasjoner. Noe av hensikten med modellen er jo nettopp at applikasjoner som er distribuerte ikke skal fortone seg annerledes enn lokale applikasjoner.



Figur 52 - PowerPoint i MS-Windows 3.0



Figur 53 - PowerPoint på Macintosh

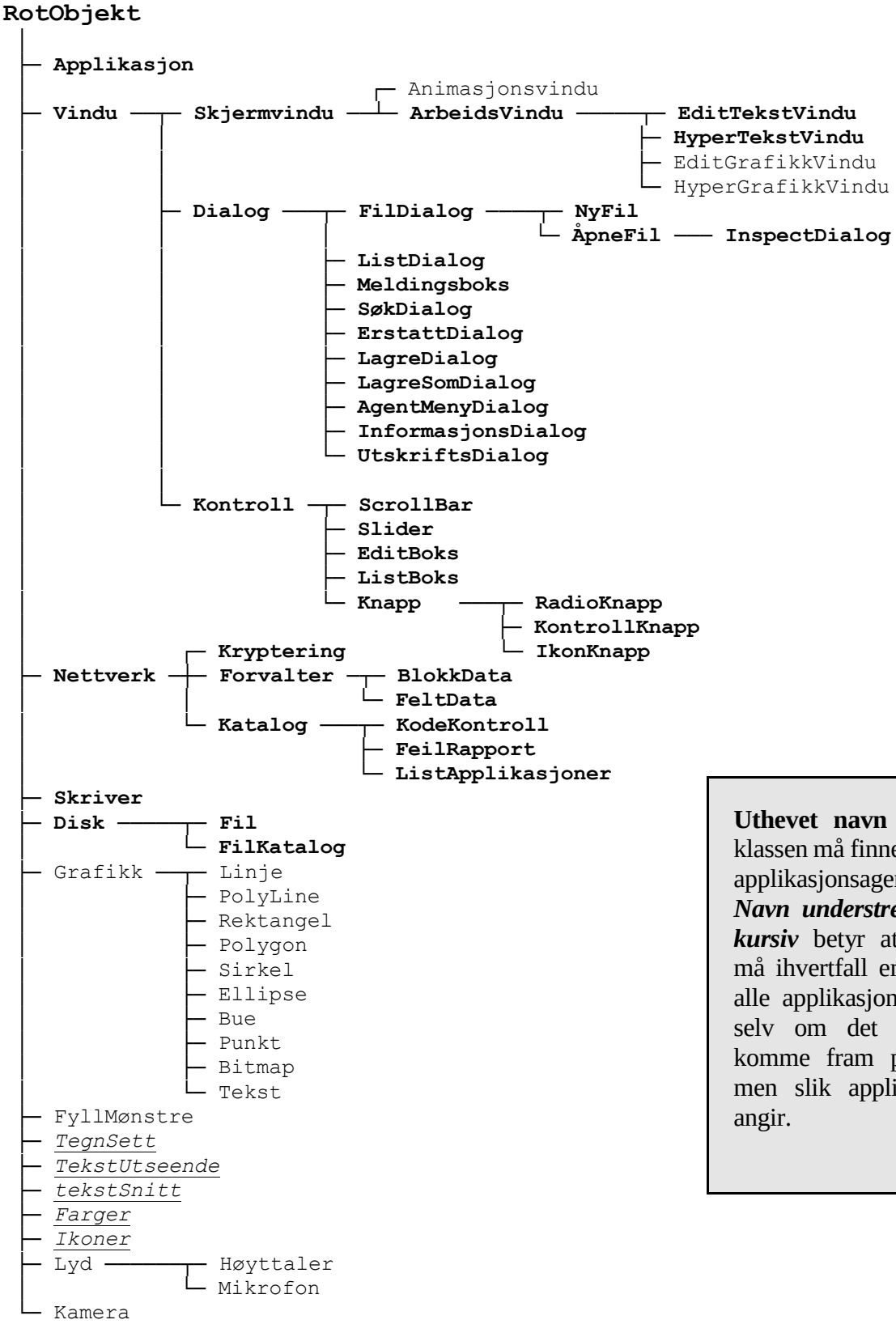
7.4. Generalisering av elementene i brukerdialogen

Et av målene med den distribuerte applikasjonsmodellen er å abstrahere funksjonene i en applikasjon tilstrekkelig til at de blir uavhengig av hvordan de presenteres for brukeren, slik at det samme brukergrensesnittet kan benyttes på mange maskinplattformer uten å kreve særlig tilpasning for hver av de tre gruppene brukergrensesnitt (se kapittel 4).

For å kunne gjøre dette, må vi finne de hovedgrupper av funksjoner en applikasjon trenger og se om de lar seg utføre under ulike vindussystemer, samt på skjermorienterte og linje orienterte brukergrensesnitt.

7.5. Basisklasser

I de følgende avsnittene vil jeg skissere hvilke grupper av klasser jeg mener trengs for å gi en tilfredsstillende implementasjon av applikasjonsagenten.



Uthevet navn betyr at klassen må finnes for alle applikasjonsagenter.
Navn understreket og i kursiv betyr at klassen må ihvertfall emuleres i alle applikasjonsagenter, selv om det ikke vil komme fram på skjermen slik applikasjonen angir.

7.6. Applikasjon

Hver applikasjon som går på applikasjonsagenten er et objekt av klassen *Applikasjon.* Dette objektet inneholder i utgangspunktet bare metodene for å starte og sette opp et enkelt *SkjermVindu* på skjermen. Utvikleren må selv legge inn de metodene som skal benyttes og hvilke meldinger applikasjonen skal kunne motta. Applikasjonsobjektet inneholder det som trengs for å kommunisere med brukeren gjennom brukergrensesnittet og for å sende og motta data til informasjonsforvalteren på applikasjonstjeneren.

For å håndtere kommunikasjonen mellom bruker og de data som applikasjonen er skrevet for å håndtere, kan utvikleren benytte de klassene som er definert, eller definere sine egne klasser som subklasser av disse. Et mål er å ha de mest vanlige funksjonene forhåndsdefinert i applikasjonsagenten, slik at selv applikasjonskoden blir så enkel som mulig.

Før applikasjonen starter kommunikasjonen mot sin informasjonsforvalter, må den finne ut om de klassene den trenger finnes i agenten. De som er obligatoriske kan den ta for gitt, men de andre må den sjekke. Ved å sende en melding til rotobjektet med spørsmål om hvilke klasser som finnes kan applikasjonen sjekke om det for eksempel finnes grafikkmuligheter, hvilke fyllmønstre som er mulig eller om agenten har mikrofon. Den trenger ikke vite hva slags maskin det er, eller hvilket operativsystem den kjører, men den bør kunne sjekke hvilken versjon den aktive applikasjonsagenten er.

Hvis man vil lure applikasjonen til å tro at agenten er bedre utstyrt enn den er, kan man godt gjøre det. Ved å definere klasser for utstyr som agenten ikke har, vil applikasjonen tro at den disponerer slikt utstyr. Man må da finne metoder for å nyttegjøre seg den informasjonen som applikasjonen forsøker å gi. Et eksempel på dette kan være at man implementerer lydklassene og viser en tekstlig representasjon i et eget vindu hver gang applikasjonen forsøker å lage lyd.

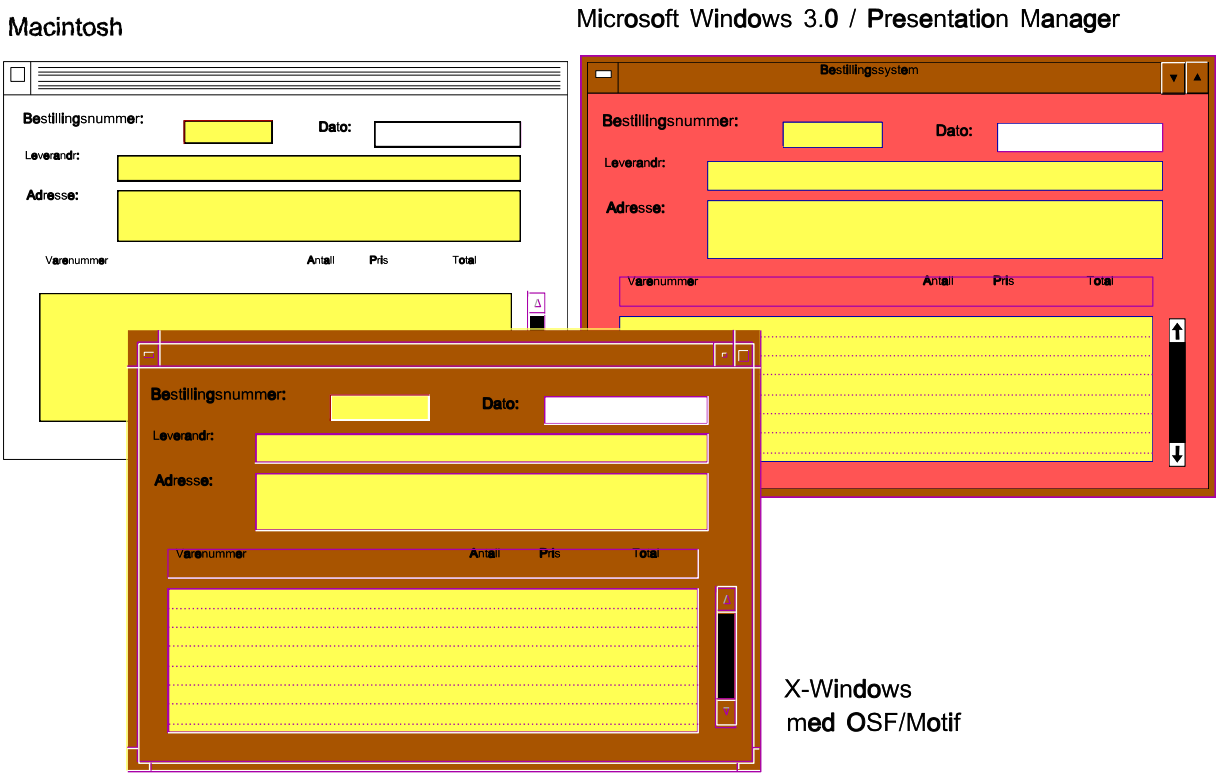
Applikasjonen kan også inneholde de funksjonene som trengs for å oppdatere den fra en tjener, for å stoppe den, for å fjerne den fra klienten, eller hvilke slike funksjoner applikasjonsutvikleren ønsker å legge inn i applikasjonen. Det kan for eksempel være nyttig om en betalingsapplikasjon sletter seg selv når prøvetiden er utgått, eller når man ikke ønsker å benytte den lenger. En applikasjon kan også inneholde nødvendige aksjoner for å starte andre applikasjoner fra en hvilken som helst tjener.

7.7. Vindusklasser

Klassen *Vindu* er et rektangulært vindu med en gitt størrelse og posisjon. Alle skjermobjekter, slik som vinduer for tekst og grafikkvinduer, dialogvinduer, scrollbarer eller kontroll er subklasser av *Vindu.* Denne klassen gir en abstraksjon av de metodene som de andre subklassene benytter for å kommunisere med skjermen, samt registrere hva brukeren gjør.

7.7.1. TekstVinduer

Alle vindusbaserte systemer har muligheten for å opprette vinduer på skjermen. Noen vindussystemer, som for eksempel X har muligheten for også å lage elipseformede vinduer, så hvis en applikasjonsutvikler insisterer på at vinduet skal være ellipseformet, vil det begrense antallet miljøer det vil kunne fungere under. Men for de fleste oppgaver vil et rektangulært vindu være tilstrekkelig.

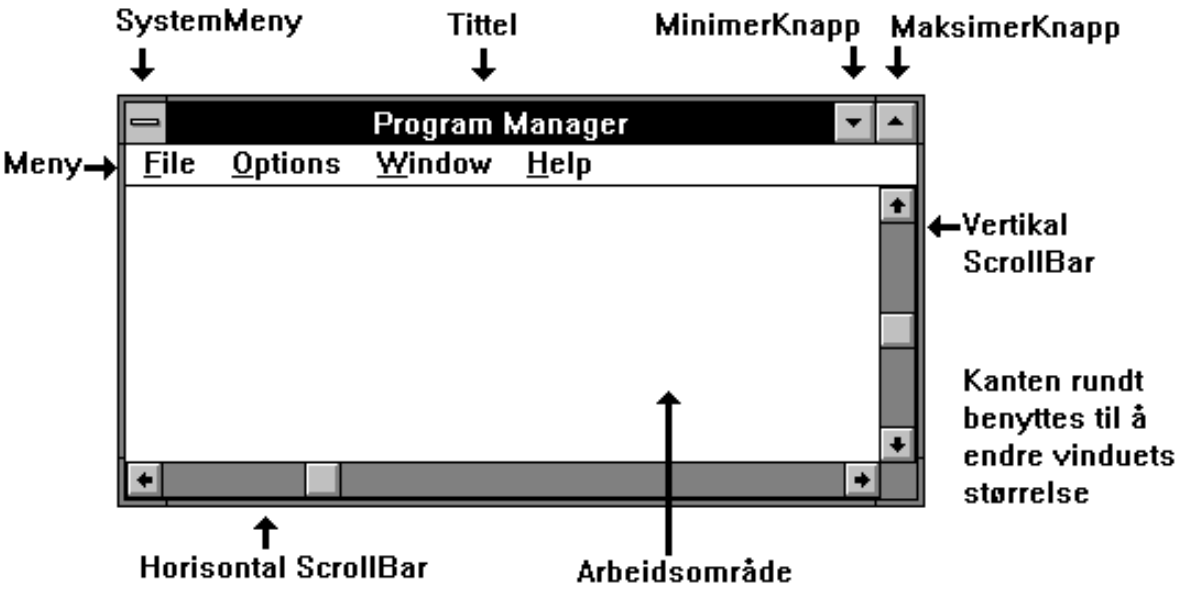


Figur 54 - Eksempel på samme database applikasjon i tre vindussystemer

Et vindu er i prinsippet ikke noe annet enn et skjermbilde, slik at også et skjermorientert system vil kunne vise et "vindu" som i det tilfellet er hele skjermen. En VT100 eller en IBM3270 terminal har denne muligheten, men er begrenset til 80x24 tegn. Krever applikasjonen mer, vil den ekskludere en del systemer. Et vindu har som oftest en tittel og gjerne også scrollbarer til å flytte skjermbildet fram eller bakover. Tittelen burde ikke være noe problem å få representert på noe system. Scrollbarene kan derimot være meningsløse på en VT100 skjerm uten mus. Funksjonen med bla opp og bla ned derimot kan godt legges til en tastkombinasjon, slik at applikasjonen får det samme signalet enten det er en VT100 skjerm hvor brukeren har trykket tasten for *skjerm-ned*, eller det er Macintosh hvor musa benyttes til å bla i skjerminnholdet. I et linjeorientert system vil det samme kunne la seg representere ved at skjermen skrives om igjen med den teksten som i andre systemer ville rullet fram i vinduet.

Et tekstvindu kan inneholde forskjellige ting. Det kan være et skjermbilde i en database applikasjon, beregnet for å oppdatere data. Det kan også være en tekstbehandler. Det kan være sider med informasjon som ikke lar seg redigere og andre anvendelser. Det eneste som begrenser det er at det ikke kan brukes til å framstille grafikk. Ulike skrifttyper (fonter), farger er det derimot ingenting i veien for å ha, men det vil ikke kunne vises på alle typer utstyr. Et dokument som inneholder grafikk burde kunne vises i et tekstvindu, men med et åpent felt der grafikken er.

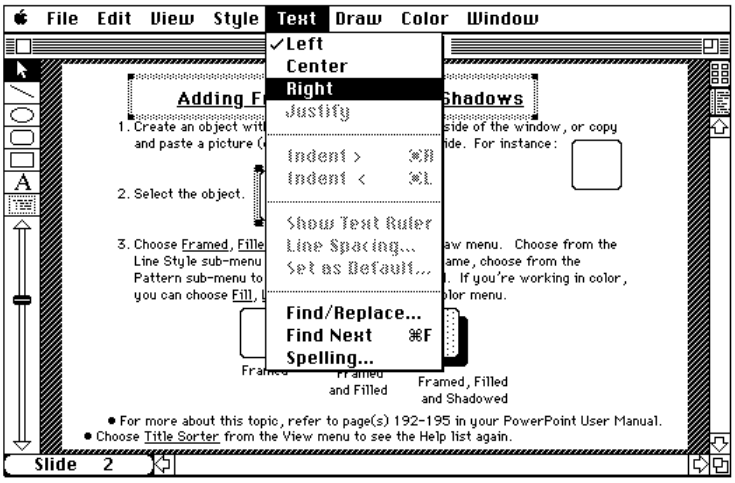
Den øverste klassen for tekst er *Skjermvindu*. Dette er et enkelt vindu med tittel, samt funksjoner for å minimalisere og maksimalisere det, endre størrelsen og et område for applikasjonen å skrive ut tekst i. Programmereren kan velge hvilke av disse funksjonene han vil ha med i vinduet ved instansiering av vindusobjektet.



Figur 55 - Elementene i et ArbeidsVindu

ArbeidsVindu er subklasse av *SkjermVindu*, og har i tillegg scrollbarer vertikalt og/eller horisontalt, samt menyer. Programmereren initierer menyene ved oppstart og gir dem innhold. Hvilke menyalternativer som til enhver tid er tilgjengelig styres fra applikasjonen ved å aktivisere og deaktivisere menyvalgene i *ArbeidsVindu.* Hvis man bare skal vise tekst på skjermen og la brukeren bla opp og ned er denne klassen tilstrekkelig. *EditTekstVindu* er en teksteditor som fungerer slik editorer normalt fungerer på det aktuelle systemet. Dette bestemmes av hvordan applikasjonsagenten selv er implementert. Fortrinnsvis bør agenten benytte de rutinene for teksteditering som finnes tilgjengelig. Her er også muligheter for å endre fonter, tekststil, farge og lignende fra menyene. Ikke alle maskiner vil kunne vise alle typer fonter eller tegnsett, men brukeren bør allikevel kunne endre dem og se hvilke attributter hvert tegn i teksten har. *EditTekstVindu* bør gjøres så avansert at det holder for de fleste formål. All editering foregår lokalt i EditTekstVindu og applikasjonen får bare beskjed om hva som eventuelt blir gjort i menyer, scrollbarer og lignende. For å få direkte kobling mellom teksten og selve applikasjonen må man benytte *HyperTekstVindu.*

HyperTekstVindu er en variant av *ArbeidsVinduer* for å kunne koble sammen tekst og applikasjon, blant annet for å kunne lage mer avanserte hjelpesystemer. Det kan være editerbart på samme måte som *EditTekstVindu.* Hensikten er å kunne markere deler av en tekst, slik at hvis brukeren velger denne teksten, får applikasjonen en melding tilbake og kan handle ut fra det. Det kan gjerne være et leksikon hvor brukeren peker på et ord og applikasjonen gir forklaring på ordet. Eller det kan være et tekstbasert informasjonssystem for det offentlige, hvor forskrifter tilhørende en lov kan vises ved at brukeren peker på bestemte deler av teksten.

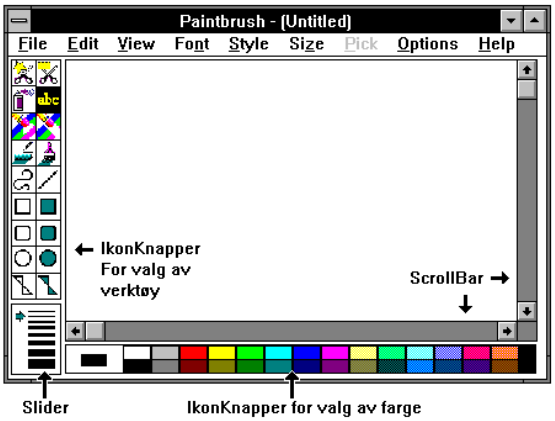


Figur 56 - Eksempel på EditTekstVindu på Macintosh

Både EditTekstVindu og HyperTekstVindu klassene og deres superklasser, kan vise grafikk, men bare på de systemene hvor grafikk er mulig. På andre systemer, vil bildet bare bli markert med et tomt felt. Det blir altså en avveining informasjonsleverandøren og brukeren må gjøre, mhp. verdien av teksten uten bildene. Hensikten med tekstvinduene er å understøtte applikasjoner hvor teksten er det primære. Det vil for eksempel være mulig å la brukeren ta en utskrift av alle bildene og få dem på papir ved siden av, hvis det er ønskelig. En slik anvendelse kan være en elektronisk avis hvor bildene stort sett er like meningsløse som i dagens tabloid presse. Brukeren kan få fram teksten og attpåtil bli spart for bildene.

7.7.2. GrafikkVinduer

Selv om grafikk er ulikt implementert i de forskjellige vindussystemene, må en applikasjon kunne be om et vindu som kan vise grafikk. Applikasjonsagenten må oversette fra et generelt grafisk format til noe som lar seg representere i de ulike vindussystemene. Tilsvarende må dialogfortolkeren kunne gi tilbake opplysninger om hva brukeren har gjort med hvilket grafikkelement på skjermen. Å lage en applikasjonsagent som håndterer grafikk uten å benytte noe vindusystem, vil bli en krevende oppgave, men ikke umulig. De funksjonene som trengs for å håndtere grafikk på den aktuelle maskinvaren må da styres av agenten selv. Et grafikkvindu kan inneholde vektor eller bitmap basert grafikk.



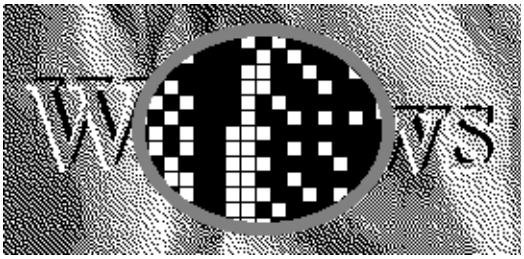
Figur 57 - Eksempel på GrafikkVindu i MS-Windows 3.0

7.7.3. Vektor grafikk

Vektorgrafikken er uavhengig av hva slags maskinvare det er laget på eller for. De grafiske elementene beskrives ved hjelp av relative geometriske vektorer. Applikasjonsagent og vindussystemet må presentere grafikken best mulig på den aktuelle skjermen. Applikasjonen lagrer bare hvilke grafiske elementer som inngår og deres respektive vektorer. Det finnes mange programmer som benytter slike teknikker. Et program som AutoCAD benytter samme filstruktur uavhengig av hva slags maskinplattform den går på. Det vil si at en bruker kan tegne en konstruksjonstegning på en Macintosh og så levere filen med tegningen til en annen som benytter en Sun arbeidsstasjon uten problemer.

7.7.4. Bitmap grafikk

De fleste vindussystemer har et format for såkalt bitmap. Det vil si at et grafikk bilde er beskrevet i form av hvordan ulike punkter ser ut innen et område. I praksis er dette den mest primitive, men også mest maskinnære grafikkformen. I siste instans blir alt som skrives til skjermen, enten dette er tekst, vektorgrafikk, vindusrammer og lignende oversatt til bitmap. Fordelen med bitmap er at det er raskere å vise på skjermen, enn om man skulle representert det samme i form vektorgrafikk. Bitmap er derimot ikke så fleksible å endre som vektorgrafikk.



Figur 58 - Eksempel på forstørrelse av bitmap grafikk

Det finnes en lang rekke forskjellige formater for bitmaps. Men det er som regel mulig å oversette fra et format til et annet format. For å generalisere bitmap er det viktig at det velges et format som gir stor grad av fleksibilitet. At det med andre ord inneholder tilstrekkelig informasjon til å gi et optimalt resultat på de mest avanserte maskinene, samtidig som det er enkelt å vise på enklere utstyr uten at for mye går tapt. For eksempel må et bilde med 256 farger kunne vises på en skjerm med bare 16 farger uten at det bare blir grums. Og et bilde som er 1024x768 punkter må kunne la seg presse inn på en skjerm som bare er 600x480 punkter. Jeg vil ikke ta stilling til hvilket format som er det beste. Uansett må det være mulig å identifisere hvilket format som eventuelt benyttes, slik at applikasjonsagenten kan oversette til det formatet som benyttes internt. Hvis det ikke er mulig eller ønskelig med et felles standard format på bitmaps, må det i alle fall være mulig for applikasjonsagenten å oversette mellom formatene på grunnlag av identifikasjonen av formatet.

Klassen *EditGrafikkVindu* og *HyperGrafikkVindu* gir mulighet for å vise og editere grafikk. Applikasjonsagentene på den enkelte type arbeidsplassutstyr må derfor forsøke å koble de grafiske muligheter applikasjoner etter den distribuerte modellen behøver mot de mulighetene som finnes i det aktuelle vindussystemet. De ulike grafikk elementene som ihvertfall bør støttes er gitt under klassen *Grafikk*. I tillegg kan applikasjonen lage sine egne klasser som er sammensatt av flere objekter av disse *Grafikk* klassene. Forskjellen på *EditGrafikkVindu* og *HyperGrafikkVindu* er at i *EditGrafikkVindu* redigerer man på grafikken til man er fornøyd og så lagrer det, skriver det ut eller hva man nå ønsker å gjøre med den ferdige illustrasjonen. Grafikk objekter i et *HyperGrafikkVindu* er koblet til applikasjonen, slik at hvis man peker på et objekt, får applikasjonen beskjed. Dette kan benyttes til å lage grafer over informasjon, organisasjoner, eller lignende. Det kan også brukes til å lage "intelligente" tegninger, hvor grafikk objektene i tegningen vet hva de representerer og ikke bare inneholder sine vektorer. *HyperGrafikkVinduer* kan sammenlignes med HyperCard på Macintosh og vil gi muligheter for å lage denne typen applikasjoner.

Heller ikke animasjon er som regel en standard del av vindussystemet. Foreløpig støttes animasjon bare av et fåtall maskinplattformer med spesiell maskin- og/eller programvare. Det finnes også utstyr for å digitalisere bilder fra video-kameraer.

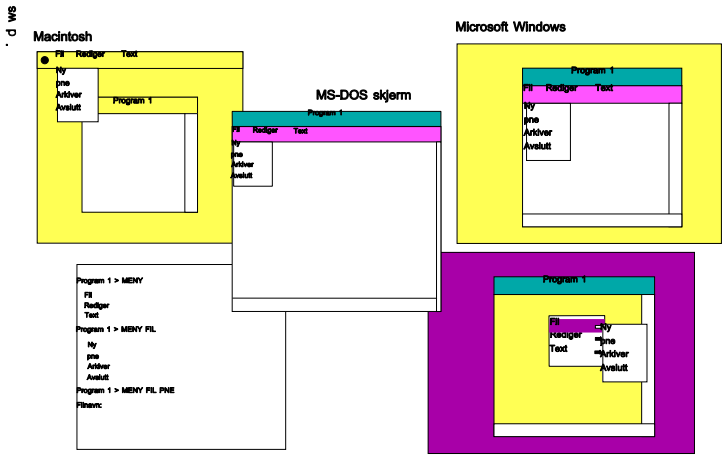
Poenget må være at animasjon generaliseres såpass at det kan benyttes på flere ulike typer maskinplattformer. Det er mulig det kan være en fordel å standardisere formatet på bilder benyttet i animasjon til det samme som i bitmaps.

Figur 59 - Eksempel på mulig billedtelefon i AnimasjonsVindu

AnimasjonsVindu er satt om en subklasse direkte under *SkjermVindu.* Hvordan dette rent faktisk implementeres vil variere etter hva slags type maskinvare som er tilgjengelig. Sett fra applikasjonens side er det nødvendig med et objekt som representerer et vindu hvor den kan sende over meldingene med de data som er nødvendig for å lage levende bilder på skjermen. Poenget er å lage en klasse med et gitt sett med metoder som er kjent utenfra og så kan det implementeres forskjellig i enkelte applikasjonsagent. AnimasjonsVinduet bør være i stand til å ta data fra *Disk, Kamera, Nett* og intern hukommelsen i form av meldinger til objektet. Brukeren kan ikke gi input direkte i Animasjonsvinduet, men andre vinduer kan brukes til å redigere animasjonssekvenser. Data fra andre applikasjoner bør kunne benyttes i en animasjon på same måte som input fra kamera.

7.7.5. Menyer

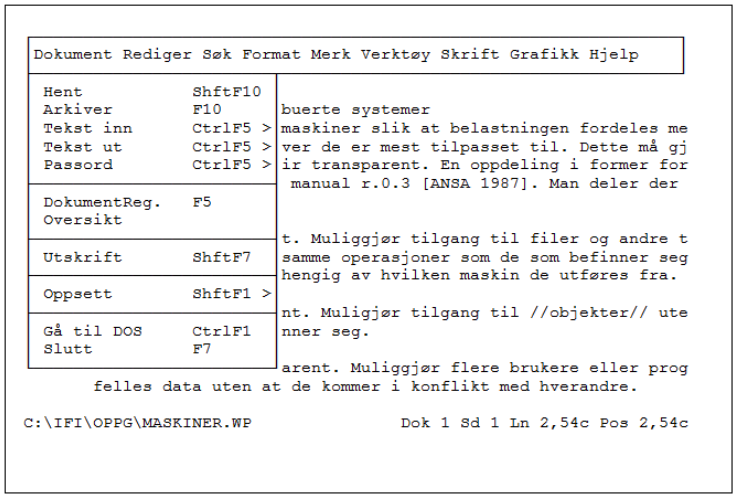
Menyer er en del av klassen *ArbeidsVindu*. I vindussystemer er såkalte rullegardinmenyer det vanligste. Menyer som ruller ned fra overkant av skjermbildet, eller vinduet, med mulighet for brukeren å velge funksjoner. I mange X-Windows applikasjoner spretter menyene fram der musmarkøren er, men forskjellen er egentlig minimal. I skjermorienterte systemer, var det tidligere vanlig at det var egne menyskjermbilder som førte fram til de ulike delene av systemet. Ettersom vindussystemer har fått økt popularitet, har såkalte rullegardinmenyer også blitt det normale i skjermorienterte systemer på mikromaskiner. I linjeorienterte systemer er det vanlig at en kommando ruller fram de valgalternativene som finnes på det nivået eller i den modus programmet befinner seg nå.



Figur 60 - Eksempler på menyer i ulike brukergrensesnitt

Menyer er muligheten for å vise brukeren hvilke funksjoner som er tilgjengelig og kunne oppfatte hvilke funksjoner brukeren ønsker utført. For applikasjonen er dette det interessante, ikke hva og hvordan brukeren faktisk har måttet peke eller trykke for å komme fram til akkurat den funksjonen.

I mange vindussystemer er det mulig å gi slike tastkombinasjoner som et alternativ til å rulle ned menyene for å velge funksjoner. Både i vindus- og skjermorienterte omgivelser kan det lages rullegardinmenyer og tastkombinasjoner, men for terminaler som VT100 og IBM3270 vil det å tegne ut menyer som oftest være lite effektivt, fordi det tar tid. På en MS-DOS maskin uten vindussystem, vil begge måter være tilfredsstillende. I linjeorienterte systemer er tastkombinasjoner eller kommandoer eneste mulighet for å velge funksjoner, men man kan selvsagt skrive ut menyen for å vise hvilke tastkombinasjoner eller kommandoer, som gir hvilke funksjoner. Poenget er at menyer kan implementeres på flere måter, men for applikasjonen er det bare en måte å få tilbake beskjed fra brukeren om hva han ønsker å gjøre. Hva som er mest effektivt og brukervennlig er ikke spørsmålet her. Poenget er at det går ann å abstrahere måten bruker kommuniserer med applikasjonen på.



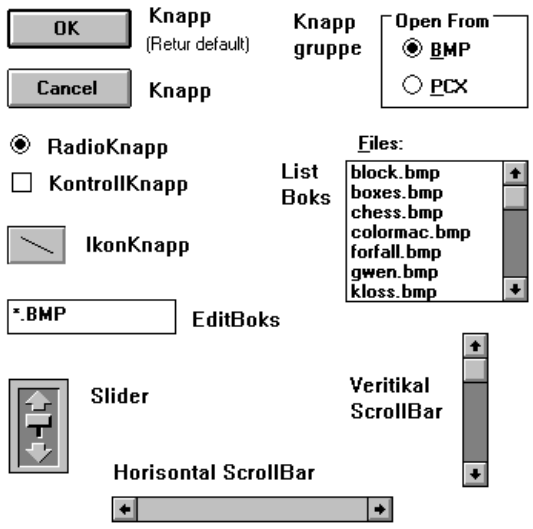
Figur 61 - Eksempel på menyer på MS-DOS skjerm

7.7.6. DialogVinduer

I linjeorienterte og tildels også i skjermorienterte brukergrensesnitt, er det vanlig at applikasjonene kommer med en melding på skjermen sammen med all den andre teksten. Tilsvarende hvis det er noe programmet må ha som input fra brukeren, blir det bare stilt et spørsmål som brukeren må besvare før programmet går videre. I vindusbaserte systemer a la Macintosh var ikke det like enkelt og kanskje heller ikke særlig pedagogisk. Teknikken som er blitt vanlig, er å lage et eget subvindu for slike meldinger og tilsvarende for å be om input fra brukeren.

Heller ikke her kan jeg se noen forskjell, sett fra applikasjonene side. Den vil gi brukeren en medling og eventuelt få tilbake noe.

For å forenkle utviklingen av applikasjoner for den distribuerte applikasjons modellen, er antallet dialogklasser temmelig høyt. Det gjør det enkelt for programmereren å instansiere objekter av den dialog typen han ønsker med minimal koding. De fleste av disse dialogtypene er tilnærmet like i Windows og på en Macintosh. Det er heller ikke videre komplisert å implementere hverken i MS-DOS skjermbaserte agenter, eller i X-Windows. Dialogvinduene tar input fra brukeren og sender det tilbake til det objektet som initsierte dem.



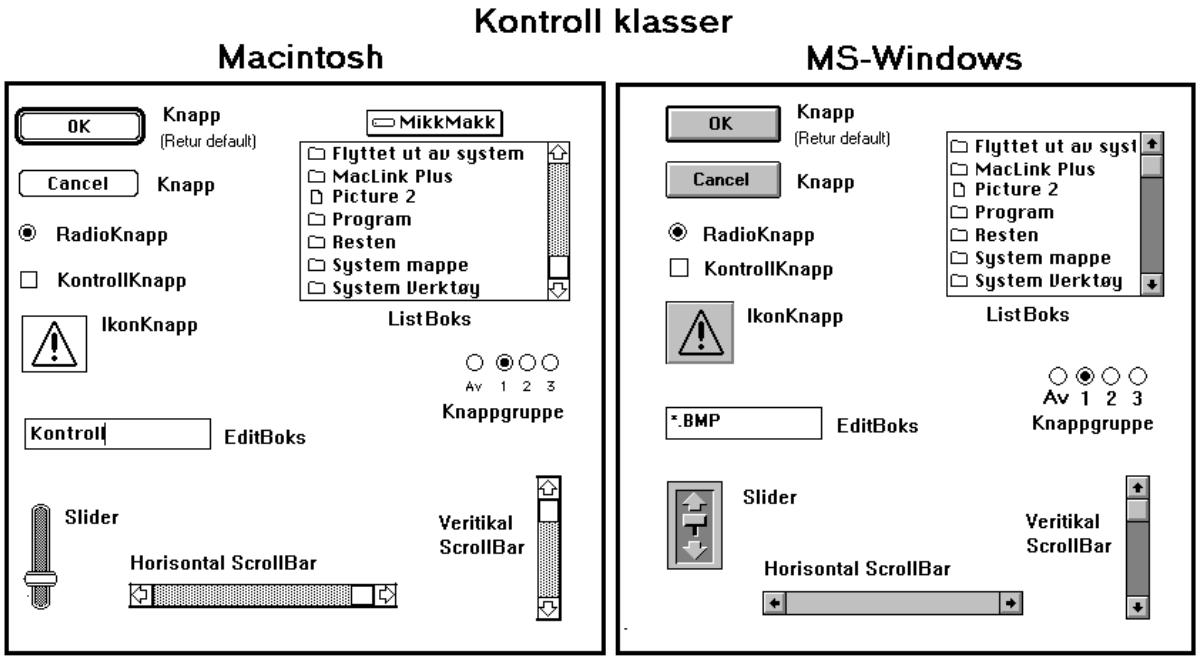
Figur 62 - Noen eksempler på dialogklasser tatt fra MS-Windows

InformasjonsDialogen er det som på en Macintosh og i Windows heter "About ...". Her bør det stå hvem som har laget applikasjonen, hvor applikasjonen henter sine data fra og hvor brukeren eventuelt kan lagre sine data og hvor feilmeldinger og lignende kan sendes. Den bør også ha inngang til et *HyperTekstVindu* basert hjelpesystem for applikasjonen.

AgentMenyDialog er litt spesiell. Den gir en egen meny for selv agent programmet. Altså det programmet som kjører alle de andre applikasjonene. Dette for å kunne få ut status, endre tilstanden på agenten, starte andre applikasjoner som agenten kjenner til, bytte til andre applikasjoner som ligger i bakgrunnen, søke i katalogen etter nye applikasjoner, eller avslutt alle applikasjoner og stoppe hele agenten. Applikasjonene kan selv legge inn nye valg i denne menyen mens de kjører, eller permanent. AgentMenyen bør kunne implementeres både som vanlige rullegardinsmenyer og som ikonbaserte menyer, alt avhengig av hvilket plattform agenten kjører på.

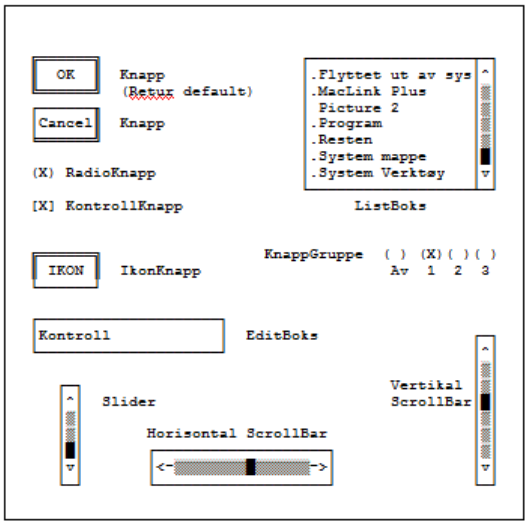
7.7.7. Kontrollklasser

Denne gruppen klasser benyttes som delelementer i vinduer. Det er ting som *ScrollBarer, Slider, EditBoks, ListBoks* og diverse varianter av *Knapper.* Ved å lage egne klasser basert på disse kan applikasjonsutvikleren lage de mest vanlige elementene i en brukerdialog. En rekke av de forhåndsdefinerte *DialogVinduene* vil inneholde slike *Kontroll* objekter, men utvikleren kan benytte disse for å lage egen og mer spesialiserte måter å kommunisere med brukeren på.



Figur 63 - Forskjellige elementer i en applikasjon. Samme funksjon, men ulikt utseende.

De fleste av disse er velkjent for dem som har benyttet en Macintosh, Presentation Manager under OS/2 eller Windows 3.0. I X-Windows programmer er det noe mer tilfeldig hvordan de hvordan slike er implementert. Programmer som er skrevet for DEC-Windows, OSF/Motif eller Open Look følger de retningslinjer som er gitt i *"guidelines"* for den enkelte *"Window-Manager"*.



Figur 64 - Kontroll klasser i skjermbasert MS-DOS system

Listbokser er en funksjon som lar brukeren velge mellom et variabelt antall muligheter, ved å presentere en liste over de mulighetene som man kan velge mellom. På en Macintosh er det for eksempel vanlig å liste filer man kan åpne i et program på den måten. Andre ting som kan la seg liste slik, kan være personer i en avdeling, produkter på lager, leverandører, maskiner med en gitt tjeneste, land og en masse annet som lar seg representere i form av lister. Brukeren må kunne bla opp og ned og kunne velge en eller flere av mulighetene som listes opp. I vindussystemer er dette implementert temmelig likt. I skjermorienterte systemer kan tilsvarende lister lages og gi omtrent samme funksjonalitet som i vindussystemer. I linjeorienterte systemer, er mulighetene stort sett begrenset til å la listen rulle over skjermen og gi brukeren en kode for et gitt valg. Samt muligheten til å fortsette listingen, eller avbryte den.

Hva brukeren velger bør kunne returneres til applikasjonen på en entydig måte. Så selv om listene kan ta seg forskjellig ut på skjermen, vil det ikke ha betydning for applikasjonen. Slike listbokser er viktige for eksempel i database applikasjoner og kan også fungere på samme måte som menyer. På mange måter er en slik listboks ikke noe annet enn en meny med variabelt og dynamisk innhold.

IkonKnapp er der for å kunne lage knapper som benytter ikoner istedet for tekst. Disse kan også benyttes til å lage ikonbaserte menyer og lignende for å lage brukergrensesnitt som kanskje er mer intuitive enn bare rullegardinsmenyer.

7.8. Nettverksklasser

Dette er funksjoner som ikke finnes i vindussystemer, men i operativsystemet. For å forenkle utviklingen av applikasjoner er det viktig at de inngår som objekter på samme måte som objekter i brukergrensesnittet. *Nettverk* benyttes til å kommunisere med informasjonsforvalteren. Den inneholder metodene for å åpne og lukke forbindelsen til ulike noder i nettverket, basert på de protokoller som agenten og det underliggende operativsystemet støtter. Det inneholder også funksjoner for å sende og motta meldinger. På samme måte som *Vindu* klassen inneholder *Nettverk* håndteringen av alle hendelser i forbindelse med nettet og er en abstraksjon av subklassene.

Subklassen *Forvalter* instansieres med de opplysninger som gjelder for informasjonforvaltere med et objekt for hver forvalter. Det er denne klassen som gjør applikasjonen i stand til å komme i kontakt med sine data på en annen maskin.

BlokkData er for å hente og sende hele filer og lignende. Denne klassen benyttes altså til å overføre større datamengder. Hvilken måte dette gjøres på kan variere fra applikasjon til applikasjon. Forvalterobjektet kan for eksempel settes opp til å benytte FTP eller FTAM for overføring. I en annen applikasjon kan det være en elektronisk melding som skal sendes til en SMTP tjener, eller hentes ved hjelp av POP. Det kan også være en spesialisert informasjons-forvalter som tar i mot data. Data til slike er da kodet i selve dataene og det er bare en "rå" overføring som gjøres på dette nivået. Et eksempel på dette kan være et arkivsystem som tar i mot artikler, brev, illustrasjoner og lignende. *BlokkData* objektet sørger bare for å overføre dataene, mens kodingen av hvor det skal lagres, med hvilke stikkord og hvem som eventuelt skal kunne oppdatere det, ligger kodet i dataene som overføres.

FeltData klassen benyttes til mer spesialisert kommunikasjon. Applikasjonen instansierer de objektene den trenger for å utføre ulike operasjoner mot informasjonsforvalteren. Hvilke meldinger de to partene kan sende seg i mellom og hvordan disse er kodet, er fritt opp til hver utvikleren å avgjøre. Fortrinnsvis bør man benytte anerkjente metoder for denne kommunikasjonen med koding i for eksempel ASN.1, men dette avhenger av hva applikasjonen skal benyttes til. I en database applikasjon vil det være svært viktig at tekst og tall kommer over slik det ble sendt. En mulig måte å gjøre dette på er for applikasjonsutvikleren å lage egne subklasser som bringer koblingen mellom det som skjer på skjermen og de data applikasjonen da har behov for opp på et høyere nivå.

Katalog klassen inneholder de metodene som trengs for kommunikasjon mot DUA'er eller selv å fungere som en DUA (se avsnitt 0). Klassen *ListApplikasjoner* er den som benyttes for å finne fram til hvilke applikasjoner som er tilgjengelig og hvordan de kan lastes over til applikasjonsagenten. *KodeKontroll* har de metodene som skal til for å overføre ny applikasjons-kode og installere denne på den lokale maskinen. Denne klassen inneholder også metoder for versjonskontroll og kontroll av hva den nyankomne applikasjonen kan gjøre av skadelige ting.

Versjonskontrollen er viktig for å kunne holde definisjonen av applikasjonen oppdatert på agentene. Det er viktig med en slik versjonskontroll hvis grensesnittet mellom applikasjon og informasjonsforvalter forandres, slik at måten data utveksles på endres. For eksempel ved at nye

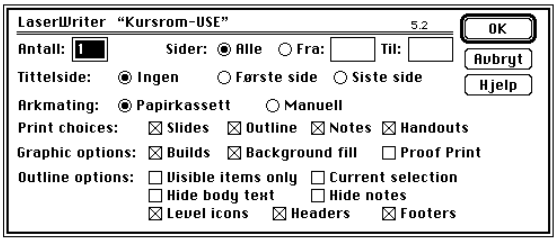
data elementer kommer til, gamle endres eller fjernes. Tilsvarende kan det tenkes at feil i en tidligere utgave er såpass alvorlig at det er påkrevd med en ny versjon. Eller kanskje har applikasjonen kommet i en ny og forbedret versjon med økt funksjonalitet. Hvis mange maskiner rundt om i verden benytter en gitt applikasjon er det ikke helt trivielt å informere alle brukerne om at de trenger en ny versjon.

Klassen for *FeilRapport* er for at applikasjonen og/eller brukeren skal kunne sende tilbake meldinger til utvikler eller systemansvarlig for informasjonsforvalteren hvis det oppstår feil under bruk av applikasjonen. Denne benytter katalogen for å sende meldingen på den måten de ansvarlige foretrekker.

Klassen *Kryptering* er metodene å sende data kryptert mellom bruker og informasjonsforvalter, eller mellom agent og tjener maskin. Når meldingene på nettet er kryptert må metodene i de andre nettverksklassene sende sine utgående meldinger innom Kryptering for å få dem kryptert og de innkommende dekryptert.

7.9. Skriver

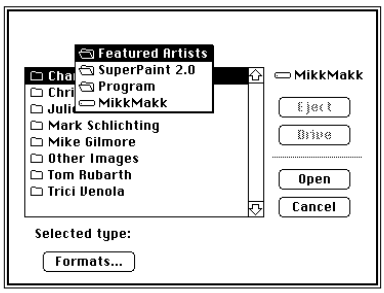
Skriverstyring er ofte dypt begravet i operativsystemet. Hensikten med å lage det som en egen klasse er at applikasjonene skal være uavhengig av hva slags skriver som finnes og hvordan den håndteres. I agenten kan det godt implementeres som en egen applikasjon etter den distribuerte applikasjonsmodellen som sender utskriften til en tjenermaskin som tar hånd om utskrifter. Det kan også være en lokal skriver som henger på maskinen, eller en nettverksskriver som benytter andre, spesialiserte metoder for utskrift. Poenget er at applikasjonen har et skriverobjekt den kan sende sine data til.



Figur 64 - Utskriftsdialog på Macintosh

7.10. Diskklasser

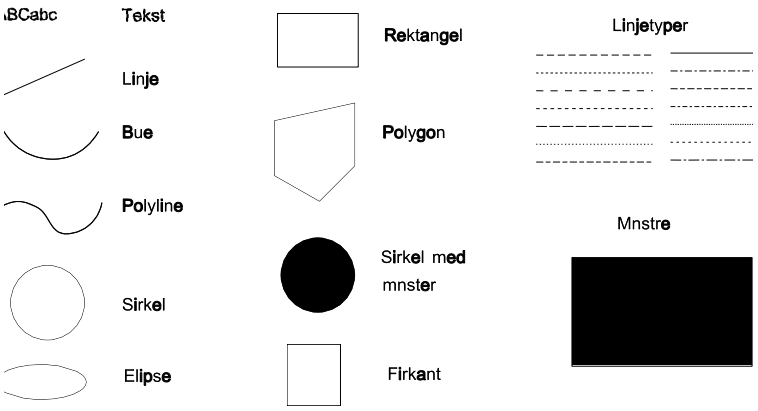
Normalt er håndteringen av filer etc. innbakt i applikasjonen som har kunnskap om hva slags filsystem som finnes på den plattformen den er skrevet for. Den benytter operativsystemkall for å utføre de ønskede operasjoner, men er som oftest ikke uberørt av større endringer i filsystemet. For applikasjoner som skal kunne fungere under ulike operativsystem på mange maskinplattformer, må filsystem og organisering av disker skjules fullstendig. Det er agenten selv som må sørge for at de meldinger som sendes til objektene for *Fil* og *filkatalog* blir utført korrekt. Typiske metoder for disse vil være å åpne, lukke, lese og skrive filer på lokale disker, samt liste hvilke filer som finnes i kataloger og lignende. Dette blir altså et slags virtuelt filsystem på over det virkelige filsystemet. Klassene må inneholde de operasjoner som er vanlige i filsystemer og agenten må emulere dem som eventuelt ikke finnes i det lokale filsystemet. De prinsippene som er nedfelt i OSI-standarden for FTAM kan komme til anvendelse her. Kanskje bør FTAM benyttes for overføring av filer mellom maskiner.



Figur 65 - Macintosh FilDialog

7.11. Grafikk

Dette er de grafikkobjektene som støttes i agenten og det tilhørende system for å vise grafikk på skjermen. Der hvor grafikksystemet ikke selv støtter visse typer grafikkelementer, må agenten selv tegne dem ved hjelp av andre grafikkelementer. For eksempel er det ikke alle grafikksystemer som støtter *Polygon*, men dette kan settes sammen av flere *Linjer*. De grafikkelementene som bør støttes er: *Linje, PolyLine, Rektangel, Polygon, Sirkel, Ellipse, Bue, Punkt, Bitmap* og *Tekst*.



Figur 66 - Eksempel på grafikk elementer

7.12. FyllMønstre

Også fyllmønstre er som oftest begravet i vindus og/eller grafikk-systemet. For at applikasjonene skal kunne benytte slike til å fylle grafikkelementer med, eller ha dem som bakgrunnsmønstre og lignende, må de være tilgjengelig i form av klasser. Applikasjonene må kunne spørre hvilke fyllmønstre som finnes og kunne lage *IkonKnapper* med hvert mønster. Hvis mønstre ikke støttes av grafikksystemet direkte, bør agenten forsøke å implementere et minimum. Fyllmønstre er stort sett bare aktuelt på systemer som støtter grafikk.

7.13. Tekst

Vindussystemer har som regel mulighet for ulike skrifttyper og skriftsnitt, samt å ha forskjellig farge og størrelse på tekst. Applikasjonsagenten må kunne håndtere slike tekstattributter, selv om den aktuelle maskinplattformen ikke er istand til å vise den på skjermen. Slikt maskinutstyr vil gi redusert funksjonalitet, men som regel er det viktigste hva som står i teksten og ikke om det er skrevet i grønn 12 punkts times eller kursiv 18 punkts Helvetica.

I praksis benytter de fleste vindussystemer en fast skrifttype for systemmeldinger etc., så det blir i tekstbehandlere og -lesere at tekst kan presenteres forskjellig avhengig av mulighetene maskinvaren har.

7.14. Tegnsett

Normalt vil et programmer forholde seg til det tegnsettet som finnes på den lokale maskinen. Problemet er at disse ikke er like for en Macintosh, MS-DOS maskin og en UNIX-arbeidsstasjon. De 128 første tegnene er som regel basert på US ASCII (ISO 646), men de siste 128 tegnene ligner lite på hverandre. Skal applikasjoner kunne fungere uavhengig av dette er det nødvendig med en egen klasse for å identifisere og oversette mellom tegnsett. Applikasjonen kan i sin kommunikasjon med sine informasjonsforvaltere muligens basere seg på ISO DIS 10646 (ny standard som inneholder muligheter for 1,2,3 eller 4 bytes tegn og støtter alle levende skriftspråk). Men for å utveksle data med lokale applikasjoner, må det finnes metoder for å oversette. Siden applikasjonen ikke skal trenge å vite hva slags tegnsett som benyttes lokalt, må agenten utføre det ved hjelp av denne klassen.

7.15. TekstUtseende

Denne klassen benyttes til å gi tekst spesielt utseende, slik som **Uthevet**, *Kursiv*, Understreket, Dobbel Understrekning, Skygge, Kontur, KAPITÉLER ol. De metodene agenten kjenner ligger her og det er agentens ansvar å sørge for at det kommer fram på skjermen så godt det lar seg gjør på den aktuelle plattformen.

7.16. TekstSnitt

TekstSnitt eller kanskje oftere omtalt som fonter er noe som har gitt brukerne nye muligheter til å lage profesjonelt utseende dokumenter. Det finnes en flere skoler innen formgivningen av tekst. For at applikasjonene skal kunne benytte ulike fonter, må det finnes standardisert navning og muligheter for å koble disse navnene med de fontene som finnes i vindusystemet. Klassen TekstSnitt inneholder metodene for å gjøre de ulike fontene tilgjengelig for applikasjonene. Hvis vindusystemet agenten benytter ikke har en gitt font installert, må den velge en font som er i nærheten og gi brukeren beskjed om at den gjør tilpasninger. Applikasjonen må kunne spørre agenten om hvilke tekstsnitt den kjenner til, slik at den for eksempel kan benytte dem i sine menyer ol. Ideelt sett burde en applikasjon være i stand til å innføre en ny font på agenten, hvis den ønsket. Dette kan gjøres ved at agenten kontakter en spesiell tjenermaskin som kjenner til det vindus- og grafikksystem som benyttes og har metodene for å installere en ny font på den maskinplattformen.

7.17. Farger

Antall farger og tilpassingen av disse på skjermen kan dessverre variere sterkt. På samme måte som for fonter, må det finnes mulighet for applikasjonen å spørre om hvilke farger som er tilgjengelig og be agenten tilpasse fargene, slik at resultatet blir så bra som mulig. Bilder i 256 farger vil lett bli temmelig meningsløse på et system med 4 eller 16 farger. Applikasjonen må da enten tilpasse grafikken til færre farger, eller la være å vise den. Farge på tekst vil ikke ha så stor betydning som farge på grafikk. Det må finnes metoder for brukeren å finstille fargene i agenten.

7.18. Ikoner

Ikoner er i praksis ikke noe annet enn bitmaps. Alle applikasjoner bør ha et eget ikon, slik at de kan vises sammen med navnet på applikasjonen i grafiske menysystemer. Tilsvarende bør de benytte egne eller standard ikoner for de datafiler og lignende de lagrer på lokale disker. Formatet bør være det samme som for bitmaps.

Agenten kan lagre et standard sett med ikoner for ulike formål for at applikasjonene skal slippe å ha disse kodet inn i seg, eller hente dem over nettet. Dette kan være ikoner som applikasjonene benytter i sine menyer, slik som mange tegneprogrammer gjør, eller det kan være ikoner som benyttes som illustrasjoner. Ikoner bør også kunne benyttes på samme måte som tekst. Dette ville kunne gjøre det mulig for en applikasjon og skrive japansk eller kinesisk tekst ved hjelp av ikoner som erstatning for de tegn som ikke støttes direkte i agentens vindus- og grafikksystem. Hvilke ikoner agenten har, må den kunne informere applikasjonen om og gjøre tilgjengelig for applikasjonen på en enkel måte. Tilsvarende bør applikasjonen kunne legge fra seg sine ikoner og dermed gjøre dem tilgjengelig i andre applikasjoner (forutsatt at de kjenner navnet). Man kan også tenke seg spesielle tjenermaskiner som inneholder store mengder ikoner og som agentene kan laste ned og lagre lokalt etter behov.

7.19. Kamera

Denne klassen er kun med i agenten hvis det finnes et kamera tilknyttet den lokale maskinen. Tilsvarende klasser må finnes for andre typer utstyr som skal tilkobles. Den må kunne ta i mot

meldinger om at applikasjonen ønsker å motta bilder og metoder for å informere om hvordan datastrømmen overføres og i hvilket format. Denne er sterkt knyttet til animasjonsvinduet. For at det ikke skal oppstå forsinkelser under overføringen kan det godt være nødvendig med en del triksing i agenten og muligens også i grafikksystemet. Poenget med å ha en egen klasse for kamera er at applikasjonen skal kunne gi meldinger om å starte og stoppe, sende bildene direkte over nettverket på en gitt protokoll eller lagre dem på en disk. Klassen er ikke ment å ta seg av alle aspekter med levende bilder på en datamaskin, men gi et standard grensesnitt som applikasjoner kan benytte på de maskiner der kamera er installert.

7.20. Lyd

Lyd er som regel ikke en standard del av vindusystemet, men de fleste maskintyper har en eller annen mulighet for å generere lyd. Kvaliteten kan imidlertid variere sterkt. Alt fra maskiner som knapt er i stand til å gi et anstendig *Beep*, til NeXT-maskinen som kan spille en symfoni med omtrent samme kvalitet som en CD-spiller.

De fleste maskiner har ikke mikrofon slik det er i dag. De som har det har som regel en mikrofon som er knyttet opp til spesiell maskin- og programvare og gjerne med egne, ikke standardiserte digitale lydformater.

Lyd må kunne la seg generalisere, slik at den kan benyttes på flest mulig plattformer med et så godt resultat som mulig. Det en bruker snakker inn på en Macintosh, må kunne spilles av på en IBM-kompatibel PC som har passende utstyr installert.
I modellen, kan lyd betraktes omtrent på samme måte som et bilde, representert ved en bitstrøm. Det bør være muligheter for å betrakte hvert lydobjekt som bestående av flere instanser av lydklassen hvor hvert lydobjekt kan aksesseres uavhengig eller samlet som en gruppe. Dette blir som på en CD-plate hvor hvert spor kan adresseres uavhengig og i vilkårlig rekkefølge. Lyder kan lagres lokalt hvis ønskelig for å spare tid på gjentatte overføringer.

I tillegg bør det også være en eller flere tekstlige representasjoner av lyden assosiert med lyden. Dette fordi det da blir langt enklere å skrive applikasjoner som benytter lyd, men som ikke er helt avhengig av en lydgenerator i terminalen for at applikasjonen skal gi mening. En annen anvendelse av en slik tekstlig representasjon vil kunne være språkopplæring. For eksempel kan man i et spanskkurs ha en setning uttalt på spansk som brukeren hører samtidig som vedkommende ser den spanske teksten og/eller den norske oversettelsen. For lyder som ikke er tale kan det enten stå hva lyden er eller en etterlignende måte å bokstavere den på. For eksempel "Boooouuhh" eller "regn på blikktak".

En lydklasse bør ihvertfall være definert med:

NAVN	Navnet på objektet
LENGDE	Lengden på lyden i sekunder (eller millisekunder?)
TYPE	Hva slags format lyden er på
TEKST	Tekstlige representasjoner av lyden på et eller flere språk
BITSTRØM	Bitmønsteret som utgjør signalene til lydgenerator/konverteringsprogram eller en peker til et annet lydobjekt.
NESTELYD	Neste lyd i sekvensen

Selve lydmekanismen bør lages slik at den kan generaliseres for bruk i flere applikasjoner. Hvis en klient alt har fått lastet over en lyd, er det unødig å overføre den nok en gang og dobbellagre den. Dette kan gjøres ved at lyd oppfattes som eksterne objekter til applikasjonene som kan kalles fra applikasjonene på navn. Ved å bygge opp et tilstrekkelig bibliotek med lyder på en klient kan en

applikasjon benytte disse til å snakke. Hvis disse generaliseres på navn vil det til og med være mulig å velge hvilken stemme man vil ha på sin datamaskin.

Jeg vil ikke ta stilling til hvilket format som er gunstigst å benytte for representasjon av lyd, men det må ihvertfall finnes et format som er grunnleggende felles. Det burde være mulig å representere lyd på en smartere måte enn bare rent å digitalisere den. Et format som sa mer om betoning, styrke, trykk og lignende burde kunne konstrueres. Det som er viktig er at det finnes muligheter for å identifisere hvilket format en lyd er lagret på, slik at applikasjonen kan tolke den. Ulike formater av lyd kan defineres som egne objekter.

7.21. Synkronisering

For å kunne gi multimediamuligheter over nettverk, må applikasjonsagentene ha muligheter for å synkronisere bilde og lyd. Det er flere muligheter for å gjøre dette. En kan være at agenten kan reservere en fast kanal i nettverk hvor det er mulig. Applikasjonen kan så disponere denne kanalen alene mens multimedia overføringen pågår. Andre metoder er å hente over lyd og bilde til den lokale maskinen før det spilles av. Uansett må det finnes mekanismer for å merke hvordan lyd og bilder henger sammen.

7.22. Annet

Ny metoder å kommunisere på kan bli aktuelt i framtiden. For eksempel tilkobling av annet utstyr. Kanskje kan det bli aktuelt å koble til forskjellig type husholdningsutstyr og la applikasjoner styre disse? Muligens kunne det være interessant å koble tyveri- og brannvarsling til en datamaskin og la denne benytte den distribuerte applikasjonsmodellen for å varsle hvis noe skjer? For å få til slike ting må det være mulig og fortrinnsvis også enkelt å kunne legge til nye måter å ta imot input og gi output til enheter som enda ikke er definert. Skal det bli mulig må det la seg generalisere slik at ulike typer utstyr kan kommunisere. Om brannvakta har et stor-maskin basert anlegg, må brannvarslerne i et bygg likevel kunne være tilkoblet en gammel Macintosh. Brannvakta må kunne få beskjeden og kunne signalisere tilbake til maskinen som styrer det lokale anlegget, at meldingen er oppfattet og kanskje også starte sprinkler-anlegget og automatisk lukke alle branndører.

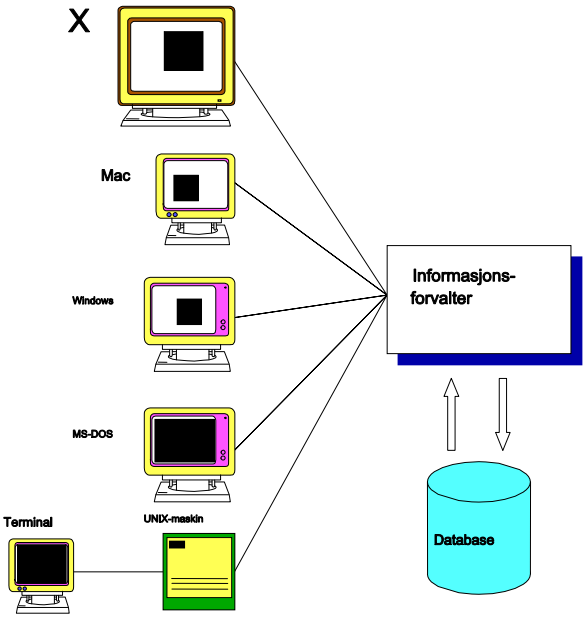
Andre ting som kan være aktuelt å koble til er telefonen, fjernstyrte døråpnere, lesere for elektroniske identitetskort, måleinstrumenter. Det er egentlig ikke grenser for hva slags utstyr som kan tenkes knyttet til. Det som kreves er klasser i applikasjonsagenten som gjør det mulig for applikasjoner å styre det utstyret som er tilkoblet. Hvordan koblingen mellom klassen og selve utstyret foregår, er det opp til leverandøren av utstyret å avgjøre. Det blir på samme måte som man i dag får med drivere for ulike programmer sammen med nye skjermer og skrivere man kjøper.

8. Anvendelse av den distribuerte applikasjonsmodellen

Jeg vil i dette kapittel ta for meg noen praktiske eksempler på hvordan jeg tenker meg at applikasjoner i den distribuerte applikasjonsmodellen kan se ut og fungere. Videre vil jeg trekke fram noen aspekter ved modellen som er viktige å få med. Til å beskrive vil jeg benytte en SIMULA aktig syntaks fordi denne er rimelig leser vennlig og forholdsvis godt kjent. Jeg vil ikke ta stilling til hva slags språk en implementasjon av den distribuerte applikasjonsmodellen bør basere seg på. Det flere av de objektorienterte språkene som kan være aktuelle, men det kan også hende at det er nødvendig å utvikle et nytt språk.

8.1. Database applikasjon

Som et eksempel har jeg valgt en enkel database applikasjon for å registrere og vise bestillinger. Selve databasen er tenkt lagt til en UNIX-maskin. Applikasjonen skal kunne kjøre på applikasjonsagenter på Macintosh, på MS-DOS maskiner (med og uten Windows), under X-Windows, på UNIX-maskiner fra en VT100 terminal. Hvordan databasen er organisert spiller ingen rolle for applikasjonen. Den forholder seg til det grensesnittet som informasjonsforvalteren for denne bestillingsapplikasjonen tilbyr. Denne applikasjonen er kun ment å illustrere sider av den distribuerte applikasjonsmodellen og er ikke ment å beskrive hvordan man kan lage en god applikasjon for et bestillingssystem. Det er derfor utelatt og lagt til ting som kanskje ikke ville vært med i en virkelig applikasjon av denne typen.



Figur 67 - Samme applikasjon fra mange maskinplattformer

For å realisere en applikasjon av denne typen trengs applikasjonskoden, applikasjonsagenter på de maskinplattformer som det er aktuelt å benytte, selve databasen,
samt en informasjonsforvalter som kan aksessere databasen og benytter en datanett-protokoll som er kjent av applikasjonen. Applikasjonsagentene kan bare kommunisere med informasjonsforvalteren ved hjelp av de tjenesteprimitivene denne kjenner og tilsvarende kan den bare vise fram på skjermen klasser som er kjent i applikasjonsagenten.

8.2. Tjeneste primitiver

Denne bestillingsapplikasjonen har få tjeneste primitiver mellom informasjonsforvalteren og applikasjonen på agenten. Det er et primitiv for å åpne forbindelsen til informasjonsforvalteren og en tilsvarende for å stenge den. For å hente data finnes det to primitiver. Et for å hente over en liste over de bestillingsnumre som er registrert og et for å hente over data for et gitt bestillingsnummer. Hvis informasjonsforvalteren har de dataene som agenten ber om, vil den sende dataene over i et kodet format som applikasjonen kan pakke ut. Hvis den ikke har dem, vil den sende over en egen pakke (*BA_NODATA*). For oppdatering av data finnes to primitiver. Ett for å endre data og et for å slette data. Informasjonsforvalteren kan svare på slike meldinger med at det er utført (*BA_CONFIRM*), eller at det ikke lot seg gjøre (*BA_NOCONFIRM*). I en virkelig applikasjon ville det trolig også finnes et primitiv for å opprette nye bestillingsnumre, men det har jeg ikke tatt med her.

Tjeneste primitiv	Beskrivelse
BA_CONNECT_DATABASE	Åpner forbindelsen til informasjonsforvalteren
BA_CLOSE_DATABASE	Stenger forbindelsen til informasjonsforvalteren
BA_GET_BESTNR	Ber om liste over bestilingsnumre
BA_GET_BESTDATA	Henter data for en bestilling
BA_DATA	Data fra informasjonsforvalter
BA_NODATA	Informasjonsforvalter har ikke data
BA_DELETE_BESTNR	Sletter en bestilling
BA_MODIFY_BESTDATA	Endrer en bestilling
BA_CONFIRM	Endring i databasen bekreftet
BA_NOCONFIRM	Endring i databasen avvist

Tjeneste primitiver for kommunikasjon mellom informasjonsforvalter og bestillingsapplikasjon

8.3. Applikasjonskoden

Eksempelet på koden er strukturert slik det er vanlig i SIMULA. De tingene som ikke lar seg beskrive enkelt i vanlig SIMULA, har beskrevet med egne konstruksjoner.

Hele applikasjonen defineres som en klasse. Muligens bør den defineres som et objekt direkte og ikke beskrives som en klassen i det hele tatt. Hvordan dette løses i en eventuell implemtasjon av den distribuerte applikasjonsmodellen, vil avhenge av det språk man velger å benytte for å beskrive applikasjonene.

Variabel definisjonene må være mer utfyllende enn vanlig SIMULA. Fordi applikasjonen skal kunne fungere på mange ulike maskinplattformer må variabeltypene kunne defineres mer fleksibelt, slik at det ikke er tvil om hva som menes. TEXT vil f.eks. ikke være det samme på en UNIX-maskin, en Macintosh og en MS-DOS PC. De har tre ulike tegnsett og applikasjonen må gi klar beskjed om hvilket tegnsett som er i bruk.

Strukturen i applikasjonen for bestillinger kan se slik ut:

```
Applikasjon CLASS Bestilling;
BEGIN
{Variabel definisjoner}

  {Definisjoner av Data i meldingene mellom klient og tjener}
  RECORD Best_Data.....
  RECORD Best_Liste.....
  RECORD Best_Nr.....

  {Klasse for kommunikasjon mellom Informasjonsforvalter og applikasjon}
  FeltData CLASS BestInfo.....

  {Vinduet for applikasjonen}
  ArbeidsVindu CLASS BestillingsVindu.....

{Versjonskontroll}
{Initiering}
{Start applikasjonen}
END
```

RECORD er en egen konstruksjon for å beskrive de meldingene som går mellom informasjonsforvalteren og applikasjonen. Dataene beskrevet i *RECORD* kan benyttes av hele applikasjonen. *FeltData* klassen i agenten får en subklasse for denne applikasjonen. Det er dette objektet som står i forbindelse med Informasjonsforvalteren. *BestillingsVindu* er en subklasse av *ArbeidsVindu* og er det grensesnittet brukeren ser. Instruksjoner for kontroll av versjonsnummer for applikasjonen, initering og oppstart ligger i slutten av applikasjonen slik det er vanlig i SIMULA.

Beskrivelsen av innholdet i datapakkene kan se slik ut:

```
  RECORD Best_Data;
  BEGIN
    Bnr   : TYPE=INTEGER,LENGTH=6;
    Dato : TYPE=DATE;
    Lev   : TYPE=TEXT(ISO8859-1),LENGTH=40;
    Adr   : TYPE=TEXT(ISO8859-1),LENGTH=40,LINES=3;
    Bdata: TYPE=MULTIPLE_RECORDS(BestListe);
  END;

  RECORD Best_Liste;
  BEGIN
    VareNr    : TYPE=INTEGER,LENGTH=6;
    VareNavn : TYPE=TEXT(ISO8859-1),LENGTH=30;
    Antall    : TYPE=INTEGER,LENGTH=6;
    Pris       : TYPE=REAL,LENGTH=7,2,POSTFIX=Kr;
    Total      : TYPE=REAL;LENGTH=7,2,POSTFIX=Kr;
  END;

  RECORD Best_Nr;
  BEGIN
```

```
    MULTIPLE
    BEGIN
      Bnr   : TYPE=INTEGER,LENGTH=6;
      Lev   : TYPE=TEXT(ISO8859-1),LENGTH=40;
    END;
  END;
```

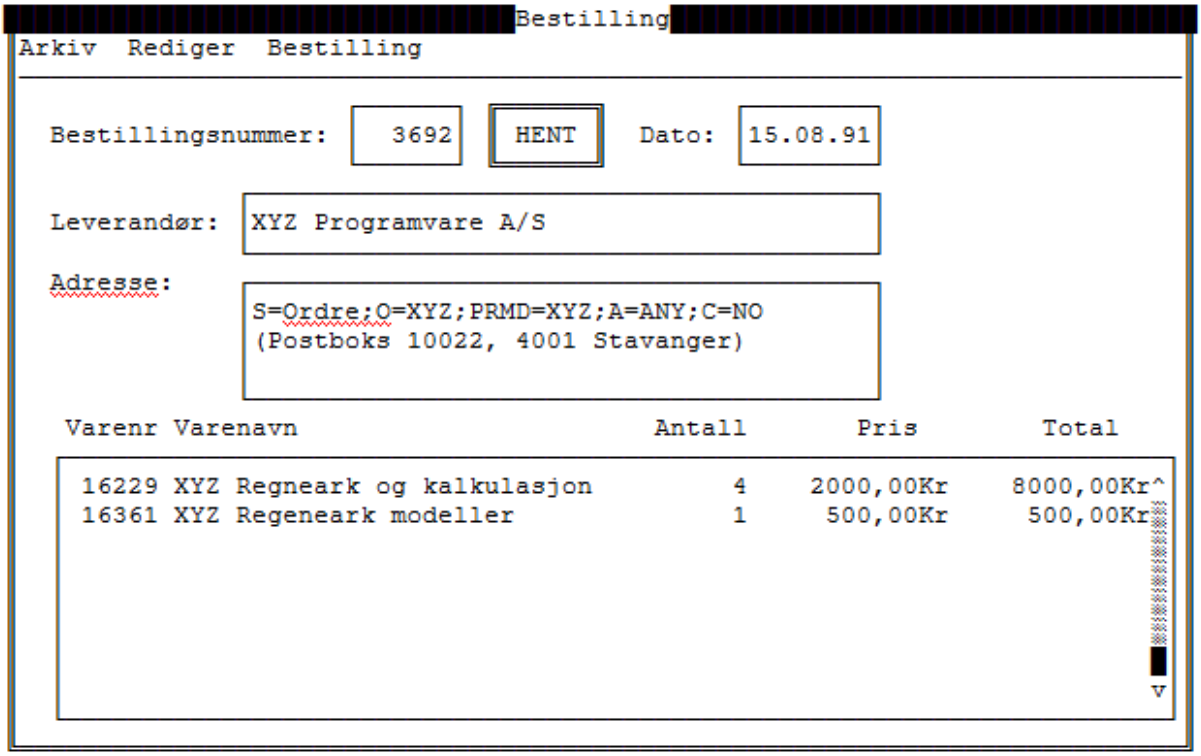
Det er to type av datapakker som kommer fra Informasjonsforvalteren til applikasjonsagenten. Den ene er *Best_Data*. Denne inneholder; Bestillingsnummer (*Bnr*), dato for bestillingen (*Dato*), Leverandørnavn (*Lev*), Leverandørens adresse (*Adr*), og en eller flere spesifikasjoner av varene som er bestilt fra leverandøren (*Bdata*). Bdata er selv en *RECORD* som er definert som *MULTIPLE* i *Best_Data*. Den består av leverandørens varenummer (*VareNr*), navnet på varen (*VareNavn*), antallet som er bestilt (*Antall*), stykkprisen på varen (*Pris*) og totalprisen for denne varen (*Total*). Det kan sammenlignes med en pekerliste med et hode (*Best_Data*) med en eller flere objekter i kjede (*Best_Liste*). Hvordan det formelt beskrives i programmeringsspråket er ikke så viktig, bare det er mulig å beskrive kontruksjoner som dette. I SIMULA kunne de vært beskrevet som klasser, men siden de egentlig er datapakker uten egne metoder finner jeg det mer tiltalende å ha en egen konstruksjon for å beskrive data som formidles mellom maskiner. Tanken er at data beskrevet i *RECORD* også skal kunne benyttes direkte i vinduet. *Best_Nr* er enklere. Den består av et eller flere bestillingsnummer (*Bnr*) og tilhørende leverandørnavn (*Lev*).

Koblingen mellom Informasjonforvalteren og applikasjonen i agenten gjøre i *BestInfo* objektet som en subklasse av *FeltData* klassen. Dette inneholder metoder for de tjeneste primitivene som er definert.

```
  FeltData CLASS BestInfo;
    Result:Pointer;
  BEGIN
    Protokoll:=OSI
    Server:="MaskinX"

    PROCEDURE Connect;
    BEGIN
      Result:-SendNet(BA_CONNECT_DATABASE);
      IF NOT {Result gir CONNECTED} THEN
      BEGIN
        NEW Feil("Informasjonsforvalter på MaskinX ikke tilgjengelig");
        {Send et event for å prøve igjen, eller terminere applikasjonen}
      END;
    END;

    PROCEDURE Close;
    BEGIN
      Result:-SendNet(BA_CLOSE_DATABASE);
    END;

    PROCEDURE Get_BNR;
    BEGIN
      Result:-SendNet(BA_GET_BESTNR);
      IF Result {gir BA_NODATA} THEN
```

```
      {Ingen bestillinger registrert}
    ELSE
      {Returner listen over bestillingsnumre i Best_Nr record}
  END;

  PROCEDURE Get_BData;
  BEGIN
    Result:-SendNet(BA_GET_BESTDATA,Param=BestNr);
    IF Result {gir BA_NODATA} THEN
      {Bestillingsnummer finnes ikke}
    ELSE
      {Returner bestillingsdata i Best_Data record}
  END;

  PROCEDURE Del_BData(BestNr);
  BEGIN
    {Sett opp en DialogBoks for å spørre om brukeren vil slette bestillingen}
    Result:-SendNet(BA_DELETE_BESTNR,Param=BestNr,ThisMachine,UserID);
    IF Result {gir BA_NOCONFIRM} THEN
      {Denne maskinen eller brukeren kan ikke slette bestillingen}
    ELSE
      {bestillingen slettet}
  END;

  PROCEDURE MOD_BDATA;
  BEGIN
    {Sett opp en DialogBoks for å spørre om brukeren vil endre data}
    Result:-SendNet(BA_MODIFY_BESTDATA,Param=OldBestNr,NewBestNr,
                    ThisMachine,UserID);
    IF Result {gir BA_NOCONFIRM} THEN
      {Denne maskinen eller brukeren kan ikke oppdatere bestillingsdata}
    ELSE
      {Oppdateringen gikk greit}
  END;

  Connect;
END; {BestInfo}
```

Vinduet brukeren ser kan se slik som vist i figuren nedenfor. En applikasjon som ikke krever mer av maskinutstyret enn denne applikasjonen, bør kunne benyttes på de fleste typer maskinutstyr. Legg merke til leverandør adressen er et felt som spenner over tre linjer. Adressen kan like gjerne være en elektronisk som en vanlig postadresse.



Figur 68 - Eksempel på Bestillingsapplikassjon i MS-DOS

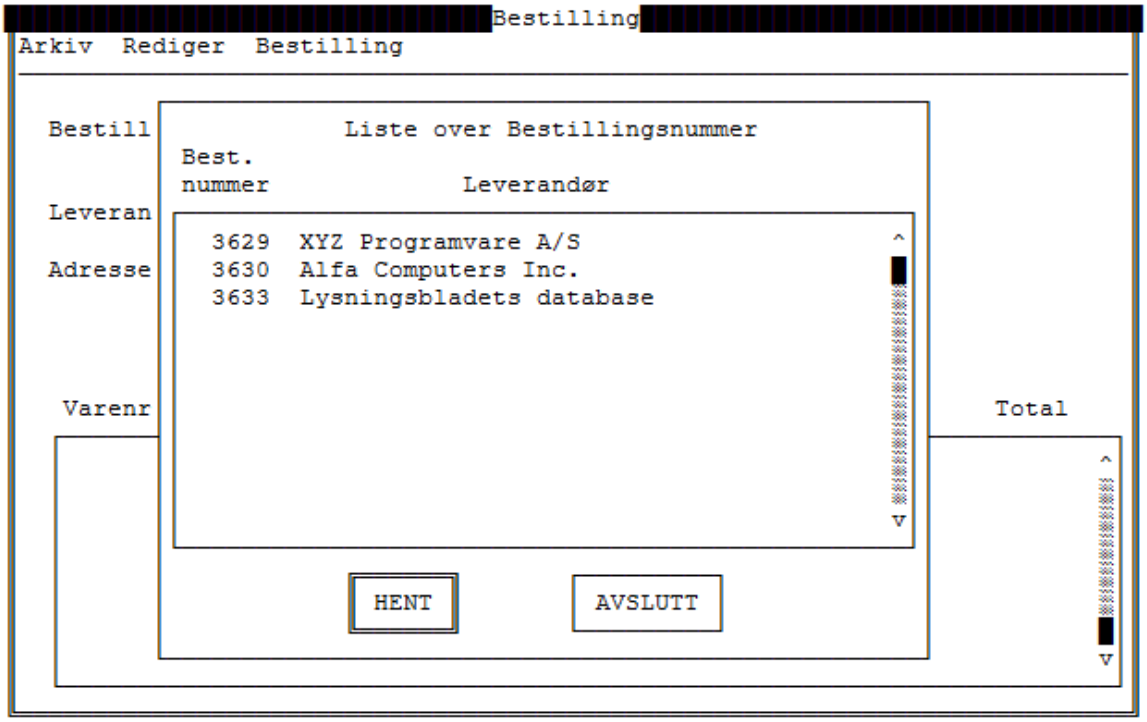
BestillingsVindu er en subklasse av ArbeidsVindu. Det er satt opp 80 tegn bredt x 24 linjer langt. Standard skriftsnitt er satt til Courier som ligner på den som vanligvis benyttes på skjermer og som har den fordelene at alle bokstaver tar like stor plass. Hvis applikasjonen benyttes på en fargeskjerm vil bakgrunnsfargen settes til blå og teksten vil komme i gult. Både tekstsnitt og farger er egne objekter som kan manipuleres. Prosedyren *InitVindu* setter opp hvordan skjermbildet skal se ut. Klassen ArbeidsVindu innholder de metodene som benyttes for å skrive ut tekst (*SetText, SetWindow, PutText og PutMenu*). Feltene hvor brukeren kan taste inn data hentes fra *Kontroll* klassen (*EditBoks, ListBoks og Knapp*).

```
ArbeidsVindu CLASS BestillingsVindu(Xsize=80,Ysize=24,BgColor=Blue);
{X og Y angir størrelse, Blue er en peker til fargen i klassen Farger}
  BnrFelt,DatoFelt,LevFelt,AdrFelt,BdataListe,HentKnapp : Pointer;

BEGIN

  PROCEDURE InitVindu;
  {InitVindu tegner opp skjermbildet}
  BEGIN
    SetText(FAMILY=Courier,SIZE=10,COLOR=yellow); {Endre standard tekst}
    SetWindow(COLOR=Blue); {Endre Bakgrunnsfarge, hvis mulig}

    PutText(x=4,y=5,TEXT="Bestillingsnummer:");
    PutText(x=42,y=5,TEXT="Dato:");
    PutText(x=4,y=8,TEXT="Leverandør:");
    PutText(x=4,y=10,TEXT="Adresse:");
```

```
    BnrFelt:-New EditBoks(x=24,y=5,DATA=Best_Data.Bnr);
    DatoFelt:-New EditBoks(x=49,y=5,DATA=Best_Data.Dato);
    LevFelt:-New EditBoks(x=17,y=8,DATA=Best_Data.Lev);
    AdrFelt:-New EditBoks(x=17,y=10,DATA=Best_Data.Adr);
    BdataListe:-New Listboks(x=5,y=17,LINES=8,DATA=Best_Data.Bdata,
                                SCROLLBAR=YES)
    HentKnapp:-NEW Knapp(x=33,y=5,TEXT=Hent,DEFAULT=YES);
    PutMenu(Arkiv,"Slutt");
    PutMenu(Bestilling,"List","Hent","Endre","Slett");
  END; {InitVindu}
```

Prosedyren *HentData* benyttes for å be om data fra informasjonsforvalteren via objektet som står i kontakt med den (*BestInfo*). Forbindelsen kobles ikke opp før brukeren har tastet inn bestillingsnummer. Dette gjør at vi kan koble ned etter et at det ikke har vært aktivitet på en stund og likevel få satt opp forbindelsen igjen når brukeren vil se på et nytt bestillingsnummer.Dataene legges i *Best_Data* og de metodene som oppdaterer skjermen kan hente sine data derfra.

```
  PROCEDURE HentData;
  BEGIN
    IF BestNR <= 0 THEN
    BEGIN
      NEW Feil("Oppgi Bestillingsnummer")
      Return:-nil;
    END
    ELSE
    BEGIN
      IF InfoTjener:-nil THEN InfoTjener:-NEW BestInfo;
      InfoTjener.Get_BData(BestNr);
    END;
  END;
```



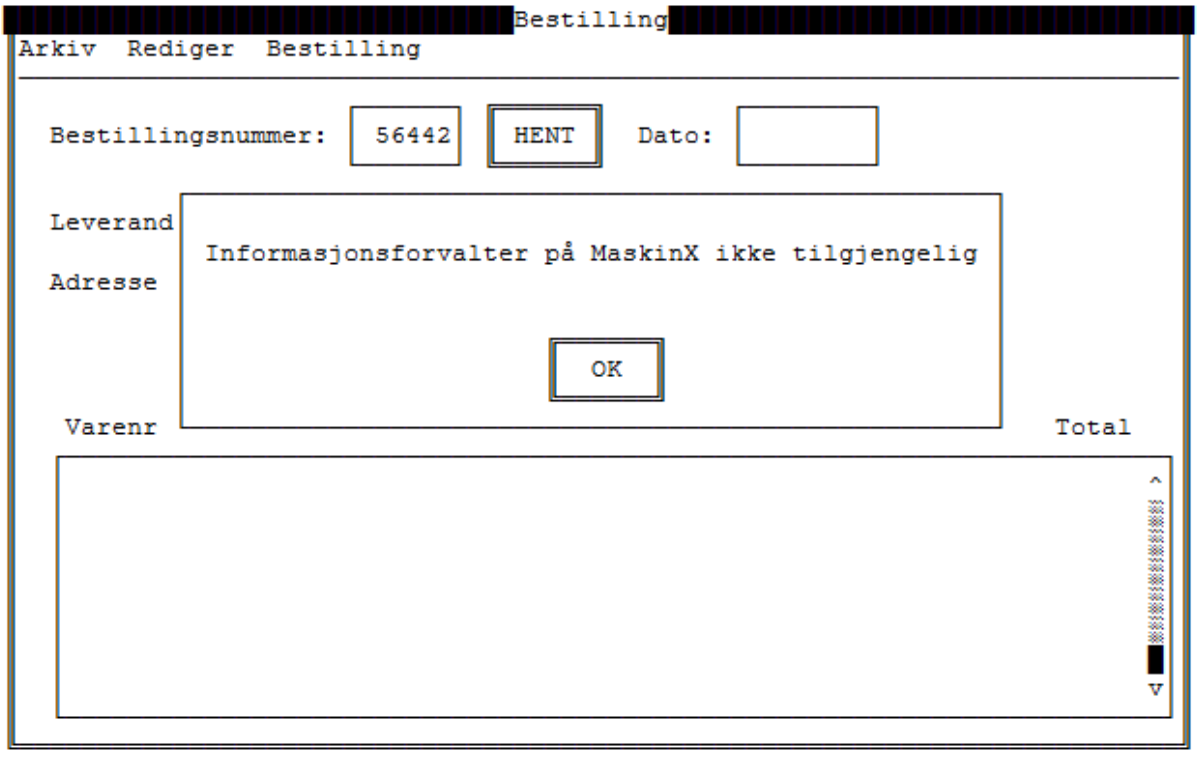
Figur 69 - Eksempel på ListBoks

```
ArbeidsVindu CLASS BestNrVindu(Xsize=52,Ysize=21,Xpos=11,Ypost=4,BgColor=Red);
  BEGIN
    {Definisjon av vindu med listboks som viser bestillingsnummer og}
    {leverandørnavn fra Best_Nr record og som lar brukeren velge et}
    {bestillingsnummer fra listen. Hvis brukeren velger Avslutt,}
    {returneres 0 som valgt bestillingsnummer}
  END

  PROCEDURE HentBnrl;
  BEGIN
    BestNr:=NEW BestNrVindu;
  END;

  Procedure Draw;
  BEGIN
    {Oppdater skjermen med de data vi har}
  END; {Draw}

  PROCEDURE Quit;
  BEGIN
    InfoTjener.Close;     {Steng forbindelsen til Informasjonsforvalter}
    {Send et event til systemet for å si at denne applikasjonen er avsluttet}
  END
```

De fleste moderne vindussystemer er styrt av hendelser (*events*) og for at applikasjonsagenten skal kunne fungere som et program under disse vindussystemene må også applikasjonene være drevet på samme måte. Prosedyren Event er selve motoren i applikasjonen. Det er her det registreres hva

brukeren gjør. Denne applikasjonen tester bare på et fåtall events som agenten får fra vindussystemet eller eget skjerstyringsdel. Legg merke til at *ReDraw* eventet kaller det samme som et *Draw* event. Et *Draw* event vil fra superklassen *ArbeidsVindu* kalle prosedyren *Draw*. *ReSize* Eventet er fullstendig koblet ut. Skjermbildet er 80*24 og brukeren kan dermed ikke endre på det.

```
PROCEDURE Event;
{Agenten får sine events fra vindussystemet}
BEGIN
  CASE OF events
    HentKnapp             : HentData;Draw;
    Meny(List)            : HentBnrl;Draw;
    Meny(Hent)            : HentData;Draw;
    Meny(Endre)           : InfoTjener.MOD_BDATA;
    Meny(Slett)           : InfoTjener.DEL_BDATA;
    Meny(Slutt)           : Quit;
    Changed(BnrFelt)      : BestNr:=BnrFelt.GetValue;
    Changed(DatoFelt)     : Best_Data.Dato:=DatoFelt.GetValue;
    Changed(LevFelt)      : Best_Data.Lev:=LevFelt.GetValue;
    Changed(AdrFelt)      : Best_Data.Adr:=AdrFelt.GetValue;
    Changed(BdataListe) : Best_Data.Bdata:=BdataListe.GetValue;
    ReDraw : Draw;
    ReSize : NOP; {ReSize kutter vi ut i denne applikasjonen}
  END;
END
```



Figur 70 - Eksempel på MeldingsBoks i MS-DOS

Ved oppstart av applikasjonen må det lages et objekt av klassen *KodeKontroll* som har ansvaret for å sjekke om det er kommet en ny versjon av applikasjonskoden. Hvis det er tilfellet inneholder

denne de metodene som skal til for å laste over den nye koden, behandle den og starte den istedet for den versjonen vi har startet her. Utvikleren kan redefinere klassen med en egen subklasse hvis han ønsker at noe annet skal skje ved nye versjoner. Tilsvarende er det under oppstart at sikkerhetskontroll må gjennomføres.

Etter at versjonskontroll er gjennomført, lages et objekt for *BestillingsVindu*. Dette er den styrende delen av applikasjonen via sin *Event* prosedyre inntil applikasjonen avsluttes.

```
Applikasjon CLASS Bestilling;
BEGIN
  BestNr : INTEGER;
  InfoTjener : Pointer;

  {RECORD og CLASS definisjoner}

  NEW KodeKontroll(Application=This Bestilling,Server="MachineY",Protocol=OSI);

  NEW BestillingsVindu;
END
```

8.4. Andre aspekter ved distribuerte applikasjoner

For å kunne oppfylle de siktemål som ble satt for den distribuerte applikasjonsmodell, er det endel forutsetninger som må være oppfylt. Skal personer som befinner seg geografiske adskilt samarbeide ved hjelp av datamaskiner, må det finnes et felles datanett mellom dem. Skal brukerne få tilgang til applikasjoner også utenfor sin egen organisasjon, må det finnes mekanismer for å finne dem. Det må også finnes måter å overføre og vedlikeholde applikasjoner slik at driftsaspektet ikke blir uoverkommelig. Mye informasjon er ofte kostbart å samle inn og vedlikeholde og eieren av slik databaser vil gjerne dekke inn igjen kostnadene. Det må derfor finnes metoder for å belaste brukeren for bruken av applikasjoner mot slike databaser. I slike åpne datanett som dette forutsetter, vil sikkerhet være et viktig aspekt. Det disse tingene jeg vil beskrive i de kommende avsnittene.

8.4.1. Nettverk

Datanett er i ferd med å komme i utstrakt bruk og i den akademiske verden finnes allerede verdensomspennende nettverk. Disse benytter i dag en rekke protokoller for overføring av data mellom maskiner, men utviklingen ser ut til å gå i retning standardisering rundt et eller to sett med protokoller [GOLDSTEIN 90]. De to protokollsettene er TCP/IP og ISO/OSI. Ved å benytte et standard grensesnitt mot det som i ISO/OSI modellen kalles transportlaget [TANENBAUM 89] vil applikasjoner enkelt kunne benytte begge protokollsett og også andre framtidige protokollsett forutsatt at det lages et tilsvarende grensesnitt mot disse (se side ??). På den enkelte klient kan man velge om man vil benytte en eller flere protokoller. Dette er ikke noe applikasjonsagenten avgjør, men operativsystem og maskinvare i den enkelte klientmaskin. Det eneste modellen forutsetter er at det finnes et nettverk med tilstrekkelig kapasitet og sikkerhet.

For enkle applikasjoner som databaseoppslag og lignende vil modellen gi svært lavt trafikkvolum mellom aksjonsfortolkeren og applikasjonen. Det er kun første gang den startes og man eventuelt må overføre den symbolske koden at det vil bli litt trafikk. Denne overføringen kan også skje ved hjelp av andre medier hvis det ønskelig å begrense trafikken. Dette burde gjøre det mulig også å benytte applikasjoner ved hjelp av modem over telefonnettet eller over ISDN [TANENBAUM 89]. Igjen er det applikasjonens natur som avgjør hvilke krav som stilles. Ettersom mulighetene

for høyere overføringshastigheter øker i de akademiske og offentlige datanettene kan avanserte grafiske og/eller audiovisuelle (multimedia) applikasjoner benyttes uavhengig av hvor i verden maskinene befinner seg.

8.4.2. Katalogen

I et verdensomspennende datanett uten en sentral driftsorganisasjon er det svært arbeidskrevende å finne fram til de spesialapplikasjonene man trenger. Hvis man for eksempel skal ha tak i en applikasjon som gir geografisk informasjon om en gruppe land i den tredje verden som enda ikke er tilknyttet noe datanett, hvor finner man da informasjonen for om dette landet?. Den ligger kanskje i endel databaser ved diverse universiteter eller i kommersielle databaser. Men for å finne fram til dem kreves endel arbeid eller forhåndskunnskap om hva som er tilgjengelig. Hvis det er en database som eieren vil ha betaling for bruken av, blir saken ytterligere forvansket.

I denne sammenheng er det katalogen basert på det som normalt omtales som X.500 (Se side 13) bør benyttes. De som har en applikasjon de vil gjøre tilgjengelig for alle, kan legge inn informasjon om denne i sin lokale del av katalogen. Denne vil da kunne være tilgjengelig fra alle maskiner med tilgang til katalogtjenesten. Uansett hvor i verden maskinene måtte befinne seg.

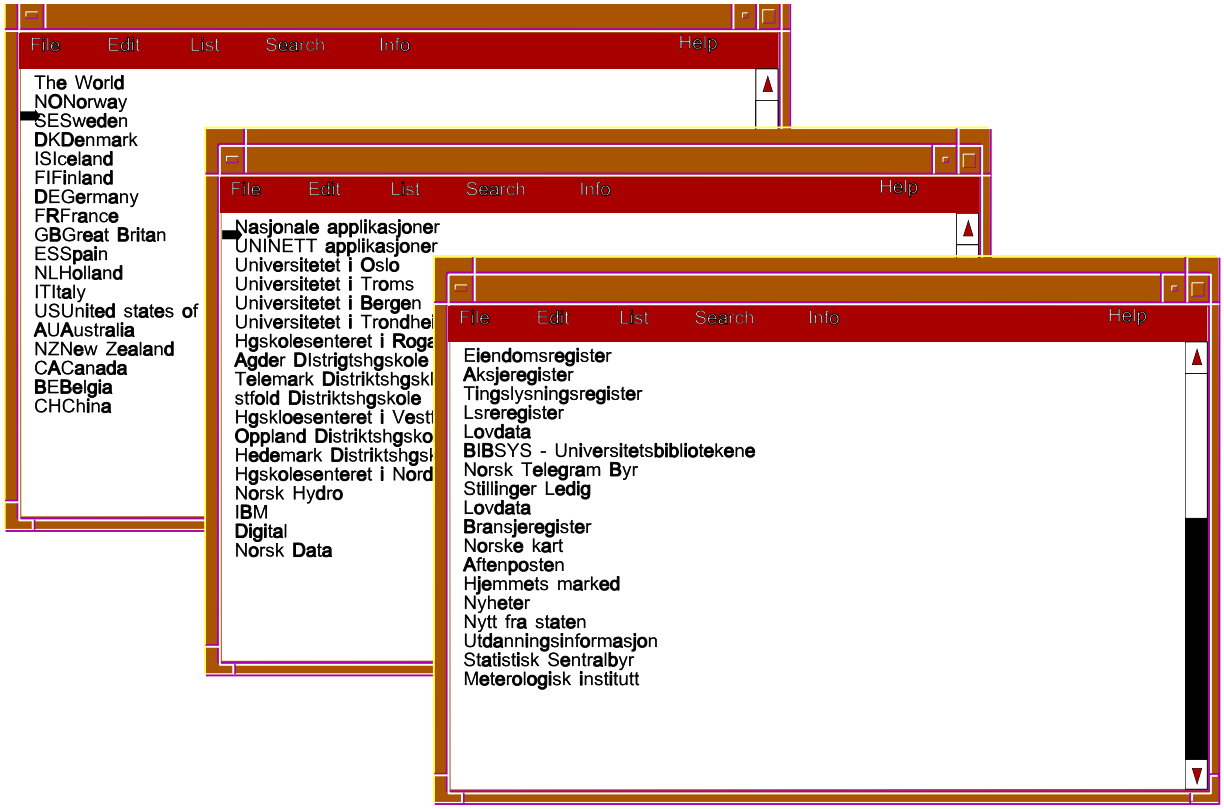
Den eneste applikasjonen en klient er utstyrt med når applikasjonsagenten installeres første gang, er den som setter brukeren i kontakt med nærmeste katalog og som derfra lister opp de applikasjoner den kjenner til, samt viser andre katalogtjenere (DSA'er). Brukeren kan derfra hente over informasjonen om hvilke applikasjoner som finnes og lagre informasjonen om hvordan de kontaktes over nettet.

En slik beskrivelse kan inneholde ting som:

NAVN	Navn på applikasjonen
TYPE	Kodet applikasjonsbeskrivelse for maskinell søking
BESKRIVELSE	En tekstlig beskrivelse av hva applikasjonen gjør på et eller flere språk.
PRODUSENT	Hvem som har laget programmet
VERSJON	Hvilken versjon som er den gjeldende
TERMINALKRAV	Hvilke krav som stilles til maskinutstyret i klientenden
IKON	Et bitmap som kan assosieres med applikasjonen i grafiske omgivelser.
KOSTNADER	Pris pr. minutt/transaksjon, engangsavgift el.
VERSJONSTYPE	Full/demo versjon
MODIFISERBAR	Kan aksjons- og/eller dialog den symbolske koden endres lokalt
UTLØPSDATO	For at brukeren skal kunne teste den i et begrenset tidsrom.
IDENTIFISERING	Applikasjonens krav til identifisering av brukeren/terminalen
NETTADR	Adressen til den eller de tjenerne som har applikasjonen
NETTTYPE	Hvilke datanett protokoller applikasjonen understøtter
BEGRENSNINGER	Hvilke begrensninger som ligger på bruk av applikasjonen
SIKKERHET	Sikkerhetsklasse applikasjonen krever av klienten.
PUBLICKEY	Krypteringsnøkkel
KODETJENER	Entydig navn i katalogen på tjenere applikasjonens kode finnes på
APPLTJENER	Entydig navn i katalogen på tjenere applikasjonens data finnes på

Feltene kan dupliseres, slik at en applikasjon for eksempel kan være kjent under flere navn, ha flere beskrivelser og lignende på ulike språk. Både data og kode kan godt befinne seg på flere tjenermaskiner for å fordele belastningen på maskinene og nettverket. Systemet bør bistå med å

finne fram til den som gir best respons til enhver tid. En informasjonsforvalter kan også hente data fra flere tjenermaskiner.



Figur 71 - Eksempel på valg av applikasjon fra katalogen

Ved for eksempel et universitet, kan man lage meny applikasjoner som ikke er annet enn oversikt over de applikasjoner lokalt/eksternt man ønsker å legge opp for sine brukere automatisk. En ny bruker kan hente over denne som første applikasjon og dermed slippe å lete rundt etter applikasjoner for å sette opp sitt eget arbeidsmiljø på sin egen arbeidsplass. Ved å lage spesial-tilpassede menyer for hver faggruppe kan man skreddersy arbeidsmiljøet etter spesielle behov. Ved å bruke automatisk oppdatering av den symbolske koden ved økning av versjonsnumre vil denne menyen kunne vedlikeholdes med et minimum av innsats både fra bruker og driftpersonell.

8.4.3. Sikkerhet

Sikring av datanett har hittil vært entydig med lukkede nett. Det gir begrenset tilgang til tjenester og informasjon. For at ulike bedrifter og organisasjoner skal koble sammen sine ulike nettverk, kreves det sikkerhet mot misbruk av data som transporteres i nettet. Skal vi få til en så stor grad av informasjonsutveksling som mulig, er det tjenestene som må sikres og ikke selve datanettet på det fysiske nivå. Det sier seg selv at ved å ha lukkede private nett, eller ved hindre alle i å kommunisere med alle, kan vi ikke få maksimal nytte av de tjenester som finnes totalt sett. Katalogen har en oppgave også i denne sammenheng. Det er identifisering av bruker, terminal, tjener og informasjonsforvalter. Alle disse komponentene i nettverket kan lagres i katalogen med en nøkkel, slik at all trafikk til denne krypteres med, men som bare eieren selv kan dekryptere (*Public key encryption*).

Brukeren må kunne være sikker på at en informasjonsforvalter virkelig er den den gir seg ut for. Det ville være katastrofalt om en forsker begynte å overføre sine forskningsresultater til en informasjonsforvalter som lå og utgav seg for å være hans arkiv, men i virkeligheten var en falsk informasjonsforvalter. Ikke bare ville vedkommende miste arkiveringen av arbeidet, men

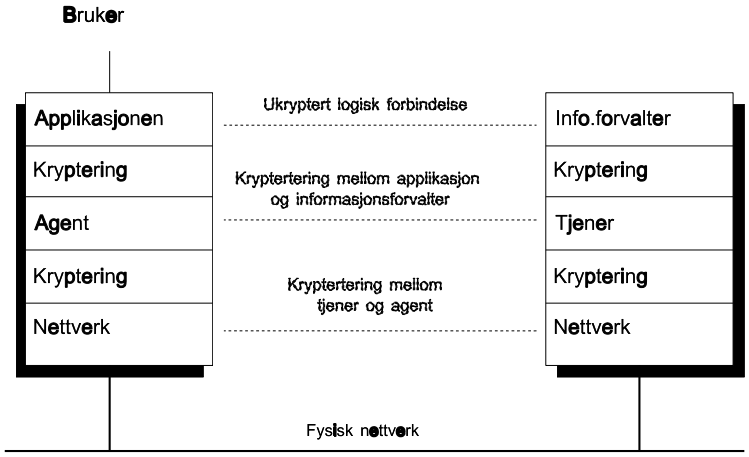
resultatene kunne også havne steder det kanskje ikke burde havne. Tilsvarende må informasjonsforvalteren kunne forsikre seg om at brukeren virkelig er den han gir seg ut for. Dette er særlig viktig for informasjonsforvaltere som inneholder informasjon som ikke skal spres til annet enn et lite antall mennesker eller for informasjonsforvaltere som eieren krever betaling for.

Ved å benytte public key kryptering [TANENBAUM 89] under oppkoblingen som signatur, og for å kryptere en eventuell nøkkel for videre synkron kryptering, vil systemet ha en rimelig grad av sikkerhet. For applikasjoner som benytter gratis informasjonsforvaltere i datanettet er ikke dette noe problem, men for ting som et lønns- og personalsystem, eller regnskapssystem er det viktig at informasjonsforvalteren kan identifisere brukeren på en standardisert måte. All trafikk mellom de to partene kan også krypteres på en per sesjons basis, eller på en per transaksjonsbasis. Hver transaksjon avsluttes da med en pakke som er kryptert etter public key og med ny nøkkel for neste transaksjon. Om noen skulle sitte og tappe trafikken vil det ta ufattelige ressurser å knekke kodene.

Brukerens system må også kunne forsikre seg om at data kommer fra den tjenermaskinen den er forutsatt å gå på. Her vil en helt analog metode kunne benyttes. Ved å finne tjenermaskinens public key i katalogen kan denne identifiseres på samme måte.

Det er den lokale klientens oppgave å forsikre seg om at brukeren virkelig er den man utgir seg for. Her er en sikkerhetsrisiko i systemet. Sensitive applikasjoner må kunne settes opp slik at de bare vil kommunisere med maskinvare av en gitt sikkerhetsklasse. Katalogen vil gi informasjon om terminalen og dennes sikkerhetsklasse og på samme måte som mellom bruker og informasjonsforvalter kan tjenermaskinen benytte terminalens public key for å sende en testmelding til terminalen og sjekke om den er den gir seg ut for. Katalogen kan også benyttes for å finne en terminal eller brukers organisatoriske tilhørighet. Dette kan brukes til å begrense tilgangen til en applikasjon på organisasjon, noe som er langt enklere å administrere enn å sette tilgangen per bruker.

Man kan her får flere lag med kryptering oppå hverandre. For eksempel ved at trafikk mellom tjener og klient er kryptert og tilsvarende mellom informasjonsforvalter og applikasjon. Den krypterte trafikken mellom applikasjon og informasjonsforvalteren blir da rekryptert mellom tjener og klient. Ved å dele opp systemet i flere krypteringsmoduler vil de øverste krypteringslagene ikke se hva slags kryptering som foregår på de lavere lagene.



Figur 72 - Kryptering mellom ulike nivåer

I tillegg til en identifisering ved hjelp av katalogen, kan selvsagt applikasjonene selv ha passord og lignende for å sikre seg ytterligere mot ikke autorisert tilgang.

Brukeren bør kunne sette opp sitt system slik at det bare er gratis applikasjoner uten sikkerhetskrav som benyttes og dermed slippe hele krypteringen. Eller man kan autentiseres til å benytte applikasjoner som har begrenset tilgang, men som er lokale ved institusjonen. Eller man kan si at

det er kun applikasjoner som har en testversjon og/eller koster mindre enn en viss sum man er interessert i. Katalogen vil da bare vise de applikasjonene som er aktuelle.

For å sikre at en applikasjon ikke forårsaker skade på den lokale maskinens data, programvare, eller sender ut ting som er lagret der, må det finnes mekanismer for å sjekke hva en applikasjon gjør. Det må også være mulig å hindre at en applikasjon endrer noe på den lokale maskinen uten at brukeren har godkjent det. Ting man kanskje ikke uten videre ønsker at en ukjent applikasjon skal utføre er å slette filer, endre navn, overskrive filer og lignende. Det må derfor finnes programmer som kan kontrollere en applikasjon og gi rapport om hva applikasjonen kan gjøre av farlige operasjoner. Dette er en viktig grunn til at koden må være i symbolsk form når den overføres fra tjenermaskinen til agentmaskinen. Symbolsk trenger strengt tatt ikke være det samme som at det er lett leselig for et menneske.

8.4.4. Begrenset spredning av applikasjoner

Enkelte applikasjoner kan være av en slik art man ikke ønsker å legge inn informasjon om dem i katalogen. Disse kan settes opp på lokale klienter ved at man laster over en dummy applikasjon som så kan konfigureres manuelt til å laste over den symbolske koden. Eller man kan lage applikasjonen slik at den aldri laster over definisjoner i det hele tatt. Den symbolske koden må da lastes inn manuelt på klientene som skal benytte den. For mindre sensitive applikasjoner kan man eventuelt bare legge informasjonen om dem i den lokale katalogen, men ikke videreformidle den til andre katalogtjenere. Det er altså en rekke metoder som kan benyttes for beskytte applikasjonene og brukerne til tross for at selve nettverket er åpent. Det må understrekes at maskinene også må beskyttes, slik at man ikke kan får tak i dataene bakveien. For eksempel ved hjelp av filoverføring.

8.4.5. Betalings applikasjoner

En nyttig mulighet med denne modellen i et stort nettverk vil være at informasjonsleverandørene kan utvikle applikasjoner som tilbys alle, men som de krever betaling for. Når applikasjonen startes første gangen er det en enkel applikasjon for betalingskvittering som startes først. Denne fyller så brukeren ut med navn, adresse, betalingsmåte og lignende. Systemet vil kunne inneholde standardopplysningene som så sendes kryptert med applikasjonens public key. Etter at de to partene har identifisert hverandre, lastes selve applikasjonen over. Man kan jo også tenke seg et system der en bedrift, et universitet eller lignende abonnerer kollektivt på en tjeneste og at straks brukeren er identifisert som tilhørende rett institusjon bli applikasjonen lastet over.

Det bør gis muligheter for en rekke betalingsformer og prismekanismer. Abonnement for en gruppe brukere er en mulighet. En engangs- eller periodisk avgift er en annen. Men man bør også kunne prise per transaksjon, oppkoblet tid, avstanden mellom bruker og applikasjon, overført datamengde, forbrukt diskplass, brukerens organisatoriske tilhørighet med mer. Betalingen må kunne skje via faktura, kredittkort, eller kunne skje via automatiske overføringer direkte fra brukerens bankforbindelse til informasjonsleverandørens, basert på EDI (Electronic Data Interchange).

Det er tjenestene applikasjonen tilbyr i forhold til informasjonsforvalteren som selges, IKKE den symbolske koden for å utføre tjenesten. Hvis man for eksempel har en generell tekstbehandler man ønsker å selge som en betalingsapplikasjon, er det ikke funksjonene i tekstbehandleren man bør selge, men tilleggstjenester som arkiver for referanse, bilder, illustrasjoner, samt trykkemuligheter, distribusjon og slike ting. Tekstbehandleren blir altså bare verktøyet for å utføre selve oppgaven med å spre informasjonen.

8.4.6. Lokale tilpasninger

For at brukeren skal kunne få et redskap som er tilpasset den jobben som skal utføres, er det viktig at applikasjonene kan tilpasses den enkelte. Siden all koden er i symbolsk form kan programmer på den lokale maskinen manipulere på den symbolske koden lokalt. Dette gjør at brukeren for eksempel kan fjerne felter fra skjermen i en database applikasjon, eller legge til egne huskelister i skjermbildet. Dette gir også gode muligheter for å oversette applikasjoner fra et språk til et annet. Å endre ting som farge, skrifttype og/eller størrelse skulle heller ikke by på problemer. For å endre på selve oppførselen kreves nok enten raffinerte programmer eller folk som kjenner systemet godt. Ting som menyer og tastkombinasjoner burde være trivielt, men logikken i applikasjonen vil nok kreve programmeringskyndige folk.

Alle bitene som styrer applikasjonens oppførsel og utseende ligger lokalt i symbolsk form og kan derfor endres etter brukerens behov og personlige smak. Det er selvsagt også en feilkilde og man kan tenke seg at applikasjoner kan forlange sjekksummer eller lignende, av applikasjonsagenten for den lokale symbolske koden, før de starter. Dette kan være aktuelt hvis det er sensitive applikasjoner som de ansvarlige ikke ønsker at brukerne selv skal kunne endre på. Tilsvarende kan brukeren nekte å ta i mot en oppdatering av en applikasjon hvis det er gjort endringer lokalt. Dette er ting modellen må kunne håndtere. Det bør være mulig både for applikasjonen og brukeren å bestemme hvordan de vil ha tingene. Hvis de ikke blir enige så er det bare for applikasjonen å nekte å starte.

Poenget er at modellen skal tilby så stor grad av fleksibilitet at brukerens behov og ønsker kan ivaretas, samtidig som man sikrer at applikasjonens integritet kan ivaretas.

8.4.7. Feil og forbedringer av applikasjoner

En ulempe med at en bruker kan hente inn applikasjoner fra, nær sagt hvor som helst er at det ikke er noen i organisasjonen som er ansvarlig for akkurat den applikasjonen. Det må derfor finnes muligheter for å få hjelp og støtte i problemsituasjoner fra et eller annet sted. Hva skal for eksempel en bruker gjøre, hvis applikasjonen stadig fyller opp den lokale disken med unødige ting? Eller hva hvis en applikasjonen for tilgang til en viktig informasjonsdatabase overhode ikke fungerer på en Macintosh, mens den fungerer helt utmerket under X-Windows? Det må finnes mekanismer for å rapportere feil, samt mangler eller ønsker om endringer av applikasjonen. Applikasjonen bør selv inneholde nødvendige adresser til dem som har utviklet den og som er ansvarlig for drift og vedlikehold. Brukeren, eller kompetansepersoner i lokal miljøet må settes i stand til å kommunisere med de ansvarlige via elektronisk post, telefon, telefax, brev eller andre medier. Slik det er i dag har man som oftest noen personer eller en avdeling lokalt som håndterer slike ting, men i et verdensomspennende nettverk, må det finnes mer sofistikerte løsninger for å løse det vedlikeholdsproblem man aldri slipper unna.

8.4.8. Multimedia muligheter

Modellen forsøker også å ta høyde for bruk av lyd og levende bilder, samt å holde muligheten åpen for at andre metoder å kommunisere på som vil/kan komme i framtiden. En viktig mulighet som følger med denne modellen, er å gi folk nye muligheter for å samarbeide. Man vil kunne benytte samme applikasjoner og dermed kunne vedlikeholde data i fellesskap, uavhengig av geografisk plassering. Det vil også bli langt enklere å ha felles arkiver for dokumenter, illustrasjoner, bilder og lignende, men det vil også kunne benyttes til kommunikasjonsmåter som i dag enten er svært dyre eller ikke gjennomførbare. For eksempel hvis to eller flere maskiner har et kamera tilknyttet vil kanskje bildene kunne sende fram og tilbake mellom to arbeidsstasjoner med mulighet for å utføre en videokonferanse ved hjelp av det utstyret som står på pulten. Tilsvarende kan begge se samme dokument på hver sin skjerm og editere i det samtidig, mens de snakker

sammen. Den ene kan da godt sitte i New York og den andre i Oslo. Dette forutsetter selvsagt at det er kapasitet nok i selve nettverket til å overføre de nødvendige datamengder i tilnærmet sann tid. Den store fordelen med å benytte et slikt system er at man ikke krever at brukerne i begge ender har likt utstyr. Ved at modellen er uavhengig av grafikk- og operativsystem vil det være mulig å kommunisere mellom alle typer maskiner som har en applikasjonsagent som er i stand til å håndtere animasjon og lyd.

For at slike ting skal bli mulig må det benyttes et standard sett med formater for overføring av bilder, grafikk og lyd. Identifikasjon av hvilket format som benyttes er viktig. Denne oppgaven tar ikke stilling til hvilke formater som bør benyttes til overføring av slik informasjon, men et viktig poeng er at det er en standard som gir muligheter for framtidige utvidelser og for konvertering til og fra andre formater, hvis ikke et format blir enerådende.

Hva må gjøres for å realisere den distribuerte applikasjons modellen?

Det som ihvertfall gjenstår er:

- Spesifisere protokollen mellom applikasjonstjener og agent. Dette er en lag 7 protokoll som bør kunne benyttes med både TCP/IP og OSI som transportprotokoller. Den eksakte protokollen ikke så vanskelig å spesifisere det er funksjonaliteten protokollen krever som er vanskelig å finne fram til.

- Utvikle eller adoptere et objektorientert programmeringsspråk som fyller de krav modellen stiller.

- Spesifisere alle basisklasser som en applikasjonsagent må kunne håndtere og mekanismene for egne subklasser og objekter.

- Utvikle applikasjonsagenter for de mest populære maskinplattformene og deres vindusystemer. Primært for Macintosh, X, MS-Windows, Presentation Manager og en skjermbasert MS-DOS agent. Store deler av arbeidet som gjøres for en applikasjonsagent kan benyttes ved utviklingen av de øvrige.

- Lage rutine biblioteker for tjenermaskiner, slik at det er enkelt for applikasjonsutviklere å støtte den distribuerte applikasjonsmodellen.

- Lage interaktive applikasjonsutviklingsmiljø for en eller flere maskinplattformer og vindusystemer, slik at det å lage distribuerte applikasjoner blir like enkelt som å utvikle applikasjoner i Smalltalk.

Det er alltid vanskelig å anslå arbeidsmengden som trengs for å utvikle programvare. Det er også tilfellet her. Jeg sitter med en følelse av at noen få, dyktige personer kunne lage det som trengs i løpet 2-3 år, uten at jeg kan begrunne det nærmere. Samtidig har jeg ikke noen problemer med å se et byråkratisk, komitéstyrt utviklingsarbeidet som tar 5-6 år og kanskje aldri kommer i mål.

8.5. Forutsetninger for å lykkes

Hvis vi ser på de systemene som har slått gjennom de siste 5-6 årene, slik som X, NFS, NIS, TCP/IP osv., har de hatt to ting felles: De har kommet ut fra universitetsmiljøer og har vært billige. Arbeidet er basert på forskning og utvikling gjort ved ulike universitet, gjerne støttet av industrien.

I utgangspunktet har det vært gratis, eller ihvertfall svært billig for leverandører å implementere systemene på sine maskinplattformer. Noe som igjen har ført til at brukerne har fått systemene billig og dermed tatt dem i bruk. Å kjøpe et UNIX-system uten disse tingene, er i dag nærmest utenkelig og for de fleste kjøpere det samme som at man heller velger en annen leverandør. Når det gjelder OSI-produkter har bildet vært omtrent motsatt. Offentlige kunder er i mange land pålagt å kjøpe OSI-produkter, men prisen har så langt avskrekket mange andre fra å kjøpe dem. At ting også er en ISO/CCITT standard er selvsagt en stor fordel. Men viljen til å betale de forholdsvis høye beløp som kreves, er ikke alltid like stor som viljen til å prise standarder.

Skal et system som beskrevet i denne oppgaven kunne realiseres, holder det ikke at noen bare setter seg ned og utvikler det. Nytten er basert på at mange benytter det. Så mange som overhode mulig. Skal et slikt system få gjennomslag på verdensbasis, må det være gratis, veldokumentert og tilgjengelig i kildekode både på tjener- og agentsiden. Det vil si at utgiftene til utvikling bør finansieres gjennom sponsorpenger fra industrien, eller gjennom offentlige bevilgninger i et eller flere land.

Det bør også forsøkes gjort til en ISO-standard. Problemet med mye av det som kommer ut av universitetene er at det ofte ikke blir ansett som seriøst i industrien før det har gått ganske lang tid. Et ISO-stempel vil trolig bidra til at industrien kommer fortere med.

9. Referanserog bibliografi

ANSA — ANSA Reference Manual Release 0.03 (Draft), Alvey Advanced Network Systems Architecture Project, 24 Hills Road, Cambridge CB2 1JP, UK

ARNOLD, Geoff — Internet protocol implementation, experiences in PC-NFS 1988, Sun Microsystems Inc. East Coast Division, Billerica, Massachusetts 01821

BOGACKI, Dariusz — Språk til objektorientert beskrivelse av informasjonssystemer, Komparativ analyse av Delta, OOA og deres editeringsverktøy i lys av et objektorientert perspektiv. Hovedoppgave ved Institutt for Informatikk, Universitetet i Oslo 1991.

COAD, Peter
YOURDON, Edvard — Object-Orientet analysis, Second Edition 1991 Object International Inc

COULOURIS, George F.
DOLLIMORE, Jean — Distributed Systems. Concepts and Design Addison-Wesley Publisher Ltd. 1988. ISBN 0-201-18059-6

DAVIDSON, John — An introduction to TCP/IP 1988 Springer-Verlag, ISBN 0-387-96651-X

DERFLER, Frank J. — TCP/IP for multiplatform networking PC-Magazine 27.juni 1989

GOLDSTEIN, — NORDUnet 90

HAUSKEN, Peter — Konferansen som vokser fra grasrota Datatid 2/90

HENDRIK, Charles L. — Introduction to the internet protocol RUDGERS, The state University og New Jersey. 3.juli 1987

HENDRIK, Charles L.
RUDGERS, — Introduction to administration of internet-based local network The state University og New Jersey. 24.juli 1988

HEINÄNEN, Juha — Introduction to the OSI Directory (NORDUNET 89) Software System Laboratory, Tampere University of Technology, P.B. 527, SF-33101 Tampere, Finland

GOLDBERG, Adele — Smalltalk-80, The Interactive programming Environment, Kapittel 5, Fundamentals of the Smalltalk-80 language Addison-Wesley Publisher Ltd.

ISO/IEC/DIS 9594 — ISO, The Directory, part 1-8, 1988 International Standard Organization 1988 International Electrotechnical Commission 1988

KOSHAFIAN, Setrag
ABNOUS, Razmik — **Object Orientation, Consepts, Languages, Databases, User Interfaces. John Wiley & Sons, Inc. 1990. ISBN 0-471-51801-8**

KUTAL, Gulay — A Comparative Survey on Windows Environments, 1990

	Hovedoppgave, Institutt for Informatikk, Universitetet i Oslo
LALONDE, Wilf R. **PUGH,** John R.	Inside smalltalk, Volume II 1991 Pretice-Hall. ISBN 0-13-467309-3
LAMPSON, B.W. et.al.	Distributed Systems, Architecture and implementation Springer-Verlag 1981. ISBN 3-540-12116-1
LARSEN, Bjørn	Fremtidens vindussystem Datatid 5/89
LEE, Ed	Window of oppertunity UNIX Review, Vol.6, No.6
NORDHAGEN 89	Objektorientering er svaret, hva var spørsmålet? Forelesningsnotat,, Senter for Industriforskning, Oslo 1989
Norsk Regnesentral	NR-rapport 840, september 1990. Norsk Regnesentral, Gaustadaleen 23, PB. 114, Blindern 0314 OSLO 3
REENSKAUG, Trygve NORDHAGEN, Else	The Design and Description of Complex, Object-Oriented Systems, version 1.0, Senter for Industriforskning 1989
SAMS, Howard W.	UNIX communication The Wait Group Inc. 1987 ISBN 0-672-2511-5
SANDBERG, Russel	The Sun Network File System, Design, implementation and experience. Sun Microsystems Inc.
SLOMAN, Morris **KRAMER,** Jeff	Distributed Systems and Computer Networks Prentice-Hall 1987. ISBN 0-13-21-5864-7
SUN Microsystems Inc.	ONC/NFS Solution guide - Open Network Computing / Network File Systems Solutions, Second Edition. 1989 Sun Microsystems
TANENBAUM, Andrew S.	Computer Network, Second Edition 1989 Pretice-Hall Inc. ISBN 0-13-166836-6
TESLER, Larry	The Smalltalk Environment, Byte aug. 91, pp 90-116
UNINETT	UNINETT - En nasjonal infrastruktur 1990 UNINETT sekretariat, ELAB-RUNIT, 7034 Trondheim
QUATERMAN, John S.	The Matrix - Computer networks av conferencing systems Worldwide. 1990 Digital Press, Order no.: EY-0176E-DP ISBN 1-55558-033-5