Gjenbruk av eksisterende edb-systemer realisert gjennom distribusjon

Kristoffer Moe

Hovedfagsoppgave Systemarbeid,
Institutt for Informatikk, Universitetet i Oslo — Høst 1997

«Gjenbruk av eksisterende edb-systemer realisert gjennom distribusjon»

Nøkkelord: Distribuerte systemer, Distribuerte objekter, Objektorientert analyse, Objektorientert design, CORBA, RM-ODP, Java, Geografiske informasjonssystemer

Keywords: Distributed systems, Distributed objects, Object-oriented analysis, Object-oriented design, CORBA, RM-ODP, Java, Geographical Information Systems

31/7/97

Copyright © 1997 Kristoffer Moe
Teksten er produsert med **FRAMEMAKER 5.1**, rollemodelldiagrammer med **OORAM PROFESSIONAL 4.0**

Til Sunniva

«Jeg antar det har hendt de fleste av dere å bli grepet i jakkekraven av en av disse pratmakerne, som, grådig etter å høre lyden av sin egen røst, leter etter en kompis hvis eneste funksjon skal bestå i å låne øre uten å åpne munnen; og det er ikke engang sikkert at denne plagsomme fyren krever at man lytter til ham, det er tlstrekkelig at man fra tid til annen anlegger en interessert mine, enten ved å nikke bekreftende med hodet, eller ved en lett mumling som forfatterne med rette kaller samtykkende, eller ved tappert å utholde det insisterende blikket til den stakkars djevelen, til tross for den ekstreme tretthet som må oppstå ved en slik muskulær spenning. (…) Vel, jeg våger å si, med forbehold om det umiddelbare og massive frafall blant leserne som denne innrømmelsen utsetter meg for, at jeg tilhører nettopp denne type pratnmakere»

Louis-René des Forêts: Pratmakeren,
Bokvennen, Oslo 1996

Forord

«I am this month one hole year older than I was this time twelve-month; and having got, as you perceive, almost into the middle of my fourth volume—and no farther than to my first day's life—'tis demonstrative that I have three hundred and sixty-four days more life to write just now, than when I first set out; so that instead of advancing, as a common writer, in my work with what I have been doing at it—on the contrary, I am just thrown so many volumes back—was every day of my life to be as busy a day as this—And why not?—and the transactions and opinions of it to take up as much description —And for what reason should they be cut short? as at this rate I should just live 364 times faster than I should write—It must follow, an' please your worships, that the more I write, the more shall i have to write· and consequently, the more your worships read, the more your worships will have to read.»

Laurence Sterne (1713–68): «The Life and Opinions of Tristram Shandy, Gentleman», Vol 4 Ch.XIII

Det har nå gått omtrent ett år siden jeg begynte på denne hovedfagsoppgaven. Som sitatet ovenfor antyder, er det tilfeller der man bare kan sette seg ned og skrive og skrive. Spesielt i et felt som denne oppgaven dreier seg om — distribuerte objektsystemer — er det en voldsom utvikling, som avstedkommer mye dokumentasjon. Flere standardiserings- og forskningsprosjekter er i gang samtidig over hele verden i dette øyeblikk, og det sier seg selv at å følge med i alt er mer enn en heldagsjobb. Faren for at stoffet derfor er ukomplett og gammeldags i det samme øyeblikk det er forfattet er overhengende. Imidlertid tror jeg denne oppgaven kan danne et utgangspunkt for å holde seg oppdatert i den videre utvikling på området.

Bakgrunn for oppgaven

Utgangspunktet og inspirasjon for valg av tema var kurset «Samvirkende Distribuerte Systemer», som ble holdt ved Institutt for Infor-

matikk ved Universitetet i Oslo høsten 1995. I forbindelse med dette kurset kom jeg i befatning med RM-ODP — Reference Model for Open Distributed Processing. Modellen foreskriver at man beskriver et distribuert system i fem perspektiver: *Enterprise-*, *Information-*, *Computation-*, *Engineering-* og *Technology viewpoint*. Min tanke var at denne oppdelingen også burde kunne brukes for å styre større systemutviklingsprosesser, helt uavhengig om systemet skulle være distribuert eller ikke. Videre syntes jeg det ville være en god idé å kombinere disse fem perspektiver med en passelig metode, og slik få et robust metodeverk.

Jeg kom inidlertid etter hvert, akkurat som Audun Killingberg [Killingberg 94], til den dobbeltbunnede erkjennelse at dette jo ikke var særlig originalt, for det samme hadde allerede vært gjort (se side side 88ff). Den dobbelte bunn redder imidlertid æren, for selv om det ikke var noe nytt og revolusjonerende jeg hadde tenkt, så kan det jo ikke vært så galt tenkt heller, siden det altså allerede var tenkt av flere andre. At en tanke er tenkt er selvsagt intet endelig kriterium på at tanken er god, men i dette tilfellet virker det som om den er et.

Jeg har senere dreid ambisjonene noe fra å finne opp noe genuint nytt, en ny metode (slik som presentert i [Hetland 97]), til å finne ut for min egen del hvilken metodisk støtte som finnes for systemutvikling for en distribuert sluttkontekst. Jeg har også ønsket å se hvordan eksisterende systemer kan tilpasses det nye og hvilke praktiske løsninger som finnes for realisering av systemer under disse betingelsene.

Målgruppe

Målgruppen for denne oppgaven er personer med interesse for objektorientering både som metode og som teknologi. Blant disse tror jeg det er studenter, forskere og systemutviklere. Det er et faktum at industrien i dag er svært konservativ med hensyn på metoder, teknologivalg og ikke minst språk. For eksempel er det stadig et stort behov for COBOL-programmerere selv om akademiske kretser ikke ofrer COBOL annet enn hån og spott. Distribuerte objekter vil her kunne brolegge gapet mellom gammelt og nytt hvis vi innser at vi i lang tid fremover må regne med å leve med de systemer som allerede er utviklet *samtidig* som vi supplerer dem med nye.

Siden case for denne studien er Plan- og bygningsetaten i Oslo, tror jeg de som beskjeftiger seg med IT-arkitektur og -strategier der vil kunne få inspirasjon av denne oppgaven. Dette gjelder også andre lignende etater i Oslo og andre kommuner, og for den saks skyld andre grener av offentlig forvaltning. Som jeg kommer til senere, i Kapittel 5: «Metoder for utvikling av distribuerte systemer» og også i Kapittel 7, vil en mulig videre vei lede i retning av å definere rammeverk for distribuerte forretningsobjekter for offentlig forvaltning.

Forkunnskaper

Tankegangen i denne oppgaven ferdes mellom tre konsepter; Teknologi, Metoder og Eksisterende systemer. Alle disse er på hver sin måte fundert på store mengder kunnskap, litteratur og erfaring. Hvilken dybde jeg så stanser ved i hvert av disse feltene vil variere, men felles for dem alle er at en viss bakgrunnskunnskap innen sentrale tema vil være en fordel. Spesielt er det nyttig med noe ballast innen *teknologi* og *metoder*:

Teknologi. Her er det nyttig med kjennskap til generell programmering, datakommunikasjon [Halsall 95] og databaseteori [Elmasri & Navathe 94], [Skagestein 91]. Inngående kjennskap til f.eks SQL (Structured Query Language) er ikke nødvendig, men det kan være nyttig å ha litt oversikt over relasjonsmodellen og forskjeller mellom relasjonsdatabaser og objektorienterte databaser.

Objektorienterte metoder og -språk. Det vil være gunstig med kjennskap til objektorienterte analyse- og designmetoder, f.eks OOram [Reenskaug et al 96], Aalborg-OOA [Mathiassen et al 93], CRC (Class-Responsibility Cards) [Taylor 95] m.fl. En god gjennomgang av noen objektorienterte analyse- og designmetoder (i allefall når det gjelder objektidentifisering) finnes i [Ressem 95]. En mer mangfoldig oversikt finnes i [Henderson-Sellers & Edwards 95]:133–92 og [Graham 94]:193–325. Aktuelle objektorienterte språk kan være Simula [Kirkerud 89], Java [Flanagan 96], Beta [Madsen 93], Smalltalk [Goldberg & Robson 83], C++ [Rudd 94], [Ellis & Stroustrup 90] m.fl.

Stilkonvensjoner

Utover i denne oppgaven har jeg brukt enkelte typografiske stilkonvensjoner som ligner de som brukes i mye annen litteratur innen informatikk. Forhåpentlig vil disse gi litt hjelp og antyde hva «rare» ord representerer:

- Firmaer, produkter og varemerker er som regel skrevet slik: SUN, HEWLETT-PACKARD, ACME.
- Programnavn, kommandoer og programelementer, f.eks variabelnavn, skrives slik: **`ls(1)`**, **`grep(1)`**, **`chown(1)`**,[1] **`alder`**.
- I interaksjon mellom bruker og edb-system angis brukerens kommandoer i *`kursiv`*, mens respons fra systemet i `normal`, f.eks:

```
$> fortune -m distributed
Law of Probable Dispersal:
        Whatever it is that hits the fan will not be evenly
        distributed.
$>
```

- I kildekode, f.eks C++ eller Java, angis nøkkelord i **`fet`**, mens kommentarer *`kursiveres`*:

1. Etter vanlig UNIX-konvensjon angir tallet i parentes i hvilken seksjon i online manualsystemet kommandoen forklares.

```
1   // Mitt første C++-program
2   #include <stream.h>
3
4   void main(void){
5      cout << "Hello world!" << endl;
6   }
7
```

- Forkortelser og akronymer forklares hovedsaklig første gangen de opptrer, f.eks TLA (Three Letter Acronyms), og opptrer også i ordlisten i Tillegg B.

I den grad det er mulig er fagutrykk oversatt til norsk, fordi nedslagsfeltet for denne oppgaven hovedsaklig er Norge, og en implisitt anvendelse av stoffet er systemutvikling for norske brukere. Som jeg kommer til i Kapittel 3, er det viktig å legge til rette for gjensidig forståelse av begreper mellom systemutviklere og oppdragsgivere/brukere, og et minstekrav må da være at de forskjellige begrepsapparater er formulert i et felles naturlig språk, f.eks Norsk. Der det ikke er noen tvil om hvilke engelske og norske uttrykk som betyr det samme (f.eks *domain* og *domene*) kan de i noen grad være brukt om hverandre for å skape en kobling mot eventuell kildetekst. I tilfeller der jeg ikke har funnet noen god norsk oversettelse brukes den opprinnelige — med tungt hjerte.

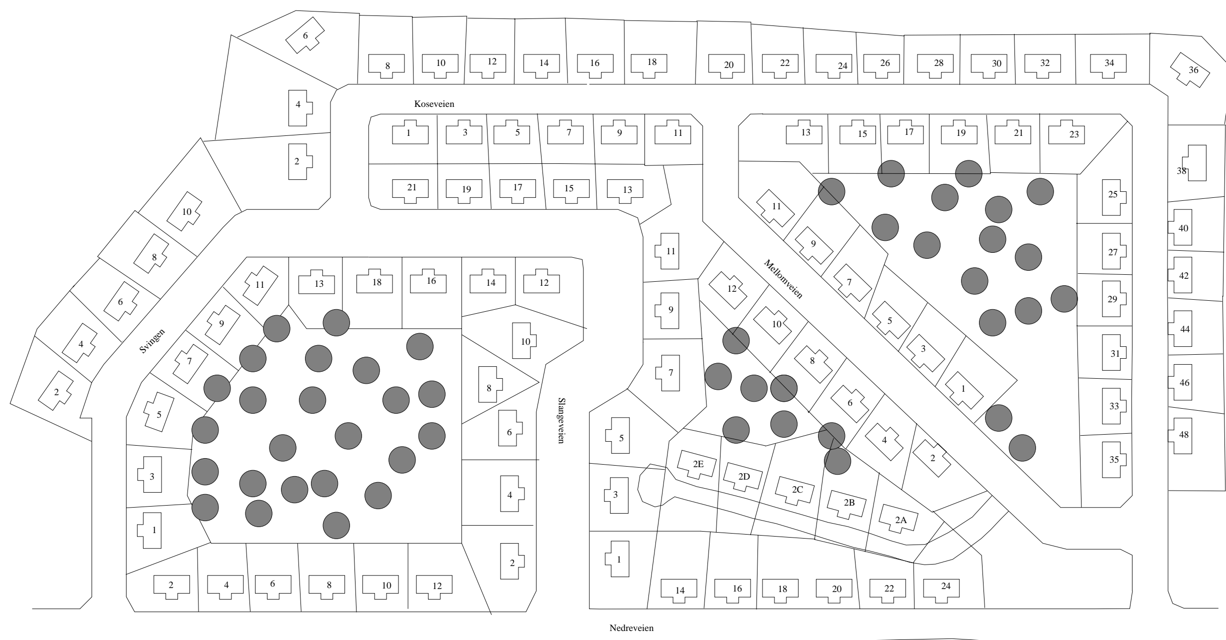
Takk til…

Det rettes stor takk til de to veilederne jeg har hatt i dette arbeidet; Jens Kaasbøll og Leikny Øgrim. Førstnevnte var primært til hjelp i igangsettelse og fungerte som samtalepartner under utforming av tema for oppgaven. Under siste halvdel av arbeidet har han veiledet fra Australia via elektronisk post, og også testet protypapplikasjonen i en virkelig distribuert sammenheng: tjener i Norge og klient i Australia! Leikny har i samme periode bidratt til styring av oppgavens gang som interessert og oppmersom leser med konstruktive kommentarer.

Videre rettes en takk til Plan- og bygningsetaten i Oslo kommune for velvillig utlån av kartdata. Prototypen ble mer realistisk med ekte data enn med den hagekoloni-lignende byen laget med tegneprogrammet `xfig(1)` i forbindelse med DSS-kurset nevnt ovenfor. (Sammenlign Figur 0.1 på neste side med Figur 6.1 på side 95.)

Jeg vil også gjerne rette en takk til instituttbibliotekets ansatte for svært velvillig bistand og assistanse med å finne bøker og artikler fra inn- og utland. Ved enkelte anledninger har de til og med gjort tapre forsøk på å spore opp bind som det viste seg at jeg ved en inkurie allerede hadde lånt…

Blindern, 31 juli 1997



Figur 0.1: Koseby — den velregulerte kolonihagebyen fra den første prototypen av kartsystemet i DSS-kurset høsten 1995

Sammendrag (Abstract)

I mange sammenhenger der det skal utvikles edb-systemer finnes det gjerne allerede edb-systemer fra før. De eksisterende systemene representerer ofte store investeringer, både i penger, tid og kunnskap, og kan inneholde data som er helt vesentlig for en organisasjons daglige drift. I denne hovedoppgaven i systemarbeid redegjøres for hvordan systemutvikling bør foregå i miljøer der slikt er tilfelle, og hvordan det eksisterende kan integreres og innbefattes av nye systemkomponenter. Det argumenteres for at distribuert objektteknologi kan løse mange av de problemer en står overfor, så som sømløshet i faseoverganger i systemutviklingsmodeller, innkapsling av eksisterende og nye komponenter samt takling av heterogene miljøer. Slik dannes broer mellom eksisterende systemer og komponentbasert systemutvikling, som virker som ett mulig remedium for den såkalte programvarekrisen. For å teste ut de rammer som presenteres i oppgaven, er en prototyp for kartinformasjon utviklet med Java og C++ som implementasjonsspråk, og Orbix fra Iona Technolgies som mellomvareprodukt. For å illustrere konsepter mer konkret i oppgaven, er problemstillinger fra Oslo Kommunes plan- og bygningsetat benyttet.

In many environments where information systems are to be developed, there often exist a number of computer systems already. The existing systems often represent great assets to the organizations in which they are used, in money, time and knowledge; crucial to the organization's daily work. This Cand. Scient thesis is set at finding out how information systems development should be conducted in environments as outlined above, and how existing system components can be integrated and incorporated into and together with new ones. It is argued that distributed object technology may be the answer to many of the problems faced, such as seamless phase transitions in system development models and encapsulation of existing and new system components in heterogeneous environments. Thus bridges are constructed, spanning existing systems and component based systems develop-

ment, seeming a promising remedy to the so-called "software crisis". To evaluate the concepts and frameworks presented, a prototype for a geographical information system is developed, by means of Java and C++ as programming languages, and Orbix from Iona Technologies as middleware. Issues concerning the municipal building control agency in Oslo, Norway, are used as case to illustrate points made in the text.

Innhold

Forord . . . . *i*

Sammendrag (Abstract) . . . . *vii*

1 *Innledning* *1*

Definisjon av problemstilling 2

Modellering og integrasjon av eksisterende edb-systemer *2*

Modellering og integrasjon av distribuerte systemer *2*

Innholdet i denne hovedoppgaven 3

Premisser for oppgaven *4*

Oppgavens avgrensning *5*

Viktige begreper 7

Metoder og systemutviklingsmodeller *7*

Objektorienterte metoder *8*

Distribuerte systemer *13*

Eksempel: distribuert prosjektstyringssystem *13*

Distribusjon og arkitektur *15*

Idealer i systemutvikling *16*

Case 17

Eiendomsinformasjon *18*

Er dette systemarbeid, da? 19

Forskningsmetode *20*

Sammendrag 21

2 *Arven fra fortiden* 23

Innledning 23

«Mission critical legacy systems» 24

Virksomhetskritiske systemer 25

«Legacy systems» 25

Strategier for videreutvikling 26

Strategi 1: kast alt og begynn på nytt 27

Strategi 2: lag nytt og kjør i parallell til alt virker 27

Strategi 3: gradvis forandring av det gamle til noe nytt 28

Reverskonstruksjon — reverse engineering 28

Program slicing —kodeslakt 29

«Screen scraping» og terminalemulatorer 30

Objektorientert reverskonstruksjon 30

Objektorientering og databaser 32

Sammendrag 33

3 *Beskrivelsesteknikker og modeller* 35

Innledning 35

OSI-modellen 37

Brødristermodellen — ECMA/NIST Toaster Model 39

Referansemodell for distribuerte systemer — RM-ODP 42

Fem perspektiver 43

Enterprise viewpoint 44

Information viewpoint 45

Computational viewpoint 46

Engineering viewpoint 46

Technology viewpoint 49

Konsistens mellom perspektivene 49

Forholdet mellom modellene 49

Hvem eier språket? 51

Felles modeller og distribusjon 53

Sammendrag 54

4 *Oss objekter imellom* 55

Innledning 55

BSD Sockets/ SVR4 Transport Layer Interface 56

Wrappers 56

Fjernprosedyrekall 57

Objektmeglere 58

Forhold til referansemodellene 59
Object Management Group 59
Object Management Architecture 60
Kommunikasjonen mellom objekter 62
Grensesnittet mellom objektene 62
Implementasjonslageret 64
CORBA tjenester — CORBAServices 65
CORBAfacilities — ekstratjenester 66
CORBA og de andre 66
CORBA vs DCE 67
CORBA vs DCOM 68
Sammendrag 69
CORBA og Plan- og bygningsetaten 70

5 *Metoder for utvikling av distribuerte systemer* 71

Innledning 71
Objektorienterte metoder 72
Sømløse reversible faseoverganger 73
Modeller og eksisterende systemer 73
Modeller og distribusjon 74
OO-metoder for distribusjon 75
OOram 75
Hovedidé 75
Tre ståsteder 76
Ti perspektiver 76
Eksempel: PBE 78
Analyse og syntese 81
Aggregering og virtuelle roller 83
Implementasjon 83
OOram og distribusjon 84
OOram og eksisterende systemer 86
DISGIS General Method 88
Organisasjonens forretningsprosesser 89
Organisasjonsstruktur 90
Organisasjonens distribusjon 90
Informasjonsmodellen 91
Bruk av informasjonen 91
Implementasjon og teknologi 92
Sammendrag 92

6 *Eksempelapplikasjon* 93

Innledning 93

Formålet med dette eksempelet ... *93*
Formålet med applikasjonen ... *95*
Bruksanvisning ... **96**
Klienten ... *96*
Tjenerne ... *96*
Analyse med OOram og de fem «viewpoints» ... **98**
Enterprise viewpoint ... *98*
Information viewpoint ... *99*
Computation viewpoint ... *100*
Engineering viewpoint ... *102*
Technology viewpoint ... *103*
Overordnet arkitektur og design ... **103**
3-lags arkitektur ... *104*
Backchannels — sidekanaler ... *105*
GAB-tjeneren ... *106*
Karttjeneren ... *106*
Objektmegleren ... *108*
Klienten ... *108*
Begrensninger ... **109**
Begrensninger i tjeneren ... *109*
Begrensninger i klienten ... *109*
Mangler og videre arbeid ... *110*
Erfaringer fra utvikling av prototypen ... **110**
Ressursfordeling ... *110*
Distribusjon og arbeidsmengde ... *111*
Forandrbarhet i systemet ... *111*
Heterogene plattformer ... *112*
Java, C og C++ ... *113*
Sammendrag ... **114**

7 *Oppsummering* ... ***115***

Metoder for distribuerte systemer ... **115**
Modellering og bruk av eksisterende systemer ... **116**
Veier til gjenbruk i systemutvikling ... **117**
Komponentbasert systemutvikling ... *117*
Rammeverk av forretningobjekter ... *118*
Veien videre ... **119**

A Referanser 121

B Ordforklaringer 131

C Kildekode til eksempelapplikasjonen 133

Grensesnittspesifikasjon 135
gis.idl 135
gis_i.h 136
gis_i.cc 136
GAB-tjener 138
gabserver.cc 138
gab_i.h 139
gab_i.cc 139
Karttjeneren 141
sosilexer.l 141
sosiparser.y 142
RTree.h 146
RTree.cc 149
stack.h 174
stack.cc 175
gis_i.h 176
gis_i.cc 176
mapserver.h 178
mapserver.cc 178
Databasehåndteringsverktøyet 181
dbm.h 181
dbm.cc 181
Klienten 183
kart.gui 183
kart.java 187

D Register 199

1 Innledning

«Most methodologies for designing object-oriented systems tend to ignore the issue of existing, conventional systems, implicitly assuming that each business design starts with a clean sheet of paper. Real companies rarely have this luxury.»

[Taylor 95]:64

«A distributed system is one in which a machine you never knew existed can stop you getting your work done.»

Leslie Lamport[1]
(Sitert i [Crowcroft 96])

Det finnes i dag et stort tilfang av analyse- og designmetoder for konstruksjon av edb-systemer. I de seneste årene har objektorienterte metoder blitt tatt i bruk i stadig større utstrekning og i flere systemutviklingsmiljøer. Mer og mer har også ikke-akademiske kretser begynt å benytte objektorienterte metoder, ikke bare til design og implementasjon av programvare, men også til analyse av problemområde for senere design. Objektorienterte metoder benyttes også for analyse som ikke nødvendigvis skal ende opp i edb-løsninger i det hele tatt, men kanskje heller omstrukturering og effektivisering av en organisasjon eller arbeidsprosess. I denne oppgaven vil jeg imidlertid ta utgangspunkt i at målet for analyse — objektorientert eller ikke — er design og implementasjon av et edb-system.

---

[1] Det er sannsynligvis flere som fortjener ære for denne definisjonen. Etter elektronisk korrespondanse med Jon Crowcroft medgir han at «this is probably apocryphal...but I heard it from people at Bell Labs (i.e. lucent....) I have also heard it ascribed to other people…»

Definisjon av problemstilling

Det later til at mange objektorienterte og ikke-objektorienterte analyse- og designmetoder gjerne deler en noe egosentrisk egenskap, nemlig en implisitt antakelse om at det som skal utvikles som resultat av arbeidet er et helt nytt edb-system og at det ikke finnes noen edb-systemer som behandler problemområdet fra før. Dette er forståelig, for når jeg selv tenker på hvordan jeg ideelt sett ville laget et gitt edb-system, så tenker jeg meg naturlig et helt konsept som gir svar på alle ønsker og eventuelt innfører en helt ny teknologi eller nye tenkemåter overalt i det fremtidige systemet. Det finnes imidlertid ofte eksisterende systemer i en virksomhet som det er behov for å integrere med nye, eksempelvis for å møte nye krav til virksomheten og dermed til de systemer som brukes.

Modellering og integrasjon av eksisterende edb-systemer

For å overleve i en hard konkurransesituasjon i dag, må bedrifter og organisasjoner være i stand til å kunne endre seg raskt [Hammer 96]. Det er derfor grunn til å tro at dette også gjelder de informasjonssystemer de er avhengig av også må kunne endres raskt [Taylor 95]. For ikke å konstruere helt nye systemer hver gang ting endres i forretningsdriften, er man nødt til å kunne bruke det som finnes fra før, enten ved å forandre på det eller bruke deler av det i en ny kontekst.

Et svar på nye krav kan derfor være integrasjon av eksisterende systemer med nye, og integrasjon av eksisterende systemer *ved hjelp av* de nye. Systemer som ikke var ment å kunne integreres med andre kan enten bygges om slik at de kan det, men det er ofte svært vanskelig, siden de ikke var ment slik i utgangspunktet. Alternativt kan de «pakkes inn» og representeres av nye komponenter overfor de andre deler av systemet. Uansett hvilken praktisk løsning som velges, må de eksisterende systemene eller delene av dem som skal beholdes i et nytt system *modelleres* på en passelig måte i en analyse før det innlemmes i en ny design.

Modellering og integrasjon av distribuerte systemer

Integrasjon — både mellom edb-systemer og mellom manuelle systemer, dvs f.eks saksbehandling eller andre arbeidsprosesser — kan i stor grad realiseres gjennom forskjellige former for kommunikasjon. Integrasjon mellom edb-systemer kan gjennomføres ved at systemene deler informasjon, f.eks gjennom felles databaser, eller mer direkte informasjonsutveksling. Kommunikasjon mellom systemkomponenter er derfor en nøkkelfaktor for systemintegrasjon. Siden eksisterende og nye edb-systemer gjerne er realisert med forskjellig teknologi, til forskjellige tider og på forskjellige steder, er det naturlig at de ikke nødvendigvis spiller sammen uten videre. En del av ressursene ved utvikling av nye integrerende systemkomponenter vil derfor gå til å binde sammen de aktuelle eksisterende og ikke-eksisterende deler til et samspillende hele. Resultatet av en integra-

sjon mellom gammelt og nytt kan derfor bli et distribuert system, der nyutviklingen består i — bortsett fra de nye funksjonskomponenter — å lage nødvendige grensesnitt mellom komponentene, gamle som nye, som kan realisere den kommunikasjon som er ønskelig.

I likhet med modellering av eksisterende systemekomponenter i nye sammenhenger som nevnt tidligere, er modellering av distribuerte systemer ikke særlig utbredt i vanlige objektorienterte metoder. Fokus ved utvikling av distribuerte systemer har generelt vært satt på tekniske utfordinger ved distribusjon, så som samtidighet, sikkerhet, pålitelighet etc. Modellering av arbeidsprosesser i organisasjoner som er distribuerte, og motivasjon for distribusjon utover tekniske årsaker som plattformmangfold og historikk som nevnt ovenfor, er heller ikke behandlet i særlig grad i klassiske OO-metoder.

Problemstillingen for denne oppgaven kan derfor formuleres som:

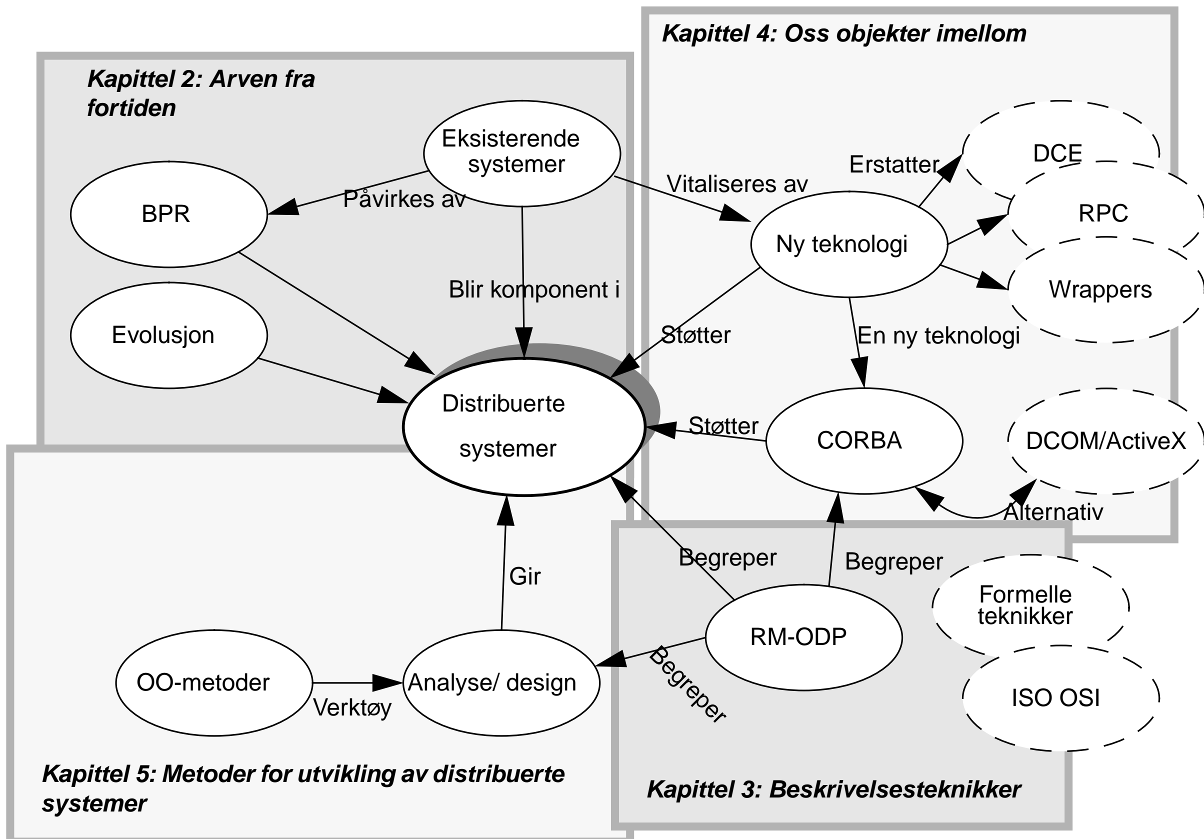
> ***Hvordan kan eksisterende edb-systemer og -komponenter modelleres og integreres med nye systemer ved hjelp av objektorienterte analyse- og designmetoder og praktiske løsninger for samvirkende distribusjon?***

I praksis vil problemstillingen i denne oppgaven belyses med en del (delvis overlappende) underproblemstillinger, som formulert som nøkkelspørsmål kan være:

- Hvordan modelleres eksisterende systemkomponenter sammen med nye?
- Hvordan kan eksisterende systemkomponenter brukes i nye sammenhenger?
- Hvordan kan eksisterende komponenter brukes i distribuerte systemer?
- Hvordan modelleres distribuerte systemer?
- Hvilken støtte finnes i eksisterende objektorienterte metoder for alle spørsmålene ovenfor?

Innholdet i denne hovedoppgaven

Figur 1.1 viser et mentalt kart over oppbygningen av denne oppgaven. Figuren gjør et forsøk på å vise hvilke konsepter som påvirker tankegang og prinsipper rundt distribuerte systemer, tanker om hva distribusjon innebærer og hvordan en tenkt konkret distribuert løsning påvirkes av de omkringliggende faktorene. Figuren følger ingen formell notasjon, men hver ellipse representerer et begrep eller emne,



Figur 1.1: Mentalt kart over emner i denne oppgaven

og pilene mellom dem viser hvordan de er relatert i behandlingen i denne oppgaven. De grå markeringsrektanglene rundt grupper av ellipser viser hvilke kapitler de forskjellige konseptene er diskutert i. Noen temaer er ikke dekket inngående, og de er representert med stiplede ellipser.

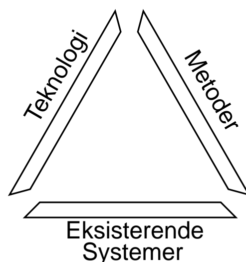
I figurens nedre venstre hjørne er det to ellipser merket henholdsvis «OO-metoder» og «Analyse/design». Hvorfor er det kun objektorienterte metoder som finnes verdige til å bli represenert her? For det første har jeg ennå ikke funnet noen behandling av distribuerthet i f.eks Strukturert Analyse, eller i noen andre metodefamilier for den sak skyld. Dessuten er for tiden all (akademisk) diskusjon rundt analysermetoder sentrert rundt objektorientering, så det er naturlig å begrense seg til det.

Premisser for oppgaven

Et gjennomgående premiss — eller kanskje mer en erkjennelse — i denne oppgaven er at det finnes mange edb-systemer fra før i denne verden. Omtrent halvparten av alle utviklingsprosjekter har som formål helt eller delvis å erstatte gamle systemer, likevel har de fleste metoder som utgangspunkt at noe nytt skal lages fra grunnen av [Kaasbøll 96b]. Vi vil imidlertid ha i bakhodet hele tiden at vi skal

utvikle et eller annet system der hvor det finnes ett eller flere fra før. Viktig her er å legge merke til at det kan være flere. Således også på forskjellige plattformer, levert av ymse leverandører og bygd etter forskjellige prinsipper. Det er grunn til å anta at det er en vanlig praksis å investere i IT-ressurser litt om litt, og dermed kjøpe det som man til enhver tid synes man har råd til, og som er gitt av utviklingen. Derfor skal man ikke la seg forskrekke av at større organisasjoner kan ha stormaskiner (f.eks IBM AS/400), minimaskiner (f.eks NOR 100) og personlige datamaskiner (MAC/PC) i tillegg til ymse filtjenere og bokser i mellomklassen for forskjellig bruk (f.eks UNIX) på samme sted. Dette betyr ikke at de spiller sammen. Det kan heller bety at det gjøres dobbeltarbeid, f.eks registreringer av samme data flere steder.

Et annet premiss som legges for denne oppgaven er at vi skal konsentrere oss om objektorienterte metoder for systemutvikling i de tilfeller der det altså finnes flere systemer allerede som man har et ønske om å integrere, og spesielt hvordan slike eksisterende systemer kan modelleres inn i en ny systemmodell.



Figur 1.2: Avgrensning av oppgaven

Et tredje premiss er valg av «lim» som skal binde sammen gamle og nye komponenter i et «nytt» edb-system. Limet som skal brukes er distribuert objektteknologi. I praksis vil det si Microsoft DCOM[1] eller CORBA.[2] Jeg har valgt å se mest på CORBA-standarden, hovedsaklig fordi den er ikke-proprietær og fordi den ligger et par år foran Microsofts DCOM. DCOM har imidlertid et svært sterkt fotfeste i kontor- og merkantile miljøer over hele verden, så jeg vil se på koblinger mellom forskjellige standarder etterhvert.

Oppgavens avgrensning

« Is it not a shame to make two chapters of what passed in going down one pair of stairs? for we are got no farther yet than to the first landing, and there are fifteen more steps down to the bottom; and for [all] I know (…) there may be as many chapters as steps (…)»

Laurence Sterne (1713–68): «The Life and Opinions of Tristram Shandy, Gentleman»[3]

Mange er sikkert enig med meg i at det vanskeligste er å avgrense seg når man skriver hovedoppgave. Objektorienterte metoder ekspanderer i stor hastighet. Nye OO-metoder, kombinasjoner av gamle, og kombinasjoner av gamle og nye publiseres oftere og oftere. Flere innflytelsesrike grupper innen objektorientert tankegang arbeider uopphoholdelig med utvikling av metodikk og tankegang. Dette kan være noe av årsaken til at deler av industrien er tilbakeholdne med å kaste

[1] Distributed Common Object Model

[2] Common Object Request Broker Architecture

[3] Finnes i Douglas Grant (ed): *«Memoires of Mr. Lawrence Sterne—The life and Opinions of Tristram Shandy, A sentimental journey, Selected Sermons and letters»*, Rupert Hart-Davis, London 1950

seg ut på en OO-galei som de ikke ser hvor går i det lange løp. Skulle jeg følge med i *alt* dette og samtidig holde meg nogenlunde oppdatert ville det selvsagt vært et sisyfosarbeid. Derfor er det viktig å avgrense seg, og simpelthen fra tid til annen nærmest late som man ikke har tilgang på all den informasjon som finnes online fra hele verden om emnet, og velge ett utgangspunkt man kan avgrense seg til. Jeg vil i tråd med dette derfor samle sammen noen metoder og forslag til metoder, en teknologi og litt til og ta utgangspunkt i dét.

De tre grensene nevnt på side 4 og illustrert i Figur 1.2 utgjør stort sett diskursuniverset for kapitlene utover i denne avhandlingen. Figur 1.1 på side 4 kan med litt velvilje tolkes som en graf, og hvis vi «grer» den ut, får vi en avhengighetsgraf eller en rettet graf som kan fortelle hvilken rekkefølge nodene bør behandles i. Man kan tolke figuren slik:

- Nye krav til EDB-systemer gjennom organisasjonsmessige endringer, f.eks BPR,[1] og evolusjon av eksisterende systemer og muligheter fører til behov for distribuerte systemer eller nye distribusjonsprinsipper.
- Det trengs et felles begrepsapparat for å kunne kommunisere og resonnere om eksisterende og nye systemkomponenter i en distribuert kontekst. Noen aktuelle begrepsapparater presenteres i Kapittel 3: «Beskrivelsesteknikker og modeller».
- Organisasjoner har ofte lagt ned store verdier i eksisterende systemer i form av utvikling og ekspertise. Dette betyr at deler av bestående systemer om mulig bør brukes videre. Jeg tar opp dette i Kapittel 2: «Arven fra fortiden».
- Det trengs analyse- og designmetoder som tar hensyn til og kan modellere distribuerte systemer. Noen metoder og verktøy presenteres i Kapittel 5: «Metoder for utvikling av distribuerte systemer».
- Det trengs en teknologi som kan binde sammen eksisterende komponenter med de nye og samtidig sikre at fremtidige tillegg og forandringer er overkommerlige å realisere. Jeg beskriver én standard for infrastruktur inngående og noen andre kursorisk i Kapittel 4: «Oss objekter imellom».

De tre siste punktene ovenfor representerer også avgrensingen som er illustrert i Figur 1.2

1 Business Process Reengineering. (se f.eks [Hammer & Champy 93])

Viktige begreper

Det eksisterer et stort mangfold av metoder for systemutvikling, og mange begreper brukes på flere måter i forskjellige sammenhenger. Dette mangfoldet gjelder også for andre emner i informatikk; både i praksis og forskning. I det følgende vil jeg derfor definere de begreper som er mest vesentlige for senere diskusjoner i denne oppgaven. Dette gjør jeg fordi jeg vil klargjøre min bruk av dem, og fordi begrepene ofte er utsatt for flertydighet. Disse begrepene vil også bli behandlet mer inngående i senere kapitler.

Metoder og systemutviklingsmodeller

Hvordan blir ribbesvoren sprø? Det finnes det en metode for. Hvordan skiftes dekk på en bil? Det finnes det andre metoder for. På samme måten finnes det metoder for å utvikle datasystemer. Felles for de fleste analyse- og systemutviklingsmetoder er oppdeling av problemet som skal analyseres, og abstrahering på passelig nivå. I strukturert analyse modelleres verden som prosesser, med tilhørende informasjonsstrømmer inn og ut av hver prosess. Prosesser kan slås sammen som subprosesser til en mer overordnet prosess, slik at detaljer altså abstraheres bort. Objektorienterte metoder modellerer verden som samspillende objekter.

I begrepsapparatet som presenteres i [Andersen et al 95] skilles mellom utviklingsmodeller, metoder og teknikker. Modeller kan være eksempelvis fossefall, spiral, iterativ etc. Metoder kan som nevnt ovenfor være strukturert analyse, objektorentert analyse/design etc. Disse kan igjen bestå av et antall teknikker, f.eks dataflytdiagrammer, rike bilder, objekt/klassemodeller osv. Mange av teknikkene er visuelle med en definert grafisk notasjon. Det finnes gjerne edb-baserte verktøy for å støtte bestemte metoder, gjerne med grafisk grensesnitt, f.eks RATIONAL ROSE, OORAM PROFESSIONAL, SELECT ENTERPRISE, OBJECTORY etc.

Ordet «Metodikk» brukes industrielt om konkrete — gjerne kommersielle — systemutviklingsmodeller. Imidlertid er en slik ordbruk ikke korrekt i henhold til norske ordbøker (f.eks [Knudsen et al 83]). Ut fra sammenhengen begrepet opptrer i er det nærliggende å anta at den intenderte meningen er den samme som «Methodology», som etter [Brown 93] kan bety «*A body of methods used in a particular branch of study or activity*». [Henderson-Sellers & Edwards 95] bruker «methodology» slik:

> *«A methodology is a set of instructions, guidelines, and heuristics that is* ***implementable*** *within a commercial environment adressing technical and managerial issues. An acceptable methodology should span the* ***whole*** *lifecycle from users' requirements through delivery and maintenance.»*

Forvirringen blir enda mer omfattende når det i [Landrø & Wangensteen 86] også er oppført «metodologi» som *«metodelære»*, mens «metodikk» er *«læren om metodene innenfor et visst arbeidsområde eller fagfelt (…)»*. Misforståelsen mellom «metodologi/metodikk» og «methodology» er derfor nærliggende.[1] Det finnes altså ikke noe norsk ord som har samme mening som «methodology». Når så komplette livssyklusmetoder, slik som f.eks MOSES [Henderson-Sellers & Edwards 95] egentlig favner over mer enn selve metoden (i [Andersen et al 95]s betydning) vil jeg velge å bruke «metode» slik «metodikk» brukes i norsk industriell betydning — dvs eng. «methodology» som i [Henderson-Sellers & Edwards 95], sitert på forrige side.

Objektorienterte metoder

> *«The object Technology experience has its roots in the 1960's when it was invented by two Norwegian university professors attempting to create an easy [way] to manipulate [a] computer model of a fjord (…)»*
>
> *Fra innledningen i «Get on the bus», informasjonsbrosjyre fra OMG*[2]

> *«'Object Oriented' is a term that has become so commonly used as to have practically no concrete meaning.»*
>
> *[Flanagan 96]:49*

Programmeringsspråket SIMULA tildeles ofte æren for å være objektorienteringens opphav, se f.eks [Ressem 95]. Utgangspunktet var å lage et språk som blant annet kunne brukes for å beskrive aktørers handlingsmønster og simulere scenarier, dvs operasjonsanalyse. Språket skulle kunne brukes til å formidle semantikken i en modell og være felles kommunikasjonsgrunnlag for programmerere *og* oppdragsgivere, dvs domeneeksperter.[3] I hvilken grad dette poenget lykkes kan diskuteres, men det er ingen tvil om at SIMULA på mange områder var omtrent 20 år forut for sin tid, men har ikke slått igjennom i kommersielle miljøer. Objektorienterte analysemetoder har ikke så lang fartstid som SIMULA, men deler mange av konseptene i større eller mindre grad.

Det er vanlig å skille mellom analyse og design. Eksempelvis er lærebøkene i Aalborg Objektorientert analyse og -design fordelt på to bind; [Mathiassen et al 93] og [Mathiassen et al 95]. En konsekvens av denne holdningen er at valg av implementasjonsspråk eller -miljø ikke skal påvirke selve analysen forut for designen. I praksis vil imidlertid systemutvikling gjerne foregå mer som vekslinger mellom analyse- og designfase, slik at implementasjonen eller betraktninger

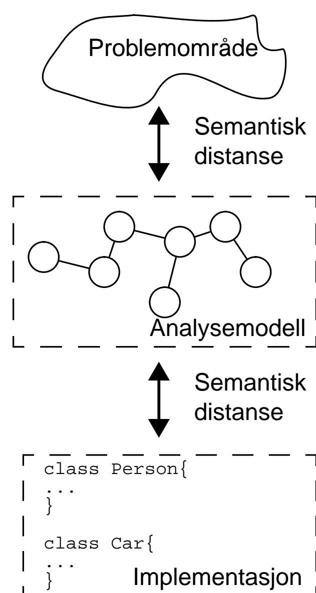
1 Ett eksempel på bruk av «metodikk» finnes i Aftenposten 6.4.97: «Sammen med professor Anne-Lise Børresen-Dala har han utviklet en metodikk for raskt å finne frem til P53-mutasjoner ved kreft.»

2 Object Management Group — se Kapittel 4.

3 Dette avsnittet er fritt etter Kristen Nygaards foredrag i Hovedfagsseminarserien høsten 96

rundt denne faktisk *vil* påvirke analysearbeidet. Mange av de konsepter som tradisjonelt tilhører objektorientering er derfor i varierende grad ivaretatt i de objektorienterte språk som finnes. Det er derfor lettere å illustrere noen av ideene med objektorientering med eksempler fra slike språk. Dette er gjort i det følgende.

Hovedformålet med en objektorientert analyse og senere -design/implementasjon er å komme frem til en hensiktsmessig *modell* av problemområdet. I diskusjonen i denne oppgaven antas det at modellen senere skal brukes i design og implementasjon av et edb-system. Generelt for alle objektorienterte analysemetoder er at den interessante del av verden (kalt «miniverden» i [Skagestein 91] eller «problemområdet» i [Mathiassen et al 93]) modelleres som objekter som interagerer med hverandre. Objektene kan ha *attributter*, slik f.eks en bil har en farge. Objekter befinner seg til en hver tid i en eller annen *tilstand*. Objektenes modelleres gjerne som *klasser*. Klassene struktureres i et hierarki etter *arv*, slik at én klasse kan være en spesialisering av en annen (*arv*) eller flere andre (*multippel arv*). Klassebeskrivelser kan også anses som maler, som viser hva som er felles for alle objekter som tilhører en klasse, eller hvordan et objekt av klassen instansieres. Forhold mellom objekter modelleres som assosiasjoner av forskjellig art, f.eks «er eier av.»



Figur 1.3: Semantisk distanse mellom problemområde, modell og implementasjon

Samhandling mellom objektene modelleres konseptuelt noe forskjellig. En måte å se samhandling på er å se på det som meldinger som sendes mellom objektene, som f.eks i OOram. (Se Kapittel 5.) Når et objekt mottar en melding, reagerer det i henhold til en bestemt metode, som beskrevet i objektets klassedefinisjonen (se nedenfor). Smalltalk er mest i tråd med denne meldingsmodellen slik at f.eks uttrykket `1+2` utføres ved at meldingen '`+`' med argumentet '`2`' sendes til objektet '`1`'. C++ er mer «tradisjonell», ved at det er funksjoner som *kalles*. I praksis går dette ut på ett. Begrepsbruken varierer altså endel her; spesielt innen de objektorienterte språk. Dette illustrerer imidlertid at valg av tankemodeller, som f.eks meldingsmodellen, vil påvirke den semantiske avstanden mellom virkeligheten — dvs problemområdet — og analysemodellen og mellom analysemodellen og design/implementasjon (se Figur 1.3).[1]

Selv om Smalltalk ikke er en analysemetode, men et programmeringsspråk, brukes meldingsparadigmet også her (se f.eks [Goldberg & Robson 83]). En annen modell for identifisering av samhandling, som finnes i Aalborg OOA, er hendelser som er felles for to eller flere objekter [Mathiassen et al 93].

[1] Kan egentlig verden modelleres som objekter som sender hverandre meldinger? Hvilke meldinger sender et pinnsvin ut mens det tasser over en vei? Og til hvem? [Cook & Daniels 94b] argumenterer for denne innsigelsen, mens f.eks [Graham 94] synes å mene at alle verdens objekters natur *er* å utveksle meldinger.

Andre grunnleggende begreper i objektorientert analyse og design er *innkapsling*, *polymorfisme*, *overlasting* og *gjenbruk*. Nedenfor er redegjort for det jeg legger i disse begrepene og de nevnt tidligere:

Klasse. Klassebegrepet er ikke entydig. En måte å se på en klasse er «Alle de objekter som har de og de egenskaper», dvs mengden av de objekter som er slik og slik.[1] En annen måte å se det på er som en beskrivelse eller en mal for instansiering av nye objekter. I en klassedefinisjon beskrives på en passelig måte egenskapene til de objekter som tilhører klassen (i den første betydningen av klassebegrepet) med attributter og metoder. Attributter er basale[2] informasjonselementer, så som `alder`, `kjønn`, `farge` etc. Metoder (aka prosedyrer eller funksjoner) definerer objektenes oppførsel eller reaksjon på stimuli.

Type. Hva er forholdet mellom typer og klasser? I mange sammenhenger regnes disse for ekvivalente, men det er mulig å definere en type som *grensesnittet* (se nedenfor) for en klasse, eller mer presist: *de grensesnitt en klasse tilbyr.* I en klassespesifikasjon defineres innmaten til en klasse, mens en type kun definerer hvordan objektet ser ut mot verden, altså det som ikke er innkapslet og skjult. Om typene til to objekter er like behøver ikke det bety at de er av samme klasse. Hva man kan bruke denne lille forskjellen til i praksis er uklart, i allefall i objektorientert analyse. I det henseendet vil jeg velge å bruke «type» og «klasse» som ekvivalente. Forskjellige klasser men samme type kan tyde på at modellen ikke er riktig, og at typer kan slås sammen til samme klasse, eventuelt plasseres i en arvsmessig sammenheng til hverandre. Dessuten kan det være at to typer som tilsynelatende er like egentlig ikke er det, fordi de er ment å være semantisk forskjellige, men har fått samme utseende. I eksempelet med `bil` og `buss` ovenfor kan de sies å ha samme type hvis de begge kun tilbyr metoden «beregn årsavgift.» Det må altså være en eller annen grunn til at de er modellert som forskjellige klasser. Det kan være at modellen ikke er komplett, slik at klassene trenger flere metoder som skiller dem fra hverandre. Eksempelvis kan en `buss` ha attributten «maksimalt antall ståplasser». En metode for å opplyse om dette vil derfor skille typene `bil` og `buss`.

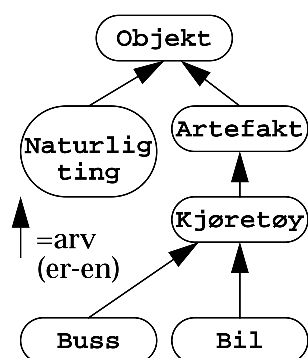
I implementasjon vil det være mer fruktbart å skille mellom implementasjon (klasse) og type (samling grensesnitt). Dette gjelder særlig i distribuerte systemer hvor eksisterende systemkomponenter er inn-

---

[1] Jorge Luis Borges refererer en klassifisering av dyr i et bestemt historisk kinesisk leksikon slik: a) De som tilhører Keiseren, b) Balsamerte dyr, c) Tamme dyr, d) Pattegriser, e) Havfruer, f) Fabeldyr, g) Løshunder, h) Dyr som er med i denne klassifiseringen, i) Dyr som skjelver som om de var gale, j) De talløse, k) Dyr som er tegnet med en tynn kamelhårspensel, l) Andre dyr, m) Dyr som nettopp har knust en blomstervase, n) Dyr som på avstand kan kan ligne fluer. ([Borges 73]:103, min oversettelse)

[2] Men ikke nødvendigvis, som antydet i Coad/Yourdon OOA, *atomiske*. ([Graham 94]:236)

kapslet, og innmaten i komponenten ikke er kjent, men det er grensesnittet.



Figur 1.4: Klassehierarki over kjøretøy

Arv. Objekter av én klasse kan arve egenskaper fra andre, slik f.eks en `bil` kan arve attributter og metoder fra `kjøretøy` (Se Figur 1.4). Det er også mulig å se på arv som en assosiasjon av typen «Er-en/et». Figur 1.4 kan derfor tolkes som at en `bil` er et `kjøretøy`, et `kjøretøy` er en `artefakt`, en `artefakt` er et `objekt`.

> *«Abstraction is a key component of everyday life. People understand concepts such as an automobile, a light bulb, freedom and trouble.»*
>
> *[Mullins 94]*

Ved generaliserings- og spesialiseringsforhold som illustrert ovenfor modelleres altså klassene etter grad av abstraksjon. Imidlertid kan ofte objekter abstraheres i forskjellige retninger. Eksempelvis kan en `bil` både være et `kjøretøy` og et `skatteobjekt`. Dette kan realiseres med *multippel arv*. Ved multipppel arv kan en klasse arve attributter fra flere andre klasser. Ved enkel arv får *subklassen* (arvingen) alle attributtene og metodene fra *superklassen* (som arves) i tillegg til de som er spesielle for subklassen. En `bil` (subklassen) er et `kjøretøy` (superklassen) med tillegg av fire hjul og motor. Ved multippel arv arves attributter og metoder fra alle superklassene, men problemer kan oppstå når metoder eller attributter fra to eller flere superklasser har sammenfallende navn og semantikk. I en analysefase vil en slik tilstand avsløre feil eller uheldig bruk av navn andre steder i modellen, eller feil i arvehierarkiet. Rent praktisk, ved design og implementasjon av modellen i et programmeringsspråk som tillater multippel arv, blir dette mer et praktisk problem. Det kan for eksempel være vanskelig å avgjøre hvilken metodeimplementasjon som skal velges hvis én metode kan arves fra flere superklasser. I Java tillates ikke multippel arv som sådan, men en klasse kan gjerne arve flere *grensesnitt.*

Grensesnitt. Et grensesnitt er en samling av signaturer til et antall metoder som på en eller annen måte hører sammen. Ved multippel arv på Java-vis kan attributter og metodeimplementasjoner arves fra én superklasse, mens kun grensesnitt fra andre superklassenes kan arves i tillegg. Subklassen må selv *implementere* de multippelt arvede grensesnitt. Slik er det mulig å referere til et objekt med forskjellige innfalsvinkler, og f.eks se på eller referere til min SAAB som en `bil` eller `skatteobjekt` etter behov. Dette gir muligheter for å modellere de samme objekter i flere kryssende forståelsesunivers. Slik semantisk heterogenitet vil kunne oppstå når flere fagdomener skal samarbeide. Eksempelvis vil tettere integrasjon av edb-systemer for trygd og videoutleie avsløre at et `person`-begrep eller -entitet kanskje ikke forstås eller brukes likt. Ved å definere ett grensesnitt for hvert bruks-

område, vil det likevel være mulig å benytte felles objekter; dvs de samme «fysiske» objektene i objektsystemet.[1]

Virtuelle metoder og abstrakte klasser. Virtuelle metoder er metoder som kan redefineres i subklasser. Slik kan oppførsel spesialiseres for å oppnå polymorfi (se nedenfor). For eksempel kan klassen `kjøretøy` deklarere en metode som beregner årsavgift. Denne kan defineres som virtuell i `kjøretøy`, og om ønskelig redefineres i klassene `buss` og `bil`. Enda mer aktuelt er å deklarere den som *ren virtuell*. En ren virtuell metode *må* redefineres i subklasser, fordi den faktisk ikke har noen implementasjon i superklassen i det hele tatt. Det får den i subklassene. En klasse som har én eller flere rene virtuelle metoder kalles en abstrakt klasse, og kan ikke selv instansieres som et objekt, men må arves i en subklasse først som implementerer alle rene virtuelle metoder. Følgelig finnes ingen rene `kjøretøy`-objekter, men de er enten `bil`er eller `buss`er. Det gir ingen mening i å beregne årsavgiften for et kjøretøy, hvis man ikke vet hva slags kjøretøy det er. Dette kan håndteres av *polymorfe* objekter.

Polymorfisme. Et objekt vil kunne sees på som tilhørende flere klasser, det vil si alle klassene langs et arvehierarki. Således kan man si at min SAAB kan sees på som en `bil`, et `kjøretøy`, en `artifakt` eller et `objekt`, hvis arvehierarkiet er slik definert. Objekter kan videre reagere på en henvendelse avhengig av plassering i arvehierarkiet. Dette vil si at samme melding til et objekt kan avstedkomme forskjellig oppførsel avhengig av hvor det er plassert i hierarkiet. Eksemeplvis kan både `bil` og `buss` være arvtaker til `kjøretøy` (se Figur 1.4). En metode `beregn_årsavgift` vil kunne implementeres forskjellig i de forskjellige klasssene, men settes i gang med samme melding.

Overlasting. Overlasting er beslektet med polymorfisme. Det kan defineres forskjellige metoder med samme navn i en klasse, men som skal brukes med forskjellige parametre. Riktig metode velges ut fra parametertypene og antall. Overlasting er mer en programmeringsmekanisme og brukes gjerne til overlasting av operatorer. Operatoren «+» kan således defineres forskjellige avhengig om parameteren eller argumentet eksempelvis er et heltall, et flyttall, en tekst eller en objektreferanse.

Innkapsling. Objekter kan skjule attributter og metoder som brukes internt, slik at ikke utenforstående kan kompromittere indre datastrukturer. Henvendelser til objekter skal foregå gjennom de offisielle kanaler klassen tilbyr, dvs som visse meldinger. I de objektorienterte metoder som baseres seg på meldingsparadigmet kan tilgang til attributter tilbys som *hent*- eller *sett*-meldinger. I praksis vil dette føre til noe forsinkelser under eksekvering, siden det må egne funksjonskall

[1] Nytten og de etiske konsekvenser av integrasjonen i dette eksempelet kan selvsagt diskuteres.

til for å lese en attributt. Denne attributten kunne (i allefall i C++) like gjerne vært hentet direkte fra minnet, men det er lett å miste oversikten og faren for å innføre feil i programkoden øker.

Distribuerte systemer

Begrepet *«Distribuerte Systemer»* er, som mange andre begreper i informasjonsteknologien, noe overlastet. Avhengig av hvem man spør vil man få forskjellige svar på hva som menes med begrepet. Mange datasystemer er å betrakte som distribuerte i fysisk forstand, for eksempel gjennom nettverk med varierende grad av ressursdeling; felles filtjenere, felles skrivere, felles administrasjon etc. Likeledes kan man alltids si at stormaskinbruk er distribuert all den tid det er flere terminaler spredt rundt geografisk, gjerne over store avstander.

Det er ikke uvanlig med kombinasjoner av disse to arkitekturer;[1] stormaskiner, minimaskiner og arbeidsstasjoner i nettverk, hvor arbeidsstasjonene fungerer som terminaler, samtidig som de kan være selvstendige arbeidstasjoner som bruker felles filtjenere. Igjen kan de være klienter eller tjenere i en klient-tjener situasjon. Det som synes å kjennetegne distribusjon er altså at noen har koblet ledninger mellom et antall datamaskiner. Dette blir imidlertid lite presist. Jeg vil i all hovedsak anta at distribuerte systemer skal forstås som *edb-systemer hvor funksjonalitet og applikasjonslogikk kan være spredt over flere fysiske datamaskiner*. Med denne definisjonen er det flere muligheter for hvor skillet går mellom det som tradisjonelt kalles henholdsvis klient og tjener. I praksis kan dette være realisert med distribuerte objekter, men det er ingen betingelse. Imidlertid vil jeg sette det litt på spissen slik: ***Etter å ha lest denne hovedoppgaven, vil leseren for en kortere eller lengre periode bli sterkt inspirert og føle trang til å utvikle distribuerte edb-systemer basert på distribuert objektteknologi.***

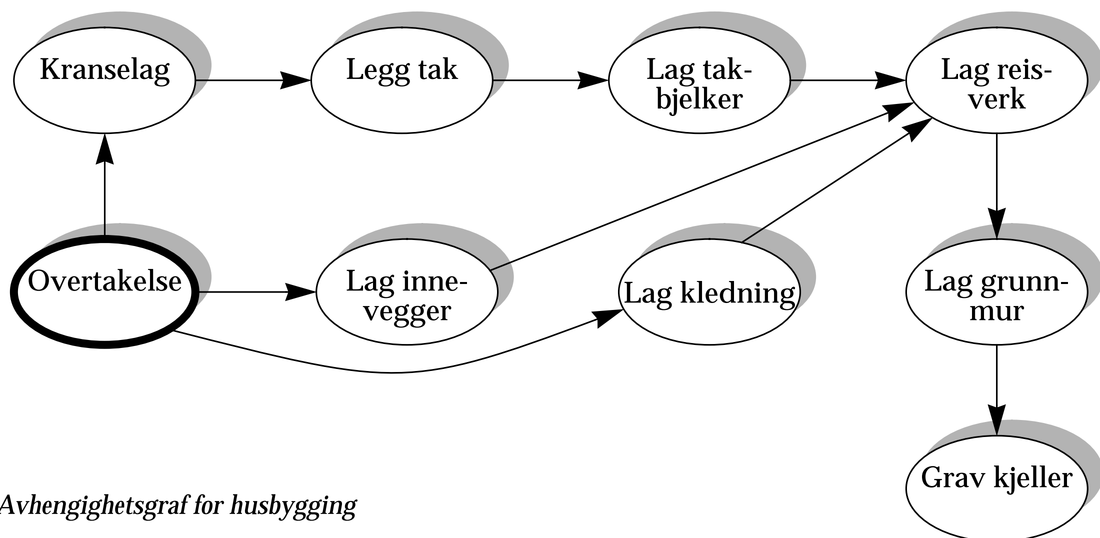
Eksempel: distribuert prosjektstyringssystem

En av de obligatoriske oppgavene i kurset IN115 (som er et kurs på grunnfagsnivå ved Institutt for Informatikk ved Universitetet i Oslo) går ut på å lage et prosjektplanleggingssystem [[Krogdahl 96]:90–96. lignende eksempel presenteres også i [Reenskaug et al 96], [Aagedal et al 97]. Kort fortalt går denne oppgaven ut på å simulere gjennomføring av prosjekter, f.eks bygging av hus, som består av et antall delaktiviteter som er avhengig av hverandre. Det er ikke mulig å legge takpapp før taket er reist, taket kan ikke bygges før reisverket osv. Aktivitetene modelleres som en rettet asyklisk graf,[2] som vist i Figur 1.5.

---

[1] Bli ikke overrasket om «arkitektur» brukes inoknsistent utover i oppgaven. Ordet er like overlastet som «distribuerte systemer», og brukes om alt fra hvordan programmer er strukturert (se f.eks [Shaw et al 96]) til maskinarkitektur (se f.eks [Tanenbaum 90]).

[2] En *syklisk* graf, derimot, dvs hvor man ved å følge en rekke piler kan ende opp der man startet, indikerer et ulaseggjørbart prosjekt à la «You can't work here if you're not in the union and you can't join the union if you're not working!»



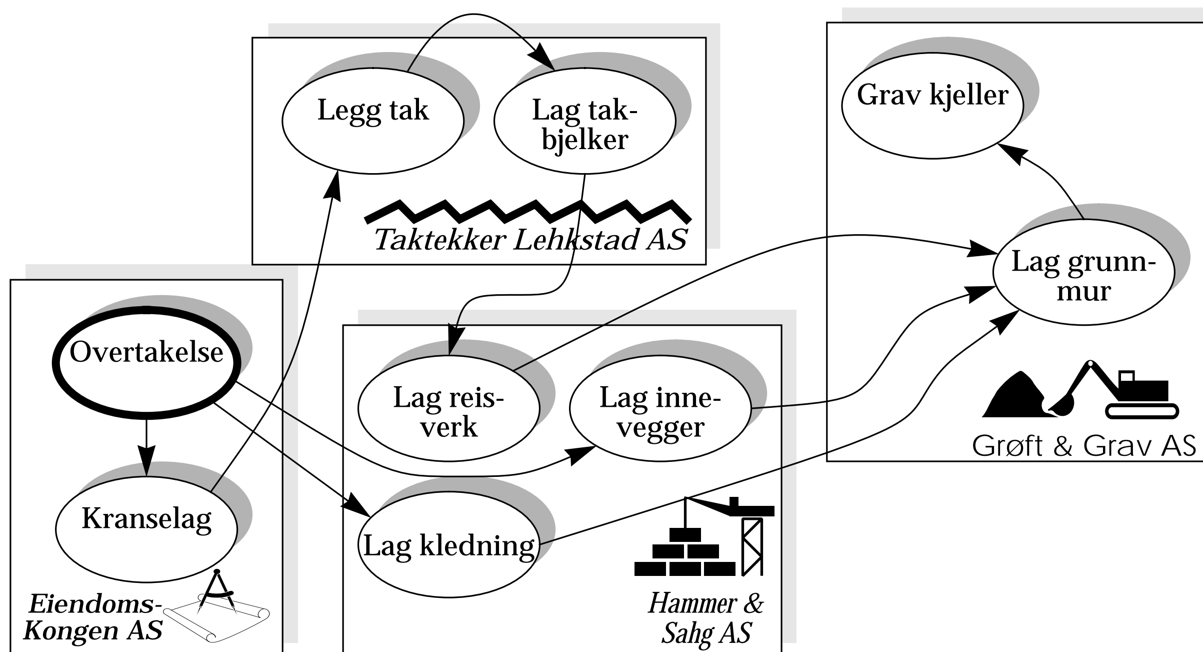
Figur 1.5: Avhengighetsgraf for husbygging

Nodene i grafen kan danne grunnlag for en topologisk liste, dvs en liste av aktiviteter hvor de kommer i lovlig rekkefølge i henhold til grafen. Siden grafen i dette eksempelet har mer enn én forgrening er det flere lovlige lister. Eksempelvis kan en topologisk aktivitetsliste bestå av: grave kjeller, lage grunnmur, lag stenderverk, lag takbjelker, legg tak, ha kranselag, lag kledning, lag innervegger, overtakelse.[1] Flere aktiviteter kan naturligvis også pågå samtidig, hvis de ikke er i veien for hverandre. I følge grafen er det mulig å lage takbjelker samtidig med innevegger. Hvis det imidlertid er fare for at noen som arbeider med innevegger kan få ting i hodet, kan dette indikere at en av disse aktivitetene bør vente på den andre og nye avhengigheter dannes.

Hver aktivitet fordrer en viss mengde ressurser, f.eks et antall dagsverk og et minimumsmannskap, som gjør det mulig å beregne hvor lang tid som brukes på hvert delprosjekt, når delprosjektet er ferdig, og eventuelt hvor effektivt arbeidsstokken utnyttes (minimalisert *slakk*).

Det er mulig å tenke seg et scenario der de forskjellige oppgaver utføres av forskjellige firmaer, noe som er svært vanlig i byggebransjen. Arbeidsstokken blir da ikke å regne som en til enhver tid tilgjengelig ressurs, men mer avhengig av hvilke andre prosjekter hvert firma er engasjert i. Dette gir altså lokal kontroll over når og hvor mange fagfolk hvert firma kan bidra med i prosjektet. Hvert firma kunne da administrert «sitt» objekt i grafen i Figur 1.5 i sitt edb-system, mens selve prosjektstyringssystemet, som formodentlig befinner seg hos oppdragsgiveren, kan kontakte objektene og utforme prosjektplanen

[1] Det er mulig at Figur 1.5 ikke yter Kranselaget rettferdighet med hensyn på dets betydning og plassering i arbeidprosessen.

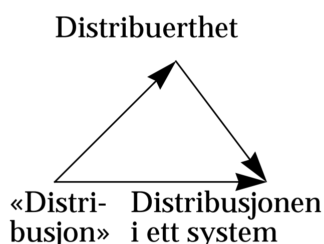


Figur 1.6: Distribuert avhengighetsgraf for husbygging

eller kjøre simuleringer slik som i oppgaven i IN115. Forskjellen er bare at objektene i datastrukturen nå er distribuert, i og med at firmaene gjerne er fysisk plassert rundt i terrenget. Figur 1.6 viser en distribuert graf, der objekter for hver arbeidsoppgave befinner seg hos og vedlikeholdes av hver underleverandør.

Distribusjon og arkitektur

Ingen avhandling er vel komplett uten en egen tolkning av Ogdens Trekant. (Se f.eks [Kaasbøll 96]:56.) Figur 1.7 viser en modifisert utgave. I forbindelse med behandling og diskurs rundt begrepet «distribuert» kan ordet spille flere ulike roller. Jeg tenker meg at *uttrykket* «distribusjon» har intensjonen *distribuerthet*, dvs det å være distribuert; de egenskaper som følger av å ha en grad av distribuerthet over seg. *Begrepet* eller *intensjonen* Distribuerthet inviterer til kontemplasjon rundt tema som samtidighet, konsistens (i data), sikkerhet, standarder, metoder osv, mens ekstensjonen av denne intensjonen manifesterer seg — eller projiserer seg — som *topografien* til ett konkret edb-system, nemlig det som skal utvikles. I tillegg til den fysiske plasseringen av nodene i systemet, kommer beslutninger som angår de generelle distribusjonskonseptene fra intensjonen i Figur 1.7, så som samtidighet, sikkerhet, feiltransparens etc. Mange av disse emnene er behandlet mer generelt i referansemodellen for distribuerte systemer RM-ODP (se Kapittel 3: «Beskrivelsesteknikker og modeller») og i standarder for kommunikasjon, f.eks CORBA (se Kapittel 4: «Oss objekter imellom»). Konteksten som uttrykket «distribuert» brukes i bestemmer altså hvilket av de to hjørnene til høyre i figuren som til en hver tid henger sammen med uttrykket som brukes.

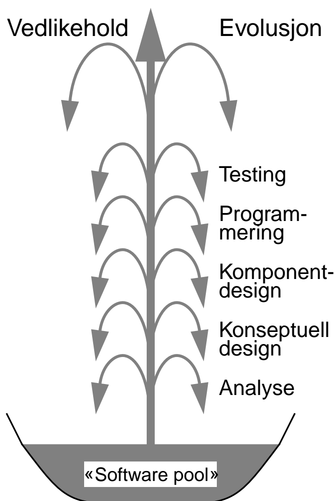


Figur 1.7: Modifisert Ogdens Trekant

Analogt med skillet mellom «distribuerthet» og konkret distribusjon fra Figur 1.7 er det mulig å skille mellom «architectural framework» og «architecture». Et *arkitekturmessige rammeverk* definerer og foreslår standarder, retningslinjer, grensesnitt og kontekst for strategiske valg av teknisk implementasjon. Et konkret prosjekteringsprosjekt resulterer i en spesifikk design, som kalles *arkitektur* [[Schulz 95].

Idealer i systemutvikling

For å kunne ta stilling til om konsepter som presenteres er gode eller dårlige, trengs et passelig sett med idealer. Som alle andre idealer opp gjennom tidene er idealene i informasjonsteknologien og utvikling ev edb-systemer vekslende, iallefall tilsynelatende. For tiden er det mulig, gjennom aktuell litteratur, å identifisere noen av de mest moderne:



Figur 1.8: Fontenemodellen (etter [Henderson-Sellers & Edwards 93])

Gjenbruk. Gjenbruk blir viktigere jo større utviklingsprosjektet er. Objektorienterte metoder tilrettelegger for gjenbruk ved at det er mulig å arve klasser som er laget tidligere. I «fontenemodellen», som presentert i [Henderson-Sellers & Edwards 93], brukes endel av ressursene i systemutviklingen til å identifisere hvilke komponenter eller klasser som kan være nyttige i flere sammenhenger enn det aktuelle prosjektet, og disse tilpasses og generaliseres slik at de kan gjøres tilgjengelig for senere bruk. Figur 1.8 viser en adapsjon av denne modellen. For å kunne utvikle edb-systemer raskt nok, med såkalt Rapid Application Development (RAD) er utviklerne avhengig av å kunne bruke store biter av eget og andres tidligere arbeid om igjen. På samme måte som det ikke er praktisk å finne opp bensinmotoren hver gang en ny bilmodell skal utvikles, er det sløsing med ressurser å skrive all programkode fra begynnelsen.

Vedlikeholdbarhet. Enda viktigere enn gjenbruk er vedlikeholdbarhet [Haythorn 94]. Selv om gjenbruk virker som et ideale med store potensielle gevinstmuligheter, er det for mange beslutningstakere vanskelig å se for seg de direkte fordelene ved å designe objektmodeller for gjenbruk. Positiv avkastningen ligger gjerne flere år frem i tid og til da kan mye ha forandret seg. Ved å forsøke å se for seg mulige forandringer — forutse det uforutsigbare — vil det være mulig å lage modeller som er mer generelle, og lettere å forandre. Slik identifiseres objektklasser som ikke umiddelbart finnes i problemområdet, slik som køer, hendelser, kommandoer etc [ibid].

Plattformnøytralitet. Ettersom flere og flere edb-systemer kobles sammen, og brukere forventer en nogenlunde homogen og gjenkjennelig oppførsel uavhegnig av produsent og maskinvareplattform, er det et ideale at systemer kan benyttes i forskjellige miljøer. Til enhver tid vil det rase religionskriger om hvilke plattformer som er best, så det er helt vesentlig at forskjellige systemer kan kjøres sammen, uten at lokale beslutninger umuliggjør dette. Eksempelvis kan det være nødvendig til å tillate at Macintosh- og Windowsbaserte systemer fungerer i samme domene, hvis de forskjellige lokale grupper selv

har valgt sine løsninger, og ønsker å stå ved disse. En del tjenester kan fungere for flere plattformer samtidig, f.eks systemer for filhåndtering, utskrift og elektronisk posthåndtering i nettverk. Java-baserte applikasjoner er andre eksempler på reslisering av plattformnøytralitet.

Skalérbarhet. Det er alltid vanskelig å spå fremtiden, spesielt når det gjelder hvordan folk tar i bruk informasjonsteknologi, eller ny teknologi generelt. Samme gjelder også graden av IT-bruk, og dermed også behov for ressurser. Skal edb-systemer overleve så lenge at de betaler seg inn, må de derfor ta høyde for utvidelser, også i ikke-forutsette retninger.[1]

Disse idealene og mange flere synes å være blant drivkreftene og er retningsgivende for utvikling av metoder og verktøy for moderne systemutvikling. De nye standarder og produkter som har kommet i de seneste årene, f.eks Java og CORBA, forsøker alle på en eller annen måte å følge opp disse idealene, og løse problemer med tidligere systemutviklingsformer, så som prosedyreorientert programmering, fossefallsmetodikk osv.

Case

For å illustrere og eksemplifisere poeng, metoder og løsninger i senere kapitler, har jeg valgt å bruke et aktuelt case. Våren 1996 utførte forskere fra Systemarbeidgruppa ved Institutt for Informatikk ved Universitetet i Oslo en analyse av behov og muligheter for langsiktig IT-strategi for Plan- og bygningsetaten i Oslo Kommune (PBE).[2] Et tidlig resultat av dette arbeidet finnes i [Greenbaum et al 96]. Etatens har i dag i bruk en rekke edb-systemer, størrelsesorden 50 [Kaasbøll 96c]. Blant disse er systemer som saksbehandling/arkiveringsstøtte (stormaskin), saksbehandling/teksbehandling (arbeidsstasjoner/PC), kartsystemer (arbeidsstasjoner/UNIX), lønnning-, regnskap- og andre administrative systemer. Etaten kan ha store fordeler av en viss integrering mellom flere av disse systemene. I tillegg vil stadig flere av etatens samarbeidende aktører og brukere *utenfor* ha nytte av integrasjon. Eksempelvis har mange arkitekter og utbyggere tatt i bruk Dataassistert Konstruksjon (DAK) i prosjektering, og kan ha glede av underlagsdata fra etatens datakilder, f.eks i

[1] Eksempelvis kan nevnes at omgivelsesvariabelen i MS-DOS baserte systemer som holder søkestien er sterkt begrenset i lengde, slik at antall kataloger med program- og annen binærkode (f.eks dynamiske biblioteksfiler) er begrenset. Konsekvensen av dette er at det er vanskelig å håndtere større antall binærfiler, f.eks mange forskjellige programmer. Dette begrenser altså skalérbarheten drastisk.

[2] «<ARC>» prosjektgruppen bestod av Joan Greenbaum, Jens Kaasbøll, Ole Smørdal og Leikny Øgrim.

form av kart, til sine prosjekter. Man kan også tenke seg at andre aktører, f.eks offentlige utvalg, kommuneadministrasjon og andre offentlige organer kan nyttiggjøre seg tettere elektronisk kommunikasjon med etaten. Et eksempel på dette er at ansatte i rådhuset som har ansvar for å legge frem reguleringssaker for politikere i utvalg. I intervju som forskere fra <ARC>-gruppen gjennomført med ansatte i Rådhuset har det kommet frem at de i møter med politikere ofte får spørsmål om andre ting enn det de hadde forberedt seg på, f.eks andre reguleringssaker i andre områder av byen. Dette kunne vært støttet med bærbare datamaskiner med tilgang til kartdatabaser og reguleringsvedtak osv.

Jeg tror etaten før eller senere må ta stilling til distribusjon, og det er kun et spørsmål om å *definere* et skille mot omverdenen og dermed bestemme seg for hva som er «her» og hva som er «der».

Eiendomsinformasjon

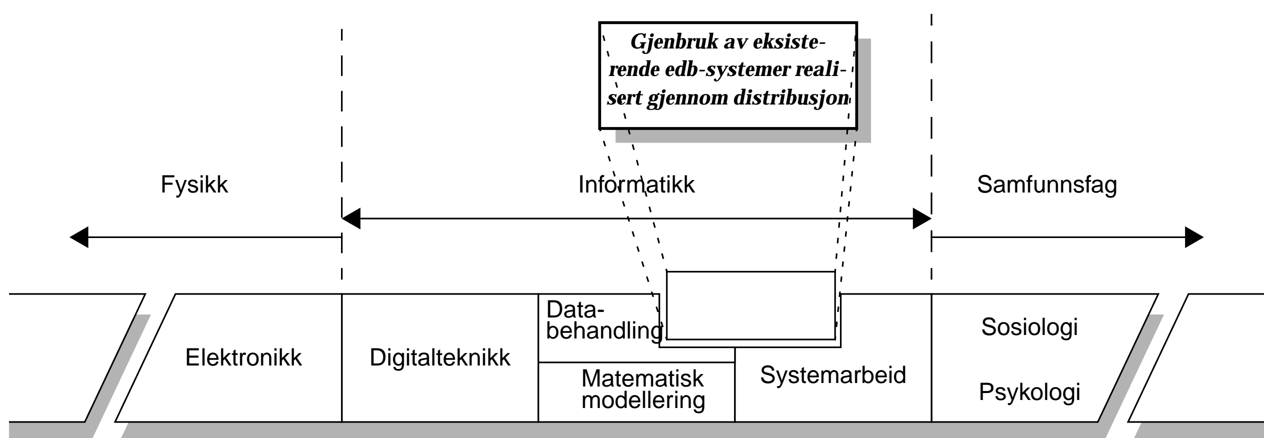
I Kapittel 5: «Metoder for utvikling av distribuerte systemer» vil jeg blant annet redegjøre for *OOram* (Object Oriented Rôle Modeling) metoden og hvordan den kan brukes på distribuerte systemer. I OOram bruker man begrepet *Area Of Concern* (AoC) for å dele opp problemområdet i passelige biter [Reenskaug et al 96]. Som et gjennomgående eksempel i denne oppgaven vil jeg ta utgangspunkt i et konkret problem (AoC); *å finne eieren av en eiendom*. Når ny garasje skal bygges til huset, ny bygning i det hele tatt, trengs tillatelse av Plan- og bygningsetaten. En av forutsetningene er at naboene har fått varsel, og godtatt planen. Slik det er nå må man henvende seg til etaten og få en liste over de som skal ha varsel. Det trenger ikke nødvendigvis være den som bor i nabohuset som skal finnes, men den som eier det, og det trenger jo ikke være den samme.

Etaten har flere datakilder som brukes for å finne eiere og dermed naboer. Først må man se etter på et kart over eiendommen, og finne ut hvilke eiendommer rundt som har rett til å uttale seg. Dette reguleres av Plan- og bygningsloven og har noen med avstander og annet rask. (finn ut dette). Hver eiendom som er registrert i kartdatabasene har et referansepunkt med et tilhørende koordinatsett. Dette kan man så bruke som nøkkel til å slå opp i GAB[1]-registeret, og slik finne ut hvem som er rette mottaker for nabovarsel. Scenarioet som er skissert ovenfor kunne godt vært en interaktiv elektronisk tjeneste som etaten tilbyr publikum. Dette er utgangspunktet for eksempelapplikasjonen presentert i Kapittel 6.

[1] Grunneiendom Adresse Bygning

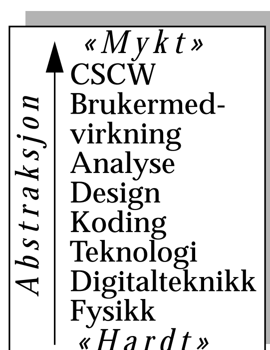
Er dette systemarbeid, da?

Jeg har brukt mye tid på å gruble på problemet *Hva er systemarbeid?* Ved Institutt for Informatikk ved Universitetet i Oslo, hvor denne oppgaven er levert, finnes for øyeblikket fire studieretninger; Digitalteknikk, Matematisk modellering, Databehandling og Systemarbeid. Hvis vi tenker oss at vi kunne plassere disse fagområdene langs en skala — eller et spekter — hvor den ene siden (digitalteknikk) berører «elektronikk-enden» av fysikkfaget, mens den andre siden (systemarbeid) berører sosiologi[1] og psykologi[2] i det samfunnsvitenskapelige domenet, vil jeg plassere denne oppgaven i grenselandet mellom Databehandling og Systemarbeid. Figur 1.9 viser en slik plassering.



Figur 1.9: Plassering av denne hovedoppgaven innen et spekter fra fysikk til samfunnsfag (Merk at det ikke ligger noen intendert rangering i f.eks legitimitet eller matnyttighet mellom disiplinene etter plassering eller tildelt boksareal)

Det er selvsagt ganske dristig å hevde at det er mulig å plassere forskjellige vitenskapsdisipliner slik i én dimensjon, men for visse formål fungerer en slik modell.



Figur 1.10: Noen tema fra praktisk systemutvikling og deres abstraksjonsnivåer.

Slik jeg ser det, består systemutviklingsarbeid i «det virkelige liv» av konstruksjon og beslutninghandlinger langs store deler av spekteret fra de «harde», nederste nivåene, hvor jeg ser for meg at maskinvare og programmering befinner seg — opp til de «myke» og mer overordnede konseptuelle modeller, og systemarbeidsdisipliner så som teori om brukermedvirkning, prosjektstyring, systemutviklingsmodeller, teorier om teknologibruk osv, som vist i Figur 1.10. Denne oppfatningen har jeg fra tidligere hovedfagsstudenter og deres erfaringer etter å ha begynt i praktisk arbeid i norske edb-bedrifter etter utdan-

[1] Mye av de samme metodene benyttes i systemarbeidsforskning og sosiologi, f.eks intervjuundersøkelser.

[2] Brukergrensesnittteori innbefatter emner fra psykologi, f.eks persepsjonsteori. Se f.eks [Dix et al 93].

nelse. I denne oppgaven vil jeg derfor forsøke å forholde meg til en forholdsvis stor del av dette spekteret, for å sikre at konklusjoner ikke låses i urealistiske forutsetninger, iallefall ikke uten å ha vurdert dem først. Dette forklarer det tilsynelatende innfløkte mønsteret i Figur 1.1 og de tre grensene i Figur 1.2.

Innen faget Systemarbeid tenker jeg meg at denne oppgaven heller mot systemutviklingsmetodikk, til fordel for brukergrensesnitt, IT-bruk og andre aktuelle tema i systemarbeidsdisiplinen.

Forskningsmetode

Denne oppgaven baserer seg hovedsaklig på kvalitative litteraturstudier rundt distribuerte objekter og objektorientert metodikk. Det finnes mye litteratur rundt objektorientering og OO-metoder, men satt litt på spissen gjør de aller fleste implisitte antakelser om at 1) man skal lage et helt nytt datasystem der intet var fra før og 2) den metoden som presenteres er en syntese av alle de andre og følgelig fremtidens metode. Det finnes følgelig ikke fullt så mye litteratur om metoder for å integrere eksisterende systemer med nye. Jeg har forsøkt å finne frem til litteratur som gjør noe med antakelsene 1) og 2) ovenfor. Slik har jeg kommet frem både til litteratur som tar opp metoder og distribusjon fra et teoretisk ståsted, som for eksempel litteratur om referansemodellen for distribuerte systemer; RM-ODP,[1] og stoff som behandler mer praktiske løsninger og erfaringer, slik som litteratur rundt CORBA og DCOM.

Siden distribuerte objekter med tilhørende teknologi og infrastruktur er forholdsvis nytt, vil mye av litteraturgrunnlaget være ganske ferskt og/eller visjonært. Mye av dette består av artikler av varierende lengder, og en skuffende liten del omhandler *erfaringer* med den teknologien og de metoder som presenteres i de mer visjonære artiklene.[2] Likevel finnes det samlinger med erfaringsmateriale fra praktiske prosjekter som er basert på objektteknologi, f.eks [Harmon & Morrisey 96], [Hetland 97]:115–157, [Brando 94].

I tillegg til litteraturstudier har jeg prøvd å orientere meg litt i markedet for mellomvareteknologi, og dessuten testet OORAM PROFESSIONAL, som er et CASE-verktøy.[3] Dette verktøyet kan generere kildekode i C++ og Smalltalk, men ikke mot f.eks ORBIX eller andre mellomvareprodukter for objektkommunikasjon. Jeg har derfor nøyd meg med å modellere eiendomsinformasjonssystemet som nevnt tidligere på et mer overordnet plan — eller altså i RM-ODP's «Information Viewpoint». (Se Kapittel 3.) Jeg har laget en prototyp i Java og

---

[1] Reference Model for Open Distributed Processing (se Kapittel 3)

[2] Bare tre av 22 objektorienterte metoder ble i en undersøkelse i 1992 regnet som «kommersielt brukbare» [Dock 92].

[3] Computer-Assisted Software Engineering

C++ av informasjonstjeneste for eiendomsopplysning som omtalt tidligere. Protoypen har to hovedformål:

- Evaluere metoder for utvikling av distribuerte systemer, som presentert i Kapittel 5;
- Få et bilde av ressursbruk ved applikasjonslogikkutvikling; både nyutvikling og endringer.

Ettersom et virkelig utviklingsprosjekt ville vært svært omfattende og således sprenge de tidsmessige rammer denne hovedfagsoppgaven er plassert i, har prosjektet tatt form av et rent laboratorieforsøk. Selv om det konkrete prosjektet er i en forholdsvis beskjeden skala og med begrenset nytteverdi, har det i allefall gitt *meg* en mer håndfast erfaring med det paradigmet som tilbys i distribuerte objekter. Mer konkret demonstrerer prototypen også hvordan flere typer teknologier (Java, C++) kan samarbeide tett.

Sammendrag

I dette kapitlet har jeg gjort rede for bakgrunnen for oppgaven, tema og valgt avgrensning; oppgaven skal bevege seg innenfor et rom avgrenset av objektorientert metodikk og -metoder, en valgt mellomvareteknologi (CORBA) og ønske om å integrere eksisterende systemer i de metoder og den teknologi som er nevnt ovenfor. Problemstillinger fra Plan- og bygningetaten i Oslo Kommune vil i noen utstrekning bli brukt som case utover i oppgaven.

Siden tema for denne oppgaven omfatter miljøer der eksisterende systemer skal integreres med hverandre og nye, er det naturlig å starte med å se på problemstillinger rundt gjenbruk av eksisterende systemer. Kapittel 2: «Arven fra fortiden» tar for seg dette. For å kunne diskutere nytt og gammelt i et distribuert samvirkende hele, trengs felles begrepsapparater. Dette presenteres i Kapittel 3: «Beskrivelsesteknikker og modeller». Det viktigste stoffet i denne oppgaven omhandler metoden OOram, med utvidelser for utvikling av distribuerte systemer. dette presenteres i Kapittel 5: «Metoder for utvikling av distribuerte systemer». For å konkretisere prinsippene fra Kapittel 3 og 5 vil jeg gjøre rede for noen infrastrukturer for objektkommunikasjon i Kapittel 4: «Oss objekter imellom». Kapittel 7 oppsummerer og identifiserer behov for videre arbeid, mens Kapittel 6 presenterer en prototyp for distribuert kartoppslag og eiendomsinformasjon. Tillegg C presenterer den viktigste kildekoden til prototypen, for spesielt interesserte.

2 *Arven fra fortiden*

Innledning

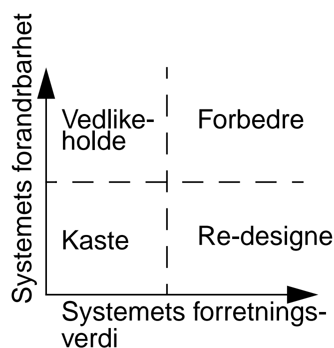
I 1989 utførte Timo Saarinen og Jukka Heikkilä en undersøkelse av systemutviklingsprosjekter i et representativt utvalg av finske bedrifter for å se hvordan systemutviklingsmetoder og -verktøy ble påvirket av typen prosjekter som ble realisert. I [Saarinen & Heikkilä 90] plasseres disse utviklingsprosjektene i tre kategorier etter prosjektenes mål:

- Utvikle nytt system der det ikke var noe fra før;
- Forbedre eksisterende systemer;
- Bytte ut eksisterende systemer.

Som nevnt i «Premisser for oppgaven» på side 4 skal jeg stort sett gå ut fra forhold hvor det allerede finnes edb-system(er) i bruk, så vi ser i det videre bort fra den første kategorien ovenfor. Etter en kombinatorisk og kvalitativ komparativ analyse kommer Saarinen og Heikkilä blant annet frem til følgende:

- Ved **forbedring av eksisterende systemer** er ferdige programpakker sjelden vellykket og grensesnitt til eksisterende komponenter er vanskelig å utvikle.
- Ved **utskiftning av eksisterende systemer** (spesielt ved anskaffelse av ferdige programpakker), er organisasjonens egen IT-avdeling en viktig ressurs. Eksisterende systemer kan fungere som kravspesifikasjon og gi et omfattende felles begrepsapparat for både systemutviklere — i denne sammenheng gjerne organisasjonens eget IT-personale — og nåværende og fremtidige brukere.

Uansett hvilken av de gjenværende to muligheter som velges, må det tas stilling til hva som *er*, og hva som *skal bli*. En pekepinn for hvilken overordnete strategi som bør inntas er presentert overflatisk i [Jacob-



Figur 2.1: Strategimatrise (etter [Jacobson & Lindström 91])

son & Lindström 91], gjengitt i Figur 2.1. Jeg vil i dette kapittelet ta for meg eksisterende edb-systemers natur og hvordan man kommer seg videre ved hjelp av re-design. I Figur 2.1 vil det si eksisterende *virksomhetskritiske* systemer som det er vanskelig å forandre. Først vil jeg imidlertid gjøre rede for noen viktige begreper, blant dem det som nettopp ble introdusert; eksisterende virksomhetskritiske systemer, eller *«mission critical legacy systems.»*

«Mission critical legacy systems»

I faglitteraturen som tar for seg emner som *distribuerte systemer, objektorienterte metoder og re-engineering* eller kombinasjoner av disse[1] finnes begreper som *«mission critical»* og *«legacy systems»*. Det er ikke til å komme fra at mye av behovet for systemutvikling i dag ikke er å lage helt nye systemer fra grunnen av der intet finnes fra før, men å forbedre gamle, eller tilpasse gamle systemer til nye omgivelser og krav.[2] Krav i denne sammenheng kan være relatert til:

- Ny funksjonalitet;
- Bedre funksjonalitet, f.eks bedret responstid og sikkerhet;
- Lover og regler utenfra eller innenfra organisasjonen;
- Brukervennlighet.

Noen av disse kan i praksis selvsagt være årsaker til hverandre, slik at f.eks nye lover medfører at ny funksjonalitet må innføres. Et eksempel på dette finnes i [Carlson et al 96]; ny forsikringslovgivning i England i 1995 dikterte endel nye krav til opplysninsplikt for forsikringsselskaper med hensyn på konkurrentenes betingelser og priser. Et annet eksempel er den nye plan- og bygningsloven som tredde i kraft 1. juli 1997 i Norge som blant annet innebærer at en større del av kontrolloppgavene i forbindelse med gjennomføring av byggeprosjekter pålegges utbyggerne. Dette innebærer et godkjennings- og klassifiseringssystem for de forskjellig aktørene, bl.a for arkitekter. Dette må formodentlig støttes av ett eller flere edb-systemer. Endringene i plan- og bygningsloven vil høyst sannsynlig også medføre endel endringer i Oslo kommunes Plan- og Bygningsetats databaseskjemaer såvel som ønsket funksjonalitet i etatens øvrige edb-systemer. I tillegg til nye krav skal ofte de nye systemer ivareta de gamle kravene på en tilfredsstillende måte, uten å gå på akkord med kvaliteten.

---

1. Se f.eks [Dietrich et al 89], [Harmon & Morrisey 96], [Jacobson et al 94], [Taylor 95].
2. I [Saarinen & Heikkilä 90] antydes at bare en firedel av prosjektene lages fra to (eller et vilkårlig partall) tomme hender.

Andre årsaker til krav om ny funksjonalitet i edb-systemer kan være organisasjonsforandringer eller forandrede driftsprosesser, f.eks som resultat av BPR-prosesser.

Virksomhets-kritiske systemer

Begrepet «virksomhetskritiske systemer» (mission critical) benyttes ofte for å trekke skillelinjer mellom (deler av) systemer etter hvilken betydning de har for organisasjonens daglige drift. Med *virksomhets-kritisk* menes gjerne f.eks. kontohåndteringssystemer i banker og finansinsititusjoner, dataregistere til f.eks Folkeregisteret etc. Hvis et virksomhetskritisk system settes ut av drift vil organisasjonens kjernevirksomhet rammes i stor grad. Hvis kjernevirksomheten er å administrere kundenes inn- og utbetalinger, er det rimelig å anta at f.eks en bankkasserer ikke kan utføre sine daglige arbeidsoppgaver tilfredsstillende dersom skrankesystemet er ute av drift.

Eksempel på *ikke-virksomhetskritiske* systemer kan være informasjonstjenester, f.eks web-tjenere eller støtteverktøy for betjeningen i en informasjonsskranke (hvis ikke organisasjonens kjerneområde er publikumsinformasjon). Grensene mellom hva som forstås med *virksomhetskritiske* og *ikke-virksomhetskritiske* systemer er imidlertid flytende. En web-tjeneste vil være et virksomhetskritisk system i en bedrift som selger annonseplass på sine web-sider, mens en web-tjener som annonserer for bedriftens egne produkter ikke vil være det. Tilsvarende er ikke beregningsapplikasjoner for lån til bruk i kunderådgivning virksomhetkritisk for en bank, men det er kontosystemene.

Naturligvis stilles gjerne strengere krav til utvikling, testing og drift av virksomhetskritiske systemer enn ikke-virksomhetskritiske systemer. Mer konservative bransjer, som f.eks bank og forsikring, har derfor være lite villige til å satse på objektorienterte metoder og teknologi, siden den er såpass ny og ofte beskyldes for å være mer akademisk og forskningsorientert enn mer etablerte metoder. En betingelse for å kunne ta i bruk objektorientering i større skala i disse sammenhengene er derfor at kvaliteten opprettholdes eller bedres, blant annet med hensyn på sikkerhet.

«Legacy systems»

> *«Typically the information systems of big organizations are large (10s of millions of lines of code), old (more than 10 years old) and have evolved in an unstructured manner with extensions added in different languages and with minimal, if any, documentation available. In addition these systems are mission-critical and should remain functional at all times. These information systems define what we today call* ***legacy information systems****.»*
>
> *[Konstantas 96]*

Definisjonen av «legacy» systemer ovenfor viser noen av de problemer man står overfor i systemutviklingsprosjekter der det finnes store

edb-systemer i bruk fra før. Slike systemer inneholder ofte forretningsregler som representerer store verdier for organisasjonen. Hvis kildekoden til disse delene er mangelfullt dokumentert, noe de nok ofte er, kan det være vanskelig å skille ut og/eller reimplementere dette på nytt [Ning et al 94]. [Konstantas 96] identifiserer videre tre viktige problemer:

- Systemer kan ofte ikke forandres på en overkommerlig måte for å gi ny funksjonalitet.
- Det er nesten umulig å flytte gamle systemer til nye plattformer. Hvis det i det hele tatt er mulig, blir det svært kostbart.
- Vedlikehold, feilretting og opplæring av personell blir stadig vanskeligere i takt med lokale tilpasninger og dårlig dokumentert utvikling. Typisk kan et virksomhetskritisk system gjerne virke tilfredstillende uten at noen vet hvorfor eller hvordan.

Som nevnt tidligere er det uungåelig i mange bransjer at forretningsdriften endrer seg. Det samme gjelder da informasjonssystemene. Uansett i hvilken grad systemene er virksomhetskritiske og/eller av eldre dato, vil de bli underlagt endinger i større eller mindre grad. Hvordan slike utfordringer kan takles vil jeg defror se på videre.

Strategier for videreutvikling

Som nevnt på side 24 vil nye krav kunne medføre initiativ til å lage et nytt informasjonssystem eller forbedre det gamle. Om det man ender opp med skal betraktes som noe helt nytt eller bare det gamle med forbedringer, er et filosofisk spørsmål. Vi kan etter eget ønske bruke det som måtte passe av kritierier for hva som er fornyet gammelt og hva som er genuint nytt. Noen forslag til faktorer som kan rettferdiggjøre brukt av termen «nytt system» kan være ny utviklingsmetode, ny teknologi, ny organisasjon, ny distribusjon, eller vesentlig nye krav eller funksjonalitet. Ser vi på det gamle som byggeklosser til det nye, vil jeg her velge å kalle det et *nytt* system. I tillegg til at dette er et filosofisk spørsmål, er det også gjerne en psykologisk faktor. Begrepsbruken kan føre til at motivasjonen opprettholdes under systemutviklingsperioden, spesielt for brukere som er involvert i prosessen. Slik kan man se frem til det som kommer som noe helt nytt og moderne, som forhåpentligvis ikke lider av de samme idiosynkrasier som de tidligere systemene led av. I det videre vil jeg derfor diskutere sider ved det å lage et nytt system som skal ivareta kravene fra det gamle på en tilfredsstillende måte.

Når en altså har valgt å lage et nytt informasjonssystem ut av et eksisterende er det tre strategier man kan velge:

1. Kaste det gamle systemet, lage et helt nytt og sette dette i drift;

2. Lage et helt nytt system som kjøres parallelt med det gamle helt til alt virker tilfredsstillende;
3. Forandre det gamle skritt for skritt ved å bytte ut og også lage nye helt nye delkomponenter ettehvert, og forsikre seg om at disse fungerer tilfredsstillende før de tas i produksjon.

Strategi 1: kast alt og begynn på nytt

Av de tre valgene ovenfor er det første det som er minst fristende å begi seg ut på, for det kan ta lang tid å utvikle et helt nytt system som skal erstatte ett som det har tatt mange år å komme frem til. Det er sjelden mulig å la bedriften eller deler av den ligge brakk til det nye systemet er ferdig, spesielt hvis det altså er et virksomhetskritisk system som skal fornyes. Vi ser derfor bort fra det alternativet i denne sammenheng.[1]

Strategi 2: lag nytt og kjør i parallell til alt virker

Det andre alternativet virker således noe mer aktuelt enn det første, men det medfører i praksis problemer når man skal kjøre systemer parallelt. Blant annet er det ofte umulig når systemene allerede kjører på heterogene platformer. [Konstantas 96] nevner to mulige årsaker til problemer i en slik kontekst; 1) grensesnittene mellom delsystemene er vanskelig eller umulig å forandre og 2) grensesnittene er ukjente eller udokumenterte. I mange tilfeller er det imidlertid aktuelt å gjøre noe med et terminalbasert stormaskinsystem (eksempelvis et banksystem). Her vil en vei å gå være å «kapsle inn» det game systemet med terminalemulatorer, slik at lokalnettmaskiner kan virke som terminaler. I følge [Ning et al 94] har dette noen fordeler:

- **Driftsikkerhet** - den originale funksjonaliteten ivaretas siden det er den samme koden som eksekveres, slik at alt virker som før.
- **Billig løsning** - det gamle systemet kjører på den samme maskinen, arbeidsstasjonenene er relativt billige i innkjøp.
- **Rask utvikling** - det er ikke behov for noen inngående analyse av problemområdet. Det ble jo gjort for lenge siden, da det gamle systemet ble laget.

Imidlertid har denne typen løsninger store begrensninger. Løsningen kan kun bruke hele den originale applikasjonen som den er, uten å kunne dele den opp. Ytelsen eller funksjonaliteten forbedres ikke, og dette er en lite fleksibel løsning. Hvis det er ønske om å tilfredstille nye krav til systemet, som nevnt på side 24, vil vi ikke denne strategien være den beste.

Imidlertid er det tenkelig at dette kan være et stadium i en gradvis utskifting av komponentene i et system, hvis det originale systemet ikke allerede kjører i heterogent miljø som beskrevet ovenfor, men på stormaskin. Man kan begynne med innkjøp av arbeidsstasjoner med

[1] Om man da ikke er så heldig i starte med carte blanche, som beskrevet i [Biffl et al 96].

terminalemulering mot den opprinnnelige maskinen i første omgang, og slik innføre ny teknologi trinnvis. Men er det lurt å kjøpe arbeidsstasjonene først? Hvis man ikke oppnår annet enn å kunne legge kabal på arbeidsstasjonen samtidig med at man gjør som man gjorde før på en dum terminal, er gevinsten liten. Dessuten vil arbeidsstasjonen raskt bli teknologisk utdatert, kanskje lenge før de andre komponentene som skal rettferdiggjøre investeringen er ferdig utviklet.

Strategi 3: gradvis forandring av det gamle til noe nytt

Den siste muligheten er en gradvis innføring og utskifting av det gamle systemet, ved å gjenbruke delkomponenter bit for bit. [Jacobson & Lindström 91] og [Taylor 95] presenterer begge en prosess der deler av de eksisterende systemer omkapsles av objekter i en ny forretningsmodell, helt til det gamle er smuldret bort. [ibid] bruker som metafor et skall rundt det eksisterende system. Skallet blir tykkere og tykkere, samtidig som kjernen (det gamle) skrumper inn.

For å kunne kapsle inn eksisterende deler er man utvilsomt nødt til å identifisere hvilke biter man er interessert i å bevare for så å plukke ut disse og bruke dem på nytt. Her vil det de eksisterende edb-systemer opptre med ulik granularitet, avhengig av i hvilken grad de passer i den Konseptuelt *reverseres* altså konstruksjonsprosessen. (*Reverse Engineering*) Strengt tatt er det ikke nødvendigvis prosessen slik den var originalt som kommer frem i en slik reverskonstruksjon, men en *pseudokonstruksjonsprosess*, som når den blir kjørt forover igjen vil kunne gi nogenlunde samme resultat som den opprinnelige prosessen gjorde. Den opprinnelige kan jo ha tatt flere år, vært mer eller mindre strukturert og være bygget på et stort praktisk erfaringsmateriale som er erhvervet gjennom mange års vedlikehold og drift. I det videre vil jeg se på hvilke muligheter som finnes for reverskonstruksjon.

Reverskonstruksjon — reverse engineering

> *«Reengineering is the process of creating an abstract description of a system, reason about a change at the higher abstraction level, and then re-implement the system»*
>
> *[Jacobson & Lindström 91]*

En bedrift eller organisasjon som ønsker å overleve i et konkurransepreget miljø i fremtiden må være fleksibel [Taylor 95]. Den må kunne tilpasse seg skiftende krav fra omgivelsene, for eksempel fra kunder og marked. En omorganisering av driftsprosessene vil også gjelde informasjonssystemene som brukes [Jacobson et al 94] (sjekk). En «reengineering» av arbeidsprosessene vil følge hånd i hånd med en

reengineering av informasjonssystemet. Selv om driftsprosesser ikke nødvendigvis er trivielle å forstå, er det sannsynlig at informasjonssystemet vil være enda mer komplisert, sett fra et programteknisk synspunkt. For å endre et informasjonssystem som følge av endrede driftsprosesser, fordrer at også systemet analyseres og forstås. En reverskonstruksjonsprosess vil kunne vise hvilke komponenter systemet består av slik det fremstår på det tidspunkt denne prosessen startes. Et eksempel på en del av en slik prosess er det arbeid som er gjort i analysen av Plan- og bygningsetaten i Oslo (se [Kaasbøll 96b]), hvor datadefinisjoner fra flere av etatens systemer er systematisert. Dette har gitt en global datamodell for hele etaten som kan brukes til videre arbeid med forbedringer av IT-løsningene der.

Reverskonstruksjon kan foregå på flere abstraksjons- eller detaljeringsnivåer. I praksis vil flere nivåer behandles, avhegnig av de eksisterende systemkomponenters art. Fra lavest til høyest abstraksjonsnivå har jeg valgt å rangere tre reverskonstruksjonsformer slik: *programforståelse*, *modellforståelse* og *bruksforståelse*.[1] Disse tilsvarer grovt henholdvis implementasjon, design og analyse fra en vanlig fossefallsmetode. Legg merke til her at disse fasene er nevnt i motsatt rekkefølge av «klassisk» fossefall. Dette kan tas som indikasjon på at det vitterlig *er* en reverskonstruksjon. Imidlertid ser jeg ingen grunn til å ta fatt på et reverskonstruksjonsprosjekt i nødvendigvis denne (omvendte) rekkefølge, spesielt ikke der hvor iterative utviklingmodeller er aktuelt. I en sammenheng der fremtidig fleksibilitet skal ivaretas, er det nødvendig at vedlikeholdere og systemutviklere inntar en iterativ eller evolusjonær holdning (finn referanse her!). Likevel vil jeg presentere de tre reverskonstruksjonsformene i stigende abstraksjonsnivå — eller logisk redesignrekkefølge om man vil.

Program slicing — *kodeslakt*

Rent teknisk kan en reverskonstruksjon av eksisterende applikajsoner bestå av såkalt *program slicing*. Kildekoden for en applikasjon parseres og parteres med et passelig verktøy, og det er mulig å f.eks forfølge interessante variabler og alt som skjer med dem, plukke ut instruksjonenene i prosedyrer eller funksjoner og slik dele opp et program i passelige biter som tilsammen gjør en delmenge av det originalprogrammet gjør. Dette kan så reimplementeres, f.eks i et annet språk eller på en annen plattform enn den originale. Denne tankegangen kalles i [Ning et al 94] *resuable component recovery* (RCR). Konsulenter ved *Anderson Consulting's Center for Strategic Technology Research* (CSTaR) har utviklet metoder og verktøy for RCR; blant annet *Cobol/SRE* (Cobol System Renovation Environment) som er et verktøy for å reverskonstruere COBOL-baserte systemer. [Hoffner 96] og [Killingberg 94] beskriver andre verktøy for *program slicing*.

[1] I problemgrupperingen presentert i [Bjerknes et al 91]:5 faller de to første reverskonstruksjonsformene i «**FIREworks**»-gruppen, mens den siste tilfaller «**FIREengine**».

«Screen scraping» og terminal-emulatorer

Program slicing og reverskonstruksjon av eksisterende kildekode er en forholdsvis dyptpløyende prosess. Et alternativ er å bruke såkalte «screen scraper» eller terminalemulatorer. Prinsippet er at et program overfor en applikasjon later som det er en bruker ved en dum terminal, og leser (skraper) av skjermbilder som applikasjonen genererer og fôrer den med kommandoer tilpasset brukergrensensnittet. Spesielt i tilfeller der applikasjonen er proprietær, kildekoden er hemmelig, eller så spagettipreget at det er vanskelig å reverskonstruere, er terminalemulatorer nyttige. Teknikken gir også et forholdsvis konsentrert og «smalt» grensenitt mellom det nye og det gamle, en slags ny fasade.[1] Generelt fungerer skjermemulatorer ved at skjermbilder mellomlagres i en buffer, som så eksamineres av emulatoren. Aktuelle skjermbilder beskrives i et passelig språk, f.eks AUIDL,[2] og programmet kan så gjenkjenne dem og hente og plassere informasjon i dem [Elwahidi & Merlo 95]. Flere produkter finnes i dag for terminalemulering og screen-scraping, gjerne tilpasset IBM stormaskinmiljøer, for eksempel AS/400 maskiner og 3270-terminaler.[3]

Imidlertid er det rimelig at utvikling og vedlikehold av driftsprosesser og tilhørende informasjonssystemer følges best ad hvis de modelleres og vedlikeholdes i beslektede paradigmer. Hvis derfor en objektorientert driftsprosessutvklingsmetode er valgt (som beskrevet i [Jacobson et al 94] eller [Taylor 95]), vil det være naturlig å velge ditto systemutviklingsmetode. Slik vil datamodellene (som jeg kommer til om senere) i de to perspektiver — dvs drift/arbeidsprosesser og informasjonssystem — være mest mulig konsistente. Jeg vil derfor i det følgende se på hvordan reverskonstruksjon av informasjonssystemer betjenes av objektorinterte metoder.

Objektorientert reverskonstruksjon

Ett utgangspunkt for objektorientert reverskonstruksjon (se også side 72) er å finne i [Gall et al 95]. Metoden som er presentert her tar sikte på å reimplementere kildekoden i det eksisterende system ved hjelp av objektorienterte metoder og teknologi. Dette gjøres ved å analysere eksisterende programkode og syntetisere en objektmodell ut fra prosedyrer og variabler i originalkoden. Gall, Klösh og Mittermeir kaller dette en *Reversely generated object-oriented Application Model* (RooAM). Det viktigste med metoden er at man i tillegg til denne analysen, som foregår mer eller mindre automatisk,[4] kompletterer denne modellen med en modell laget sammen med eksperter på problemområdet, *Forward object-oriented Application Model* (FooAM).

---

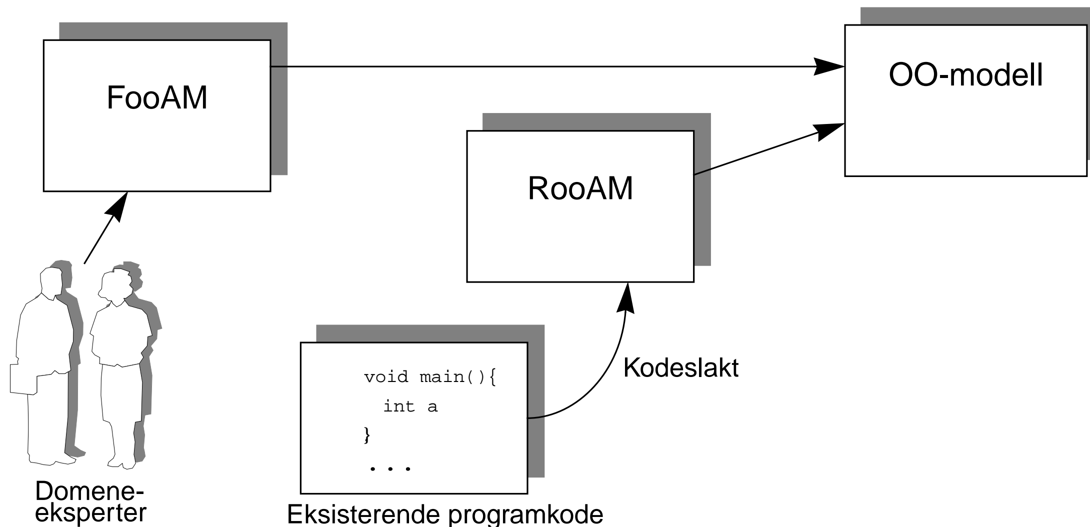
1 Et «fasademønster», som presentert i [Gamma et al 95]:185–193, kan brukes for å illustrere innkapsling og ordning av flere mer eller mindre definerte grensesnitt til ett slik som en screen scraper. Mønsteret passer imidlertid like bra for en tilsvarende oppgave ved program slicing.

2 Abstract User Interface Language

3 Se f.eks `http://www.conextions.com/`, `http://www.gesoft.com/`

4 Grad av automatikk vil selvsagt være avhengig av programmeringsspråk og andre betingelser.

Begge modellene syntetiseres sammen til en ny objektmodell. (Se Figur 2.2) Slik kan man ta vare på intensjoner nedlagt i eksisterende kildekode, intensjoner som er mer eller mindre dokumentert samtidig som dette smeltes sammen med nye intensjoner eller kravspesifikasjoner. Kodeslakting alene altså ikke nok for å generere en fullstendig objektmodell, men også mennesker med kunnskap fra problem- og applikasjonsområdet.



Figur 2.2: FooAM-modellen syntetiseres sammen med RooAM-modellen til en total objektmodell (Forenkling av Fig.1 i [Gall et al 95])

Hvordan finner så denne metoden frem til hva som skal bli objekter? Problemet løses initielt ved å finne ut hvilke objekter som er de samme i FooAM-modellen og RooAM-modellen — rent semantisk ut fra domeneekspertenes kunnskap om problem- og applikasjonsområdet. Objektene kan gjerne ha forskjellige signaturer[1]/navn i utgangspunktet, fordi objekter og klasser i RooAM-modellen får sine navn og signaturer fra kildekoden, og preges dermed av konvensjoner i implementasjonsspråket og den aktuelle programmeringsstilen. Tilsvarende vil FooAM-modellen påvirkes på sin måte av analysemetoden og de konvensjonene systemutviklerne og problemområdeekspertene finner for godt å bruke i sin analyse. Selve *mappingen* mellom disse to signaturrommene må nok til syvende og sist utføres av mennesker, men metoden gir visse retningslinjer for hvordan dette kan gjøres.

Imidlertid vil ikke alle klasser finne sin «partner» i «den andre» modellen. Noen må da grupperes *inn* i andre objekter, eller de kan bli til

[1] Med «signatur» menes her det som entydig definerer objekter/klasser. Mer presist vil jeg tolke et objekts signatur som klassens navn pluss type (se side 10–11). I en modelleringskontekst som her vil imidlertid semantikken være mer sentral.

nye. Spesielt hvis kravene til systemet har endret seg siden koden ble laget er dette aktuelt. Det er hovedsaklig foroveranalysen — altså FooAM modellen — som gir slike nye tilskudd til objektparken.

Til sist på fatet ligger globale variabler og -funksjoner. Globale data kan modelleres som felles dataobjekter. Den siste rest av funksjoner modelleres som en *«functional reminder»*, som inneholder objekter av typen *«main»* (som f.eks representerer `main()` funksjonen i et C-program) og diverse husholdsobjekter, f.eks for sortering, søking etc.

På denne måten blir den gamle applikasjonen redesignet som en objektmodell, og det er nå mulig å føye til ønskelige forandringer i et nytt system. Nye og gamle deler settes sammen og får nytt liv. Ved å holde modellen konsistent med edb-systemet, spesielt ved bruk av CASE-verktøy, er det sannsynlig at vedlikehold blir lettere i fremtiden [Strand 96].

Objektorientering og databaser

Til nå har jeg sett på reverskonstruksjon av prosedyrebaserte programmer og applikasjoner, f.eks skrevet i C, Pascal eller COBOL.[1] Imidlertid er ofte en viktig komponent av eksisterende systemer databaser, og da gjerne relasjonsdatabaser.[2] Det er gjerne satset store summer på utvikling og vedlikehold av relasjonsdatabaser i mange bedrifter, og teknologien er nå forholdsvis moden og stabil. Objektorienterte databaser kan ikke vise til samme suksess ennå, selv om det er et «impedansmisforhold» mellom relasjonsmodellen og OO-tankegangen som objektbaser i høy grad løser [Loomis 94], [Mullins 94]. [Kappel et al 94] identifiserer to strategier for gjenbruk av relasjonsdatabaser:

1. Reverskonstruere databaseskjemaene og omskape dem til objektorienterte modeller. Dette implementeres på nytt i ikke-relasjonell teknologi, f.eks objektorienterte databaser.
2. Kombinere relasjonell teknologi med objekt-teknologi. Relasjonsdatabasen lagrer informasjonen i objektene.

Den første tilnærmingen ovenfor har et par ulemper. For det første vil *alle* eksisterende relasjons-baserte applikasjoner i bruk måtte reimplementeres med objektetekologi. Dette passer ikke inn i strategi nr 3 på side 28: gradvis utskifting av systemkomponenter. For det andre, som det ble nevnt ovenfor, er ikke objektdatabasetekologien moden nok ennå — i allefall ikke så moden at industrien velger å satse på den.

Tilnærming 2. ovenfor er det nok lettere å argumentere for. Et problem som gjenstår er imidlertid å overkomme det impedansmisforhold som nevnt tidligere. Spesielt gjelder dette når det er ønskelig å

---

1. Det finnes anslagsvis 50 milliarder linjer COBOL-kode i bruk i USA alene [Jeffery 96].
2. Relasjonsdatabaseteori behandles utførlig i [Elmasri & Navathe 94].

representere komplekse objekter eller objekt- og klassestrukturer. [Narasimhan et al 94] gir en ganske rett frem metode for å generere objektorienterte strukturer fra ER-modeller eller relasjonsdatabase-skjema. Tilnærmingen i denne metoden er imidlertid å helt erstatte relasjonstekologien med en tilsvarende objektteknologi, men det fordrer at OO-teknologien kan håndheve persistens på lik linje med relasjonsdatabaser i produksjon.

[Kappel et al 94] referer til eksempler på tre eksisterende muligheter for objektifisering av relasjonsdatabasetabeller: Streams++, Smalltalk/SQL og Persistence Software. Den første og den siste av disse tre oppnår imidlertid kun å lagre og hente objekter fra filer, og Smalltalk/SQL hjelper til å *lese* data fra en relasjonsdatabase i objektorienterte applikasjoner, men støtter ikke komplekse objekter, arv eller polymorfisme. Det fjerde alternativet, som altså er presentert i [Kappel et al 94], tar sikte på å utvide det eksisterende relasjonsdatabase-skjemaet med et metaskjema, som ivaretar korrespondansen mellom relasjonsdatabaseentitetene i databasen og objektene i objektmodellen (som vi antar er generert f.eks med ER➜OO metoden i [Narasimhan et al 94]).

Bibliotekene i COMan-rammeverket, som er det forslag som presenteres i [Kappel et al 94], sørger for — ved hjelp av metadatabasen — at relasjonsdatabasen fortsatt kan eksistere og vedlikeholdes som før, og enda viktigere: *den kan fortsatt brukes av eksisterende applikasjoner på vanlig måte*. Dette sikrer at man kan følge en utviklingsstrategi hvor delkomponenter i store edb-systemer skiftes ut gradvis, og dette er helt i tråd med strategi nr 3 på side 28.

Sammendrag

I dette kapittelet er det presentert generelle betraktninger og retningslinjer for gjenbruk av eksisterende systemer og programvare i ny kontekst. I praksis vil ikke de forskjellige prinsippene utelukke hverandre, men kunne brukes om hverandre i varierende grad, på forskjellige steder, kanske i forskjellige stadier av utviklingsprosessen. Typisk kan eksisterende systemer «spises opp» litt etter litt, slik som antydet i [Jacobson & Lindström 91] og [Taylor 95]. Hver munnfull blir omkapslet eller omdannet til objekter, som inngår i en objektorientert design av det nye edb-systemet.

For at de nye objektmodeller skal bli riktige, er det imidlertid nødvendig at modeller fra det eksisterende systemet brukes sammen med modeller slik brukerne og domeneeksperter ser det. Slik kan implisitt kunnskap og bruksmåter som ikke var påtenkt da de opprinnelige systemer ble laget komme til nytte.

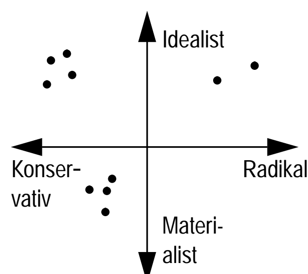
3 Beskrivelsesteknikker og modeller

«Models are not right or wrong; they are more or less useful.»

[Fowler 97]:2

Innledning

En forutsetning for at deltakere i systemutviklingsprosjekter — f.eks systemutviklere, oppdragsgivere, programmerere og brukere — sammen skal kunne utvikle eller forbedre et edb-system er at de kan kommunisere seg i mellom om det, slik det er nå og slik det blir i fremtiden. Som nevnt tidligere er mange begreper i praksis blitt ganske upresise, siden de ofte brukes i mange forskjellige sammenhenger og på forskjellige måter. Én strategi for å motvirke dette problemet er å bli enige om beskrivelsesteknikker, referansemodeller eller rammeverk for områder der det er viktig å kunne utveksle informasjon og konsepter på en presis måte. Referansemodeller og rammeverk bør definere viktige begreper klart og entydig, og presentere konseptuelle modeller for aktuelle problemområder, i denne sammenheng datakommunikasjon og distribusjon av data og funksjonalitet.



Figur 3.1: Markedssegmentering

For enkelhets skyld har jeg i dette kapittelet avgrenset aktørene til to grupper; systemutviklere og domeneeksperter. Systemutviklere innbefatter i denne sammenheng designere, programmerere etc, mens domeneeksperter blant annet inkluderer brukere og oppdragsgivere. Litt uformelt kan beskrivelsesteknikker i denne sammenheng plasseres langs to akser, analogt med segmenteringsdiagrammer som brukes i markedsundersøkelser og demografiske undersøkelser. To kryssende akser representerer plassering mellom f.eks materialisme og idealisme og konservatisme og radikalsime, henholdsvis. Figur 3.1

viser eksempel på et verdisegmenteringdiagram med noen forekomster.

På samme måte kan modeller og rammeverk plasseres innenfor to andre akser:

1. I hvilken grad tar teknikk/modell/rammeverk *X* hensyn til distribusjon?
2. I hvilken grad gir *X* forståelse for både domeneeksperter og systemutviklere, og ikke bare én av dem?

Jeg skal ikke her gjøre noen forsøk på plassering innefor disse aksene, kun antyde at blant de idéverk som kan plasseres i henhold til spørsmålene ovenfor kan være:

- Analysemønstre, se f.eks [Fowler 97];
- Designmønstre, som presentert i f.eks [Gamma et al 95]
- Taksonomier for distribuerte systemer, se f.eks [Martin et al 91]
- Forskjellige modelleringsteknikker, slik som

  ER (Entity Relationship, se [Elmasri & Navathe 94], [Skagestein 91]);

  NIAM (Neijssen Information Analysis Method, se [ibid], [Ressem 95], [Normann & Ressem 94]);

  ORM (Object-Rôle Modeling, se f.eks [Halpin 96])
- Referansmodeller, slik som:

  ISO's OSI-modell (Open Systems Interconnection), se f.eks [Halsall 95];

  NISTs[1] OSE (Open Systems Environment), se f.eks [[Schulz 95];

  ECMA's [2] «Reference model for frameworks of software engineering environments»/«Brødristermodellen», se [ECMA 93];

  RM-ODP (reference Model for Open Distributed Processing), se [RM-ODP 1], [RM-ODP 2], [RM-ODP 3];

Selv om ikke alt som representeres av punktene i listen ovenfor omhandler distribusjon som sådan, er de alle interessante som hjelpemidler for forståelse og samarbeid mellom og innenfor de to gruppene jeg definerte tidligere; systemutviklerne og domeneekspertene.

I dette kapitlet vil jeg i hovedsak gjøre rede for den nyeste referansemodellen fra siste del av listen ovenfor; RM-ODP. Denne referansemodellen gir felles språk for beskrivelse av distribuerte systemer (faktisk fem) og definerer endel begreper og hva som bør forstås med dem. Modellen følger en objektorientert tankegang, og betrakter derfor verden som samspillende objekter, i harmoni med stoffet presentert på sidene 7–13. Først vil jeg imidlertid nevne noen tidligere referansemodeller som jeg mener i stor grad er interessante når det gjelder distribusjon; ISO's OSI-modell, NIST's Open Systems Envi-

---

[1] National Institute of Standards and Technology

[2] European Computer Manufacturers' Association

ronment modell og «brødristermodellen» — eller ECMA/NIST-modellen. Jeg vil ikke ta opp mer formelle beskrivelsesspråk som Z, LOTOS, ESTELLE etc.

OSI-modellen

Formålet med International Standardisation Organisation (ISO) sin referansemodell for datakommunikasjon *Open Systems Interconnection* (OSI) er å gi et standard begrepsapparat for å kunne resonnere og kommunisere tanker om datakommunikasjonssystemer. Modellen representerer en lagdelt arkitektur[1] med syv lag, som danner en mal for strukturen i kommunikasjonssubsystemer [Halsall 95]. Lagene ligger konseptuelt over hverandre. Hvert lag *N* tilbyr en øket tjenesteverdi til laget *N+1* som ligger over *N*. Økning av tjenesteverdi kan eksempelvis være en øket *Quality of Service* (QoS) [ibid], f.eks pålitelighet eller feilkorreksjon. Det kan også tilby en passelig grad av abstraksjon til lagene over, slik at f.eks valg av nettverkstype (Ethernet, Token Ring etc.) på Link-laget ikke er av vesentlig betydning for tjenestene i lagene over. Tilsvarende vil Transportlaget tilby en transportkanal til Sesjonslaget uten å bry dette med lengde på datapakker, feilkorreksjon osv.

Applikasjon
Presentasjon
Sesjon
Transport
Nettverk
Link
Fysisk

Figur 3.2: OSI-stakken

OSI-modellen kalles gjerne en stakk. Samme begrep brukes også om praktiske kommunikasjonsimplementasjoner, så som TRUMPET WINSOCK, som er en TCP/IP-stakk[2] for MICROSOFT WINDOWS 3.X. Stakker finnes også for andre protokollsuiter, f.eks IPX/SPX (Brukes av NOVELL), NetBIOS[3] (MICROSOFT/IBM), SNA (IBM). Stakker som følger ISO/IEC X-anbefalinger finnes også, men har nok blitt satt i skyggen av TCP/IP, i allefall når det gjelder utbredelse og Internettbruk. Dette kan komme av at ISO/IEC-X baserte stakker er forholdsvis kompliserte i bruk [Crowcroft 96], og at de gjerne er tregere enn tilsvarende TCP/IP-alternativer [Moldeklev 90]. Enkelte deler av OSI-teknologien har overlevd, men stort sett synes det som om *«(…) OSI has disappeared from the corporate radar screen.»* [Eckerson 95], noe som synes å stille spådommen i [Hannemyr 92]:111; *«Problemet er (…) ikke* ***hvorvidt*** *man skal satse på OSI, men* ***når****»*, til skamme. Likevel er OSI-modellen godt egnet til å beskrive andre stakker, inkludert de som er nevnt ovenfor. Faktiske implementerte kommunikasjonsstakker er av praktiske årsaker ikke alltid delt opp i like mange lag som i

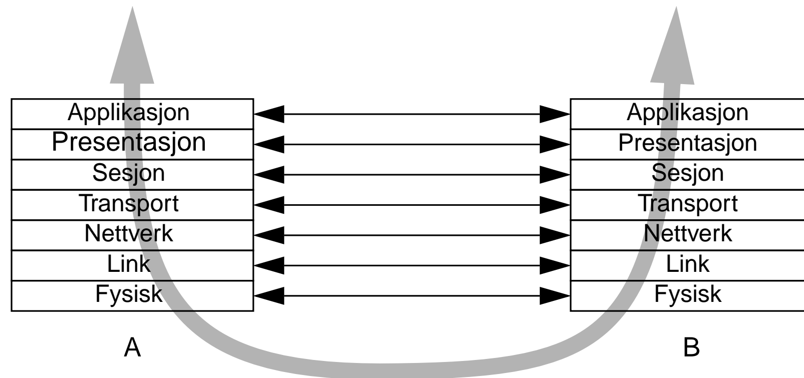
1 Se f.eks arkitekturtaksonomien i [Shaw et al 96].

2 Transport Control Protocol / Internet Protocol (TCP/IP) er en forbindelsesorientert komunikasjonsprotokoll på transportlaget i OSI-modellen som brukes i stor utstrekning innen forskning- og akademiske miljøer, samt Internett.

3 NetBIOS og IPX/SPX er presentert i f.eks [Schwaderer 92].

OSI-modellen, men kan gjerne ha slått sammen og/eller splittet opp noen av dem.

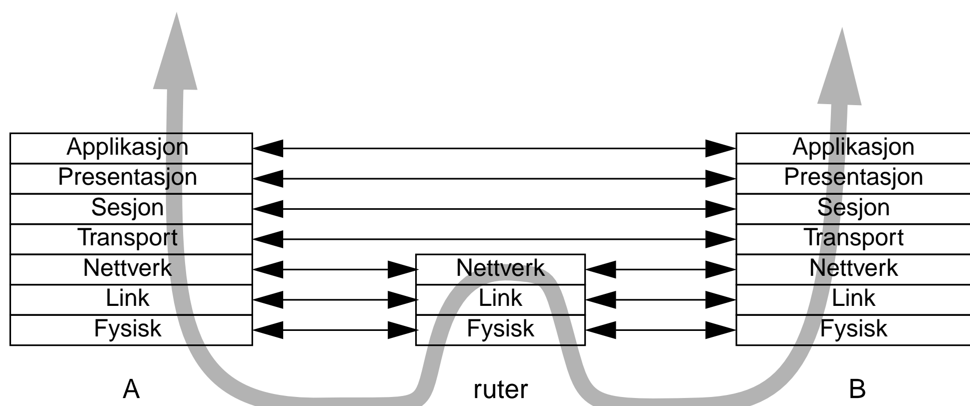
Ett godt eksempel på hvordan modellen gir en enklere beskrivelse og lettere forståelse av et kommunikasjonsscenario, er ved å tenke seg at et lag N på maskin A (N_A) alltid kommuniserer med et tilsvarende lag N_B på maskin B (se Figur 3.3). Riktignok går kommunikasjonen



Figur 3.3: Hvert lag sin protokoll

egentlig via lag N-1_A og N-1_B og så videre ned gjennom den ene stakken og opp gjennom den andre, men de detaljene skjules. Applikasjonslaget på den ene maskinen snakker med Applikasjonslaget på den andre, Presentasjonslag med Presentasjonslag osv.

Som et eksempel viser Figur 3.4 systemet i Figur 3.3 utvidet med en ruter. En ruter har ansvaret for å dirigere datapakker mellom to eller flere nettverk. Den bryr seg ikke om innholdet i pakkene, bare hvor de kommer fra og hvor de skal.



Figur 3.4: Kommunikasjon gjennom en ruter

Poenget her er at protokollene mellom de fire øverste lagene i de to endenodene A og B ikke påvirkes av ruteren i mellom. Med andre ord

kan all programkode som kun opererere på disse lagene i prinsippet se bort fra de forhold som ruteren tar seg av. Det kan til og med tenkes at protokollene på de tre nederste lagene er forskjellig på hver side av ruteren.

Et par andre eksempler: Over en *bro* mellom to lokalnett, mellom f.eks en Token Ring og et Ethernet segment, vil de fem øverste lagene i nodene på hver side av broen kommunisere med de samme protokoller, mens de to nederste vil kunne ha forskjellig protokoll på hver side av broen. For en konverterende *hub* mellom to Ethernettsegmenter, hvor det ene består av koaksialkabel (10Base2/10Base5), mens den andre bruker tvunnet parkabel (10BaseTP), vil bare den nederste laget i OSI-modellen, det fysiske laget, ha forskjellige protokoller på hver side.

Brødristermodellen — ECMA/NIST Toaster Model

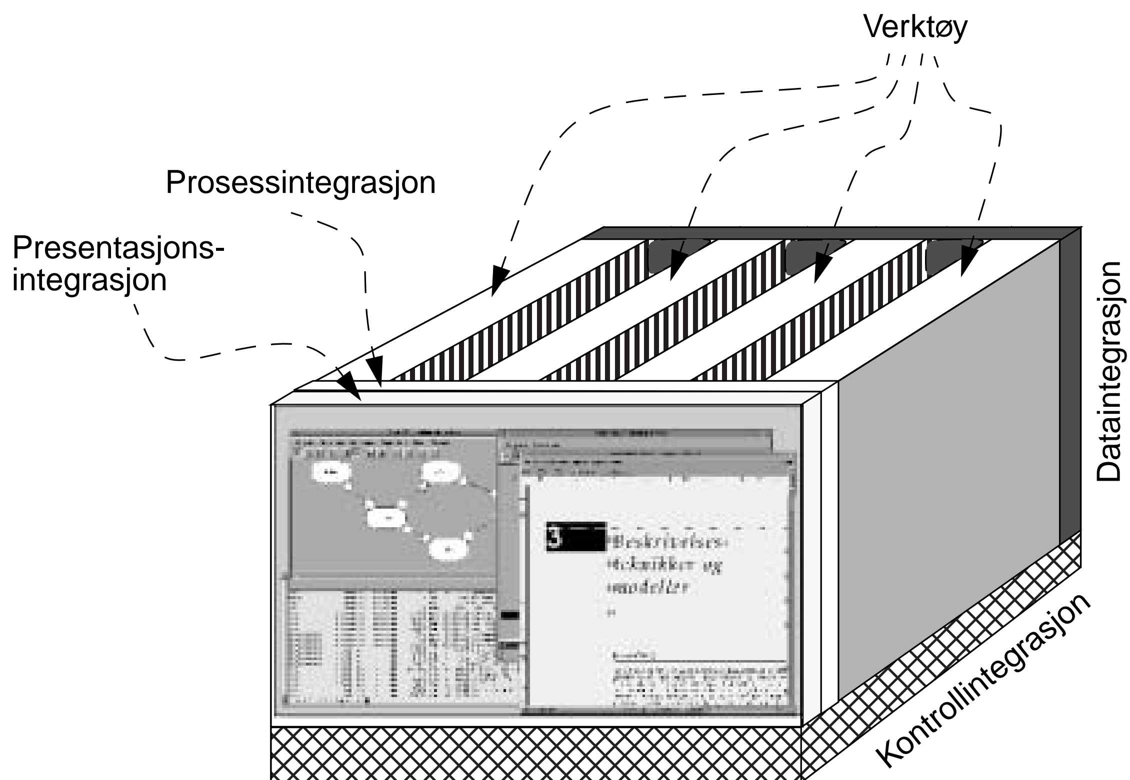
Svært sentralt i denne oppgaven er integrasjon av eksisterende og nye edb-systemer. Som nevnt tidligere er betydningen av mange begreper ofte noe utvannet. Dette ikke minst med «integrasjon». Hva menes med å integrere noe? I [ECMA 93] er integrasjon:

- I hvilken utstrekning forskjellige verktøy effektivt kan kommunisere med hverandre innen et gitt miljø;
- Et mål på forhold eller avhengighet mellom komponenter i et miljø;
- Hvilken letthet, interoperabilittet, portabilitet, skalerbarhet og andre «-het»'er som er resultat av sømløs interaksjon mellom et sett komponenter i et miljø.

De tre punktene ovenfor virker kanskje ikke så håndgripelige ved første øyekast. [ibid]:16–17 presenterer derfor tre viktge bestanddeler av integrasjon:

- Integrasjon er et sett av tjenester. ECMA-referansemodellen beskriver et antall felles tjenester som bidrar til integrasjon.
- Integrasjon er en ny dimensjon for hver tjeneste. Et felles grunnlag for tjenester gir mulighet for, men påtvinger ikke, integrasjon.
- Integrasjon kan være en «policy», dvs felles regler som gjelder i det integrerte miljøet.

ECMA/NIST's «brødristermodell» gir et forståelsesramme for integrasjon i forskjellige dimensjoner, se Figur 3.5. Modellen plasserer former for integrasjon fra informasjons-/datanivå til brukergrense-



Figur 3.5: NIST/ECMA brødristermodell

snittnivå. De forskjellige integrasjonsdimensjonene støttes av hverandre, slik at de sammen og hver for seg bidrar til å løse arbeidsoppgaver:[1]

- I **Data/Informasjonsdimensjonen** kan data og datamodeller integreres ved at forskjellige informasjonsmodeller og lagringsstrukturer bindes sammen både semantisk og i praksis. Sentralt i integrasjon på dette nivået er felles forståelse og tolkning av data, slik at de kan utnyttes på så mange vis som mulig. Eksempelvis vil globale datamodeller, som den utviklet i [Smørdal 96 — har O. Smørdal skrevet noe om Betamodellen sin av PBE? Finn ut!]. Utveksling, konvertering, filtrering og tilpassing av data faller inn under denne integrasjonsdimensjonen.
- **Prosessintegrasjon** støtter arbeidsprosesser, dvs styring og tilrettelegging av de enkelte arbeidsoppgaver brukeren utfører for å nå konkrete mål i sitt virke gjenneom interaksjon med edb-systemet.

[1] Merk at delkapittelet om «Brødristermodellen» er min tolkning av referansemodellen presentert i [ECMA 93]. Referansmodellen i [ibid] inneholder formelt ikke Brødristermodellen lenger, men består av en katalogisering av flere *tjenester* som tilsammen bidrar til systemintegrasjon. Dette ligner på CORBA's Services (se side 65). Jeg har imidlertid kun tatt tak i Brødristermodellen, så ECMA får ha meg tilgitt her.

- **Presentasjonsintegrasjon** integrerer delarbeidsoppgavene som inngår i de ankelte arbeidsoppgaver fra prosessintegrasjonen, slik at delopgaver kan realiseres. Et eksempel på presentasjonsintegrasjon er vindussystemer, slik som MICROSOFT WINDOWS, X WINDOW SYSTEM eller andre brukergrenesnittstandarder. Ved å utstyre applikasjoner med lignende brukergrensesnitt gjennom samme «look-and-feel», la dem følge felles designregler og gjennom bruk av visuell datautveksling, f.eks klipp-og-lim, kan arbeidsoppgavene integreres tettere.
- **Verktøy** binder datarepresentasjon til prosess- og presentasjonsintegrasjonen ved å binde sammen datarepresentasjon og arbeidsoppgaver som påvirker de forskjellige elementene i datamodellene. Som modellen i Figur 3.5 viser er det plass til mange verktøy i «brødristeren». Dette kan tolkes som at dette er den mest fleksible integrasjonsdimensjonen. Det er også her mye av funksjonaliteten i edb-systemet ligger, dvs mesteparten av forretningslogikken.
- Alle integrasjonsdimensjonene nevnt ovenfor trenger imidlertid støtte for å kommunisere seg imellom, så grunnmuren i hele modellen er **kontrollintegrasjonsdimensjonen**, som ligger under og støtter alle de andre. Ideelt sett burde denne integrasjonsformen være usynlig, slik at utviklere og brukere kun behøver å vurdere og diskutere de øvrige integrasjonsformene. Det viktigste er jo først og fremst forretningsreglene og arbeidsprosessene og de arbeidsoppgaver de består av. For å kunne modellere og designe fornuftige og effektive arbeidsprosesser, trengs en arkitektur hvor det er lagt til rette for dette, og hvor detaljer er mer skjult. Kontrolldimensjonen representerer altså infrastrukturen som integrasjon i de andre dimensjonen benytter seg av.

Som nevnt ovenfor finnes allerede flere mekanismer for presentasjonsintegrasjon ved hjelp av grafiske brukergrenssnitt. Gruppevareprodukter finnes for prosessintegrasjon (f.eks LOTUS NOTES, MICROSOFT EXCHANGE). Dataintegrasjon i praksis støttes av blant annet databaseadministrasjonssystemer og transaksjonsmonitorer (se [Orfali et al 96a]:145–317).

Alle disse elementene integreres gjennom et «lim» som altså utgjør kontrollintegrasjonen. Dette limet kan realiseres med blant annet fjernprosedyrekall, delt hukommelse eller annen inter-prosess kommunikasjon, meldingsutveksling eller distribuerte objekter. Det siste alternativet ser for øyeblikket ut til å være det mest lovende, siden det a priori passer best sammen med objektorientert analyse, design og implementasjon. Selv om eksempelvis vindussystemer, slik som X WINDOW servere, i praksis er realisert monolittisk — dvs at de består av et optimalisert stykke kompilert C-kode som viser frem grafiske elementer (eksempelvis vinduer) på en skjerm som respons på meldinger — er gjerne applikasjonutvikling mot disse systemene basert på objektorienterte rammeverk, slik som f.eks BORLAND TURBO C++

OBJECTWINDOWS LIBRARY (OWL), SUN's JAVA WORKSHOP VISUAL JAVA wrapper klasser eller Java's standard abstrakte vindusklasser (`java.awt`.* — se f.eks [Flanagan 96]). Applikasjonsutvikling og -integrasjon tror jeg derfor i stadig større grad vil foregå med objektorienterte metoder og verktøy.

Selve brødristermodellen er egentlig ment som en huskliste for forskjellige integrasjonsdimensjoner i «Software Engineering Environments», presentert i tidligere utgaver av [ECMA 93]. Brødristermodellen er imidlertid eksplisitt fjernet derfra, fordi forfatterne av rapporten mente at den ble oppfattet som selve referansemodellen, og altså ikke som bare en huskeliste [ibid]:118. Likevel mener jeg brødristermodellen gir et intuitivt bilde av — og et begrepsapparat for — systemintegrasjon. Jeg tror også referansemodellen er nyttig i andre problemområder enn systemutvikling, selv om denne modellen i utgangspunktet er ment for dette spesifikke bruksområdet.

Referansemodell for distribuerte systemer — RM-ODP

Reference Model of Open Distributed Processing er en standard som er felles for ISO og ITU-T.[1] Standarden beskrives i ISO 10764-1, -2, -3 og -4 og ITU-T X.901, -2, -3 og -4. Standarden gir et begrepsapparat for overordnet beskrivelse av distribuerte systemer, uten å legge føringer på valg av teknologi, protokoller eller andre kommunikasjonsstandarder. Arbeidet med utviklingen av referansemodellen har tre hovedmål [Raymond 95]:

- Lette portabilitet av applikasjoner mellom heterogene plattformer;
- Lette samvirke og kommunikasjon i og mellom heterogene distribuerte systemer;
- Opprettholde en passelig grad av transparens overfor konsekvenser av distribuert informsajonsssystemarkitektur, tilpasset aktuelle interessegrupper, som f.eks systemdesignere, programerere og brukere.

Et annet bruksområde er som sammenligningsgrunnlag for distribuerte systemer og infrastrukturer for støtte til slike. Slik kan referansemodellen brukes til å vurdere konkrete implementasjoner og standarder, som eksempelvis DCE og CORBA (se f.eks [Plowiec 96] og [Halvorsen 97], henholdsvis).

[1] International Telecommunications Union — Telecommunication Standardization Sector.

Som antydet tidligere er standarden delt og beskrevet i fire deler:

- «Part 1: Overview and guide to use», ISO10764-1/ITU-T X.901
- «Part 2: Descriptive model», ISO10764-2/ITU-T X.902
- «Part 3: Prescriptive model», ISO10764-3/ITU-T X.903
- «Part 4: Architectural semantics», ISO10764-4/ITU-T X.904

Fem perspektiver

Den sentrale konstruksjonen i RM-ODP er de fem perspektiver (*viewpoints*). Hvert perspektiv kan brukes til å beskrive eksisterende eller fremtidige distribuerte systemer ut fra forskjellige hensyn. De fem perspektiver er:

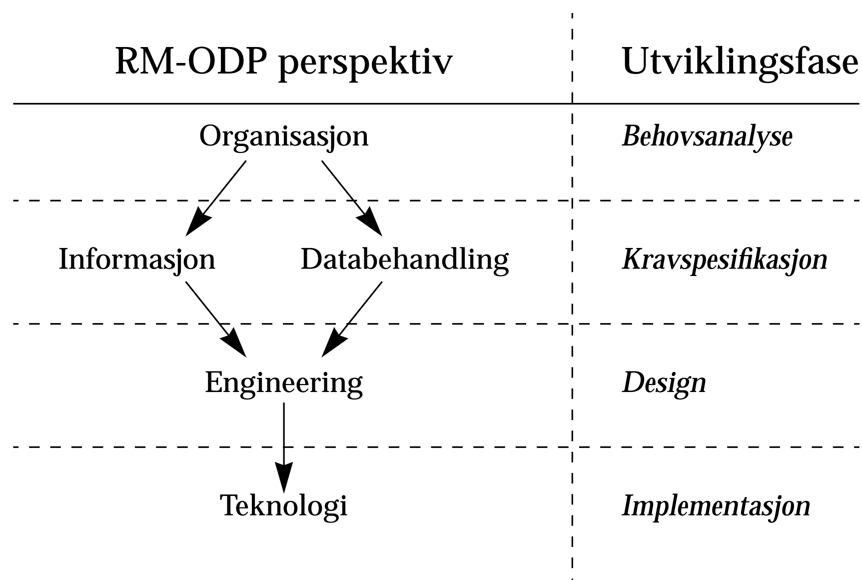
- Enterprise Viewpoint (Organisasjon) — formål, anvendelsesområde og overordnede forretningsregler;
- Information Viewpoint — sematikk i informasjon og informasjonsmodeller;
- Computational Viewpoint (databehandling) — databehandling, funksjonell design;
- Engineering Viewpoint — infrastruktur for å støtte den konkrete distribusjonen;
- Technology Viewpoint — valg av teknologi.

Ved å dele opp diskurs, analyse og problemløsing rundt et eksisterende eller fremtidig distribuert system etter disse perspektivene, er det formodentlig lettere å plassere rett tema på rett plass og til rett tid. Eksempelvis vil valg av kommunikasjonsprotokoll (f.eks TCP/IP vs SPX/IPX) ikke være relevant i Organisasjonsperspektivet (*Enterprise Viewpoint*), men vil være passende i Teknologiperspektivet.

Hvert perspektiv tilbyr et språk — eller et begrepsapparat — som kan brukes til resonnement, diskusjon og analyse/design innen det aktuelle perspektivet.

De fem perspektivene er i utgangspunktet ikke rangert etter abstraksjon, slik som OSI-modellen kan tolkes. Hvert perspektiv belyser heller problemområdet fra forskjellige sider, slik at beskrivelsene i hvert perspektiv til sammen utgjør en utfyllende og komplett beskrivelse av det distribuerte systemet. Likevel er det klart at påvirkningen flyter fra Organisasjonsperspektivet «nedover» til Teknologiperspektivet, slik at f.eks organisatoriske avgjørelser til syvende og sist vil bestemme hvilke objekter som gjør hva i systemet og ikke omvendt.

Dette gjør at det er mulig å plassere perspektivene innenfor en systemutviklingsprosess som grovt er delt inn i behovsanalyse, krav-



Figur 3.6: De fem perspektiver i en systemutviklingsprosess

spesifikasjon, design om implementasjon, som vist i Figur 3.6.[1] De fem perspektivene er altså ikke helt uavhengige av hverandre. For å sikre konsistens mellom perspektivene, er det et sett nøkkelelementer som knytter dem sammen. I tillegg er det mulig å plassere de forskjellige perspektivene langs oppdelingen mellom problemområdet og andvendelsesområde i [Mathiassen et al 93] ved å tenke seg at problemområdet dekker ovenfra og nedover i Figur 3.6, mens andvendelsesområde dekker nedenfra.

I hvert perspektiv håndheves visse transparenser, dvs at hensyn som må tas i et praktisk fungerende system skjules. Spesielt er feiltransparans viktig, men også andre typer, så som fysisk plassering av dataobjekter, replikering av data, valg av protokoller etc. Mange av disse transparenskravene vil i en praktisk løsning ivaretas av infrastrukturen, eksempelvis CORBA (se Kapittel 4).

Enterprise viewpoint

Enterprise Viewpoint fokuserer på formål, bruksområde (scope) og overordnede regler for edb-systemet og bruken av det. Det sentrale i dette perspektivet er hva som skal oppnås, og i mindre grad hvordan. Det identifiseres forretningsobjekter og -roller og samspill defineres som kontrakter og ansvarsforhold mellom disse. I tilknyting til objektene kan eventuelt eierskap eller andre rettigheter som gjelder identifiseres. Interessenter og aktører grupperes i fellesskap — eller *communities* — som på en eller flere måter har felles interesser. Eksempelvis vil en gruppe brukere kunne anses som et fellesskap. Communities samles administrativt i *domains.* Domener kan baseres på

[1] En slik bruk av RM-ODP's Viewpoints var egentlig utgangspunktet for denne hovedoppgaven. (Se side i.)

forskjellige kriterier, f.eks tekniske eller administrative domener, hvor henholdsvis felles teknologiske beslutninger er tatt eller hvor det gjelder bestemte sikkerhetsregler. Domener slås eventuelt sammen i *føderasjoner*, hvor domenene har ett eller flere felles mål og ønsker å samarbeide og utveksle informasjon.

Plan- og bygningsetaten kan sees på som en føderasjon av domener. De tidligere adskilte enhetene (Heiskontrollen, Bygningskontrollen og Oppmålingsvesenet) kan hver betraktes som domener. Disse består igjen av felleskap/communities, så som saksbehandlere, kontrollører, kartspesialister osv. Etterhvert som elektronisk integrasjon av stadig flere domener gjør seg gjeldende, vil formodentlig føderasjoner bli større og fler, og domener vil bli medlem av flere føderasjoner. Organisasjonens grense utad kan derfor bli flytende og ansvarsforholdene kompliserte. Fellesskap-, domene- og føderasjonsbegrepene kan derfor bidra til å spesifisere disse tilfellene mer nøyaktig.

Et interessant tema er hvordan grensene mellom føderasjoner, domener og fellesskap skal trekkes. Aktuelle teknikker for å bestemme dette kan være vegg-grafer, rike bilder etc. Disse teknikkene kan bidra til å avdekke saksgang og papirflyt, både internt og i samarbeidet med andre eksterne aktører, som f.eks kommuneadministrasjonen og politikerne i Rådhuset i Oslo.

Information viewpoint

I Information Viewpoint beskrives semantikken i informasjonene og informasjonsbehandling i det distribuerte edb-system, form av atomiske informasjonselementer og strukturer mellom dem. Mer presist beskrives objektenes informasjonsinnhold som attributter og beskrankninger som gjelder for dem. Informasjonemodellen beskrives i tre skjema:

Invariant skjema. Beskriver tilstand som alltid må gjelde. Eksempelvis skal en bankkonto ikke kunne overtrekkes, dvs ingen operasjon på den kan resultere i at saldoen blir negativ.

Statisk skjema. Beskriver lovlig tilstand i på gitte tidspunkt, f.eks skal en bankkonto ha kr 0,- i balanse når den opprettes.[1]

Dynamisk skjema. Beskriver hvordan tilstanden kan forandre seg på lovlig vis. Saldoen på bankkontoen fra de foregående eksemplene skal f.eks ved uttak senkes like mye som det beløpet som er tatt ut. Her beskrives også hvordan strukturen mellom informasjonsobjektene, f.eks aggregering, påvirkes av lovlige operasjoner. Imidlertid

[1] Det er slutt på de tider da nydøpte barn fikk innsatt kr 10,- i gave av Sparebanken ved åpning av konto…

må reglene i det dynamiske skjema ikke bryte med det invariante skjema.

Computational viewpoint

I Computational viewpoint beskrives den logiske distribusjonen i systemet, dvs hvordan systemet oppdeles i objekter som samarbeider gjennom veldefinerte grensesnitt. Hvert objekt utfører en nøye avgrenset del av systemets funksjonalitet. Det er altså her samarbeidet mellom objektene — og dermed også applikasjonens funksjonalitet — defineres og diskuteres. Hvordan objektene så spres rent topografisk berøres *ikke* i dette perspektivet. Heller ikke tas det stilling til hvordan de «klumper» seg sammen i moduler og komponenter.

Interaksjon mellem objektene kan ta tre former:

- **Operasjoner** ligner mest på den «vanlige» klient-tjener situasjonen; klienten sender en forespørsel til tjeneren, og får et svar tilbake, eventuelt en bekreftelse på at henvendelsen er mottatt. Klientobjektet venter mens operasjonen utføres i tjenerobjektet. Fjernprosdyrekall (se side 57) passer best til denne beskrivelsen.
- **Datastrømmer** kan f.eks være lyd/bilde. Avhenggig av krav til overføringskvalitet og –pålitelighet kan man eventuelt akseptere litt tap av eller feil i datapakker, så lenge flyten opprettholdes og forsinkelsen er så liten og konstant som mulig.
- **Signaler** er atomiske, og går bare én vei fra sender til mottaker. De to andre formene ovenfor kan bygges opp av signaler.

Det kan diskuteres om man stadig befinner seg i problemområdet i dette perspektivet, eller om modelleringen nå foregår i anvendelsesområdet. I utgangspunktet virker det naturlig å si at vi nå er i førstnevnte, siden vi ikke har tatt stilling til distribusjonen eller implementasjonsdetaljene ennå. Imidlertid er det ikke usannsynlig at en del av objektene i utgangspunktet stammer fra et allerede eksisterende anvendelsesområde, f.eks fra eksisterende systemkomponenter:[1]

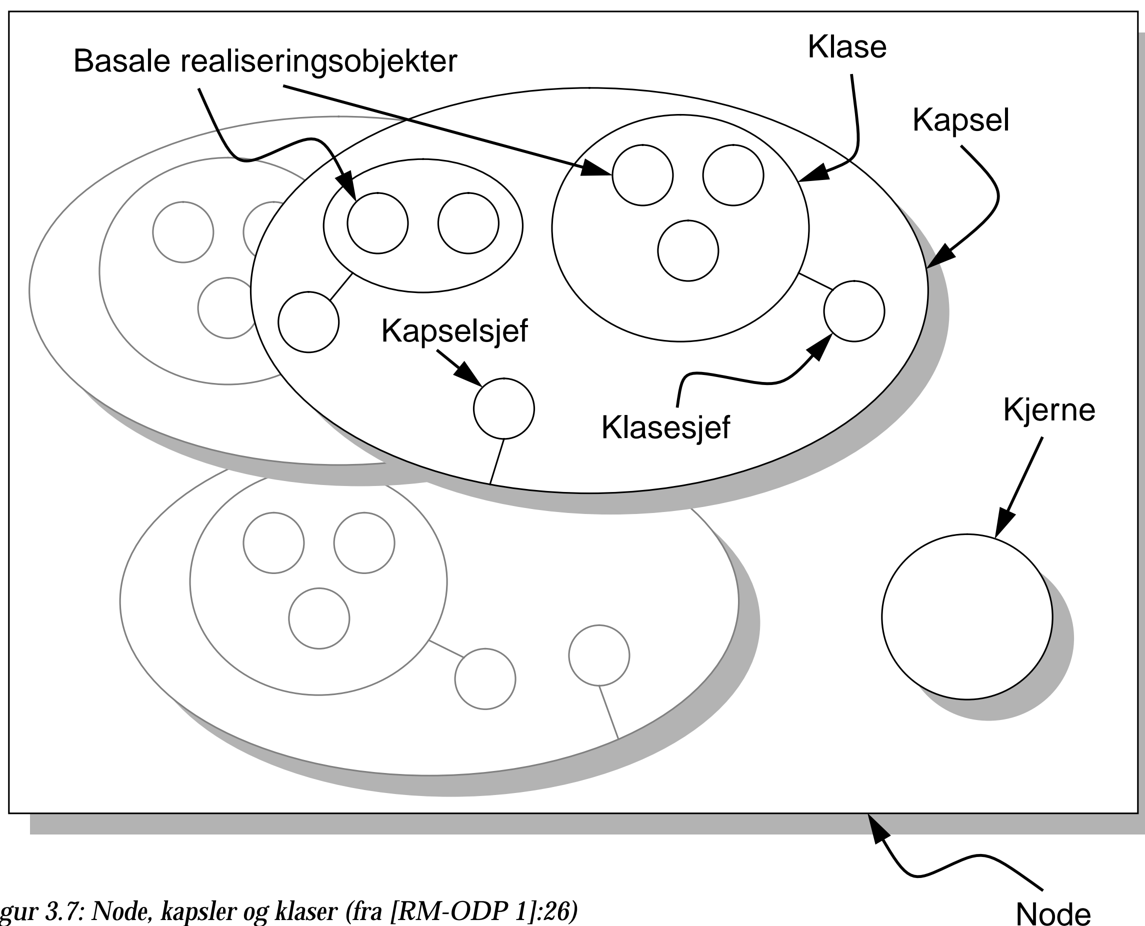
> *«[The compuattional language] allows for encapsulation of existing applications as (…) components of a larger, distributed application.»*
>
> *[RM-ODP 1]:20*

Engineering viewpoint

I motsetning til Computational Viewpoint, som behandler *hva* objektene skal oppnå med samhandling, er man i Realiseringsperspektivet mer interessert i *hvordan*. Målet er å gi støtte til samhandlingen mellom objektene som beskrevet i forrige perspektiv. Objekter fra forrige perspektiv brukes i dette som basale realiseringsobjekter (basic engi-

[1] Se forøvrig Kapittel 2.

neering objects) og kobles sammen med andre realiseringsobjekter her. Objektene grupperes hierarkisk, på samme måte som i Enterprise Viewpoint[1] (se Figur 3.7):



Figur 3.7: Node, kapsler og klaser (fra [RM-ODP 1]:26)

- **Noder** samler objekter fysisk på maskiner. Hva som danner grenser mellom nodene er noe uklart, så [RM-ODP 1]:25 definerer en node slik: *«anything which has a strongly integrated view of resources, as long as the system designer can consider it as a hole.»* Tett koblede parallellsystemer[2] kan slik anses som en node. Hver node kontrolleres av en *kjerne*. Kjernens oppgave er å administrere kommunikasjon med andre noder, og andre basale tjenester. I praksis vil kjernen være operativsystemkjernen.
- **Kapsler** er neste nivå i grupperingen. En node kan ha flere kapsler, som hver er beskyttet mot hverandre. Kapslene har sitt eget adresseorom og deler nodens ressurser med andre kapsler, i henhold til regler håndhevet av kjernen. Kapsler er sammenlignbare med *prosesser* i UNIX, og er den minste enhet av feilkontroll som tas hånd

[1] Enterprise Viewpoint: føderasjoner, domener, felleskap

[2] MISD (Multiple Instructions, Single Data) i [Tanenbaum 90]:491, se fotnote 1 på side 60.

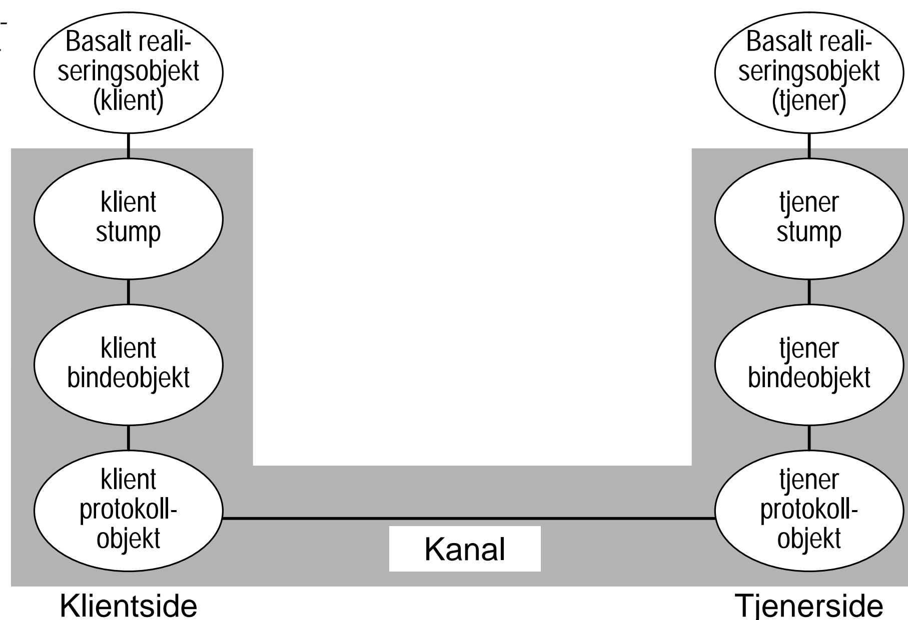
om av operativsystemet. Hvis eksempelvis en UNIX-prosess forsøker å manipulere hukommelse utenfor sitt eget adresserom, vil prosessen normalt termineres med en feilmelding. Hvis én kapsel krasjer, skal den altså ikke kunne påvirke andre. Kapselen styres av en *kapselsjef.* I praksis vil det være vanskelig å konkret identifisere kapselsjefen og dennes innflytelse, så dette objektet brukes kun som en deskriptiv forståelse av denne delen av modellen.

- **Klaser** er den laveste grupperingen av objekter. Klaser i en kapsel er ikke beskyttet mot hverandre av nodens kjerne slik som mellom kapslene, så dette må kapselen selv ta seg av. Objektene inni en klase administreres av en *klasesjef.* Klasesjefen er, i likhet med kapselsjefen, ikke nødvendigvis et virkelig objekt, men kan modelleres som det for forståelsens skyld. Hensikten med oppdeling i klaser er å redusere kostnadene ved behandling av objekter som er forholdsvis tett knyttet sammen, f.eks ved at de hentes og lagres på fil sammen, flyttes sammen el.l.

I tillegg til de basale objektene fra Computational Viewpoint behandles bindingene mellom dem som *kanaler*, som består av kjeder med objekter, som vist i Figur 3.8. Man kan tenke seg én retningslinje for hvordan objekter grupperes i klaser som *objekter som kan dele en kommunikasjonskanal.* Dette reduserer kostnadene ved kommunikasjon i en finkornet objektstruktur. Eksempelvis kan samlinger av grafiske objekter som kan hentes ut fra kartdatabasen i eksempelapplikasjonen presentert i Kapittel 6 derfor betraktes som objektklaser.

I dette perspektivet har vi definitivt beveget oss inn i anvendelsesområdet i [Mathiassen et al 93]s oppdeling som nevnt tidligere. De objekter som opptrer i diskusjonen finnes nå som deler av implemen-

Figur 3.8: Kommunikasjonskanal mellom to basale realiseringsobjekter



tasjonen, og i den grad de reflekterer «virkeligheten», som f.eks objekter fra Enterprise perspektivet, er de stedfortredere for objektene der. Det er altså her selve designen av edb-systemet foregår, og objekter fra problemområdet blandes sammen med objekter fra applikasjonsområdet, f.eks kanalobjekter som vist i Figur 3.8. I tillegg kommer andre praktiske objekter som f.eks brukergrensesnittobjekter og objekter fra eksisterende systemer.

Technology viewpoint

I teknologiperspektivet beskrives det distribuerte systemet gjennom valg av teknologi og praktisk implementasjon. Siden dette i liten grad påvirker de konseptuelle modeller av systemet skal jeg ikke gå nærmere inn på dette perspektivet her.

Konsistens mellom perspektivene

Selv om de fem perspektivene presentert ovenfor later til å være rangerbare innen en abstraksjonsdimensjon fra høy til lav (Enterprise til Technology) *og* i en enkel fossefallsmodell (se hhv Figur 1.10 på side 19 og Figur 3.6) er de i prinsippet likeverdige perspektiver på ett og samme system. RM-ODP definerer derfor et sett med korrespondansepunkter mellom perspektivene, slik at det er mulig å sjekke om de forskjellige beskrivelsene virkelig dekker samme fenomen. Jeg skal imidelertid ikke gå nærmere inn på disse sidene av RM-ODP her.

Forholdet mellom modellene

Det er naturligvis en sammenheng mellom de forskjellige modellenes kompleksitet/omfang og forståelighet, men for dem alle synes jeg at de ikke er lette å få hånd på ved første møte. I teksten ovenfor er presentasjonen forholsvis summarisk, men formodentlig grundig nok til å kunne se dem i forhold til hverandre.

Det er rimelig å anta at hver ny referansemodell ideelt sett skal erstatte alle foregående modeller på samme område, og litteraturen som behandler disse modellene referer også i stor grad til andre modeller som tidligere arbeid. Siden oppfatning og bruk av modellene imidlertid ikke er helt forutsigbar, er forholdet mellom modellene og den praktiske betydningen av dem heller ikke klar. Eksempelvis er OSI-stakken ikke blitt noen stor suksess som praktisk nettverkskommunikasjonsprotokollsuite (se side 37), men som begrepsmessig referanseramme har den vist seg som brukbar. Likeledes har Brødristermodellen vist seg å overskygge referansemodellen den var en del av, og er som tidligere nevnt eksplisitt fjernet fra denne standarden.

Objektorientering er uten tvil dominerende innen «moderne» litteratur og diskurs rundt systemutvikling og design/implementasjon. Et

viktig poeng — eller ideale — ved objektorientering er, ved siden av gjenbrukbarhet, *sømløshet* (se side 73). Objektorientering kan altså dominere alle aspekter og faser i systemutviklingen, fra analyse til implementasjon. Konsekvensen av dette er at også begrepsapparatene bør være «objektorienterte», og dette er for øyeblikket kun ivaretatt av RM-ODP. Selv om dette egentlig er det eneste av de ovenfor nevnte referansemodeller som er ekte objektorientert, passer imidlertid de fleste andre inn på sitt vis:

- **Brødristermodellen** gjør ingen antakelser om objektorentering er tatt i bruk eller ikke, men kombinert med andre begrepsapparater, f.eks Design Patterns, gir den likevel et verdifullt bidrag til konseptuell vurdering av integrasjon.
- **NIAM/ORM-modellering**[1] later i utgangspunktet til å være på kollisjonskurs med den objektorienterte tankegang, siden målet med en NIAM- eller ORM-modell etter *gruppering*[2] er en normalisert relasjonsdatabase. Som nevnt i [Kappel et al 94] og [Narasimhan et al 94] er kobling mellom relasjonsmodellen og OO-modellen ikke helt triviell. Imidlertid kan selve modelleringsprosessen bidra til å identifisere objekter og deres klasser [Ressem 95], spesielt for RM-ODP's Enterprise- og Information viewpoints.
- **Design Patterns**, som presentert i f.eks [Gamma et al 95], er i praksis objektorienterte, siden de fleste eksemplene som mønstrene er «destillert» fra er objektorienterte systemer, som f.eks objektorienterte vindushåndteringssystemer.
- **OSI-modellen** faller i første omgang utenfor det gode objektorienterte selskap. Riktignok gjør modellen ingen eksplisitte antakelser om objektorienting verken konseptuelt eller i implmenetasjon, og i praksis er nok de fleste kommunikasjonssystemer som ligner på OSI-stakken ikke objektorienterte. På den annen side ligger som regel de konsepter som OSI-stakken berører utenfor — eller kanskje heller nedenfor — den sfære hvor objektorientering har mest å tilby, dvs i og over Applikasjonslaget (se Figur 3.2). Objekter, f.eks distribuerte objekter, bryr seg ikke om hvordan kommunikasjonen ordnes for dem, og som regel skjer dette til syvende og sist ikke-objektorientert.[3] Om OSI-modellen skal kunne plasseres innen RM-ODP, må det være i det «nederste» perspektivet; Technology viewpoint.

Det er altså stort sett vanskelig å plassere de forskjellige modellene i forhold til hverandre, og det er mer nærliggende å se på dem som forskjellige *perspektiver* på distribuerte og ikke-distribuerte edb-syste-

---

[1] Object-Rôle Modeling (ORM) er i stor grtad inspirert av NIAM, se f.eks [Halpin 96].

[2] Begrepene (tilsv. enititeter i ER) i et NIAM/ORM-diagram grupperes til attributter i relasjonsdatabasetabeller etter en nøye beskrevet metode som den presentert i [Normann & Ressem 94].

[3] Husk f.eks at C++ som regel bare er en front-end til C, dvs at C++-programmer gjerne oversettes til C-kode før det kompileres ordinært [Rudd 94], [Carlson et al 96].

mer, på samme måte som RM-ODP definerer sine fem perspektiver på distribuerte systemer. Det blir mer naturlig å bruke modellene som det passer for anledningen, og eventeult plassere en konkret situasjon i forhold til flere andre referansemodeller og standarder, slik som i [Berre 96].

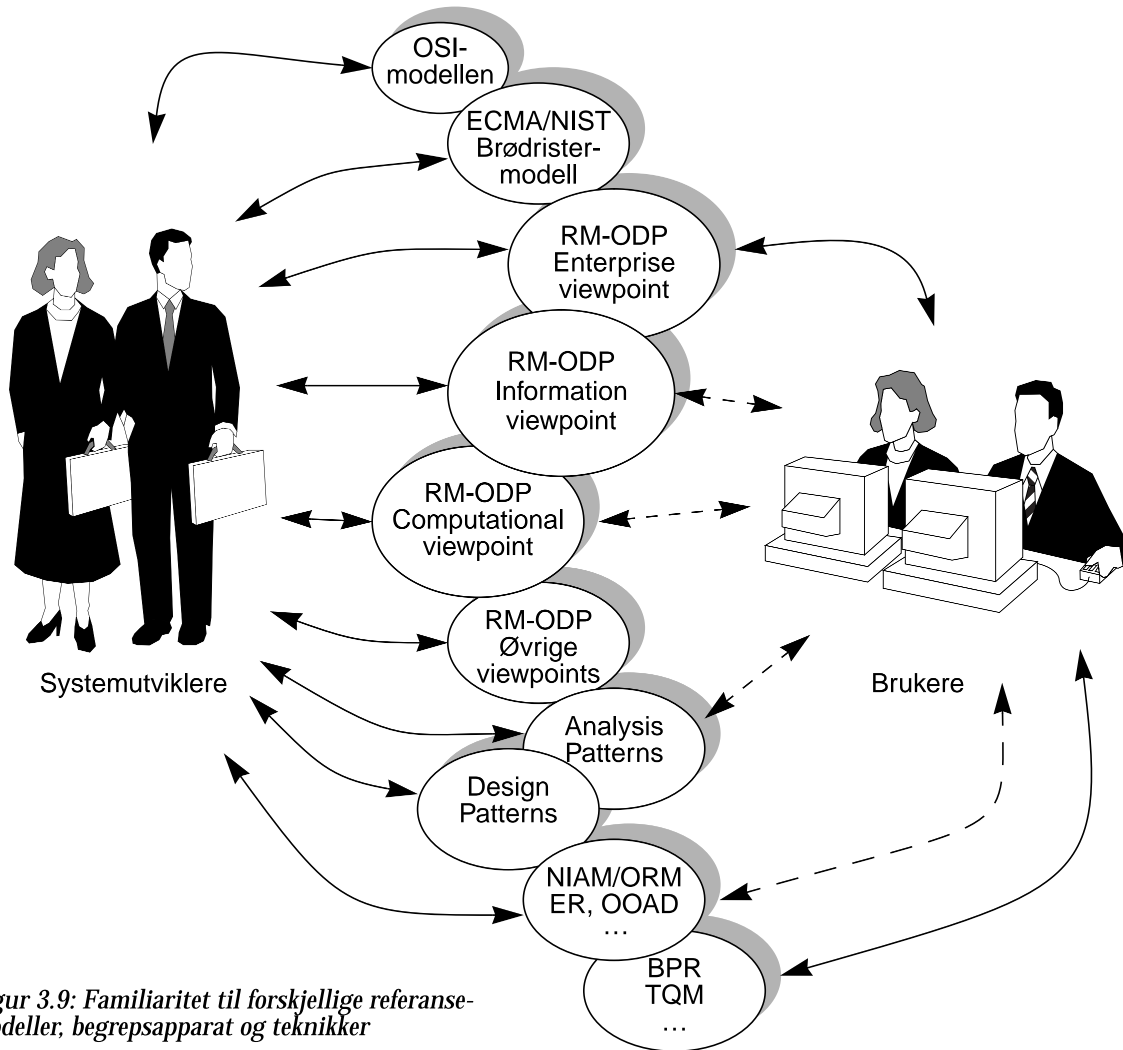
Hvem eier språket?

Som nevnt innledningsvis i dette kapittelet er det viktig å kunne kommunisere og resonnere omkring eksisterende og nye edb-systemer på en måte som gjør at alle innvolverte parter i prosessen kan delta på like vilkår. I møte mellom forskjellige grupper i et systemutviklingsprosjekt er det derfor stor sjanse for misforståelser og forskjellig begrep- og språkbruk ikke bare mellom de ulike gruppene, men også internt i gruppene (referanse til INCA/INSA her).

Det kan lett oppstå en situasjon der én bestemt gruppe mer eller mindre tilraner seg ***modellmakt*** [Bråten 92], og slik overkjører andre deltakere i prosjektet. Denne makten får de fordi det er de som behersker terminologi, verktøy og metoder — i allefall når det gjelder store deler av selve systemutviklingeprosessen og løsningsrommet. En aktuell situasjon kan være når konsulentene ender opp med modellmakten, siden de er engasjert for å lage et nytt edb-system, og de behersker de nyeste begrepene.

Mye av begrepsbruken i referansemodellene omtalt ovenfor kan lett bidra til en slik situasjon. I en stor grad kommer selvsagt dette av at modellene er forholdsvis tekniske, spesielt OSI-modellen. ECMA/NIST-modellen *er* i utgangspunktet ment som en modell for integrasjon av systemutviklingsomgivelser, så den også tar for gitt at «brukerne» av modellen allerede behersker de begreper, tankemåter og uttrykk som brukes. Designmønstre, som presentert i f.eks [Gamma et al 95], gir glimrende begrepsapparat for systemuviklere og programmerere, mens medlemmer av andre grupper nok ikke får så mye ut av dem. Samme gjelder i noen grad analysemønstre som presentert i f.eks [Fowler 97], men siden mønstrene her ligger nærmere problemområdet (eller på et høyere abstraksjonsnivå — se Figur 1.3 og 1.10), vil de sannsynligvis ha størrre nytte for begge aktørgruppene identifisert tidligere. Kun deler av RM-ODP, spesielt i Enterprise perspektivet, later til å kunne forene de forskjellige interesser i et utviklingsprosjekt. I dette perspektivet er det anledning til å diskutere bedriften eller organisasjonens planer, mål og strategier i termer som ligger nærmere for de som planlegger driftsprosesser og de som arbeider med dem til daglig, dvs brukerne av de edb-baserte systemene.

Litt rundhåndet er det mulig å sammenligne de forskjellige tilgjengelige teknikker og referansemodeller og se på i hvilken grad de bidrar til å danne bro mellom grupper i systemutvikling og i størst mulig grad spre modellmakten. Figur 3.9 viser a priori hvem som behersker de forskjellige referansemodeller, begrepsapparat og teknikker. Figu-



Figur 3.9: Familiaritet til forskjellige referansemodeller, begrepsapparat og teknikker

ren antyder at systemutviklere behersker flere av de tilgjengelige mulighetene, og det er grunn til å anta at disse derfor i mange tilfeller sitter med modellmakt i en samarbeidsituasjon med domeneeksperter. Imidlertid er det noen konstruksjoner som er felles; RM-ODP Enterprise Viewpoint og også delvis Information- og Computational viewpoint, analysemønstre og modelleringsteknikker som NIAM/ ORM eller ER. I hvilken grad disse virkelig er felles mellom gruppene er naturligvis varierende, derfor er enkelte «beherskelsespiler» i figuren markert med stiplete linjer.

Selv om de felles tankekonstruksjoner og teknikker ikke alle er referansemodeller eller begrepsapparat i samme forstand som f.eks RM-ODP eller OSI-modellen, er dette teknikker som kan brukes til å formulere slike. Eksempelvis er RM-ODP formulert i et objektorientert paradigme. Både NIAM og ORM egner seg for informasjonsmodellering fordi modellene lett kan tolkes og drøftes med naturlig språk, f.eks norsk. Dermed er det lett å danne felles forståelse mellom domeneeksperter og systemutviklere [Ressem 95].

De gjenværende konstruksjonene i Figur 3.9 er hver for seg brukbare referanserammer for sitt bruk, slik f.eks designmønstre gir et forholdsvis presist begrepsapparat for å kommunisere og resonnere om program- og applikasjonsdesign i objektorienterte termer. Tilsvarende gjelder for begrepsapparater som taksonomien presentert over distribuerte systemer spresentert i [Martin et al 91]. På samme måte som BPR, TQM[1] og andre organisasjons-, arbeidsprosess- og domenespesifikke begrepsapparater synes å domineres av domeneeksperter, er disse imidlertid mest brukbare for systemutviklere og programmerere.

En av mine overordnede ideer med denne oppgaven var i utgangspunktet også å se hvordan organisasjonsmessige forhold og endringer, så som Business Process Reengineering, har innflytelse på konkret distribusjon. I en BPR-sammenheng vil typisk domeneeksperter og brukere sitte med modellmakt i forhold til systemutviklere, siden BPR ikke nødvendigvis er knyttet opp mot innføring av ny informasjonsteknologi.[2] Selv om det ovenfor er identifisert begrepsapparater som synes å kunne bygge broer mellom de to aktuelle aktørgruppene nevnt tidligere, er det ikke gitt at disse reduserer modellmonopolet til en av gruppene overfor den annen. Selv om man skulle tro at modellformidling skulle fordele modellmakten mer rettferdig mellom deltakerne i samarbeidet, argumenterer [Bråten 92] for at det motsatte skjer. Hvordan skal man da oppnå en utjevning av kontrollen? Både systemutviklerne og domeneekspertene har sine distinkte modellparker i utgangspunktet, men de vil ikke nødvendigvis handle om det samme, noe som synes å være en forutsetning for tilstedeværelse av modellmakt i det hele tatt. For å finne de områder modellmakt råder, og slik identifisere områder hvor tiltak for å redusere skjevheter bør settes inn, er det derfor nødvendig å finne ut hvilke modeller som har overlappende referentområde, eller mer presist de områder bare den ene part har utfyllende modeller.

Felles modeller og distribusjon

Imidlertid later det altså til at ikke alt som kan brukes på tvers av faggrensene, dvs mellom systemutviklere og brukere/domeneeksperter, omhandler gjenbruk av eksisterende systemer enn si distribusjon direkte. Det koker ned til RM-ODP's Enterprise Viewpoint, som altså er den eneste arena som innbyr til dialog om distribusjon hvor ikke systemutviklerne sitter med en uforholdsmessig stor del av modellmakten. Det eneste generelle råd som kan gis for å redusere modellmonopol fra noen av sidene, synes således å være å gjøre de involverte partene oppmerksomme på modellmaktteorien, og slik spille med så åpne kort som mulig.

---

[1] Total Quality Management

[2] Tvert imot; i allefall i Norge viser det seg at IT i mange BPR-prosjekter ikke tillegges stor vekt [Iden 95]. På den annen side er det gjort mye arbeid for å «innlemme» BPR-tankegang og metoder inn i objektorientert analyse og design, se f.eks [Jacobson et al 94], [Taylor 95].

Sammendrag

I dette kapitlet har jeg gjort rede for noen viktige referansemodeller, ganske forskjellige hva gjelder kompleksitet og bruksområde. Modellene later ikke til å være i opposisjon til hverandre, men ser — i likhet med de forskjellige perspektivene i den mest sentrale modellen presentert: RM-ODP — på integrerende systemutvikling fra forskjellige ståsteder. Spesielt har jeg erfart at modellene egner seg til bevisstgjøring av problemstillinger gjennom f.eks foredrag og presentasjoner, der flere modeller kan presenteres samtidig, og slik belyse emnet distribuerte systemer fra forskjellige sider.[1]

Ved konkret systemutvikling gjennom analyse, kravspesifikasjon, design og implementasjon kan modellene konkretiseres med beskrivelsesteknikker, som fungerer som felles språk for større eller mindre grupper i arbeidet. Et ideale er selvsagt at språket er brukbart for så mange som mulig, og slik være i stand til å effektivisere samarbeidet mellom systemutviklere og domeneeksperter. Teknikkene har vist seg å ha varierende evner på dette området, slik at valg av teknikk bør vurderes ut fra hvordan de aktuelle aktørgruppene kan tilegne seg dem. Som et utgangspunkt bør derfor velges de teknikker og modeller som det er lett å omsette og diskutere i naturlig språk, som f.eks. NIAM og Enterprise Viewpoint i RM-ODP.

Suksess i større systemutviklingsprosjekter er istor grad betinget av at brukerne er fornøyd og mener de har fått det rette edb-systemet. Dette oppnås med størst sannsynlighet hvis de er med på beslutninger i viktige deler av utviklingsprosessen. En måte å oppnå dette på kan være mest mulig åpen samtale, og da trengs felles begrepsapparater. Kandidater til deler av et slikt apparat er presentert i dette kapitlet, noe såpass nytt at det finnes forholdsvis liten erfaring i systematisk bruk.

[1] I [Berre 96], f.eks, belyses én arkitektur gjennom flere modeller, blant annet RM-ODP.

4 Oss objekter imellom

Innledning

Det utvikles stadig sterkere og mer pålitelige programmeringsomgivelser for distribuerte systemer. Felles for alle disse er at de gir muligheter til å programmere samvirkende systemer på et høyere abstraksjonsnivå, dvs i de øvre lagene i OSI-modellen. Nederst på abstraksjonskalaen finner vi systemkall til selve operativsystemkjernen.[1] Disse tilbyr tjenester på transportlaget, dvs i grenen mellom lag 4 og 5 i OSI-modellen (se Figur 4.1). For lettere å kunne designe og programmere distribuerte systemer over dette nivået finnes flere hjelpemidler for kommunikasjon, f.eks BSD sockets (UNIX)/ WinSock (Windows) og SVR4[2] TLI (Transport Layer Interface), named pipes og FIFO-kanaler [Schmidt 92b] som bringer abstraksjonen opp på sesjonslaget. Over dette igjen finnes mekanismer for fjernprosedyrekall og objektmegling. Disse tilbyr tjenester på applikasjonsnivå, slik at alt fra datarepresentasjon, oppkobling og håndtering av forbindelsen kan overlates til organismer på de lavere nivåer. Akkurat som abstraksjonsnivåer i programmeringsspråk vil nivået i mekanismene som brukes påvirke kompleksiteten i kildekoden, og dermed også muligheter for feil.

Lag i OSI-modellen	
Applikasjon	CORBA
Presentasjon	DCOM
Sesjon	RPC Wrappers
Transport	Sockets/TLI
Nettverk	Operatativ-system-tjenester
Link	
Fysisk	

Figur 4.1: Abstraksjonsnivå blant forskjellige kommunkasjonsmekanismer

I dette kapittelet skal jeg først se på noen av de byggestener som finnes tilgjengelig på de forskjellige abstraksjonsnivåer, og hvordan de håndterer kompleksiteten. Jeg vil også se på hvordan de passer

[1] Jeg antar for anledningen at vi opererer i et miljø som støtter nettverkskommunikasjon, f.eks UNIX, Windows NT etc. I enkelte andre operativsystemer er dette tilleggsutstyr.

[2] Berkeley Software Disitribution (BSD) og AT&T System V rev. 4 (SVR4) er to vanlige UNIX -varianter som danner mønster for flere kommersielle og ikke-kommersielle UNIX-produkter. Socket-familien (BSD) bruker stikkontaktmetaforen; man kobler seg til nettverket med et «støpsel», og operativsystemet gjør resten. SVR4, i større grad enn BSD representert av kommersielle leverandører, bruker en lignende variant; TLI (Transport Layer Interface).

sammen med en objektorientert tankegang, og hvordan faseovergangene påvirkes i systemutviklingsprosjekter som benytter distribusjon. Til slutt vil jeg sammenligne noen av de forskjellige mekanismer for objektkommunikasjon som er tilgjengelige i dag.

BSD Sockets/ SVR4 Transport Layer Interface

BSD sockets og AT&T TLI tilbyr begge tjenester som ligger i overkant av sesjonslaget, dvs at det tilbys funksjonskall for oppretting av forbindelse og transport av binære data. Disse funksjonsporteføljene gjør det mulig å behandle kommunkasjonskanaler på samme måte som filer, fordi enkelte funksjoner ligner på filhånderingsfunksjoner. Det vil si at man bruker lignende syntaks og semantikk på skrive- og lesefunksjoner, og refererer til kommunkasjonskanaler på samme måte som til filer. Både sockets og TLI abstraherer vekk detaljer lenger nede i kommunikasjonssystemet, slik som nettverksprotokollering, som f.eks kan være TCP/IP, OSI, eller IPX/SPX. Bruken av bibliotekene blir stort sett den samme uansett. Valg av protokoller og adresseringsmekanismer velges med heltallsparametre (C preprosessor makroer) til funksjonskallene.

Grovt sett tilbyr Sockets/TLI mulighet til å sende og motta blokker med binære data. Tolkning av innhold og pakking/rekkefølge på dataelementene må programmerereren imidlertid besørge selv. En vanlig måte å sende en abstrakt datatype, som for eksempel består av noen heltall, et flyttall og en tekst osv, er å definere datatypen som en C `struct`. Elementene i denne kan så settes med setninger som `min_struct.heltall_1 = 1` etc. Når så alle elementer er fyllt inn, sendes en kopi av hele det minneområde som strukturen opptar til mottakeren med noe i retning av `send(min_struct, sizeof(min_struct))`.[1]

Imidlertid er det lett å gjøre feil i bruken av disse funksjonskallene. Dette kommer av at alle funksjonene som er tilgjengelige for kommunikasjon på dette nivået er tilsynelatende presentert bare på ett abstraksjonsnivå [Schmidt 92a], siden grensesnittet er tilgjengelig i C. Man må selv passe på at det brukes riktig. Derfor er det fort gjort å ikke oppdage at man f.eks feilaktig pålegger en tjener-konstruksjon klient-oppførsel før noe skjærer seg under kjøringen av applikasjonen. Den som bruker biblioteksfunksjonenen må også beherske en unødvendig stor del av finessene, for ikke å risikere at noe blir galt. Programmereren må altså sette seg inn i mange detaljer, som ikke direkte vedkommer selve applikasjonen og applikasjonslogikken som skal utvikles.

Wrappers

En mulig løsning på problemet som er skissert ovenfor er å «pakke inn» socket- eller TLI-kallene i klasser som sørger for at ting gjøres på

[1] Denne fremstillingen er noe forenklet.

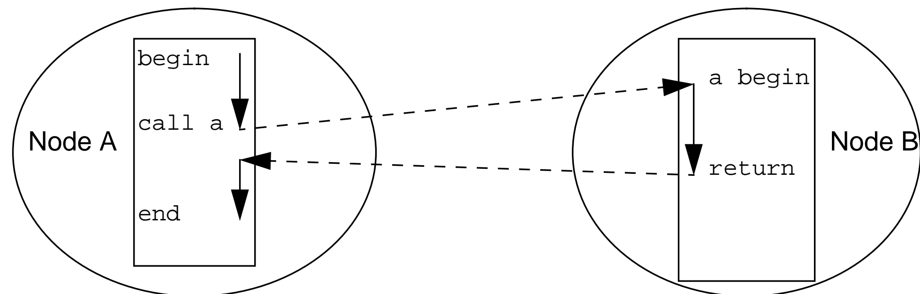
riktig måte. Ved å innføre et objektorientert rammeverk som kapsler inn socket/TLI-kallene kan detaljer i underliggende kommunikasjonssystem skjules, og man får en lavere feilprosent. I forslaget til innpakningsklasser som presenteres i [Schmidt 92b] brukes C++ parametriserte klassedefinisjoner (templates), slik at valg av riktig type objekter bestemmes av parametre. Dermed kan man oppnå en høy grad av portabilitet, og samme applikasjon kan kompileres lettere på flere platformer. Målene med forslaget er formulert som: *«(…) easy to use (…) correctly, hard to use incorrectly, but not impossible to use it in ways the class designers did not anticipate originally»*. Dette representerer altså en bottom-up filosofi, som ikke sperrer for fremtidige utvikling.

Av de fordeler med innpakningsklasser som identifiseres i [Schmidt 92a] og [Schmidt 92b] kan noen nevnes:

- *Høyere grad av riktighet (correctness):* Rammeverket gir fordeler ved sterkere typing som høynivåspråk gir. Applikasjoner får ikke muligheter til å kalle lavere C-funksjoner uten videre og dermed stå i fare for å klusse ting til.
- *Lettere å lære:* Objektorientert innpakning med hierakisk klassetruktur er lettere å lære fordi de gir mer fornuftig og intuitiv gruppering av funksjonalitet.
- *Portabilitet:* Samme applikasjon kan implementeres på flere plattformer ved at plattformspesifikke detaljer skjules av innpakningsklassene, og ikke spesialiseres før kompilering/linking.
- Programkode blir lettere å lese og forstå, samtidig som kodesegmenter som besørger kommunikasjon kan bli ca 50% kortere enn tilsvarende kode som bruker sockets eller TLI.

Fjernprosedyrekall

På et nivå over sockets, TLI og wrappers som beskrevet ovenfor finnes fjernprosedyrekall (Remote Procedure Call — RPC). Mekanismen er blant annet beskrevet i [Birrell & Nelson 83] og [Comer & Stevens 93]. Fjernprosedyrekall fungerer i prinsippet som vanlige funksjonskall, bortsett fra at koden som utfører selve kallet kan befinne seg på en annen node, gjerne på en annen plattform. Som Figur 4.2 viser, overføres programkontrollen og eventuelle parametre til kallet fra node A til node B, og etter at det er utført returneres kontrollen til A sammen med eventuelle returverdier. Bibliotek og hjelpeprogrammer for RPC besørger alt som har med parameterinnpakking, rekkefølge på dataelementene, fragmentering og defragmentering av datapakker, finne frem til mottaker, sikkerhet etc. [OSF/DCE 92]:3-9. Disse detaljene må man selv ta seg av når man bruker sockets/TLI eller andre mer primitive mekanismer.



Figur 4.2: Fjernprosedyrekall mellom to noder

Under eksekveringen av applikasjoner med RPC[1] må tjenerprogrammer registrere seg og sine tjenester hos en demon på vertsnoden.[2] Programmer og fjernprosedyrer identifiseres med nummer, både på programmet, versjonen og prosedyren. Offentlige programnummer skal være entydige, så de forvaltes av SUN. Versjonsnummer brukes for å velge riktig versjon av prosedyrer, slik at tjener og klient ikke trenger å være av samme versjon for å kunne spille sammen. Eksempel på applikasjon som er implementert ved hjelp av RPC er SUN NFS (Network File System) [Comer & Stevens 93], som blant annet brukes her på Instititutt for Informatikk. Mange av tjenestene definert i DCE (Distributed Computing Environment), f.eks DCE's distribuerte filsystem DFS (Distributed File Service) er tilsvarende implementert med DCE RPC.

Applikasjoner som skal distribueres ved hjelp av fjernprosedyrekall lages på nogenlunde samme måte som om de var ikke-distribuert, men man bruker hjelpeprogrammer[3] for å generere RPC-spesifikk kode. Distribusjonen beskrives i et eget språk, RPCL (Remote Procedure Call Language) eller IDL (interface Definitions Language) for SUNrpc og DCE RPC, henholdsvis. For hver funksjon som skal gjøres tilgjengelig med fjernprosedyrekall lager hjelpeprogrammet en stump (*stub*) som kalles på klientsiden istedenfor den opprinnelige funksjonen. Samtidig genereres et tjenerskjelett som kaller den opprinnelige funksjonen når det ankommer fjernprosedyrekall. Under linking av binærkoden etter kompilering byttes så den originale med stumpen i klinten. Originalen linkes så inn i tjeneren i steden, slik at den kalles *der* når tiden er inne. Hvis klient- og tjenerplattformene er forskjellige, separatkompileres originalfunksjonen etter behov.

Objektmeglere

Objektmeglere (Object Request Broker — ORB) kan betrakes som en videreføring av fjernprosedyrekall. Gangen i implementasjonen av et distribuert system er ganske lik: En egen fil beskriver grensesnittet

1. Ettersom jeg har mest erafring med RPC på SUN-plattformer tar jeg utgangspunkt i SUNrpc her.
2. SUNrpc bruker **`portmap`** demonen.
3. For eksempel **`rpcgen(1)`** (SunOS)

mellom klienter og tjenere. Denne filen kompileres, og det genereres stumper for klientene og skjelett for tjenerne. Faktisk så er enkelte implementasjoner av objektmeglere basert på underliggende RPC-mekanismer.

En stor forskjell mellom bruk av objektmeglere og fjernprosedyrekall ligger imidlertid i at sistenvnte i mye større grad er i tråd med objektorientert tankegang. Dette ivaretar sømløse faseoverganger i systemutviklingen på en helt annen måte enn ved fjernprosedyrekall, som vanligvis er implementert i C-kode. Slik vil ikke objekttankegangen så lett stoppe ved implementasjonen av systemet. Det vil også koste vesentlig mindre å lage distribuerte objektsystemer, siden kommunikasjonen mellom dem ivaretas av velprøvde standardmekanismer. Som ved bruk av RPC er distribusjonen ikke i fokus ved selve utviklingen av applikasjonen, men den kommer til syne ved bruk av hjelpeprogrammer.[1] Det genereres også her stumper og skjeletter, og en demon (selve objektmegleren) besørger kontakt mellom klient- og tjenerobjekter.

Forhold til referansemodellene

Plasseringen av de forskjellige kommunikasjonsparadigmene presentert ovenfor er allerede antydet for OSI-modellen på side 55. Tilsvarende kan de også i varierende grad plasseres i forhold til de andre modellene fra Kapittel 3. Eksempelvis vil objektmeglere, fjernprosedyrekall etc utgjøre kontrollintegrasjonen i Brødristermodellen i Figur 3.5 på side 40. Dette viser én måte referansemodeller kan brukes på, nemlig som en felles forståelseshorisont for formidlere og mottakere av tanker om distribuerte edb-systemer. En annen er å sammenligne systemer for støtte av distribusjon ved å se på samsvarighet med referansemodeller, som f.eks arbeidet presentert i [Plowiec 96].

Object Management Group

Figur 4.3: OMGs gode merke

Object Management Group ble formelt dannet i 1989. Initiativtakere da var DATA GENERAL CORPORATION, HEWLETT-PACKARD og SUN. Siden da har antall medlemmer i gruppen vokst til over 700 medlemmer. Medlemslisten i dag inneholder navn som IBM, MICROSOFT, NETSCAPE COMMUNICATIONS, SINTEF, NORSK REGNESENTRAL m.fl. Gruppens mål er ved enighet å komme frem til standarder for infrastruktur mellom samvirkende objekter i distribuerte systemer. Jevnlig holdes det messer rundt om i verden i regi av OBJECT WORLD CORPORATION (OWC) for å promotere objektteknologi. OWC legger spesiell vekt på kommersielle og praktiske muligheter og sider ved

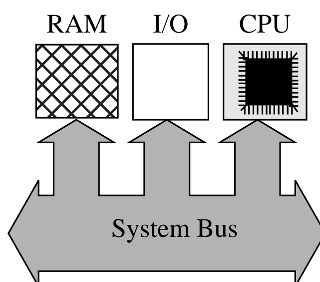
[1] F.eks `idl(1)` som hører med i Orbix suiten fra Iona Industries.

objektteknologi. OWC hedrer også vellykkete utviklingprosjekter innen flere kategorier. De beste i 1995 er f.eks presentert i [Harmon & Morrisey 96].

En av de viktigste mekanismene for å komme frem til standarder i OMG-arbeidet er at medlemmene av konsortiet sender inn forslag til løsninger på problemer som defineres av sekretariatet, og deltakerne blir enige om den beste løsningen som foreslås, eller en kombinasjon av flere gode løsninger. Et viktig poeng er at implementasjon av forslagene *skal* kunne leveres kommersielt innen 1–1,5 år. Dette kan bidra til at standarden utvikles forholdvis raskt, og siden det skal finnes implementasjoner ved aksept av en delstandard eller senest etter halvannet år, sikres at standarden ikke så lett blir en sovende en.

Object Management Architecture

Det viktigste resultatet av arbeidet som Object Management Group har ledet de siste årene er standarden for kommunikasjon mellom distribuerte objekter: Common Object Request Broker Arcitechture (CORBA). Grunnlaget i CORBA-arkitekturen er ideen om en felles buss for alle objekter som skal kommunisere i distribuerte edb-systemer. Lik en utvidelsesbuss i en datamaskin, f.eks ISA- eller PCI-bussen i Intel x86 baserte maskiner, tilbyr bussen visse tjenester til de enheter som er koblet til. En maskinvarebuss tilbyr f.eks strøm (5V, 12V) til å drive logikkretsene på enheten, adresseringslinjer, datalinjer og kontrollinjer osv. De forskjellige enhetene som er koblet til bussen (se Figur 4.4) kan kommunisere med hverandre og med hovedprosessoren i maskinen gjennom bussen. Hukommelse og inn/ut-porter har hver sin adresse i adresserommet. Dette kalles gjerne en von Neumann-arkitektur etter John von Neumanns «IAS» maskin [Tanenbaum 90]. Tilsvarende hjelper objektbussen alle objekter som vil kommunisere slik at de kan løse sine oppgaver i fellesskap.[1] Det er viktig å erkjenne at det ikke nødvendigvis trenger å være noen busstopologi mellom nodene i systemet i fysisk forstand, slik f.eks Ethernet 10Base2-kabel[2] som trekkes fra maskin til maskin.



Figur 4.4: von Neumann bussarkitektur

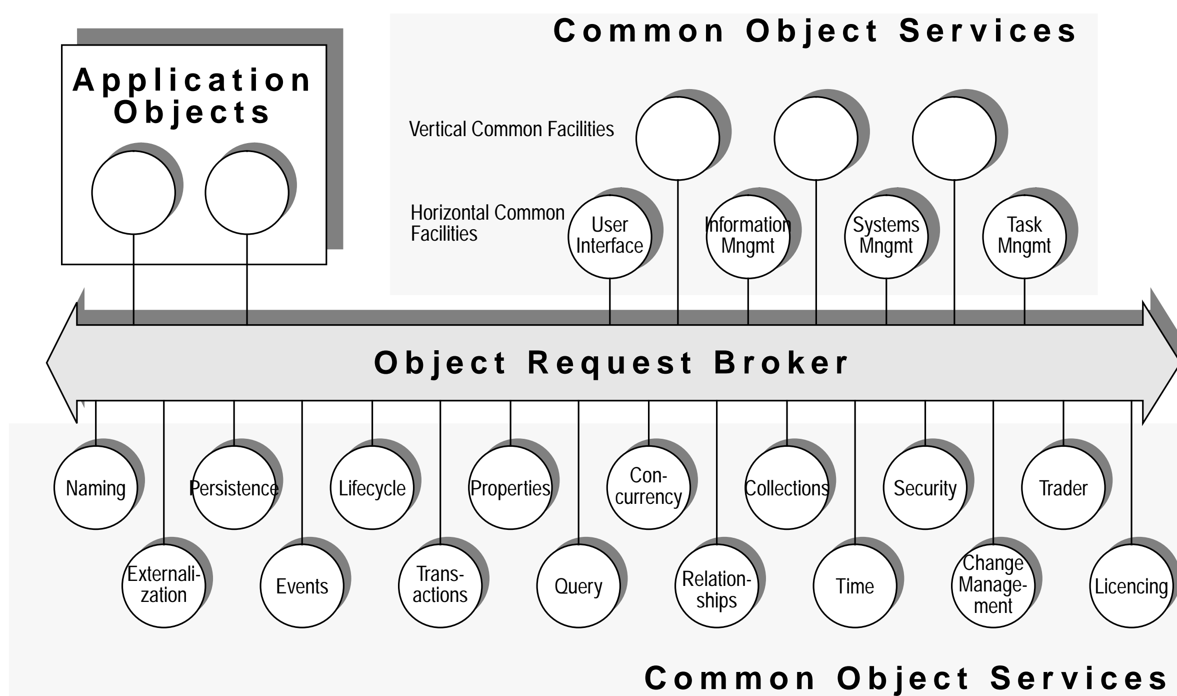
Kommunikasjonen mellom objektene organiseres av en megler (*broker*). Objektmegleren er det sentrale element i et CORBA-basert distribuert system, og er det aller minste en CORBA-implementasjon må tilby. Megleren kan i seg selv være distribuert, slik at den opptrer på alle de noder som kan tilby objekter utad. I praksis løses meglingen gjerne ved at megleren fanger opp alle forespørsler mellom de forskjellige objektene — både de som ligger lokalt og de som befinner

1 Analogien mellom objektbussen og von Neumanns arkitektur strekker seg hit og ikke lenger. Sistnevnte er en typisk *SISD* (Single Instruction Single Data) arkitektur med tett kobling, mens et system av distribuerte objekter (over flere noder) må kalles en *MIMD* (Multiple Instruction Multiple Data) arkitektur med løs kobling. Disse finnes som motpoler i taksonomien presentert i [Tanenbaum 90]:491.

2 10Base5 og -2 er begge forholdsvis umoderne Ethernetstandarder, men brukes gjerne i laboratorier eller rom med flere maskinen som kobles i lokalnett. Mer moderne er 10BaseT, som kobles sammen til oppsamlingspunkter.

seg andre steder — og sørger for å finne ut hvor riktige målobjekter finnes og at de er tilgjengelige. Hvis så objekter flyttes, skal ikke de applikasjoner som har bruk for disse objektene være nødt til å rekompileres eller rekonfigureres. Megleren vil sørge for at flyttingen er transparent. I et UNIX-miljø vil megleren typisk være en demon på hver node som samarbeider med demoner på andre noder, tilsvarende RPC-demonen i SUNrpc.

Figur 4.5 viser hvordan Object Management Architecture (OMA)



Figur 4.5: OMG Object Management Architecture (etter [Orfali et al 96b] s 54)

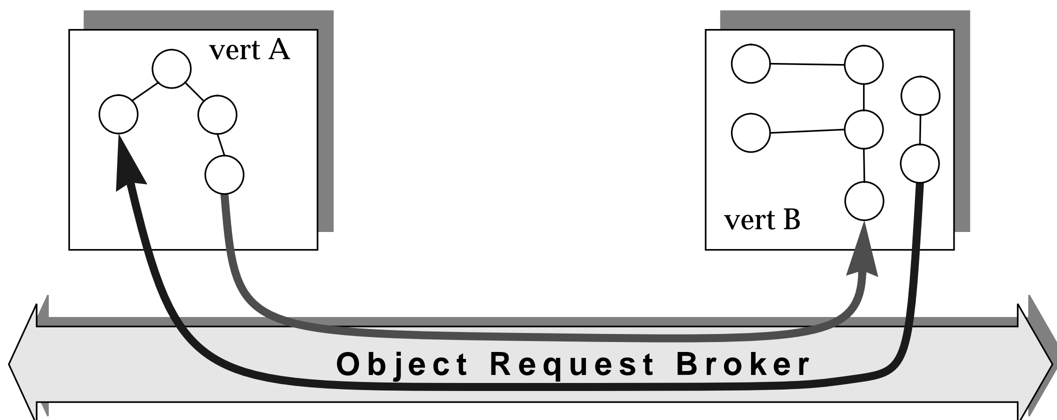
gjerne fremstilles i litteratur, både fra OMG og andre kilder, f.eks [Orfali et al 96b] og [Crowcroft 96]. Dette er, i likhet med OSI-modellen (se «OSI-modellen» på side 37) en utmerket modell for samtale på et rimelig høyt abstraksjonsnivå. Ved første øyekast virker den imidlertid like lite innlysende som OSI-modellen virker ved første møte, men etterhvert vil det være en passelig beskrivelse av CORBA-arkitekturen og plassering av tjenestene som forventes av et CORBA-standardisert mellomvareprodukt.

Jeg tror imidlertid det er lurt å erkjenne hva som ligger i bunnen av det hele først. Det distribuerte objektsystemet skal tross alt implementeres på et antall fysiske noder, og disse kan kobles sammen gjennom et medium, f.eks et lokalnett — eller over Internett for den saks skyld. Man kan da forestille seg at det man i første omgang får hjelp til av en objektbuss er kommunikasjon mellom det man velger å kalle

objekter, slik at de kan sende data seg imellom. All formatering, pakking og protokollering blir imidlertid tatt hånd om av objektbussen, slik at systemutviklerne kan konsentrere seg om semantikken i kommunikasjonen og selve applikasjonslogikken.

Kommunikasjonen mellom objekter

Figur 4.6 viser et eksempel på hvordan objekter på vertsnoder kan samarbeide. Verdt å merke seg er at det ut fra figuren ikke er lett å se hvem som er klient og hvem som er tjener hvis man ser det som et klient/tjener-forhold. Dette er også noe av poenget med distribuerte objektsystemer; at alle objektene blir likeverdige (*peers*). Selvsagt kan man identifisere hvem som spiller hvilken rolle ved å se på hvem som hadde initiativet til kommunikasjonen, dvs hvem som spurte hvem først, men det er ingen hindring for at dette kan snu seg i neste omgang. Distribuerte objekter kan altså være både klienter og tjenere, gjerne samtidig! Modellen i Figur 4.6 illustrerer på en ganske generell måte hvordan objekter forholder seg til hverandre over objektbussen. I praksis vil koblingen mellom objektene og bussen foregå ved hjelp



Figur 4.6: Kommunkasjon mellom objekter på to vertsmaskiner

av en CORBA Application Programming Interface (API), objektadaptere eller *stubs*.

Når man har erkjent denne modellen av hva som skjer, passer det å ta for seg CORBA-arkitekturen slik den vanligvis fremstilles og diskuteres. Forståelsen av CORBA-arkitekturen blir som OSI-modellen og RM-ODP lettere hvis man bruker en tilnærming hvor man pendler mellom abstrakte prinsipper og konkrete løsninger. Etter å ha presentert objektenes interaksjon med hverandre på et rimelig grovt plan, vil jeg derfor i det følgende se mer teknisk på hvordan dette brukes.

Grensesnittet mellom objektene

Grensesnittene mellom objektene beskrives i et eget språk; *Interface Definition Language* (IDL) som definerer hvilke tjenester objektene

tilbyr. IDL er en del av CORBA-standarden. En IDL-beskrivelse gir således en spesifikasjon i Computational Viewpoint i RM-ODP. Grensesnittene kan også betraktes som en *typebeskrivelse* av objektene, uten implementasjonsdetaljer. Selve innmaten, dvs implementasjonen av metodene og provate datastrukturer inni objektene kan om ønskelig beskrives som klasser et annet sted. Det er imidlertid ikke noe krav at objektene er implementert som objekter. Hva som skjuler seg inni dem kan opphavet om ønskelig holde for seg selv. Man velger altså å *se* alt som objekter fra utsiden, på samme måte som en alt ser ut som spikere for en som har en hammer.

En IDL-spesifikasjon av en programvarekomponent forteller på en formell måte hvilke grensesnitt komponenten tilbyr interesserte. Det fortelles ikke *hvordan* objektsystemet er implementert — om det i det hele tatt *er* et objektsystem. IDL-formatet er inspirert av C++-syntaks, og faller således mellom C, C++ og Java. IDL-grensesnitt for karttjeneren i eksempelet fra Plan- og bygningsetaten kan da inneholde noe slikt:[1]

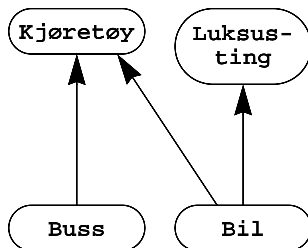
```
// To skråstreker betyr at resten
// av linjen er en kommentar

interface Karttjener {
      Kart hent_kartbit(in Punkt Nordvest, in Punkt Sørøst);
      Punkt find_adresse(in string addresse);
};

...
```

Dette beskriver altså et grensesnitt som tilbyr to metoder/funksjoner: ett som henter en bit av et kart innenfor et rektangel som er avgrenset av **`Norvest`** og **`Sørøst`** parametrene, og ett som finner referansepunktet til bygningen på en gitt adresse. Implisitt her er at datatypene **`Punkt`** og **`Kart`** er definert et annet sted i IDL-spesifikasjonen, som f.eks:

```
...
struct Punkt{
      long x;
      long y;
}
struct Kart {
      // Et kart er en mengde grafiske objekter, f.eks punkter,
      // linjer, kurver, sirkler, tekster etc.
      // For punkter og linjer:
      ...
      sequence<Punkt> Punktene
      sequence<Linje> Linjene
      ...
}
...
```

[1] Se Tillegg 6 for fyldigere beskrivelse.

Et viktig poeng er at selve implementasjonen av objektsystemet bak et grensesnitt er helt opp til den som har laget den delen av systemet. Det eneste som forventes er at det er et samsvar mellom grensesnittet definert med IDL og de tjenester som faktisk tilbys. Det legges heller ingen føringer på valg av programmeringsspråk, teknologisk plattform eller for den saks skyld analyse- og designmetodikk bak objektsystemene som tilbys, så lenge objektenes grensesnitt følges. Det trenger ikke engang være noe objektsystem bak fasaden i det hele tatt, men gjerne en applikasjon skrevet i maskinspråk, C, Pascal eller COBOL for den saks skyld. Imidlertid bør man benytte de kraftfulle og utrykksfulle mekanismene som finnes i objektorientering, og f.eks utvikle komponenter, dvs frittstående byggestener på er rimelig høyt nivå, som senere kan sys sammen med andre, med objektbussen som «lim».



Figur 4.7: Klassehierarki over kjøretøy II

Et annet poeng med CORBAs IDL-grensesnitt er at de er multippelt arvelige, på samme måte som klasser og grensesnitt er det i objektorienterte språk, f.eks Java. Således kan man f.eks tenke seg IDL-grensesnitt for kjøretøymodellen i Figur 4.7 — inspirert av Figur 1.4 på side 11. **`Bil`** er her ikke bare et **`Kjøretøy`**, men også en skattbar luksusgjenstand. Dette kan representeres i følgende IDL-grensensnitt:

```
interface Kjøretøy {
     float attribute[1] vekt;
     float attribute pris;
     ...
};

interface Luksusting {
     float beregn_skatt();
     ...
};

interface Bil: Kjøretøy, Luksusting {
     // her er vekt, pris og beregn_skatt() arvet
     ...
};
```

På samme måte som i objektorienterte språk kan her bil-objekter refereres til enten som **`Bil`**, **`Kjøretøy`** eller **`Luksusting`**, avhengig av sammenhengen.

Implementasjonslageret

Når objekter gjør sine tjenester tilgjengelige via megleren, er det denne som registrerer grensesnittene til objektene i grensesnittlageret (Interface Repository). Dette fungerer som en distribuert database med informasjon om alle objekter som er tilgjengelige og deres gren-

[1] Når IDL-spesifikasjon kompileres for C++ målkode, blir **`attribute`** til en operator/funksjon med samme navn [ORBIX 95]. For Java målkode legges på en **`get_`** eller **`set_`** foran attributtnavnet for henholdsvis lese- eller skriveoperasjon, f.eks **`get_pris()`**.

sensnitt. Dette er spesielt interessant i de tilfeller der grensesnitte ikke er kjent på forhånd, dvs ved bruk av dynamisk grensesnittbinding (Dynamic Invocation Interface — DII/ Dynamic Skeleton Interface — DSI) fremfor statisk (Static Invocation Interface). Jeg vil imidlertid ikke ta opp dynamiske binding videre her.

CORBA tjenester — CORBAServices

Et mellomvareprodukt som skal tilfredstille CORBA 2.0 standarden, må ivareta et passelig utvalg av de 16 basale tjenester som er definert i *CORBAServices*. Disse 16 presenteres er:

Navnetjeneste. Skal sørge for å koble navn med objekter.

Eksternalisering. Gir muligheter for konvertering av objektstatus slik at det kan transporteres som en strøm for eksempel for lagring på fil.

Hendelseshåndtering. Koordinerer hendelser (events) mellom objekter. Objekter kan generere eller «abonnere på» hendelser.

Livssyklus. Støtter generering, kopiering, flytting og destruksjon av objekter.

Trader. Hjelper klienter med å finne sine tjenere.

Egenskaper. Tjeneste som registrerer tilleggsattributter til objekter.

Transaksjoner. Sørger for at transaksjoner som innbefatter mer enn ett objekt følger ACID-krav (Atomicy, Concistency, Integrity, Durability)

Samtidighet. Beskytter objekters integritet når de utsettes for flere klienter samtidig.

Persistens. Lagrer objektstatus transparent for klienten.

Spørring. Tilbyr muligheter for å søke etter objekter.

Asossiasjoner. Håndterer forhold mellom objekter, navigasjon mellom dem og struktur.

Mengdekontroll. Støtter generering og håndtering av mengder av objekter.

Sikkerhet. Støtter autentisering, autorisering, integritet og privatliv.

Tid. Sørger for en uniform oppfatning av tid blant alle objektene.

Versjonshåndtering. Støtter identifisering og konsistent evolusjon av objektsystemer.

Lisensiering. Sørger for å ta betaling for bruk av objekter.

CORBAfacilities — ekstratjenester

I tillegg til de sentrale tjenestene ovenfor, ønsker man flere tjenester. Disse kalles CORBA facilites og deles inn i to grupper: horisontale og vertikale. De horisontale ser man som nyttige for mange typer virksomhet, mens de vertikale er mer bransjespesifikke. De vertikale fasiliteter kan være større objektrammeverk som er definert ut fra spesielle bransjebehov, f.eks helse/omsorg, finans, oljeleting etc. Innen hver av disse finnes ofte lignende behov, så man kan ta utgangspunkt i et ferdig rammeverk og tilpasse det sin egen situasjon. Til nå har lite av dette blitt implementert, ikke engang fullt ut spesfisert.

Det finnes heller ingen forslag til rammeverk for offentlig forvaltning. Her mener jeg det kan ligge en stor gevinst ved å på sikt definere et standard objektrammeverk for offentlig forvaltning, og så innføre i kravspesifikasjonen i systemutviklingsprosjekter på oppdrag fra offentlige etater at dette rammeverket skal følges. Man kan tenke seg rammeverk for saksbehandling, arkivering og annen administrasjon, som kan spesifiseres med CORBA IDL. Hvilken plattform de forskjellige leverandører av systemene så velger å bruke er opp til dem, men man sikrer seg at systemene er interoperatible (gjennom Internet Inter-Orb Protocol — IIOP) slik at integrering nå eller i fremtiden vil bli lettere å gjennomføre. Selv om f.eks KOARK-standarden [KOARK 95] setter endel konkrete krav og føringer på design av arkiveringssystemer, sier den lite om integrering mellom forskjellige edb-systemer.

CORBA og de andre

Det finnes i dag flere andre tilgjengelige teknologier for nettverksintegrasjon, f.eks MICROSOFTs DCOM (DISTRIBUTED COMPONENT OBJECT MODEL), NETSCAPE ONE (OPEN NETWORK ENVIRONMENT),[1] JAVA RMI (Remote Method Invocation). Disse er i varierende grad modne, dvs implementert som stabile produkter. Generelt regnes CORBA som den mest gjennomtenkte standarden, men de tilgjengelige implementasjoner er ikke alltid komplette. Som regel tilbys i utgangspunktet kun de viktigste objektmegertjenester, men flere tjenester vil ventelig komme, og noen kan allerede kjøpes som ekstrautstyr. Siden både DCE og DCOM er basert på eksisterende teknologi,

[1] Netscape ONE innbefatter imidlertid bl.a. også CORBA

har de fordel av å være tidlig ute med fungerende løsninger. I det følgende er noen av disse andre standardene sammenlignet med CORBA.[1]

CORBA vs DCE

«DCE is just a fancy RPC with security and directory service.»

[Orfali et al 96b]:89

OSF (Open Software Foundation) DCE (Distributed Computing Environment) har blitt en industristandard for distribuerte systemer. Miljøet tilbyr tjenester som fjernprosedyrekall, navnetjenester, sikkerhet, tidstjeneste, nettverksfilsystem mm., se f.eks [OSF/DCE 92], [Plowiec 96]:106–138. I likhet med CORBA og DCOM brukes et eget grensesnittspråk, DCE IDL, som ikke støtter arvelige grensesnitt slik som CORBA. CORBA og DCE er forøvrig forskjellige på noen vesentlig punkter:

- DCE er vinklet mot distribuerte systemer primært realisert i C, mens CORBA i så måte passer bedre sammen med C++.[2] DCE er altså i utgangspuktet *ikke* objektorientert, i motsetning til CORBA. Dette gjør at CORBA vil passe lettere inn i objektorientert systemutvikling.
- DCE bygger på allerede eksisterende løsninger, som betraktes som referanseimplementasjoner. CORBA forutsetter på sin side at foreslåtte løsninger skal kunne implementeres av forslagstilleren innen 18 måneder fra forslaget er godtatt.
- DCE betraktes gjerne som mer moden og komplett enn CORBA. Dette er ingen overraskelse, tatt i betrakting hvor omfattende CORBA-arkitekturen er [Carlson et al 96], i tillegg til at DCE-standarden som nevnt i forrige punkt er basert på fungerende løsninger.

DCE har også en del andre begrensninger, blant annet [Mock 93]:

- Grov granularitet; minste distribuerte enheter er klienter og tjenere på kapselnivå, som kan bli ganske store «klumper».
- Assymetrisk kommunikasjon; støtter vanlig klient-tjener arkitektur, men det er ikke helt trivielt å bytte om disse rollene.

Siden DCE altså ikke er objectorientert slik som CORBA, vil det være vanskeligere å opppnå sømløse faseoverganger i objektorienterte utviklingsprosjekter, fordi implementasjonen på et for tidlig tidspunkt — eller for «høyt» i abstraksjonsmodellen i Figur 1.10 på side 19 — må tilpasses ikke-objektorientert implementasjon.[3] Imidlertid er distribusjonsmekanismene i DCOM (se nedenfor) basert på

[1] Java Remote Method Invocation (RMI) er ikke behandlet her, siden denne mekanismen ikke er sørlig språknøytral.

[2] — og en mengde andre objektorienterte språk, f.eks Smalltalk, Java m.fl

[3] Objektorienterte tilpasninger til DCE finnes imidlertid, f.eks DCE++ [Carlson et al 96].

mekanismer fra DCE, blant annet sikkerhet og fjernprosedyrekall [COM 95], [Brown & Kindel 96].

CORBA vs DCOM

«Creating a new object model on top of DCE (…) is totally ridiculous. The world needs another object model like it needs a hole in its head.»

[Orfali et al 96b]:89

«[DCOM] is an application-level protocol for object-oriented remote procedure calls (…) layered on the [DCE] RPC specification (…)»

[Brando 95]

Jeg skal ikke gå nøye inn på hvordan Microsofts DCOM modell fungerer, for den er ganske omfattende. DCOM er Microsoft og DEC (Digital Equipment Corporation.) sin standard for distribuerte objekter. DCOM tilbyr basal objektkontroll; livsløpskontroll,[1] formidling av kontakt mellom — og støtte kommunikasjon mellom — DCOM-objekter og deres klienter. OLE (Object Linking and Embedding)/ ACTIVEX er basert på COM/DCOM igjen. Kommunikasjon mellom klienter og objekter over nettverk er basert på DCE RPC-mekanismen. Grensesnitt kan beskrives med to mulige språk; IDL (Interface Definition Language) og ODL (Object Description Language). Førstnevnet er en utvidelse av DCE IDL, mens ODL er mer objektorientert. DCOM-grensesnitt støtter, i likhet med DCE, *ikke* arv av grensesnitt, i motsetning til CORBA. Grensesnitt kan heller delegeres eller aggregeres. Ved delegering vil ett DCOM-object videresende metodekall til andre DCOM objekter. Ved aggregering av grensesnitt tilbys innkapslede objekters grensesnitt direkte utad. Mer om dette i f.eks [COM 95], [Orfali et al 96b]:449–52.

DCOM eies og styres av MICROSOFT og DEC, og det kan bety at definisjon av standarden formodentlig er mye mer smertefri og ubyråkratisk enn f.eks CORBA, som er dannet på konsensus blant en stor menge innflytelsesrike selskaper. Selv om DCOM er en «åpen standard» [Brown & Kindel 96], er det lite som tyder på at Microsoft vil la andre kontrollere den. Dette er nok også grunnen til at DCOM ikke er foreslått som del av CORBA, selv om Microsoft er medlem av OMG. Formodentlig er dette for å unnga at Microsoft mister kontroll over standarden. Det er også viktig å huske på at det er opp til MICROSOFTs forgodbefinnende å forandre på standarder rundt DCOM, men dette er et problem som forsåvidt gjelder mange proprietære standarder.

DCOM er en videreføring av OLE-teknologien, som har utviklet seg over de siste årene ut fra et økende behov for applikasjonsintegrasjon. Dette viser en klar top-down filosofi bak utviklingen, dvs at utviklingen av paradigmet til en hver tid styres av umiddelbare behov eller gode ideer og innfall. Dette viser seg blant annet i historikken bak

[1] dvs bl.a oppretting og fjerning av objekter.

teknologien, fra dynamiske bliblioteker, gjennom enkle datautvekslingsmekanismer til COM-modellen. Forskjellige problemer har oppstått som følge av «uheldig» design, slik at OLE-teknologien i vesentlig grad er endret nå til sammenligning med forløperne, f.eks DDE (Dynamic Data Exchange) og OLE1.0 [Brown & Kindel 96]. Generelt angriper CORBA problemstillingen fra den en annen kant; bottom-up. Byggeklossene defineres heller *før* det er klart hvordan de skal brukes. Sideeffekten av dette er at ambisjonsnivået kan bli noe høyt, og det kan ta lang tid før noen komplette CORBA-implementasjoner foreligger. Dette er imidlertid den tilnærmingen jeg tror man heller bør ha når man skal utvikle rammeverk og verktøysett, for ikke å bli for spesifik og «kit»-orientert.[1]

Microsofts dominans i kommersielle og private miljøer samt i de fleste administrative kontormiljøer gjør at mange nok velger Microsoftløsninger for sikkerhets skyld eller av gammel vane. Microsoft later således til å ha overtatt IBMs slagord *«Nobody was ever fired for bying IBM»*. Dessuten var DCOM-teknologien tilgjengelig kommersielt på et tidspunkt mange programvarehus begynte å ta i bruk distribuert objektteknologi, mens det var altså ikke CORBA. Eksempelvis er Nauticus-prosjektet ved Det Norske Veritas — som blant annet innebærer en langsiktig objektmodell for skip klassifisert av DNV — basert på DCOM-teknologi. Hadde prosjektet startet i dag, hvor stabile CORBA-implementasjoner begynner å dukke opp, er det ikke usannsynlig at valg av objektinfrastruktur ville blitt annerledes enn det ble.

Sammendrag

> ***«There is a natural tendency in a network environment to create entirely new application-level protocols as each new or seemingly unique combination of client, user agent, and server requirement arises.»***
>
> ***[Brando 95]***

Mellomvare, slik som DCE, DCOM og CORBA-baserte produkter, regnes gjerne for å være det som ligger bak bindestreken i klient-tjener arkitektur. Når krav til hurtig og feilfri systemutvikling gjør seg stadig mer gjeldende, er det viktig å kunne bruke så lite ressurser som overhodet mulig på selve «rørleggerarbeidet» i distribuert systemutvikling, og heller mer på selve applikasjonslogikken. Med de kommunikasjonsmekanismer som er tilgjengelig, f.eks fjernprosedyrekall, kan en vesentlig del av kommunikasjonsdetaljene overlates til slike teknikker. Imdlertid passer ikke vanlige fjernprosedyrekall sær-

[1] Forholdet mellom «kits», rammeverk, mønstre og systemutvikling er behandlet i f.eks [Tepfenhart & Cusick 97].

lig godt inn i objektorientert systemutvikling, i allefall ikke like godt som distribuert objektteknologi, slik som f.eks de basert på CORBA-standarden.

Selv om de forskjellige standardene dyrkes av forskjellige leverandører og miljøer, gjøres det mye arbeid med å lage muligheter for koblinger og samarbeid mellom dem. Valg av én standard burde således ikke bety at alle muligheter for skalering og kobling mot andre systemer er utelukket. Jeg tror man skal vurdere valg av åpenhet i standardvalg som en funksjon av de oppgaver edb-systemet skal ivareta. Hvis en vesentlig del av systemets funksjonalitet er å betjene andre eksterne domener som kanskje for øyeblikket er ukjente, vil nok en bra stratergi være å velge mest mulig åpne løsninger. Likeledes virker det fornuftig for en organisasjon å velge en åpne standarder hvis en selv har begrenset innflytelse.

CORBA og Plan- og bygningsetaten

Som nevnt annensteds i denne oppgaven tror jeg at forskjellige edb-systemer i f.eks offentlig forvaltning, så som systemene på Plan- og Bygningsetaten, vil knyttes sammen på én eller annen måte, slik at føderasjoner dannes. Føderasjoner er samlinger av domener med ett eller flere felles mål, men med stor sannsynlighet forskjellig politkk på flere punkter, f.eks valg av teknologi. Helt konkret er det en kjennsgjerning at kreative yrker (som f.eks arkitekter) har en tendens til å velge Macintosh utstyr, mens offentlig forvaltning gjerne har satset på PC'er. Hvis de klassifiseringssystemer med selvpålagt kontroll, som er nedfelt i den nye Plan- og Bygningsloven som trådte i kraft 1. juli 1997, skal støttes av et distribuert edb-system, må dette nødvendigvis i en viss grad bli plattformnøytralt. Dette kan tale for at systemets grensesnitt bør defineres og implementeres i en så åpen standard som mulig, f.eks CORBA. Gode objektmeglere gir gjennom IIOP muligheter for kontakt med andre, gjerne fra andre leverandører. Dette gir muligheter for bedre integrasjon mellom alle disse forskjellige edb-systemene.

5 Metoder for utvikling av distribuerte systemer

Alan:	*And this week on 'How to do it', we're going to show you how to (…) rid the world of all known diseases.*
Jackie:	*Well, first of all become a doctor and discover a marvellous cure for something, and then (…) you can jolly wll tell [the medical profession] what to do and make sure they got everything right, so there'll never be any diseases ever again.*
Alan:	*Thanks, Jackie. Great Idea.(…)*

Fra "Monty Pythons Flying Circus Just the Words", Vol 2 s. 63

Innledning

Distribusjon i systemutviklingsmetoder er forholdsvis nytt. Svært få vanlige metoder har innebygget aktiviteter som tar hensyn til at brukere faktisk ikke sitter i en stor klump på ett kontor og arbeider. Selv om det finnes mange flerbrukersystemer i drift og under vedlikehold og utvikling virker det som det faktum at organisasjoner kan være spredt over store avstander ikke tas tilstrekkelig hensyn til under analyse og modellering. Dette har vært et aspekt ved systemet som utsettes til implementasjonen, og det vil sannsynligvis innføre «magiske» overganger [Smørdal 96] mellom faser i systemutviklingsprosessene. Slike usynlige skritt gjør det vanskelig å forandre eller forbedre systemet og samtidig opprettholde konsistens mellom analyse-/designmodeller og implementasjonen som forandres.

Det er vanlig å sammenligne metoder for strukturert analyse med objektorienterte metoder, og så komme til at de objektorienterte metodene er bedre. Sammanligningen blir spesielt urettferdig når det

kommer til distribusjon, for det er først i de seneste årene at dette har blitt mulig å innføre distribuert databehandling til overkommerlig pris og med en akseptabel ytelse for vanlige organisasjoner.

I et skifte fra sentrale databasesystemer, som gjerne kalles en IRM-strategi (Information Resource Management), til mer desentraliserte løsninger ([Goldkuhl et al 93]: VBS — Verksamhetsbaserad System-strukturering), vil man ha behov for utviklingsmetoder som behandler distribusjon som et viktig aspekt av applikasjonsområdet.[1]

Dette kapitlet vil ta for seg objektorienterte metoder, siden det er i disse man skulle kunne forvente seg at distribusjon var et tema. Spesielt vil jeg se nærmere på OOram og DISGIS-utvidelsen for distribuerte systemer. Av plasshensyn er andre OO-metoder som behandler distribusjon, slik som MOSES [Henderson-Sellers & Edwards 95] m/ utvidelser [Henderson-Sellers & Graham 96] og OPEN/Mentor, som for tiden er under utvikling eller nær fullførelse, ikke behandlet i særlig utstrekning her.

Objektorienterte metoder

«There are many ways to find [the key classes to build truly extensible systems] including:
1) be a genuis;
2) have lots of time;
3) solve the problem three times;
4) know the pattern from previous experience (…);
5) have a method that helps you get there.»

[Haythorn 94]

Objektorienterte systemutviklingsmetoder fremstiller ofte på en svært tilforlaterlig måte hvordan man bør analysere, designe og implementere objektorienterte systemer. Det er vanlig å liste opp et antall aktiviteter utviklingen bør bestå av slik som i [Madsen 93] og [Henderson-Sellers & Edwards 95]. Det er imidlertid ingen tvil om at utviklere trenger mye erfaring med en metode før de kan bruke den på en effektiv måte. Dette gjelder for eksempel problemet med å identifisere hvilke objekter som skal med i analysemodeller, hvilke som blir med videre til design og hvilke som implementeres i det endelige produktet — om det da i det hele tatt implementeres i et objektorientert språk. Man kan altså analysere problemområdet med en passelig objektorientert metode, f.eks Jacobsons «use cases» [Jacobson et al 94], og lage en objektmodell. Ett utgangspunkt for objektidentifisering kan være å streke under alle substantiv i samtaler med domeneeksperter, og supplere eller fjerne etthvert som analysen tar form. Objektklassene herfra kan senere brukes som klasser i design og implementasjonen av et edb-system som skal støtte arbeidsoppgavene i problemområdet. Et vesentlig spørsmål reiser seg imidlertid her: Er det virkelig de samme objektene vi finner i ana-

[1] «Applikasjonområdet» er hentet fra begrepsapparatet presentert i [Mathiassen et al 93].

lysen som vi skal bruke i implementasjonen? Dette spørmålet skal jeg ikke ta opp videre her, i alle fall ikke i en ordinær analysekontekst. Identifisering av objekter er behandlet mer utførlig i [Ressem 95]. Imidlertid er det mer interessant i denne omgang å se hva som blir objekter når man modellerer et allerede eksisterende system.

Sømløse reversible faseoverganger

I tillegg til idealene i systemutvikling identifisert på side 16 kan anføres mulighet for å gjøre overgangene mellom de forskjellige fasene av utviklingsprosessen sømløse og reversible, slik at alle modeller er konsistente med det virkelige edb-systemet, inkludert alle endringer [Smørdal 96]. Dette prinsippet er tilsynelatende ikke nødvendig å realisere i prosjekter som styres etter fossefallsmodellen, men virker mer som en fordel i iterative modeller. Imidlertid vil gjerne en stor del av ressursene i et utviklingsprosjekt etterhevrt brukes på vedlikehold og utvidelser, så en mulighet for reversering og sømløse faseoverganger vil likevel være en stor fordel [Haythorn 94].

I en iterativ utviklingsmodellen må det legges til rette for at de forskjellige fasene kan foregå samtidig eller sammenviklet. Alle vet at det er fristende å tenke design på et tidlig tidspunkt i en utvikingsprosess, og det er ikke uvanlig å skifte mellom analyse- og designaktivitet hele tiden, med minutter eller sekunders mellomrom [Henderson-Sellers & Edwards 95]. Overgangene mellom fasene i systemutviklingsprosjektet bør altså bli mer eller mindre utvisket. Dette betyr at man lettere kan bruke reverskonstruksjon innad i prosjektet, og gjøre endringer i én fase som så avspeiler seg i andre — gjerne tidligere — faser. Dette er innført i flere modelleringsverktøy, f.eks RATIONAL ROSE, OBJECTORY og SELECT ENTERPRISE [SELECT 96].

Modeller og eksisterende systemer

Konteksten i denne oppgaven innebærer, som nevnt i Kapittel 1, aspektet *systemutvikling i miljø der det finnes systemer som er i bruk allerede.* Ettersom det svært ofte finnes eksisterende systemer som man må ta hensyn til, burde dette vært innebygd i de metoder som benyttes. Men det er jo så mange forskjellige systemer, så det vil være vanskelig å finne en felles måte å behandle og modellere dem på. Dessuten vil graden av gjenbruk variere fra prosjekt til prosjekt. Hvis vi antar en objektorientert tilnærming til analysen og modellereringen er det to hovedmåter å behandle gamle systemer på:[1]

- Reverskonstruere det gamle systemet og bruke de enkelte bestanddeler derfra, som klasser, objekter og prosedyrer i en ny modell.
- Innkapsle større komponenter som objekter som lever sitt eget liv, f.eks databaser.

[1] Se også «Objektorientert reverskonstruksjon» på side 30

I praksis vil det være en glidende overgang mellom disse to standpunktene. Ikke bare er de ikke-dikotomiske, men også ikke-ekskluderende. Man kan tenke seg at forskjellige deler av et system behandles på forskjellig måte, hvert tilfelle plasserbart på en skala mellom de to standpunktene ovenfor.

Plasseringen på denne skalaen gir større eller mindre problemer ved modelleringen. Hvis man velger den første tilnærmingen, og remodeller det eksisterende til sine enkelte bestanddeler, vil byggestenene mest sannsynlig bli rimelig sammenlignbare med de nye man innfører som en del av det nye systemet. På den annen side er man her avhengig av hvordan det opprinnelige systemet ble modellert/implementert. Hvis det ble implementert med spagetti-kode, i maskinspråk, FORTRAN, COBOL eller i et annet programmeringsparadigme med liten slektskap med objekt-orientering, er faren stor for at det blir en begrepskløft mellom det gamle og det nye. Det gamle må derfor gis et objektorientert ytre, hvis man da ikke skal modellere alt helt på nytt.

Modeller og distribusjon

I tillegg til den usikkerhetsdimensjonen som ble presentert ovenfor kommer distribusjonsinstansen, dvs hvorfor og hvordan det ferdige systemet skal bli i det distribusjonemsssige henseendet. Som nevnt på side 15 kan man skille mellom distribusjon generelt, dvs hvilke hensyn utvikling av et distribuert system må ta — altså «Distribuerthet» i Figur 1.7 på side 15 — og den konkrete distribusjon i ett system, dvs det vi vil lage. Generelle hensyn kan være:

- Samtidighet
- Transaksjonsproblematikk
- Ikke-determinisme (Spesielt interessant i test-sammenheng.)
- Sikkerhet
- Replikering
- Feiltransparens
- Lokaliseringstransparens

De fleste av disse hører imidlertid hjemme i implemenasjonsfasen i et utviklingsprosjekt. En del kan også i praksis ivaretas av mellomvare, f.eks objektmeglere som følger CORBA standarden. (Se Kapittel 4.) Som systemutviklere vil vi i så stor grad som mulig konsentrere oss om selve applikasjonslogikken og objektmodellen og dermed skjermes fra de trivialiteter som ikke har med selve applikasjonen å gjøre. En mellomvare for distribuerte objekter vil også gi en viss transparens i flere dimensjoner (se Kapittel 3), slik at man slipper å ta hensyn til det som skjer på lavere abstraksjonsnivå. I prinsippet skulle vi for eksempel kunne ignorere *hvor* objekter befinner seg i verden, og heller tenke på *hvordan de samarbeider*. Det er akkurat dette som er målet med å bruke objektmeglere.

Hvis det nå er likegyldig hvor objektene befinner seg — kunne de ikke da like gjerne eksistert på samme sted alle sammen, f.eks på én stormaskin? Er det noen forskjell i tankegang når man lager objektmodeller for distribuerte systemer kontra ikke-distribuerte systemer?

Som vi skal se senere er det liten dekning i vanlige systemutviklingsmetoder for *hvorfor* man distribuerer et system.

OO-metoder for distribusjon

I det følgende skal jeg ta for meg DISGIS (DIStributed Geographical Information Systems) General Method, som er basert på OOram. Deretter vil jeg se på noen strategier for gjenbruk som forsåvidt ikke tilhører noen spesiell metode; klassbibilioteker, rammeverk og komponenter. Selv om OOram ikke er noen «stor» metoderetning i betydningen antall brukere eller prosjekter som har brukt den — i motsetning til andre mer «kurante» metoder f.eks basert på UML (Unified Modeling Language) — er den interessant fordi den kobler utviklingsmetode med RM-ODP. Som nevnt i Forordet (under «Bakgrunn for oppgaven» på side i) har utforsking av en slik kobling vært noe av utgangspunktet for hele oppgaven.

OOram

OOram — Object Oriented Rôle Analysis and Modeling — er utviklet av personer ved SINTEF, TASKON A/S og Institutt for Informatikk ved Universitetet i Oslo. Metoden støttes av CASE-verktøyet OORAM PROFESSIONAL [OOram 96a] som er utviket av TASKON.

Hovedidé

Hovedideen med OOram er modellering av roller. Det sentrale er således rollemodellen, som modellerer samspillet mellom objektene gjennom de roller de spiller. Samspillet gir en synergieffekt som overgår enkeltobjektenes evner hver for seg. Hvert objekt kan spille én eller flere roller i ulike sammenhenger. Dette ligner på rollemodellering med NIAM [Ressem 95], bortsett fra at NIAM modellerer strukturen mellom begreper mens OOram altså legger vekt på samspillet. OOram benytter i tillegg en splitt-og-hersk teknikk (Separation of concern) for å dele opp problemområdet. Dette deles derfor opp i passelige «Area of Concern» (AoC), som kan sammenlignes med Ivar Jacobsons «use-cases» [Jacobson et al 92]. Disse delproblemområdene analyseres og modelleres hver for seg med de forskjellige teknikker og notasjoner OOram tilbyr, og syntetiseres så sammen til en komplett objektrollemodell. Denne kan så danne utgangspunkt for implementasjon. OOram støttes av en grafisk notasjon såvel som et formelt språk: *OOram Language*.

Tre ståsteder

OOram ser på modellen med tre forskjellige ståsteder eller «utkikks-punkter»:[1]

- *Fra omgivelsene*, hvor observatøren altså befinner seg i omgivelsene rundt systemet. Herfra kan han eller hun observere og modellere interaksjonen mellom omgivelsesrollene og systemet;
- *Ytre plassering.* Her er observervatøren plassert blant objektene, og betrakter meldingsflyt mellom dem;
- *Indre plassering.* Observatøren sitter inne i en rolle og kan betrakte rolleimplementasjonen direkte.

Merk at det siste ståstedet er det som gjerne er utgangspunkt i vanlig objektorientert programmering i f.eks SIMULA; man beskriver objektene ved deres klasser innenfra.

Ti perspektiver

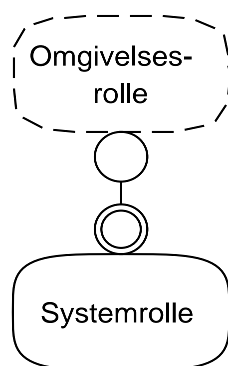
OOram bruker ti perspektiver («views») for å beskrive problemområdet ut fra forskjellige hensyn. De ti perspektivene er (etter [Reenskaug et al 96]):

1. **Area of Concern perspektiv:** en tekslig beskrivelse av problemområdet oppdelt i passelige AoC;
2. **Stimulus-respons perspektiv:** Beskriver hvordan roller utenfor systemet påvirker interne roller. Eksterne roller sender meldinger til roller i systemet som systemet reagerer på;
3. **Rolleliste:** tekstlig liste over alle roller med forklaringer og attributter;
4. **Semantisk perspektiv:** viser statisk forhold mellom roller. Kan sammenlignes med Entity-Relationship-modellering (ER) (se f.eks [Elmasri & Navathe 94], [Skagestein 91]);
5. **Samarbeidsperspektiv:** viser hvordan roller samarbeider ved hjelp av meldingsutveksling;
6. **Grensesnittperspektiv:** definerer alle meldinger som kan sendes lovlig mellom objektene;
7. **Scenarioperspektiv:** viser lovlige interaksjonsmønstre mellom ulike roller, dvs meldingsprotokoller mellom ulike roller;
8. **Prosess-perspektiv:** viser hvordan data flyter mellom rollene;
9. **Tilstandsdiagram:** hver rolle kan ha hvert sitt tilstandsdiagram; hvilke signaler roller kan motta i hvilke tilstander og hvilke som sendes ut;
10. **Metodeperspektiv:** beskriver prosedyrene (metodene) som utføres når roller mottar forskjellige meldinger.

[1] Egentlig bruker OOram «view» der jeg bruker «perspektiv» og «perspektive» der jeg bruker «ståsted». Dette kan virke forvirrende, men jeg synes det gir en bedre forståelse med denne oversettelsen. OOram er ikke langt unna: «We may observe the system (…) from different observation points (…) called perspectives» [Reenskaug et al 96] s. 60. [Ressem 95] bruker også samme oversettelse som meg.

De ti perspektivene ovenfor for kan hver behandles i ett eller flere av ståstedene. Imidlertid er ikke alle kombinasjoner like meningsfulle, men jeg skal ikke gå nærmere inn på denne matrisen her. Se heller [Reenskaug et al 96]:61.

De perspektiver man først kommer i kontakt med i OOram modellering er Area of Concern-, samarbeid-, grensesnitt- og scenarioperspektivene. Jeg skal bare kort gjøre rede for disse her. En komplett behandling av alle perspektivene og bruken av dem finnes i [ibid]. Det første perspektivet er en ren tekstlig beskrivelse i et naturlig språk av det aktuelle delproblemområdet (AoC). De tre andre er støttet med grafisk notasjon, og også med OORAM PROFESSIONAL 4.0. Dette verktøyet slår forøvrig sammen noen av perspektivene i brukergrensesnittet, slik at det fremstår som tre redigeringsverktøy.



Figur 5.1: To roller med meldingsvei

Samarbeidsperspektivet. I samarbeidperspektivet modelleres rollene og meldingsveiene mellom dem. Det skilles mellom systemroller og omgivelsesroller. Omgivelsesrollene stimulerer systemet, og kan være f.eks brukere eller andre systemer. Dette tilsvarer «Actor» i Jacobsons notasjon i forbindelse med «use-case» modellering [Jacobson et al 92]. Figur 5.1 viser en systemrolle og en omgivelsesrolle med en meldingsvei definert mellom de to.[1] Hver ende av en meldingsvei representerer en *port*. Porter opererer således i par, og hvert portpar symboliseres med sirkler på hver deltakende rolle forbundet med en linje. Kardinalitetene på portene, dvs hvor mange mottakere en port vet om, representeres med konsentriske eller enkle sirkler. Dette betyr henholdsvis mange og én. Ingen sirkel i det hele tatt betyr at det ikke er noe grensesnitt den veien, dvs ut av rollen langs denne linjen. Kardinalitetsnotasjonen i OOram synes jeg personlig ikke er like intuitivt som f.eks den som brukes i ER-dialekten som presenteres i [Skagestein 91], hvor en kråkefot tydelig indikerer at det er flere i den enden kråkefoten indikerer. Den kan mer sammenlignes med NIAM-notasjon (også presentert i ibid.) der en entydighetsskranke i en binærsetning kan sammenlignes med «kråkeføtter». Nå skal det sies at disse teknikkene ikke er direkte sammenlignbare i *samarbeidsperspektivetet.* I OOrams semantiske perspektiv blir semantikken i kardinalitetene mer sammenlignbar med de andre notasjonene. Det semantiske perspektivet er imidlertid ikke så mye i bruk, for tilstrekkelig informasjonen finnes som regel i samarbeidsmodellen [Reenskaug et al 96].

interface 'Omgivelsesrolle<Systemrolle'
message synch 'Hallo'
explanation "Melding som sier Hallo"

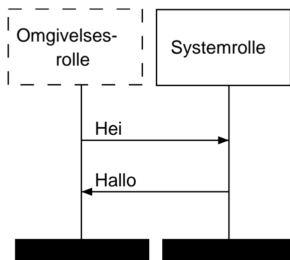
interface 'Systemrolle>Omgivelsesrolle'
message synch 'Hei'
explanation "Sier Hei"

Figur 5.2: Tekstlig beskrivelse av grensesnitt for Figur 5.1

Hver port støtter ett eller flere grensesnitt. Hvert grensesnitt kan sende et sett meldinger *ut* gjennom porten det tilhører. I eksempelet i Figur 5.1 kan flere objekter som spiller omgivelsesrollen således sende meldinger til én systemrolle. Denne kan igjen sende meldinger til flere omgivelsesroller.

[1] I [Aagedal et al 97] er superellipsene fra OOram [Reenskaug et al 96] erstattet av rektangler.

Grensesnittperspektiv. Dette perspektivet gir en tekstlig og/eller grafisk beskrivelse av hvilke grensesnitt som støttes. Figur 5.2 på forrige side viser en tekslig beskrivelse av grensesnittene som støttes i portene fra Figur 5.1. Notasjonen følger OOrams formelle syntaks for tekslig beskrivelse. Det er også mulig å beskrive grensensnittene grafisk, i samme diagram som samarbeidsperspektivet. Dette gjøres ved at hver port får tilknyttet en forklaringsboks med grensesnittene listet opp i. Imidlertid blir det raskt for komplisert og det blir lett for trangt på en dataskjerm eller papirkopi.



Figur 5.3: Scenario med to roller

For hver melding i grensesnitt-perspektivet kan man lage en kommentar som på en lettfattelig måte som indikerer *hensikten* med meldingen. I Figur 5.2 angir linjene med **explanation** dette.

Scenarioperspektiv. I scenarioperspektivet modelleres eksempler på samhandling mellom ulike roller i systemet og i omgivelsene. Notasjonen ligner noe på Jacobsons «use-cases» [Jacobson et al 92] med tilhørende scenarionotasjon. Figur 5.3 viser et scenario med eksempelrollene fra Figur 5.1. Verdt å merke seg er at den første interaksjonen alltid kommer fra en omgivelsesrolle (stimuli).

Eksempel: PBE

Som eksempel på modellering i OOram, har jeg valgt eksempelet fra case i denne oppgaven; Plan- og bygningsetaten i Oslo kommune (se «Case» på side 17).

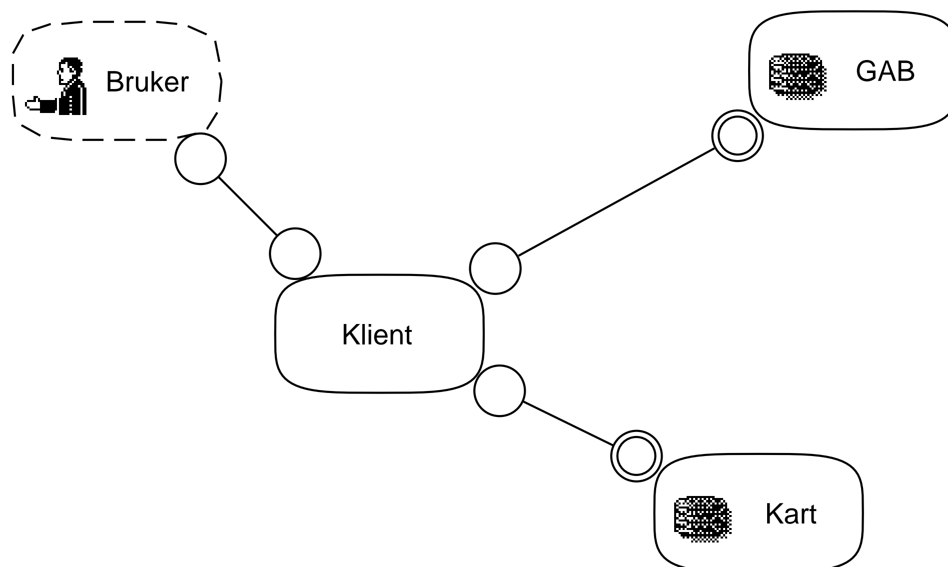
AoC-perspektiv. Area of Concern-view for en del av naboproblemet kan formuleres som: *«Denne modellen beskriver hvordan brukere får hentet kartbiter, adresser og informasjon om eiere av bygninger fra PBE's databaser.»*

Samarbeidsperspektiv. Samarbeidsmodellen for det AoC ovenfor har fire roller: Bruker, klient, kartdatabase og GAB-register. Figur 5.4 viser dette. Modellen hevder altså at hver bruker har én klient. Klienten henter informasjon på vegne av brukeren fra Kartdatabasen og GAB-registeret. Verdt å merke seg er at vi allerede her har gjort visse antakelser om hvordan det skal designes. Vi antar at det skal finnes et klientprogram, og at det finnes to databaser: kart- og GAB. De to siste indikerer at vi har tenkt å beholde endel av datastrukturen som etaten vedlikeholder i stor grad som den er.

Grensesnitt perspektiv. I grensesnittperspektivet for dette AoC kan rollenes grenesnittene gis i en tekslig fremstilling:[1]

---

[1] Merk at formatet for grensesnittenenavnene ved første øyekast ser litt bakvendte ut. I f.eks **interface** 'Klient<Bruker' er det «Klienten» som er mottaker og angitt først. '<'-tegnet angir retningen av meldinger. Likevel *tilhører* grensesnittet avsenderen, slik OOram vil ha det. Man står imidlertid fritt til å definere navn på grensesnittene etter eget forgodtbefinnende, men OOram verktøyet som jeg har brukt lager dem nå engang slik.



Figur 5.4: Samarbeidsperspektiv for kartoppslag

interface ‘Klient<Bruker’
    message synch ‘Vis eiendom’
        explanation “Viser den eiendommen brukeren klikker på”
        param ‘Musklikk’ **type** ‘skjermkoordinater’

interface ‘Bruker<Klient’
    message synch ‘VisEier’
        explanation “Skriver ut på skjerm eier ev en eiendom”
        param ‘Eier’ **type** ‘tekst’

interface ‘Klient<Kart’
    message synch ‘Send Kartbit’
        explanation “Sender et utsnitt fra kartet”
        param ‘Kartbit’ **type** ‘Kartobjekter’

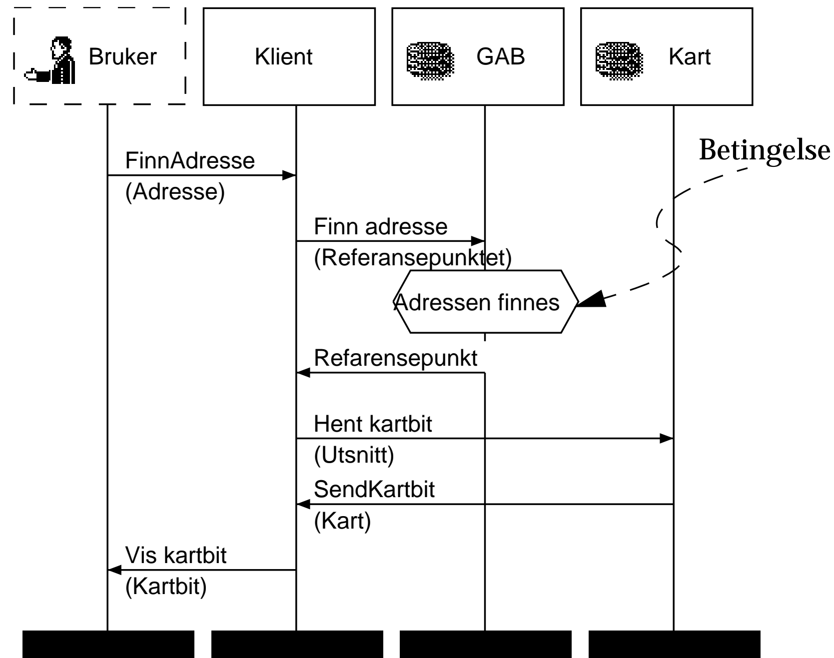
interface ‘Kart<Klient’
    message synch ‘Hent kartbit’
        explanation “Hent kartutsnitt avgrenset av et gitt rektangel”
        param ‘Utsnitt’ **type** ‘rektangel’
    message synch ‘Finn naboer’
        explanation “Finn naboene rundt et referansepunkt”
        param ‘Bygning’ **type** ‘referansepunkt’

interface ‘GAB<Klient’
    message synch ‘Finn adresse’
        explanation “Finner adressen fra et referansepunkt”
        param ‘Referansepunktet’ **type** ‘referansepunkt’

```
interface 'Klient<GAB'
    message synch 'Eier'
        explanation "Sender data om en eier"
        param 'Eier' type 'tekst'
    message synch 'Referansepunkt'
        explanation "Sender et referansepunkt"
        param 'Referansepunktet' type 'referansepunkt'
```

Scenario-perspektiv. Figur 5.5 viser scenarioperspektiv for systemet — et «use-case» hvor en bruker taster inn en adresse. Systemet

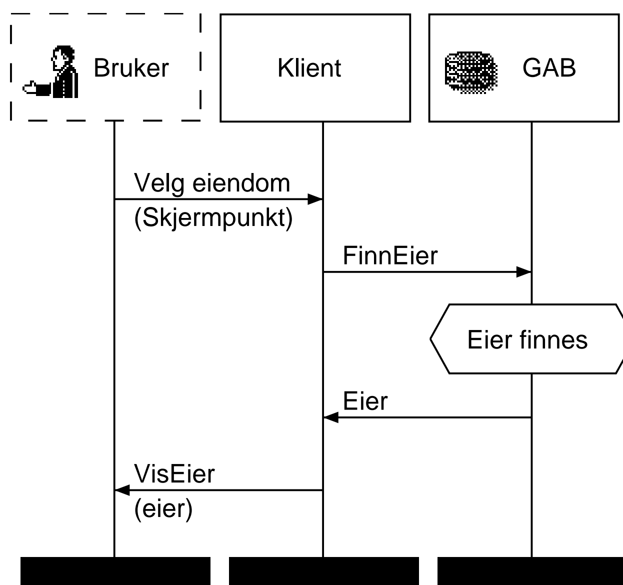
Figur 5.5: Scenario for å vise kart rundt en adresse



finner så adressens referansepunkt, hvis adressen finnes i GAB-registeret. Systemet viser så et passelig kartutsnitt hvor bygningen brukeren ville se på er i midten.

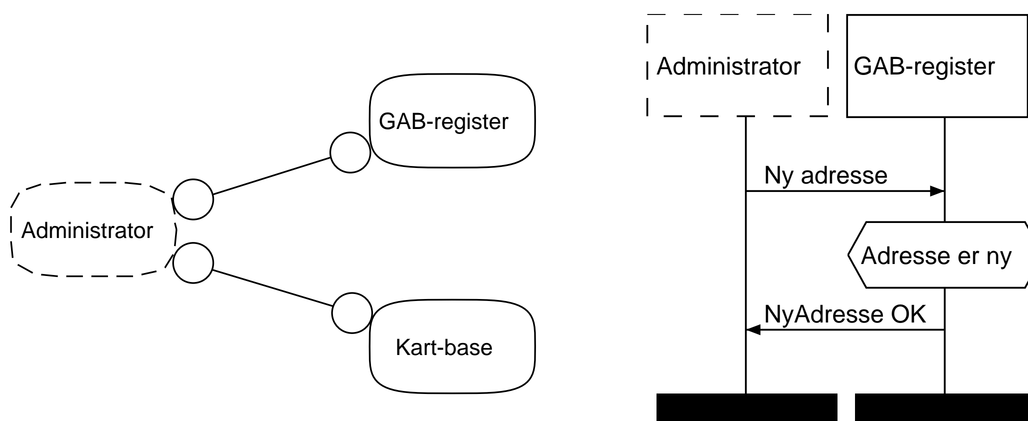
Figur 5.6 viser et scenario der brukeren klikker på en eiendom av interesse. Systemet finner så ut hvem som er eier av bygningen eller bygningsdelen brukeren klikket på.

Figur 5.6: Scenario for å finne eier av en eiendom



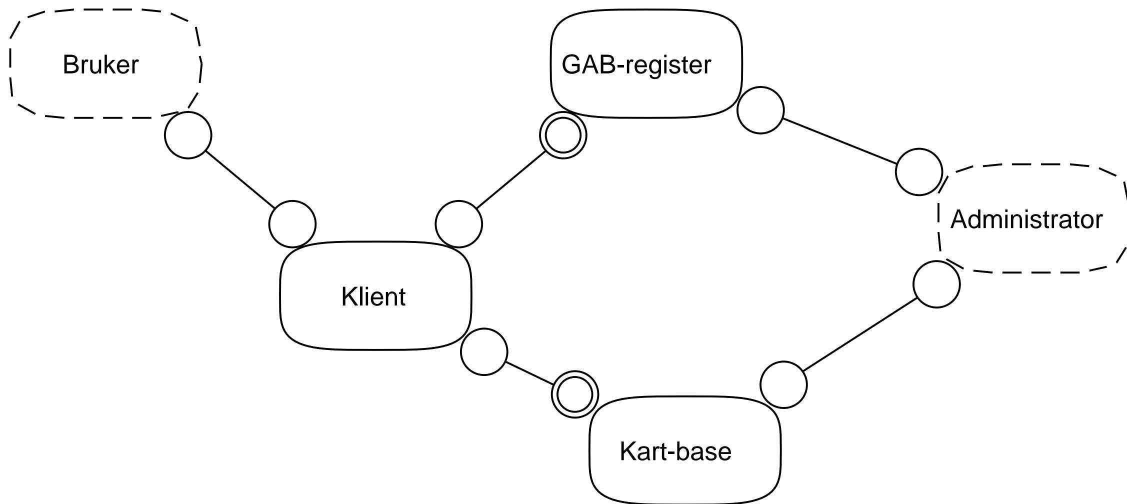
Analyse og syntese

Svært sentralt i OOram er samspillet mellom *analyse* og *syntese*. Ved å dele opp problemområdet i flere subområder — Area of Concerns (se ovenfor) — kan hvert problem med sine scenarier analyseres hver for seg. Disse forskjellige AoC er ikke nødvendigvis disjunkte siden de gjerne deler felles objekter. Et objekt kan spille flere roller. Som i modellene i Figur 5.4–5.6 viser, er det fire roller i systemet så langt. Vi skal imidlertid innføre et AoC til for anledningen: *Administrasjon av databasene*. Figur 5.7 viser rollemodell og ett scenario for dette AoC.



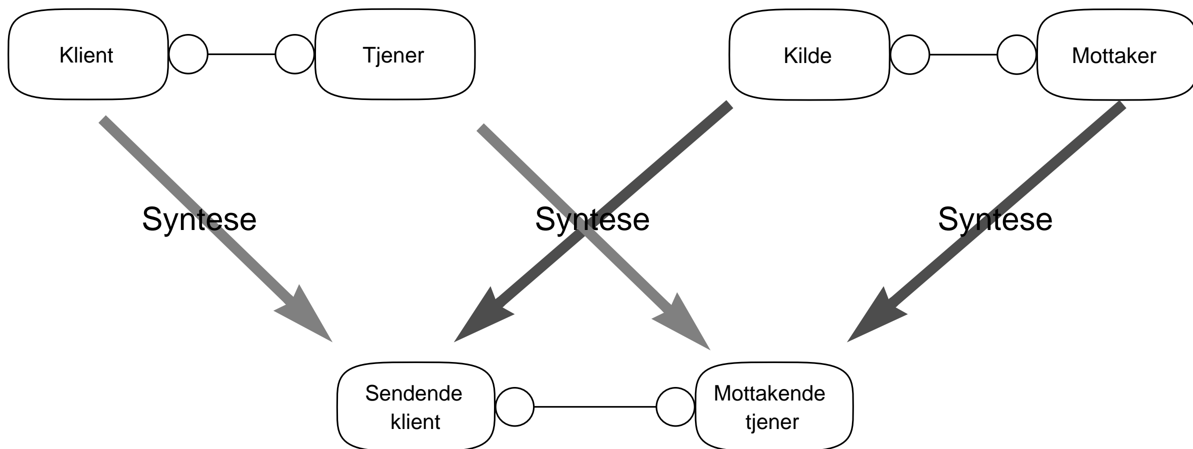
Figur 5.7: Rollemodell og scenario for databaseadministrasjon

OOrams kongstanke ligger nå i syntesen mellom disse to modellene. GAB-registeret og kartdatabasen «medvirker» altså i to rollemodeller som syntetiseres sammen til en. Denne syntesen er vist i Figur 5.8. Syntesen er OOrams alternativ til multippel arv. Mekanismen er mer



Figur 5.8: Syntese av rollemodellene i for kartoppslag og databaseadministrasjon (Figur 5.4 og 5.7)

slående i andre eksempler, f.eks ved syntese mellom generiske og problemspesifikke rollemodeller. Et av eksemplene i [Reenskaug et al 96] demonstrerer syntese mellom generiske modeller for henholdsvis klient/tjener- og kilde/mottaker-forhold. Figur 5.9 viser en adapsjon



Figur 5.9: Syntese av klient/tjener rollemodell med kilde/mottaker modell (etter eksempel i [Reenskaug et al 96]:81–82).

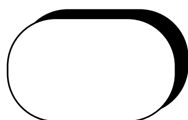
av dette eksempelet. Ved å kombinere forskjellige syntesekombinasjoner (ikke vist her) kan man oppnå forskjellig funksjonalitet. Mulighetene er:

- Klienten kan sende data til tjeneren
- Klienten kan motta data fra tjeneren

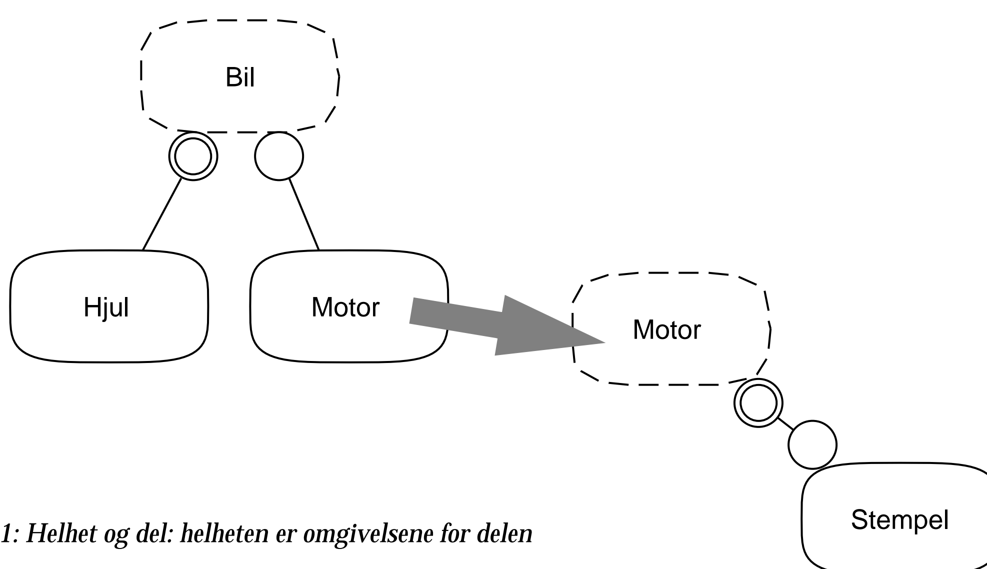
- Klienten kan sende og motta data fra tjeneren

Aggregering og virtuelle roller

OOram og OOram Professional støtter ikke aggregering over flere abstraksjonsiváer direkte. Likevel kan man innkapsle delmodeller i virtuelle roller. Et delsystem kan således kollapses til én virtuell rolle som ivaretar interessene utad for alle rollene den omfatter. Figur 5.10 viser notasjon for dette. Selv om systemer og deler av systemer kan betraktes som et hierarkisk system er dette bare noe som oppstår i analytikerens hjerne, og derfor ikke en del av virkeligheten [Reenskaug et al 96]. Da foreslås det heller å modellere delsystemer som egne AoC, hvor kontaktpunktene med helheten modelleres slik at helheten opptrer som omgivelser for delen. Eksempelet i Figur 5.11



Figur 5.10: OOram virtuell rolle



Figur 5.11: Helhet og del: helheten er omgivelsene for delen

viser hvordan dette kan gjøres med en modell av en bil modellert som to AoC. Ved syntese vil så disse kombineres til én rollemodell.

Implementasjon

I utgangspunktet er OOram rollemodeller uavhengig av det programmeringsspråk systemet skal implementeres i. For øyeblikket finnes verktøy (OORAM PROFESSIONAL) for å generere Smalltalk og C++ kode fra analysemodeller, men man kan tenke seg andre målspråk — også ikke-objektorientert. Figur 5.12 (etter [Reenskaug et al 96] s. 117)

OOram	Smalltalk	C++	Java
Rollemodell	—	—	—
Rolle	Objekt	Objekt	Objekt
Objektspesifikasjon/type	class	class	class
Port	Variabel	Peker attributt	Referanse

Figur 5.12: Mapping mellom OOram og programmeringsspråk

OOram	Smalltalk	C++	Java
Grensesnitt	Protocol	Abstrakt klasse eller protokollklasse	interface
Melding	Message	Funksjonskall	Metodekall
Metode/handling	Method	Funksjon	Metode
Avledet modell	Subklasse	Subklasse	Subklasse
Basismodell	Super-klasse	Basisklasse	Superklasse

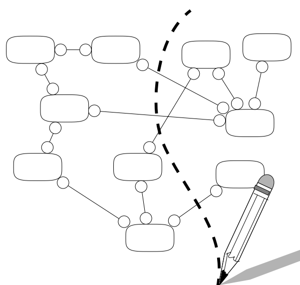
Figur 5.12: Mapping mellom OOram og programmeringsspråk

viser denne sammenhengen.[1] Generelt blir *porter* til variabler som peker til andre objekter. Kardinaliteten til en port avgjør om variabelen er vanlig, ved kardinalitet lik én, eller en mengde — for eksempel en liste — ved mange-kardinalitet.

Et sentralt punkt i koblingen mellom en OOram-modell og implementasjonen av den er hvordan grensesnitt implementeres. Siden grensesnitt i OOram modelleres som meldings*kilder*, dvs «hull» som meldinger strømmer ut av, kan man på et vis si at dette viser hva objektene *ønsker*. Imidlertid er utgangspunktet i de fleste objektorienterte språk å spesifisere hvilke tjenester et objekttype eller en klasse objekter kan *tilby*. Dette er altså et spørsmål om tilbud og etterspørsel. Denne tankegangen er et resultat av at en OOram-modell modellerer samarbeidet mellom objekter. Når en rolle omskapes til en klasse i et objektorientert språk senere må man derfor sørge for at klassen kan motta og forstå alle de meldinger de nære omgivelsene kan finne på å belemre den med. Denne tankegangen vil kunne ha den heldige effekt at man oppdager nye samarbeidsformer — og dermed tilbud som klasser burde implementere hvis den skal delta tilstrekkelig i samarbeidet. Dette aspektek ville man kanskje ikke oppdaget like lett med et mer vanlig klassesentrert perspektiv.

OOram og distribusjon

I utgangspunktet har ikke OOram noen spesielle mekanismer for å modellere distribusjon. Imidlertid ligger det nokså nært i dagen at veien fra en rollemodell — som fokuserer på interaksjon mellom objekter modellert ved hjelp av meldinger — til et system av distribuerte objekter, burde forholdsvis kort. Man kan tenke seg at man kunne trekke distribusjonsgrensene etter eget forgodtbefinnende mellom objektene med en blyant (Figur 5.13). De metoder i klassebeskrivelsene som følger av modellen kunne så utstyres med en passelig kommunikasjonsmekanisme og sende de meldinger som er definert i de respektive grensesnitt.



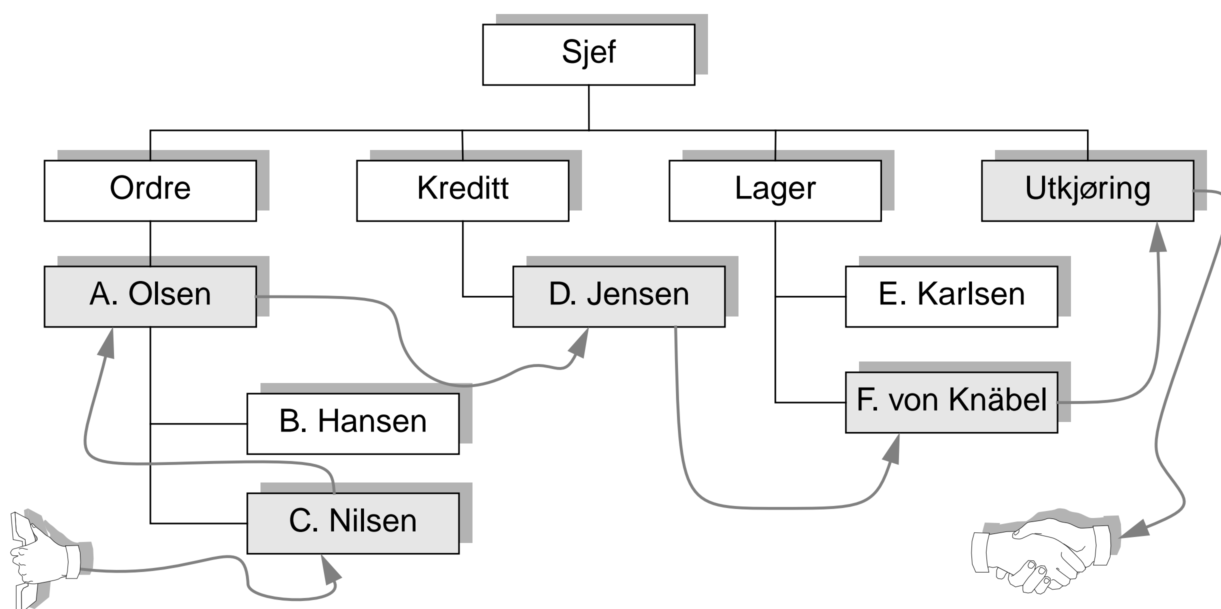
Figur 5.13: OO-Nirvana: vi trekker opp distribusjonsgrensen med egnet verktøy: blyant

Imidlertid gis det liten støtte for å bestemme *hvordan* distribusjonen skal være — dvs distribusjonsinstansen eller topografien i systemet.

[1] Jeg har lagt til Java mapping for egen regning.

OOram viser forsåvidt hvordan det er mulig å distribuere systemet, men altså ikke *hvorfor*. Her må man bruke andre metoder for å finne frem til den distribusjonen som er riktig for organisasjonen, f.eks Business Process Rengineering (BPR) eller andre organisasjonsutviklingsprosesser. En slik prosess kan avstedkomme en ny geografisk distribusjon av organisasjonen; fra sentralisering til desentralisering, vica versa, eller ingen forandring i det hele tatt.

Ved en BPR prosess vil man typisk forandre måten en organisasjon arbeider på. Som eksempel vil jeg bruke en organisasjon som er hierarkisk oppbygget. Dette er ikke uvanlig. Det første man ser når man åpner årsmeldinger og presentasjoner av større organisasjoner er et organisasjonskart, som viser hvem som er mest sjef, nest mest sjef og



Figur 5.14: Organisasjonskart med saksgang

nedover gjennom divisjoner og avdelinger. I en BPR-prosess vil man kanskje se på hvordan en kundeordre behandles, og hvem som virkelig har ansvaret for den. Ikke sjelden viser det seg at det er mange personer involvert; salg, ordremottak, kredittvudering, fakturering, lager (hvis vi f.eks snakker om en bedrift som selger ting) levering etc. Som illustrert i Figur 5.14, er det mange som er involvert i dette «use-case». Én mulig geografisk distribusjon av denne oganisasjonen kunne være at hver avdeling holder til i hver sin bygning eller etasje. De ansatte sender således papirer seg imellom helt til oppgaven er gjennomført. Kunder føler ofte at de ikke vet hvem som er ansvarlig for handelen, og når de spør får de ikke alltid noe godt svar.[1] Med OOram straight up ville man modellert dette som et antall roller;

[1] Dette er kanskje et enda større problem for «kunder» av offentlig forvaltning.

ordremottak, kredittvurdering etc., og så modellert samhandlingen mellom dem. Dette løser imidlertid ikke problemet med hvem som er ansvarlig for et helt «use-case», men med rollemodellene fra studier av arbeidet i organisasjonen kan man så *syntetisere* en passelig modell hvor det legges til rette for at ansvaret mer konsist kan plasseres hos én person der det for kunden virker naturlig at det er slik. En slik modell vil gi litt andre delingsmuligheter enn en modell der disse hensyn ikke er ivaretatt.

Hvis gevinsten av BPR, OOram og medhørende distribusjonsbetraktninger ikke fører til annet enn at kunder og brukere nå vet hvem som er ansvarlig for at alt går tregt, er kanskje lite vunnet. Imidlertid kan prosesser som BPR føre til at organisasjonen arbeider mer effektivt, og informasjonsteknologi kan være en nøkkel til suksess på dette området. Pussig nok mener et stort antall konsulenter i Norge at dette ikke er tilfelle; at IT heller spiller en mindre rolle eller er til direkte hinder for en BPR-prosess [Iden 95]. Dette står i sterk kontrast til amerikanske BPR-prosjekter og -litteratur som forfekter at informasjonsteknologi er en klar suksessfaktor [Hammer & Champy 93], [Davenport 93], [Harmon & Morrisey 96], [Illiaifar et al 95].

OOram og eksisterende systemer

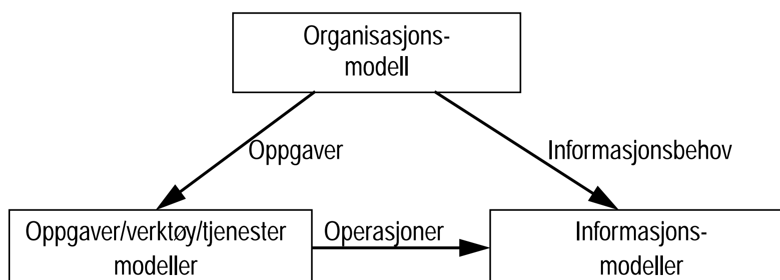
Hvordan modelleres så eksisterende systemkomponenter med OOram? OOram gir i seg selv ingen oppskrift for hvordan man modellerer eksisterende systemer sammen med nye. Som nevnt tidligere kan man bestemme seg for et passelig granularitetsnivå på de forskjellige delmodeller; om det eksisterende skal splittes opp i sine enkelte bestanddeler på kodenivå, eller om man skal modellere det som selvstendige svarte bokser som tilbyr tjenester til nye og gamle komponenter. Noe av nøkkelen til dette spørmålet kan liggge i organisasjonsmodelleringen. Én vei å velge i et BPR-prosjekt er å bruke en objektorientert modelleringsmetode, som f.eks. den som foreslås i [Jacobson et al 94]. En objektorientert prosessinnovasjon vil kunne gi følgende fordeler:

- OO-modellene av de nye driftsprosessene vil hjelpe til med å identifisere granularitetsnivået i de enkelte delkomponenter av det bestående og det nye.
- Man kan få en sømløs overgang til andre faser i BPR-prosjektet for eksempel å lage nye informasjonssystem eventuelt ved å tilpasse de gamle.

Med andre ord vil man få mer klarhet i hvilke komponenter idet gamle systemet som kan bestå som de er (eventuelt i en innpakket variant), og hvilke som må i støpeskjeen og gjenoppstå som nye eller deler av nye systemkomponenter.

Utgangspunktet for prosessmodellering i OOram er å lage seg tre modellgrupper:

Figur 5.15: Forhold mellom de tre modellene (etter [Reenskaug et al 96])



- Organisasjonsmodeller (enterprise model)
- Oppgave/verktøy/tjeneste modeller (task/tool/services model)
- Informasjonsmodeller (information model)

Organisasjonsmodellen modellerer organisasjonens aktører overfor edb-systemet og den prosess som skal støttes. Aktørene kan passelig modelleres som omgivelsesroller i forhold til systemet. Organisasjonens samling av aktuelle verdikjeder (som kan være et resultat av en BPR-prosess) modelleres som arbeidsprosesser i OOrams prosessperspektiv (se side 76). Under eller før denne aktiviteten kan man vurdere hvordan delaktivitetene i prosessene passer inn i de omstrukturerte arbeidsprosesser. Eventuelt kan de forandres hvis det er grunn til det. Dette vil si at denne modelleringen kan være et ledd i en BPR-prosess, hvis en slik er etablert for organisasjonen for det aktuelle AoC.

I og med at prosessmodellen er basert på hva de ulike *roller* foretar seg som ledd i en verdikjede, gir dette den åpenbare fordel at man senere kan omdisponere rollelisten etter skiftende behov, selv om modellen forholder seg konsistent, trass nye konstellasjoner og ansvarsfordeling som oppstår i systemet og dets omgivelser. Dette gir en fleksibilitet som er helt nødvendig for å kunne holde organisasjonens arbeidsprosesser oppdatert og i takt med varierende eksterne krav. Organisasjoner i konkurranseutsatte bransjer må belage seg på å måtte fornye sine driftsprosesser med bare års mellomrom [Hammer & Champy 93].

Organisasjonsmodellen som utvikles med eller uten BPR vil etterhvert sette krav til de to andre modellene i Figur 5.15. Den dikterer hvilke oppgaver som skal støttes, identifisert ved gjennomgang av arbeidsprosessene og rollefordelingen. Modellen identifiserer dermed hvilke verktøy og tjenester som er ønskelige. Samtidig identifiserer organisasjonsmodellen hvilke informasjonsbehov organisasjonen og de nye driftsprosessene har. Informasjonsmodellen kan så tilpasses disse behovene, og verktøymodellen kan lages som bindeledd mellom organisasjonsmodellen og informasjonsmodellen. Faktisk så er det her OOram har noe av sin kreative styrke. Ved å modellere hvilke behov arbeidsprosessene har, vil man gjennom disse behovene

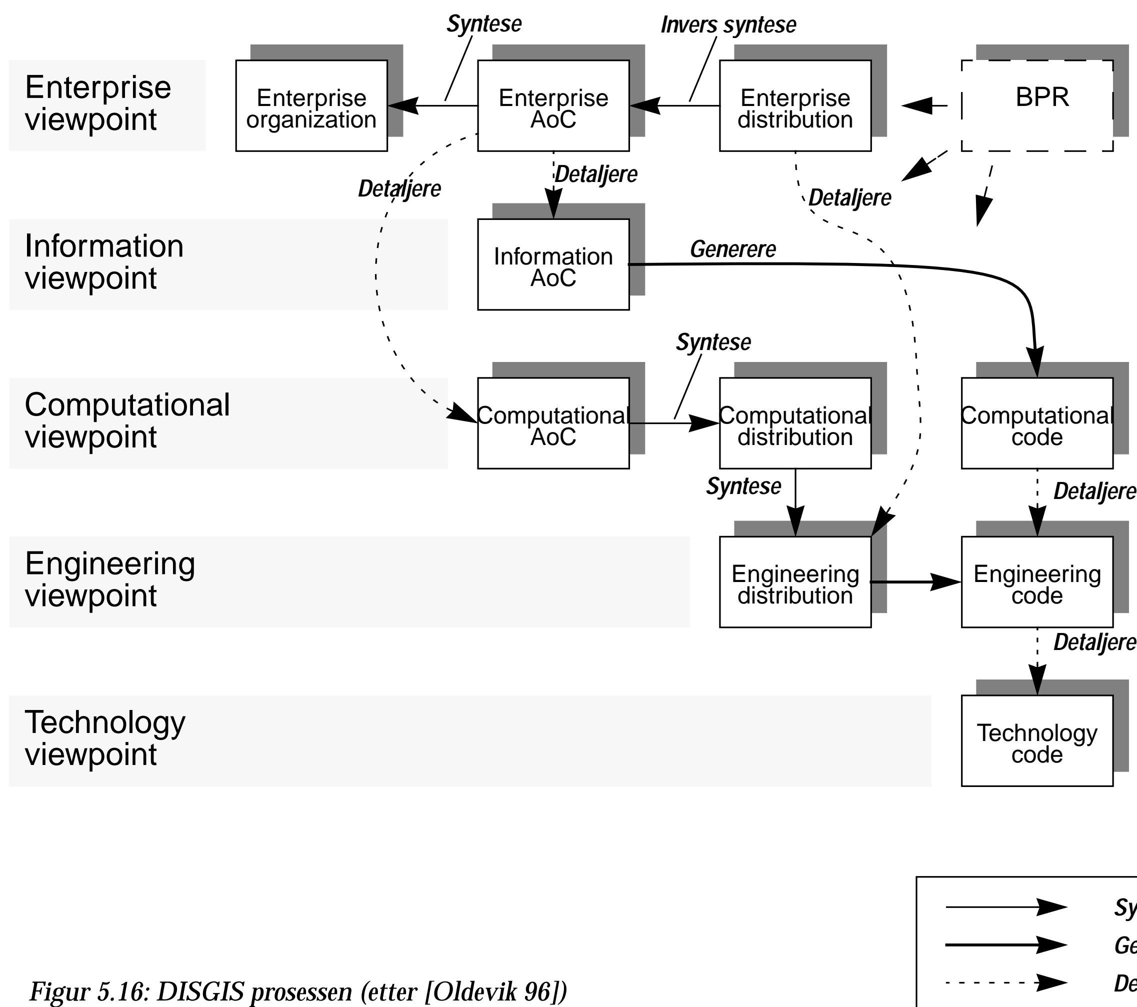
kunne identifisere kravene til både verktøy og informasjonstjenester. På den annen side er det nok ikke like lett å se i hvilken grad de eksisterende systemkomponenter passer inni disse kravene, og eventuelt hvor mye som må spesiallages for at ting skal passe sammen. Man kan tenke seg at mindre endringer i kravene fra arbeidsprosessene vil kunne spare mye arbeid og tilpasning av eksisterende komponenter. For å tilfredstille denne motsetningen, bør man bruke en iterativ utvikling også her, og se på samspillet mellom det nye og det eksisterende på en heuristisk måte.

Metoden beskrevet hittil tar i utgangspunktet lite hensyn til hva som eksisterer og hva som blir nytt. Det er heller ikke vesentlig i hvilken grad systemet er distribuert eller ikke. Imidlertid vil de organisatoriske aspekter ved den nye driftsprosessmodellen sammen med krav og ønsker vedrørerende informasjonsmodellen gi en distribusjonsinstans som er definert av brukernes- og de eksisterende systemkomponentenes geografiske plassering. I dette regnes også administrasjon og drift av systemkomponentene som en del av arbeidsprosessene. Dette kan f.eks tilsi at sentrale systemtjenester, så som databaser, plasseres i nærheten av driftspersonellet av praktiske årsaker.

DISGIS General Method

DISGIS — DIStributed Geographical Information Systems — er et ESPRIT-prosjekt med samarbeidspartnere over hele verden. Blant disse er SINTEF og TASKON. I [Oldevik 96] presenteres DISGIS General Method. Metoden kan betraktes som en syntese av OOram og RM-ODP. OOram selv består av ni «views» mens RM-ODP bidrar med fem «viewpoints». Det kartesiske produktet av disse (med 45 «views») er ikke hensiksmessig, men metoden identifiserer 11 aktiviteter og forholdene mellom dem. Resultatet av hver aktivitet er utgangspunktet for en eller flere andre.

Figur 5.16 (etter [Oldevik 96]) viser aktivitetene i DISGIS modelleringsprosessen. Utgangspunktet for en slik prossess i forslaget er organisasjonen og dens distribusjon slik den foreligger, men man kan også tenke seg at en ny organisasjonstruktur kan utvikles parallelt med systemet, for eksempel i forbindelse med et BPR-prosjekt. Jeg har derfor selv tilføyet BPR-aktviteten øverst i høyre hjørne på figuren, slik at den påvirker både organisasjonens distribusjon og driftsprosesser. Avhengig av om BPR-initiativet er en del av utviklingen av det distribuerte system eller omvendt, vil BPR-aktiviteten i større eller mindre grad gjennomsyre flere av de aktivitetene som opprinnelig er presentert i metoden.



Figur 5.16: DISGIS prosessen (etter [Oldevik 96])

Organisasjonens forretnings-prosesser

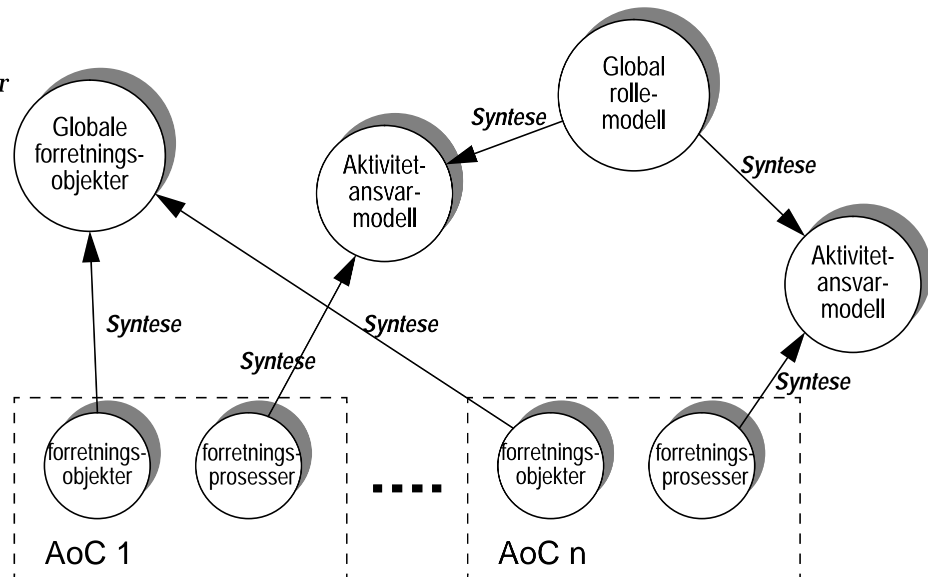
Det første skritt i prosessen består av å lage forretningsmodeller av organisasjonens forretningområde. Hver prosess kan sees på som en verdikjede, og et eget AoC for organisasjonen. Hvert AoC består av et antall deltakende forretningsobjekter (Business objects) og forretningsprosesser som disse objektene inngår i. Forretningsobjekter er typisk objekter som finnes i problemområdet, som for eksempel fakturaer, kontrakter, konti etc. For Plan- og bygningsetaten kan det i tillegg dreie seg om søknader, godkjennelser, høringsuttalelser, byggetegninger osv.

Allerede på dette stadiet er det sannsynlig at et eventuelt BPR-initiativ vil påvirke hvordan disse prosessene utformes, slik at de blir lønnsomme eller i alle fall optimalt formålstjenelige verdikjeder for organisasjonen. Dette vil igjen påvirke hvilke forretningsobjekter som identifiseres og hvordan de arter seg. Hvert AoC som identifiseres på dette stadiet blir grunnlaget for senere delaktiviteter i prosessen.

Organisasjons-struktur

Den neste aktiviteten i DISGIS-metoden er å beskrive organisasjonens struktur og hvordan arbeidsprosessene utføres. I denne aktiviteten legger man ennå ikke vekt på distribusjon. Alle forretningsobjektmodellene syntetiseres sammen til en global forretningsobjektmodell, og de forskjellige prosessmodellene syntetiseres sammen med en rollemodell av organisasjonsstrukturen til aktivitetsansvarmodeller for hvert AoC. (Se Figur 5.17.)

Figur 5.17: Modellering av organisasjonens forskjellige Area of Concern og struktur



Organisasjonens distribusjon

I aktiviteten «Enterprise distribution» (se Figur 5.16) beskrives hvordan organisasjonen er distribuert i en bestemt organisatorisk struktur. Uansett hvordan distribusjonen tilkommer, er den i dette perspektivet eksplisitt[1] i organisasjonen. En eventuell BPR-prosess vil også her kunne ha innvirkning på hvordan denne distribusjonen arter seg. En BPR-prosess kan både føre til ny distribusjon eller gjøre arbeidsprosessene mer sentraliserte enn de var fra før. Dette er opp til hvert enkelt BPR-prosjekt, og kan gjerne forandre seg neste gang organisasjonen setter igang en tilsvarende prosess.

I denne aktiviteten identifiseres på grunnlag av den globale forretningsobjektmodellen et antall «fellesskap» (RM-ODP: «Communities»), dvs grupper av roller i organisasjonsmodellen som har felles interesser ved at de bruker de samme objektene. Dette gjøres ved hjelp av *invers syntese* fra forretningsobjektmodellen. Én kilde til distribusjon kan således være lokalisering av de forretningsobjekter som

[1] Å bruke «implisitt» istedenfor «eksplisitt» kan også rettferdiggjøres her. Avhenngig av hva som er den primære årsak til organisasjonens geografiske distribusjon vil begge disse begreper med motsatt betydning passe like bra. Man kan si at «distribusjonen til Postverket jo er implisitt — det ligger i postverks natur; hva skal man med et postverk med ett postkontor?». Likeledes kan man si at «Postverket er eksplisitt distribuert; det er en helt nødvendig faktor for et vellykket postverk at det distribueres».

aksesseres av roller som hører til to eller flere felleskap. Dette gir altså en touch av PAKS (Process- , Aktivitets- och Komponentbaserat Systemstrukturering) [Goldkuhl 96].

Felleskapene identifisert på denne måten kan så syntetiseres sammen til domener og føderasjoner. Regler for hvordan dette gjøres eller hva domener og føderasjoner skal brukes til, gir ikke DISGIS noen klare retningslinjer for. Eksempel på et domene kan være en samling felleskap som administreres av felles sikkerhetspolitikk, maskinplattform, nettverk osv [RM-ODP 1]. Domener har en viss grad av selvstyre. En føderasjon kan være en samling domener med et felles mål. Føderasjopnsregler kan begrense domenenes selvstyre. Således kan vi kalle hele Plan- og bygningsetaten for en føderasjon, mens hver deletat (heiskontrollen, bygningskontrollen og byplankontoret) er hvert sitt domene. Domenene består igjen av et antall fellesskap, som består av f.eks saksbehandlere, funksjonærer på forværelser, i informasjonsskranker, i bibliotek etc.

Informasjonsmodellen

Prosessen beveger seg nå fra Enterprise viewpoint til Information viewpoint. I dette perspektivet modelleres informasjonsbehovet til de arbeidsprosesser som ble definert tidligere, ut fra en sammensatt modell av forretningsobjektene i alle de forskjellige AoC i organisasjonen. For hvert AoC blir forretningsobjektene beriket med detaljer, og så syntetisert til en global informasjonsmodell. I en sammensatt etat som PBE vil man imidlertid oppleve at enkelte felleskap ikke har samme begrepsoppfatning på tvers av felleskapsgrensene. I Plan- og bygningsetetens tilfelle er dette til en viss grad tilfelle. Enda større forskjeller kan finnes mellom domenene eller føderasjonene. Man kan tenke seg at f.eks Folkeregisteret har en annen oppfatning av hva som forstås med «person» enn et offentlig bibliotek.

Bruk av informasjonen

På grunnlag av modellen av organisasjonens arbeidsprosesser og informasjonsmodellen kan nå bruken av informasjonen i informasjonsmodellene modelleres. Byggestener her er «computational objects» som kan grupperes som verktøy eller tjenester. Objekter modelleres som tjenester og aktiviteter i driftsprosesser blir til verktøy. Her defineres kommunkasjonssemantikk[1] og scenarier. Merk at det her ikke finnes noen ren fysisk distribusjonsdimensjon i modellen, annet enn en logisk verktøy/tjenesterollemodell i form av verktøy som aksesserer tjenester i klient-tjener forhold. Således kan man f.eks tenke seg at en byggesøknad kan registreres med ett egnet registreringsverktøy, og behandles og godkjennes av andre. Imidlertid er det ikke noe i det inneværende perspektivet («Computational AoC» i Figur 5.16) som sier at disse ikke til syvende og sist kan ivaretas av det samme verktøyet eller applikasjonen. Logisk er de imidlertid

[1] Eksempelvis asynkrone eller synkrone kanaler, strømmer, signaler, hendelser etc.

separate inntil videre. Verktøy/tjenestemodellene for alle AoC'ene danner så grunnlaget for en global modell av samhandlingen mellom verktøy og tjenester.

Implementasjon og teknologi

Informasjonsmodellen fra «information AoC», sammen med den logiske distribusjonsmodellen, danner grunnlaget for å generere kildekode for systemet i en ikke-distribuert versjon. Samtidig kan man lage en spesifikasjon over hvordan distribusjonen skal håndteres i «Engineering viewpoint», med utkrystallisering av kapsler, noder og kanaler mellom objektene. Denne modellen er avhengig av den aktuell distribusjonsinstans, som stort er bestemt ut fra organisasjonens eksplisitte, geografiske distribusjon og deling av og bruksmønster for forretningeobjektene (se side 90).

Sammendrag

For øyeblikket finnes få objektorienterte metoder som tar hensyn til og tilbyr teknikker for modellering av distribusjon i edb-systemer. I og med den økede interesse for distribuerte objektsystemer, kommer imidlertid flere slike etterhvert, og én av dem, DISGIS General Method, er presentert i dette kapittelet. Metoden kombinerer OOram modelleringsmetode, som er basert på roller, med perspektiver fra RM-ODP referansemodell for distribuerte systemer. Metoden virker lovende, siden det er så kort vei fra OOrams verktøy/tjener-prinsipp til klient-tjener i distribuerte miljøer. Imidlertid må selve topografien i edb-systemet bestemmes med andre og mer uformelle metoder, og nok endel tekniske og ressursmessige betraktninger. I tillegg kan organisatoriske aspekter, så som resultater av BPR-prosesser, tillegges betydning.

6 Eksempel-applikasjon

Innledning

Som et eksempel på bruk av distribuerte objekter over en CORBA-basert infrastruktur, har jeg laget en eksempelapplikasjon for fremvisning av kartutsnitt fra Plan- og bygningsetatens kartdatabaser. Applikasjonen er basert på en klient-tjener arkitektur der klienten er tenkt å integreres i en web-tjeneste. Dette kapittelet redegjør for applikasjonen og utviklingen av denne applikasjonen. Noen deler er basert på en obligatorisk oppgave i forbindelse med kurset DSS — Distribuerte Samvirkende Systemer, som ble undervist første gang ved instituttet høsten 1996.[1] Brukergrensesnittet og klientdelen ble i utgangspunktet laget av Oddvar Kolset, men jeg har videreutviklet og tilpasset det en del. Her har jeg brukt JAVA WORKSHOP fra SUN,[2] som er et utviklingsverktøy, spesielt egnet til design av brukergrensesnitt i Java.

Andre eksempler på systemer med sammenlignbar detaljeringsgrad finnes i f.eks [Hetland 97] og [Brando 94].

Formålet med dette eksempelet

Hovedhensikten med eksemplet som presenteres i dette kapittelet er å få litt erfaring med hvordan det er å utvikle edb-systemer under betingelser som er skissert tidligere i oppgaven; i omgivelser der det finnes systemer fra før, og hvor disse skal knyttes sammen med en passelig mellomvareteknologi for å oppnå ny funksjonalitet gjennom integrasjon. Siden denne prototypen er utviklet helt uavhengig og utenfor Plan- og bygningsetaten, er det ingen fakitsk eksisterende systemer som er integrert her. Disse blir derfor simulert med enkle

[1] Kurset har egen hjemmeside: `http://www.ifi.uio.no/~samvsys/`

[2] Java WorkShop finnes for Solaris og WindowsNT/95 på: `http://www.sun.com/workshop/`

programmer. Likevel vil det forhåpentlig være interessant og lærerikt som en erfaring med distribuert objektteknologi. Spesielt har jeg under utviklingen forsøkt å legge merke til fordelingen av ressurser — i form av tankevirksomhet, problemløsing og programmering — mellom problemstillinger som vedkommer selve applikasjonslogikken kontra distribusjon og datakommunikasjon. Strategien jeg har valgt for å finne ut noe om dette er å forandre på detaljer i systemets applikasjonslogikk og distribusjon *etter* at en fungerende versjon er implementert, for å se hvordan dette påvirker kompleksitet og arbeidsmengde.[3]

Det har også vært et mål å evaluere de metoder som har vært behandlet tidligere i oppgaven, spesielt DISGIS. Imidlertid er dette prosjektet så lite at det har vært lettere å tenke design før analyse. Den objektorienterte analysen blir således fort en etterrasjonalisering, og viser desverre ikke hvordan metoden og de verktøy som støtter den (f.eks OORAM PROFESSIONAL) kan bidra til den optimalisering og strukturering som større prosjekter vil være avhengig av for å oppnå suksess. I den forbindelse er det også verdt å merke seg at eksemplene som presenteres i litteraturen jeg har brukt som behandler metodene nevnt tidligere heller ikke er særlig omfattende. Dernest kan den versjon jeg har brukt av modelleringsverktøyet nevnt ovenfor kun generere kildekode for Smalltalk og C++. Ønskelig ville vært om det kunne lage CORBA IDL-spesifikasjoner, som grunnlag for distribusjon.[4] Delsystemer kunne etterpå implementeres i andre språk, som de to nevnt ovenfor. Siden detta altså ikke er mulig for øyeblikket, blir hele prosessen ganske oppstykket, og bruken av de forskjellige modeller og dokumenter usystematisk. Overgangen mellom fasene blir heller ikke helt sømløs.

Selve evalueringen av dette prosjektet vil være kvalitativ og ganske uformell, og jeg har ikke foretatt tester av systemet når det gjelder robusthet, ytelse, nytteverdi eller overfor aktuelle brukergrupper. Dette fordi jeg tror det er urealistisk å gjennomføre en evaluering av et fullskala utviklingsprosjekt i en hovedoppgave, siden det i så fall ville tatt en uforholdsmessig stor del av arbeidsmengden. Gyldigheten til de konklusjoner som kan trekkes av erfaringene med dette lekeprosjektet kan derfor være noe begrenset, spesielt med hensyn på evaluering av metoden som brukes, som nevnt ovenfor. Dernest vil erfaringer med samarbeidsproblematikk i en prosjektsammenheng ikke bli belyst i særlig utstrekning, fordi jeg stort sett har gjennomført prosjektet alene. Forhold mellom ressursbruk i applikasjonslogikkutvikling og distribusjonsutvikling, derimot, kan gi et bedre grunnlag for konklusjoner. Dette kommer jeg tilbake til mot slutten av kapitlet.

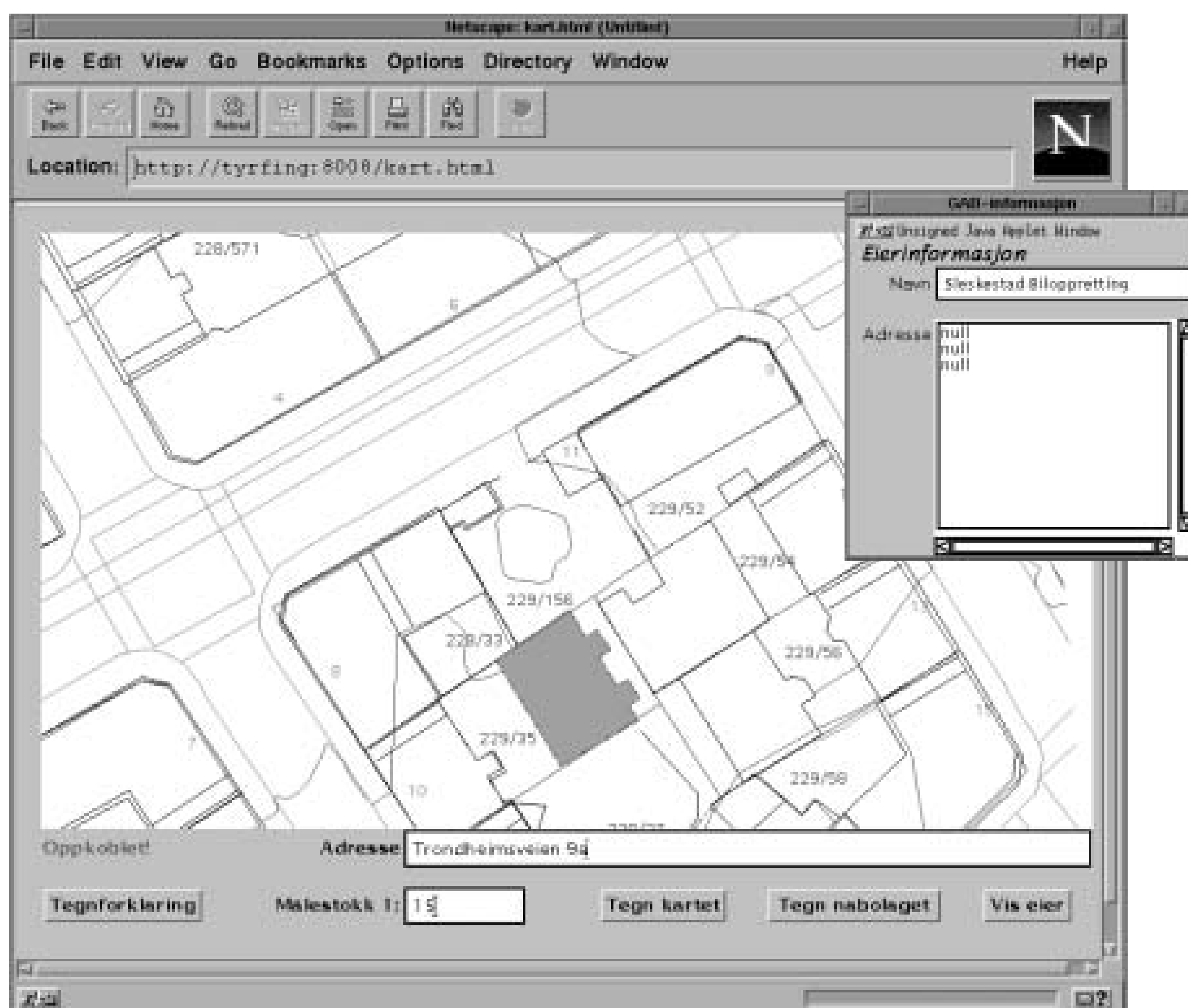
---

[3] Merk at analysen og designen som kommer senere tar utgangspunkt i systemet *etter* forandringen.

[4] En kobling mellom OOram modeller og CORBA IDL/ DCOM IDL er foreslått i [Berre et al 96].

Formålet med applikasjonen

Utad består systemet av en enkel Java applet som er tenkt brukt i en web-tjeneste rettet mot publikum enten som potensielle utbyggere eller nysgjerrige borgere. I tillegg kan man tenke seg at etaten kan benytte verktøyet internt som informasjonstjeneste i en intranettløsning. Brukeren taster inn en adresse vedkommende er interessert i, og skjermen viser deretter et passelig stort område rundt denne adressen, hvis den finnes i GAB-registeret. Figur 6.1 viser et typisk skjermbilde.



Figur 6.1: Kartutsnitt rundt Trondheimsveien 11 i Oslo sentrum. Plan- og Bygningsetaten (Tr.vn. 5) befinnner seg litt utenfor nedre venstre hjørne av kartutsnittet. En bygning er markert, og eiendomsinformasjon for denne vises i et eget vindu. Merk at eiendomsinformasjonen for denne bygningen i dette eksempelet er fiktivt. (På den annen side har befaring i området avslørt at bygningen nå er revet…)

Ved å markere en av bygningene med musen kan brukeren få opplysninger fra GAB-registeret om bygningen, i første omgang hvem som eier den. En tenkt utvidelse av dette er mulighet til automatisk å få svar på hvem som er naboer som må kontaktes hvis man skal bygge eller forandre noe på en tomt.

Siden etaten tar seg betalt for enkelte kartprodukter, er det ikke rimelig at de skal tilby sine kart gratis til publikum i samme kvalitet som de selger. Det kan derfor være en mulighet at de kartblad som man får frem på web-tjenesten er av enklere kvalitet hva angår detaljrikdom, nøyaktighet og oppdateringsgrad. I tillegg kunne tjenesten differensieres ved en senere anledning, slik at kartutsnitt som er i profesjonell kvalitet, som etaten ellers ville brukt ressurser på å lage papirutskrifter av, kunne formidles mot betaling i en eller annen passende elektronisk form. Her vil f.eks en CORBA-basert lisensieringsfunksjon kunne brukes. Flere arkitekter og utbyggere ønsker å kunne motta kartgrunnlag elektronisk og senere tegne inn sine prosjekter med DAK. Til nå har dette vist seg vanskelig, uvisst av hvilken grunn.[5]

Bruksanvisning

Klienten Applet'en lastes ned i web-leseren når nettsiden hentes, og eksekveringen startes. Kontakt med GAB-tjeneren opprettes, og det skrives ut melding om dette og dessuten ledetekster til tekstfeltene. (Se Figur 6.1) Knappen «Tegn kartet» blir gjort tilgjengelig. Det kan oppgis en adresse og/eller ønsket målestokk for det kartutsnittet man vil ha vist fram. Knappen «Tegn kartet» klikkes og kartdataene for det aktuelle området tegnes ut. Man kan oppgi ny adresse og/eller skaleringsfaktor og få nye kartutsnitt tegnet ut. Hver gang hentes de grafiske objektene fra tjeneren. Brukeren kan klikke på en bygning i kartutsnittet. Da markeres denne bygningen med en annen farve. Ved deretter å klikke på knappen «Tegn nabolaget», vil programmet beregnet et passelig stort område rundt det huset som er klikket på, og kartdata for dette området hentes ut fra tjeneren og tegnes ut. Ved å trykke på «Vis eier», vil et vindu vise informasjon om eieren av den bygningen som er valgt.

Ved å trykke på «Tegnforklaring», vises et vindu med de viktigste tegnforklaringer, dvs fargekoder for bygninger, tomtegrenser, veier etc.

Tjenerne Forutsatt at en ORBIX-demon er i drift på tjener-noden startes karttjeneren slik:

```
$> server kart &
```

[5] Dette har jeg hørt av arkitekter som selv har henvendt seg til etaten med forespørsel om å få situasjonskart på diskett. Imidlertid er det mange forskellige formater i bruk; AUTOCAD, MINICAD, ARCHICAD osv, og etaten kan formodentlig ikke forholde seg til alle disse.

```
Reading kart...
Warning at line 11: Empty file header
OK.
Registering mapServer...
[mapServer: New Connection(tyring.ifi.uio.no,IT_daemon,*,root,
pid=20927,optimised) ]
[mapServer:Server "mapServer" is now available to the network ]
[      Configuration tcp/1595/xdr ]
$>
```

Dette gjør at tjeneren leser inn alle kartfilene etter hverandre, og registrerer sine tjenester hos objektmegleren (ORBIX-demonen).

GAB-tjeneren startes slik:

```
$> gabserver &
Registering gabServer...
[gabServer: New Connection(tyrfing.ifi.uio.no,IT_daemon,*,root
,pid=20927,optimised) ]
[gabServer: Server "gabServer" is now available to the network]
[      Configuration tcp/1598/xdr ]
$>
```

Selve databasen administreres med hjelpeverktøyet **dbm**. Dette enkle verktøyet legger inn og fjerner tupler fra eiendomsdatabasen. **dbm** tar opsjonene **-a**, **-d** eller ingen ting.

For å legge til en eier i databasen:

```
$> dbm -a
Address: Drammensveien 1
Owner: Kong Harald V
Location (north):123456
Location  (east):789011
Database updated!
$>
```

For å slette en adresse:

```
$> dbm -d
Address: Drammensveien 1
Drammensveien 1 deleted from database!
$>
```

For en dump av hele databasen:

```
$> dbm
Address: Trondheimsveien 9b
    Owner: Sleskestad Biloppretting
    Location (North/East): 68746, 256371

Address: Trondheimsveien 9a
    Owner: Skurkerud's Bakgårdsverksted AS
    Location (North/East): 67026, 254651

Address: Trondheimsveien 4
    Owner:
    Location (North/East): 68546, 251151
```

```
Address: Herslebsgate 11
    Owner: Olsen Vask og Rens
    Location (North/East): 68770, 256350

Address: Herslebsgate 9
    Owner: Oslo Kommune
    Location (North/East): 70400, 256120

Address: Herslebsgate 7
    Owner: Skurkerud Bil & Brems
    Location (North/East): 68800, 256320
$>
```

Analyse med OOram og de fem «viewpoints»

For å bruke noe av stoffet som er presentert tidligere i denne hovedoppgaven, vil jeg her modellere det distribuerte eksempelet med OOram og de fem «viewpoints» fra RM-ODP, som ble presentert i henholdvis Kapittel 5 og 3. Problemet kunne passelig vært analysert og modellert med DISGIS General Method [Oldevik 96], men som i likhet med mange andre eksempler av overkommelig størrelse, er imidlertid dette så begrenset i omfang at det er liten vits i å benytte alle aspekter av et større metodeverk. Jeg vil likevel komme inn på hvert av de fem viewpoints fra RM-ODP rammeverket, inspirert av [Berre et al 96].

Enterprise viewpoint

I Enterprise view defineres de aspekter ved systemet som angår organisasjonens mål, prosesser og aktiviteter både internt og i interaksjon med omgivelsene som omgir organisasjonen. Prototypen som presenteres i dette kapittelet er tenkt som et verktøy som blant annet kan brukes til å finne opplysninger som er interessante i forbindelse med byggesaker. Som en illustrasjon går jeg derfor ut fra et svært forenklet bilde av de involverte parter i godkjennelsen (aprobasjon) av et byggeprosjekt. Figur 6.2 viser noen av de *roller* som kan finnes i en slik sak.

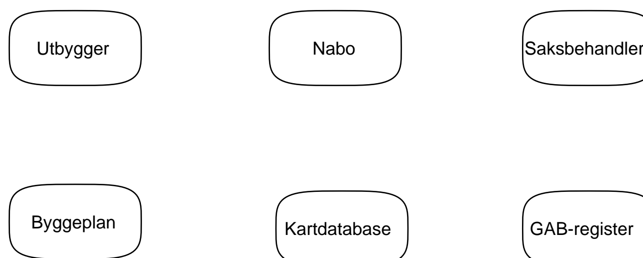
Det er to åpenbare «Ara of Concern» (AoC) i den forbindelse:

- Behandling av byggesøknader
- Informasjonsbank for byens innbyggere. Bibliotek for stedfestede data.

Kikker vi først litt mer på søknadsbehandling som en forretningsprosess og hvilke funksjoner som inngår i behandling av byggesøknader, finner vi at det er flere forretningsobjekter i samarbeid om saksbe-

handlingen. Målet her er å produsere byggesøknaden. En del av denne prosessen består i å finne ut hvilke naboer som må varsles.

Figur 6.2: Roller i enterprise Viewpoint



Vi kan definere forretningsregler og -objekter, og felleskap:

- Forretningsregler: *Nabovarsel skal sendes til dem som er oppført som eiere av naboeiendommer etter visse kriterier (avstand, felles grenselinjer etc);*[6]
- Fellesskap/interessegrupper: *utbyggere, saksbehandlere, naboer;*
- Forretningsobjekter: *byggemeldingstegning, nabovarsel, situasjonskart, byggemelding, vedtak …*

Det mest sentrale forretningsonbjektet for prototypen er *situasjonskart* fra siste punkt i listen ovenfor.

Information viewpoint

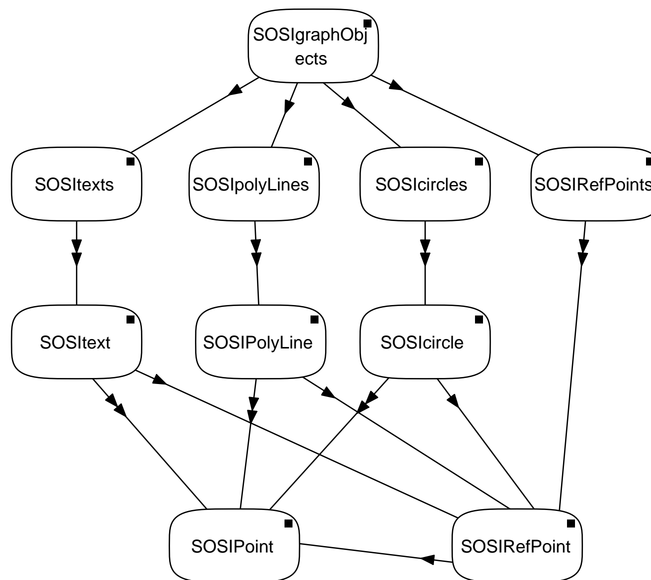
I dette perspektivet legger vi innhold i forretningsobjektene. Et krav kan være at *kartdatabasen* skal lagre data på SOSI-standard.[7] Dette er et tekstlig format som beskriver geografiske data. Denne standarden beskriver også datastrukturen med nestede ER-diagrammer [SOSI 95]. Datamodellen i Figur 6.3 er basert på disse ER-diagrammene, men med en del forenklinger. Grensesnittene er formalisert i IDL-spesifikasjonen i Figur 6.5 på side 101.

I et *information viewpoint* i en mer fullstendig analyse ville man tatt hensyn til flere forretningsobjekter. Jeg skal ikke gå nærmere inn på andre forretningsobjekter som faller utenfor kartoppslagdomenet i denne omgang. Her kan man modellere strukturen mellom for eksempel saksbehandlere, saker, journaler, saksmapper, vedtak, utvalgsforberedelser osv — og selvsagt objekter i kartsystemet.

[6] Denne fremstillingen er selvsagt sterkt forenklet. Mange av de virkelige forretningsreglene er bestemt av lovverk blant annet nedfelt i Plan- og Bygningsloven og Forvaltningsloven.

[7] SOSI er imidlertid på vei ut, og vil bli erstattet av et internasjonalt format etterhvert [GEOLOK 95]. Temakoder beholdes likevel i stor grad fra SOSI.

Figur 6.3: Datamodell basert på ER-modell i [SOSI 95]



Computation viewpoint

For å sende nabovarsel trengs et verktøy for å finne ut hvem naboene er, eller mer presist: ***hvem som eier naboeiendommene og som man må sende nabovarsel til.*** For å løse dette problemet (og også forøvrig oppgaven som informasjonsbank/bibliotek som nevnt på side 98, «Area of Concern») lages systemet som ett verktøy og to tjenere; én som leverer kart-data, og én som leverer opplysninger om adresser/eiendommer med plassering i et koordinatsystem.

Figur 6.4 viser et scenario hvor en bruker henter frem en kartbit etter å ha tastet inn adressen han/hun ønsker å se nærmere på. Det er uklart hvilken rolle som kan løse selve nabosøkproblemet; om det er GAB-registerets oppgave eller kartbasen. Man kan også tenke seg at klienten selv finner ut hvilke tomter som grenser inntil, eller en innen en viss avstand fra tomten det er spørsmål om. Jeg har imidlertid avgrenset meg fra løsning av akkurat dette problemet, men antyder bare dette som en ønsket funksjonalitet.[8]

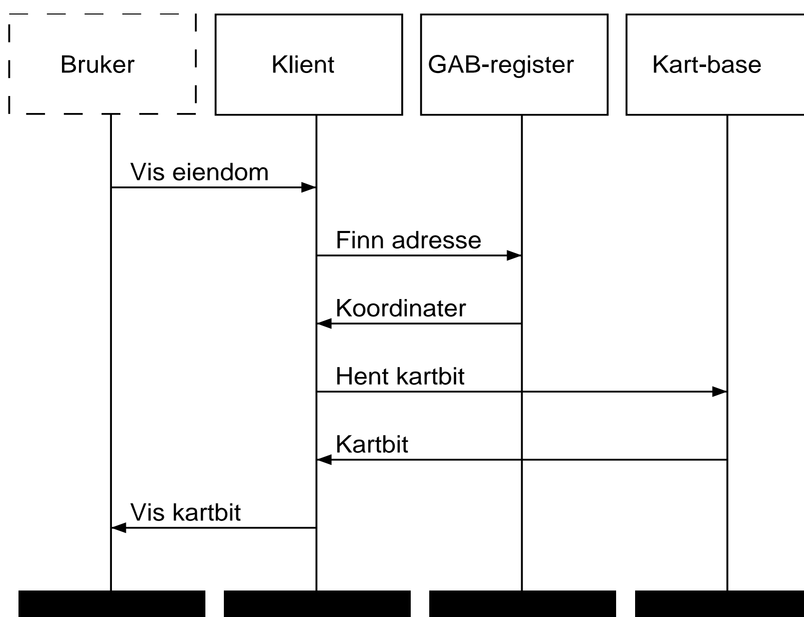
Etter dette scenarioet kan vi definere et IDL-grensensnitt for applikasjonen. Figur 6.5 viser dette. Dette kunne også vært gjort med OOrams eget edfinisjonsspråk, men siden det skal implementeres med CORBA, har jeg brukt CORBA IDL i steden.[9]

Først i IDL-spesifikasjonen er endel typedefinisjoner. ER-modellen av grafiske objekter fra SOSI er implementert som abstrakte datatyper. Den grunnleggende elementtype er punkt, modellert som strukturen

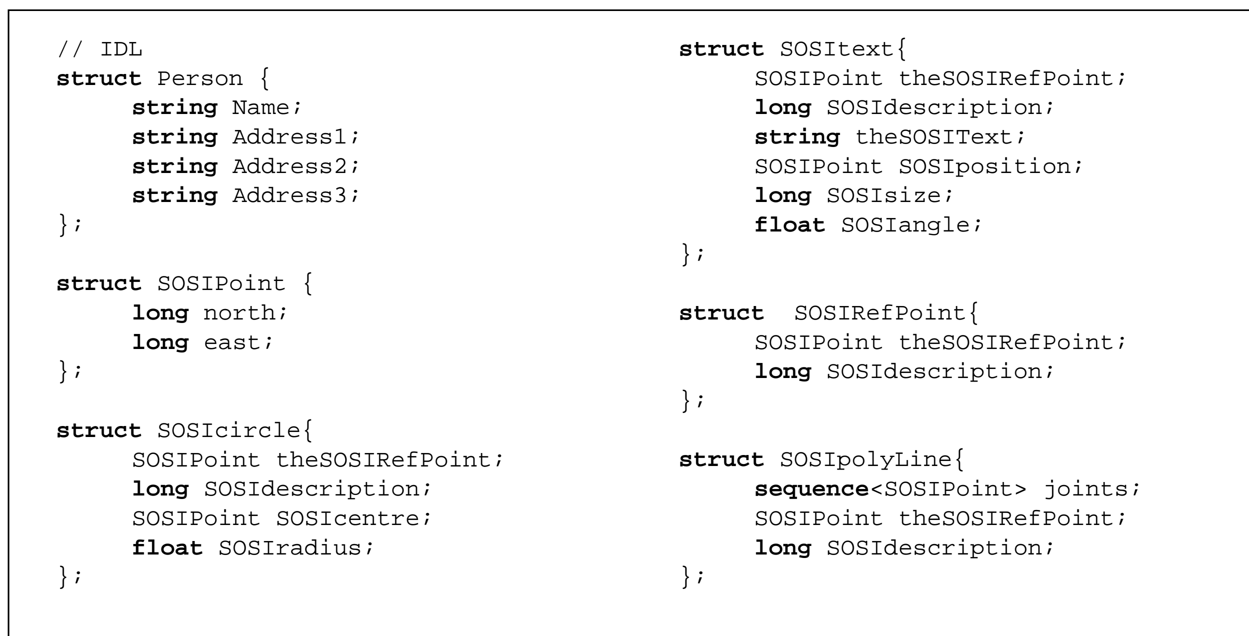
[8] Generelle nabosøk i flerdimensjonale R-trær er behandlet bl.a i [Roussopoulos et al 95].

[9] Merk at valg av beskrivelsesspråk i dette tilfellet (CORBA IDL) legger føringer for valg av infrastruktur, dvs et eller annet CORBA-produkt. Dette er egentlig feil viewpoint å gjøre dette i, men verktøyet som er brukt til modellering generer ikke IDL-kode.

Figur 6.4: Verktøy og Tjenester



`SOSIPoint`. Av disse bygges det opp typer for sirkler, polygoner etc. Sistnevnte består blant annet av en sekvens av punkter.



```
// IDL
struct Person {
	string Name;
	string Address1;
	string Address2;
	string Address3;
};

struct SOSIPoint {
	long north;
	long east;
};

struct SOSIcircle{
	SOSIPoint theSOSIRefPoint;
	long SOSIdescription;
	SOSIPoint SOSIcentre;
	float SOSIradius;
};

struct SOSItext{
	SOSIPoint theSOSIRefPoint;
	long SOSIdescription;
	string theSOSIText;
	SOSIPoint SOSIposition;
	long SOSIsize;
	float SOSIangle;
};

struct  SOSIRefPoint{
	SOSIPoint theSOSIRefPoint;
	long SOSIdescription;
};

struct SOSIpolyLine{
	sequence<SOSIPoint> joints;
	SOSIPoint theSOSIRefPoint;
	long SOSIdescription;
};
```

Figur 6.5: IDL-beskrivelse av datatyper for personinformasjon og grafiske objekter

IDL-spesifikasjonen definerer også to grensesnitt; ett for GAB-tjeneren og ett for karttjeneren. GAB-grensensittet (**`interface`** `gabSer-`

ver) har metoder for å finne koordinat ut fra en adresse, og eier fra koordinat. Kartgrensesnittet (**interface** mapServer) tilbyr en metode for å returnere grafiske objekter som overlappes av et gitt søkeområde. Datatypen som returneres er en abstrakt datatype som består av fire sekvenser av henholdsvis sirkler, polygoner, tekster og representasjonspunkter. Ideelt sett er det mer ønskelig at datatypen som representerer en samling grafiske objekter virkelig var en samling eller mengde, dvs en sekvens av generelle grafiske objekter (Figur 6.6). Hvilken type hvert objekt i virkeligheten var, kunne så

```
struct SOSIgraphObjects{
      sequence<SOSIcircle> SOSIcircles;
      sequence<SOSIpolyLine> SOSIpolyLines;
      sequence<SOSItext> SOSItexts;
      sequence<SOSIRefPoint> SOSIRefPoints;
};
```

Figur 6.6: Datatype for en samling grafiske objekter

bestemmes av hvilken subklassse de tilhørte. På denne måten kunne polymorfien blitt utnyttet bedre, og det ville ikke vært nødvendig å behandle de forskjellige objektene forskjellig i klienten. Av praktiske årsaker er den herliggende løsningen valgt likevel.

```
interface mapServer{
      SOSIgraphObjects getRegion(in SOSIPoint NW, in SOSIPoint SE);
      SOSIgraphObjects getOrigoRegion();
};
interface gabServer{
      Person findOwner(in SOSIPoint building);
      SOSIPoint findLocation(in string address);
      string findAddress(in SOSIPoint location);
};
```

For å teste ressursbruk ved forandring er metodene her skilt ut fra **interface** mapServer

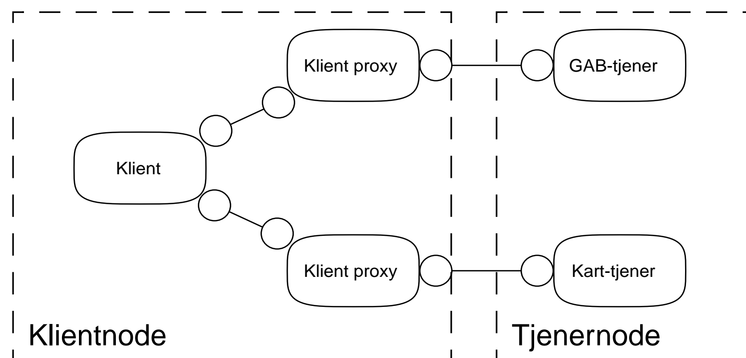
Figur 6.7: Grensesnitt for karttjener og GAB-tjener

Engineering viewpoint

I Engineering Viewpoint er det den praktiske distribusjonen som er sentral. Her bestemmes strukturen i systemet, som noder, kapsler, klaser og kanaler introdusert i RM-ODP (side 46ff). I praksis blir dette en struktur med «proxy»-objekter,[10] som vist i Figur 6.8. Proxy-mønsteret er identifisert i [Gamma et al 95]:207–17, og formålet er å fungere som et surrogat for det virkelige objektet, som i dette tilfellet

[10] proxy (eng): stråmann

Figur 6.8: Engineering viewpoint



(muligens) befinner seg et annet sted, f.eks på en annen node. Klientproxyene finnes igjen i kildekoden i linje 41–42 i **`kart.java`** på side 187 som objektreferansene **`mapserver`** og **`gabserver`**.

Technology viewpoint

I det siste viewpointet beskrives valg av teknologi i systemet, f.eks implementasjonsspråk, operativsystem og mellomvareprodukter. Her er valgt Java applet for klienten, C++ for tjenerne og ORBIX som objektmegler. Dette er et CORBA-produkt. Teknisk plattform for tjenerne er SUN ULTRASPARC.

Overordnet arkitektur og design

Infrastrukturen i dette prosjektet er mye av den samme som under løsningen av oppgaven i DSS-kurset nevnt tidligere. Selv om klient-tjener paradigmet er noe oppløst i tankegangen rundt distribuerte objekter, er det i praksis snakk om en Java applet klient som aksesserer to databasetjenere; en kartdatabase og en GAB-database. Koblingen mellom kartdata og GAB-data skjer med geografiske koordinater.

Oversikt over bygninger basert på gårds- og bruksnummer finnes i GAB-registeret. Dette registeret har opplysninger over bygninger og deres «rettighetshavere»,[11] hvor hver bygning kan stedplasseres med et koordinatsett. Dette koordinatsettet er således et referansepunkt til fysiske punkter representert i kartmassen som også vedlikeholdes av etaten. Etaten har nå et system («VG») som kan vise kartblad med varierende detaljerinsgrad, med ting som veier, hus, ledningsnett, tomtegrenser etc. Det finnes koblinger mellom referansepunkter i dette systemet og GAB-registeret, men ikke noen kobling videre i saksbehandlingsregisteret som er konsekvent. Jeg skal imidlertid ikke ta med saksarkivet i denne omgang.

[11] Rettighetshaver er en som f.eks eier en eiendom, har rett til å bygge vei på den etc.

I dette prosjektet er GAB-registeret og kartdatabasen endel forenklet. «GAB-simulatoren» er en enkel database med attributt/verdipar, slik at det simulerer en enkel relasjonsdatabase med to tabeller. Kartdatabasen består av en enkel datastruktur som bygges opp når tjeneren initialiseres ved at den leser inn kartdata fra datafiler i forenklet SOSI-format.

I det løsningeforslaget som ble utarbeidet i samarbeid med Oddvar Kolset i DSS-kurset, ble Java benyttet til realisering av både klient- og tjenerside. I den herliggende varianten har jeg imidlertid brukt C++ i implementeringen av sistnevnte. Grunner til dette er:

- **Ytelse.** Java er pr i dag i størrelsesorden 10–20 ganger så tregt som tilsvarende C++-kode. Hvis tjeneren skal søke gjennom store mengder kartdata og finne riktige geografiske objekter med en rimelig responstid, kan dette være viktig.
- **Heterogenitet.** Jeg ønsker å teste kommunikasjon mellom forskjellige implementasjonsplattformer.[12]
- **Mer realistiske geografiske data.** I den tidligere implementasjonen i forbindelse med DSS-kurset brukte vi fiktive geografiske data som vi tegnet med med `xfig(1)` tegneprogram (se Figur 0.1 på side v). I denne versjonen ønsket jeg å bruke realistiske data. Disse foreligger gjerne på SOSI-format [SOSI 95] som må parseres på en mer skikkelig måte enn parseringen av xfig-koden, som i forrige versjon var ganske primitiv. Derfor har jeg brukt `flex(1)` og `bison(1)` som genererer C-kode for parsere til BNF-spesifiserte språk (Bachus-Naur Form) [Levine et al 92].

3-lags arkitektur

Tolkning og presentasjon av data fra karttjeneren er opp til klienten. I filer som følger SOSI-standarden er det blant annet registrert en del grafiske objekter. Hver av disse objektene har bl.a. et «tema» som forteller hva det er snakk om; tomtegrense, hus, vei, trikkeskinne, innsjø, elv osv [SOSI 95]. Det er opp til klienten å presentere dataene på en passelig måte, med farger, linjetykkelse etc. På denne måten blir det en tre-lags arkitektur, hvor det er én eller flere database(r) som håndterer lagring av de stedbestemte data, en tjener som på forespørsel finner frem et riktig utsnitt av kartmassen, velger ut riktige objekter og sender dem til klienten.[13] Klienten presenterer i sin tur disse på en

[12] Heterogeniteten i denne sammenheng begrenser seg imidlertid til Java kontra C++. Mellomvareproduktet er det samme; Orbix fra Iona Industries. Riktignok er Java og C++-variantene av Orbix-pakken noe forskjellige, men noen full heterogenitetstest blir dette ikke. For å teste en mer heterogen løsning, kunne det vært en idé å formidle én eller begge tjenernes tjenester med f.eks CHORUS' COOL ORB for LINUX, men det får bli til en annen gang.

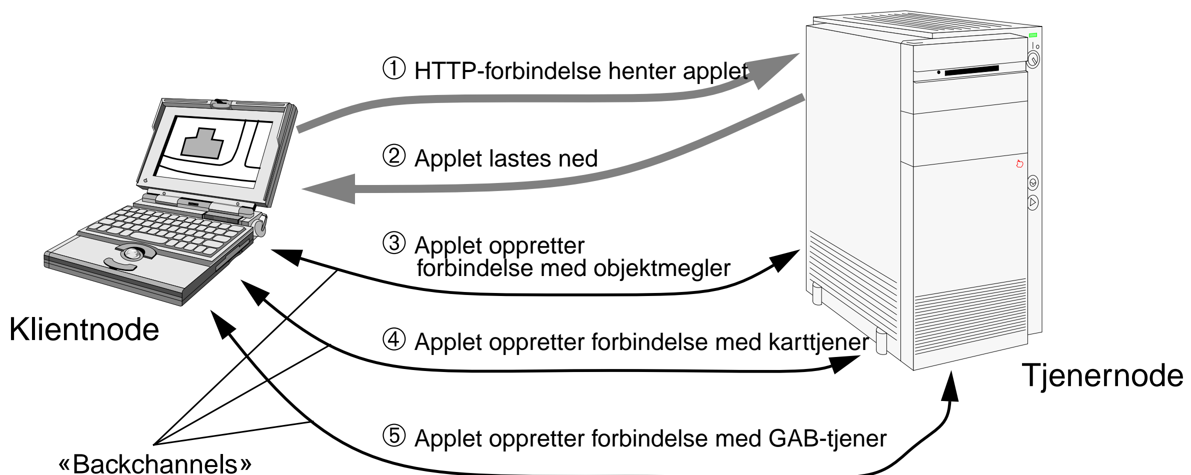
[13] Merk at tre-lags arkitektur *ikke* er det samme som treskjemaarkitektur fra databaseteorien. Det første beskriver ansvarsfordeling mellom klienter og tjenere [Orfali et al 96a]:19, mens sistnevnte skiller mellom de tre skjemaer i en databasebeskrivelse; konseptuelt-, internt- og fysisk skjema [Skagestein 91]:203. Imidlertid kan disse samkjøres, og slik brukes som en måte å bestemme delingen mellom klienter og tjenere [Skagestein 94].

tiltalende måte i et grafisk brukergrensesnitt. Slik er det mulig å differensiere klienter mot forskjellige brukergrupper eller bruksområder. Eksempelvis kan «vanlig» publikum få en polularisert variant med mindre detaljer, men mer naturrealisme (farger på trær, gater, hus etc), mens interne brukere, arkitekter og utbyggere kunne få en mer detaljert og teknisk variant. Bruksområdet og de aktuelle brukernes krav kan derfor bestemme hvordan data presenteres.

I tillegg er det mulig at etaten ønsker å bearbeide data før de gjøres tilgjengelig for publikum, for eksempel ved å filtrere vekk enkelte grupper grafiske objekter, og kode om tema-informasjonen i de som leveres. I SOSI-formatet er temainformasjon kodet med firesifrete verdier med en stor grad av detaljering. For mange «hverdagslige» formål er kanskje ikke denne detaljrikdom nødvendig, og kan forenkles på tjenerensiden. Dette gjør at etaten ved å filtrere og styre datakvaliteten kan skjerme sine data mot uatorisert kommersiell bruk som de ellers vil ta betaling for.

Backchannels — sidekanaler

Java applet'en lastes ned til web-leseren med HTTP (Hypertext Transfer Protocol) som alle andre websider, indikert som forbindelse ① og ② i Figur 6.9. Når eksekveringen starter, opprettes «backchannels» — dvs nye kommunikasjonsforbindelser — først til objektmegleren, dvs Orbix-demonen, med forbindelse ③. Denne formidler kontakt til tjenerkapslene på den samme noden som applet'en ble hentet ned fra. (Forbindelse ④ og ⑤.) All videre dataoverføring foregår så over de tre siste forbindelsene, spesielt over ④ og ⑤. Av sikkerhetsmessige årsa-



Figur 6.9: Arkitektur i prototypsystemet

ker er det vanligvis ikke mulig å opprette nettverksforbindelser til andre noder enn den som applet'en ble hentet fra.[14] Slike forbindelser må i så fall gå igjennom den samme noden som leverte applet'en. Slik kan tjenerne pakke inn og formidle kontakt til f.eks andre eksisterende systemer. Dette viser hvordan tjenerobjekter i det ene øyeblikket er tjener og det neste klient. Dette oppløser altså til en viss grad klient/tjener-paradigmet.

GAB-tjeneren

Tjeneren for eiendomsinformasjon er i dette prosjektet svært enkel, og kun ment som en illustrasjon. Databasen simulerer en enkel totabell relasjonsdatabase, og er implementert med `ndbm(3)`-funksjoner. Dette er et bibliotek av C-funksjoner som kan brukes til å lagre og hente to-verdi tupler, slik som f.eks `adresse/koordinat` fra hashindekserte[15] filer på en rask og effektiv måte. Databasen består av én tabell for adresser og koordinater, og én for koordinater og eiere. Adressen er primørnøkkel i den ene, og koordinat i den andre. «Databasen» tilbyr gjennom sitt grensesnitt en metode for å finne koordinatpunkt ut fra en adressetekst, og en for å finne eierdata ut fra et koordinat.

Karttjeneren

Karttjerenen består av en SOSI-parser, et R-tre som lagrer alle kartdata i en passelig effektiv flerdimensjonal søkestruktur og funksjoner som søker i R-treet og sender data på forespørsel.

SOSI-parser. Parseren er bygget opp ved hjelp av `flex(1)` og `bison(1)`.[16] Den er svært enkel, og gjenkjenner kun et subsett av SOSI-formatet. Jeg har testet den med en SOSI-fil som er eksportert fra VG Innsyn programmet. På mer omfattende geodatafiler vil det garantert oppstå problemer. Parseren må derfor utvides i slike tilfeller.

R-tre. Parseren leser SOSI-data og leverer grafiske objekter til en *R-tre* datastruktur. (Se Figur 6.10.) R-trær er beskrevet bl.a i [Guttman 84], [Panagopoulou et al 96]. R-trær er en generalisering av *B-trær* (se f.eks [Stubbs & Webre 89]) slik at det er mulig å indeksere og søke etter objekter med utstrekning i flere dimensjoner, i motsetning til B-trær som bare kan håndtere én dimensjon.[17] Implementasjonen i dette eksempelet er basert på C-kode skrevet av Daniel Green.[18] Jeg har oversatt den til C++ og spesialisert den noe for dette propsjektet.

---

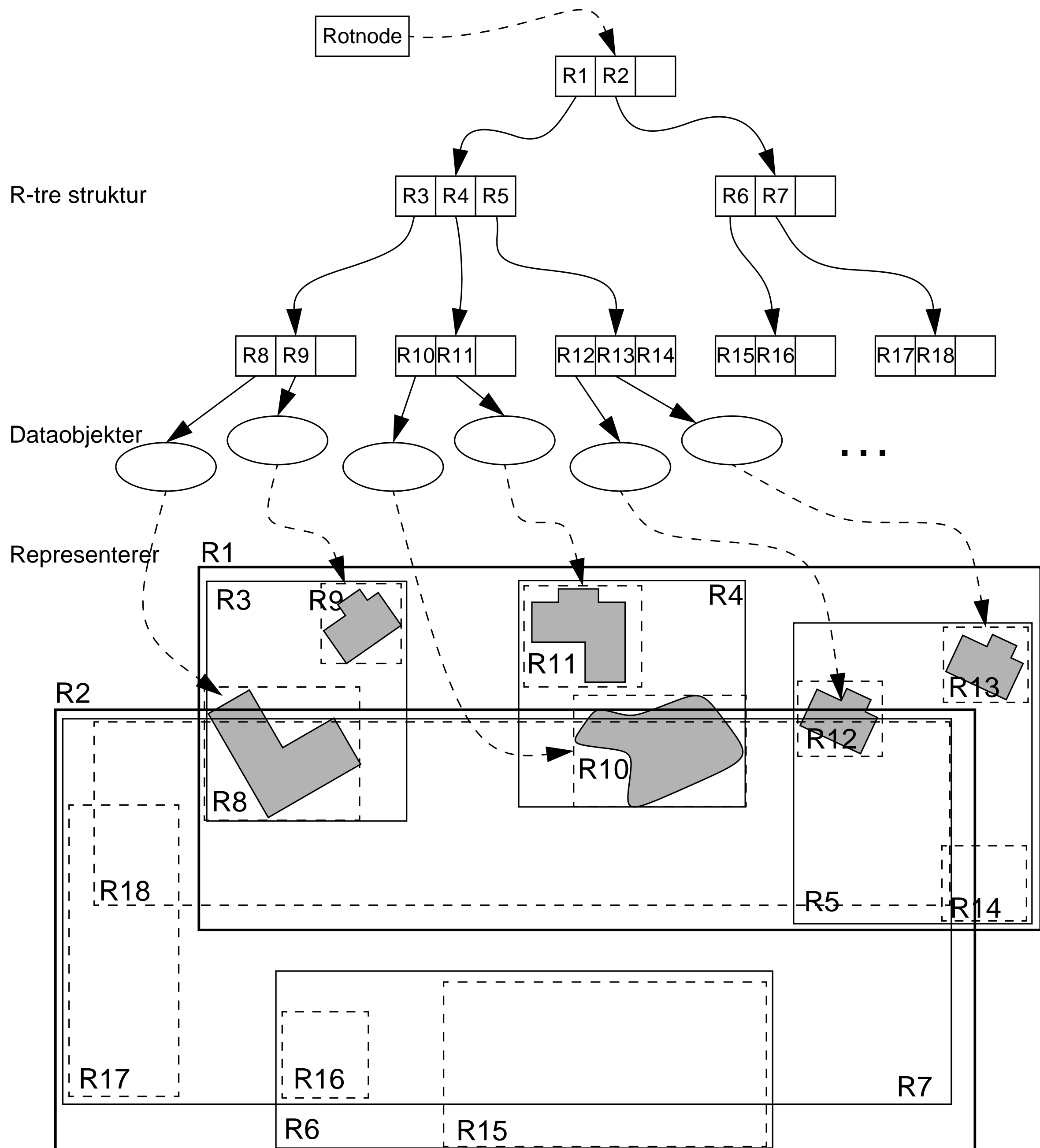
[14] Disse og andre restriksjoner vil kunne lempes på ved «signering» av applet'er, dvs registrerte produsenter garanterer for at applet'er og komponenter de har produsert ikke representerer noen sikkerhetsrisiko for brukeren.

[15] Hashing er en lagre/søkemetode som er beskrevet i blant annet [Stubbs & Webre 89].

[16] Dette er Free Software Foundations varianter av AT&T's `lex(1)` og `yacc(1)`.

[17] R-tre implementasjonen som er i bruk i dette prosjektet kan enkelt generaliseres til å plassere og søke etter objekter med utstrekning i vilkårlig antall dimensjoner.

[18] Opprinnelig kode er tilgjengelig på: `http://glass.ucsc.edu/~tonig/rtrees/dgreen`



Figur 6.10: R-tre datastrukturen med noen forekomster

Som i et B-tre har R-treet noder som hver inneholder informasjon om området noden dekker, og hvilke undertrær som dekkes av dette området. Løvnodene i treet inneholder referanser til selve dataobjektene. Størrelsen på hver node tilpasses gjerne maskinens *sidestørrelse* (eller diskblokkstørrelse hvis data lagres på disk) for å optimalisere ressursbruken i maskinen. R-treet beregner minste rektangel som omslutter hvert grafisk objekt, og plasserer objektreferansen i den node i treet som har ansvaret for det området dette raktangelet faller

inn under. Hvis en node er full når det skal settes inn et nytt objekt, splittes den i to som i et B-tre, og splittelsen forgrener seg oppover i treet (husk at trær vokser nedover!) slik at balansen opprettholdes.

Objektmegleren

For begge tjenerne gjelder at de etter at er klare til å betjene sine klienter registrerer sine tjenester hos Orbix-demonen lokalt på maskinen de eksekveres på, i dette tilfellet er `tyrfing.ifi.uio.no`. På forespørsel fra klienter om levering av kartustnitt finner karttjeneren ved hjelp av søk i R-treet alle objekter hvis omsluttende rektangel overlapper søkerektangelet. Disse objektene kopieres ut fra datastrukturen og returneres til klienten. Klienten pakker så ut denne strukturen, traverserer den og tegner opp objektene på skjermen.

Klienten

Klienten i prosjektet er en Java applet, som kan kjøres i Netscape, Appletviewer og i prinsippet alle andre nettlesere som kan eksekvere applets.[19] Programmet vedlikeholder to referanser til henholdsvis GAB-tjeneren og karttjeneren, som begge bindes mot sine respektive tjenere ved programstart. Ut fra størrelsen på tegneflaten og målestokken som er oppgitt i målestokkfeltet, beregnes et passelig rektangel som sendes som argument til karttjeneren når klienten ber om kartdata. Når kartdata hentes fra tjeneren, opprettes objektlister med kopier av alle de grafiske objektene som ble returnert. Alle hus, dvs de objekter med tema-kode for vanlige bygninger (5000-5999), registreres i tillegg i en egen husliste. Kartet tegnes opp på tegneflaten med farger avhengig av tema-kode.

Når brukeren klikker på et sted på tegneflaten med musen, sjekkes om punktet som ble trykket på befinner seg inni et hus.[20] Hvis så er tilfelle markeres huset ved at det tilsvarende polygonet fylles med et ekslusiv-eller fyll. Fargene blir på denne måten invertert i polygonet. Dette innebærer at polygonet (og det som måtte befinne seg inni) får tilbake sine respektive farger neste gang samme fylloperasjonen utføres.

---

[19] Uvisst av hvilken grunn fungerer ikke systemet med MICROSOFT INTERNET EXPLORER.

[20] Algoritmer for å avgjøre om et punkt befinner seg inni et polygon eller ikke finnes i f.eks [Nordbeck & Rystedt 67], [Brockschmidt 96]. Kort går de ut på å trekke en tenkt linje fra punktet og gjerne langs en av koordinataksene. Hvis denne linjen skjærer noen av linjene i polygonet et *odde* antall ganger, f.eks 1, er punktet inni. En slik funksjon finnes allerede i Javas standardbibliotek og det er den som brukes i dette prosjektet. Imidlertid vil det være mange linjer å sjekke mot hvis det er mange hus på tegneflaten. Det kan gå raskere hvis punktet først kontrolleres mot et rektangel som omslutter polygonet [Nordbeck & Rystedt 67]. Slike rektangler er allerede beregnet i R-treet i tjeneren, så de kunne også tilbys av karttjeneren. Det får bli neste gang.

Begrensninger

Siden dette er en tidlig prototyp, finnes en del begrensninger og problemer som ikke er behandliet i særlig utførlig grad.

Begrensninger i tjeneren

Implementasjonen slik den foreligger bygger opp en privat objektstruktur av all tilgjengelig SOSI-informasjon, uten å bruke noen underliggende database eller filhåndtering. Dette medfører at programmet kan ta stor plass i nodens arbeidshukommelse, og bruke lang tid på å initialisere seg selv, særlig fordi den skal fylle opp og balansere R-strukturen under veis. Siden programmet er realisert i C++, er det sannsynligvis ikke like driftsikkert som om det var skrevet i Java, og hvis tjeneren krasjer, må den begynne om igjen og lese inn data på nytt. På den annen side vil bruk av virtuell hukommelse[21] i praksis fungere som en objektdatabase. Tjeneren kunne f.eks oppdateres ved at den leser inn data fra SOSI-filer hver natt, og på den måten ikke belaster etatens databaser mer enn nødvendig. Dette fordrer at tjeneren er tilstrekkelig driftssikker.

Begrensninger i klienten

Klienten får alle data fra tjener(ne) ved å henvende seg til dem som distribuerte objekter. Ideelt kunne man tenke seg at objektsystemet i et kart (Derivert fra SOSIs ER-struktur) kunne videreføres som distribuerte objekter, slik at *hvert objekt* i kartsystemet opptrådte som en tjener på samme måte som kart- og GAB-tjenerne gjør nå. Dette ville være mer «ren» objektorientert design. Det ville vært mer nærliggende å modellere problemområdet slik, fordi de objekter som finnes i en kartmasse gjerne representerer faktiske objekter i virkeligheten også. Hvert objekt, for eksempel bygninger, kunne så kontaktes og spørres om ting og oppdateres uavhengig av hvor de fysisk var lagret. Dette kan også utvides til et distribuert tegnesystem, hvor nye objekter kan opprettes etter behov, og andre kan redigeres eller slettes. Imidlertid blir dette en alt for finkornet struktur for dette prosjektet, og vil i praksis føre til stort kommunikasjons-overhead. Hvis hver objekttype i SOSI-standarden — og det er svært mange forskjellige! — fikk sitt eget grensesnitt ville det i praksis blant annet medføre like mange Java-klasser. Siden klasser lastes ned én for én til web-leseren, vil det blitt svært tregt. Derfor er bare de mest grunnleggende grafiske elementene fra SOSI-standarden modellert som abstrakte datatyper (`struct` i CORBA IDL), og karttjeneren forsyner dermed klienten med en enkel samling passive dataelementer av overkommelig mangfoldighet.

[21] Dvs at sekundærminne, så som platelager, benyttes som utvided hukommelse. Deler av minnet lagres og hentes (såk. swapping) frem fra platelageret ved behov. Ved søk i R-treet vil det derfor være en stor fordel hvis nodene i treet har samme størrelse som sidestørrelsen som brukes ved swapping, slik at swappingen reduseres så mye som mulig.

Mangler og videre arbeid

Ting som ikke fungerer helt som de burde:

- For øyeblikket er det ikke samsvar mellom målsetokken som oppgis i målestokkfeltet og den reelle målestokken.
- Metoden for å finne et passelig nabolag til et utpekt hus er laget rimelig enkel. Her kunne man utvide med å lage mer avanserte metoder for å sikre seg å få med alle nabohusene uansett hvor store de er etc. .
- Skjermoppdateringen og kontroll av applet'ens livssyklus er ikke konsistent
- Java applets kan ikke kjøres på WINDOWS 3.X plattformer, så en stor del potensielle brukere er utelukket. Dette er imidlertid et vanlig problem, og en årsak til at Java ofte ikke brukes i prosjekter hvor brukerne er ukjente.

Erfaringer fra utvikling av prototypen

En del av tiden i det året jeg har brukt på denne hovedfagsoppgaven har gått til implementasjon av prototypen slik den er presentert ovenfor. Dette arbeidet har vært spredt noe utover i tid, og totalt har det gått med omtrent to månedsverk. Nøkkelpoeng i evalueringen av erfaringene fra dette delprosjektet er:

- Fordeling av ressurser mellom arbeid som er relatert til datakommunikasjon og arbeid med selve applikasjonslogikken.
- Mengde arbeid som resultat av distribusjon.
- Hvor lett kan spesifikasjoner forandres, og hvor lett kan de implementeres?
- Hvordan påvirker distribusjonen prosjektorganisering?
- Erfaringer med heterogene plattformer
- Erfaringer med Java og C++

I det følgende vil jeg vurdere hver av disse.

Ressursfordeling

Slik jeg ser det, er den viktigste gevinstene ved å bruke distribuert objektteknologi at distribusjonen blir så transparent som mulig i visse henseende. Dette vil ideelt sett føre til at det trengs mindre ressurser til å modellere og implementere mekanismer og protokoller for datakommunikasjon. Ved å bruke et standard kommunikasjonsparadigme, som f.eks CORBA eller DCOM/ActiveX, vil mange detaljer knyttet til utviklingen av kommuniasjonsdetaljer være behandlet allerede. Ressursene som frigjøres kan heller brukes på å utvikle *funksjonaliteten* i systemet.

Min erfaring med CORBA og Orbix/OrbixWeb meglerteknologi fra denne oppgaven — og forøvrig også fra kurset DSS fra tidligere — er at det er en relativt høy terskel for å komme i gang og få ting til å fungere, men når det først gjør det, er miljøet rimelig robust. Jeg har tatt i bruk denne teknologien i et forholdsvis «fiendtlig» miljø, iallefall rent sikkerehetsmessig. Klienten i systemet autentiseres ikke på noen måte, siden den kan eksekveres på en hvilken som helst maskin i verden tilkoblet Internett.

Totalt sett vil jeg anslå arbeidet med å få kommunikasjonen til å virke til noen dagers arbeid, og det innebærer å få både klientsiden (som er skrevet i Java) og tjenersiden (skrevet i C/C++) til å spille sammen. Den resterende tiden er gått med til programmering av SOSI-parseren og R-tre databasen som i all hovedsak er hva tjeneren består av. Hvis jeg skulle arbeide videre med prosjektet, anslår jeg at det er lite behov for å løse flere kommunikasjonsproblemer, annet enn detaljer i Orbix omgivelsene jeg ennå ikke har brydd meg med. Dem er det jo en del av.

Distribusjon og arbeidsmengde

Det er ikke til å komme fra at distribusjonsapsektet i dette prosjektet innfører en viss kompleksitet. Verdt å merke seg er at det ikke uten videre går an å splitte opp applikasjonsområdet slik som antydet i Figur 5.13 på side 84, og så gå hver til sitt for så å implementere hver sin del og koble det hele sammen etterpå. Min erfaring — og spesielt fra arbeidsdeling i forbindelse med oppgaven i DSS-kurset høsten 1996, der vi bestemte oss for å starte med å skrive klienten og tjeneren hver for oss — er at man trenger testklienter og -tjenere for å se om komponentene implementerer IDL-spesifikasjonen på en tilfredsstillende måte. Selvsagt er ikke disse testprogrammene så omfattende som deres virkelige stedfortredere, men det går tross alt noe arbeid med til å programmere disse også. Konklusjonen er derfor at den totale arbeidsmengden går noe opp, også hvis de forskjellige deler av systemet utvikles ett sted.

Siden jeg i denne versjonen av det aktuelle distribuerte systemet har hatt kontroll med alle deler, har klienten gjerne blitt brukt til å teste tjeneren og omvendt. Likevel var det i tidligere faser av utviklingen nødvendig å lage noen testprogrammer.

Forandrbarhet i systemet

Et viktig poeng i «moderne» systemutvikling er mulighet til hurtig å kunne forandre et edb-system for å oppfylle nye krav fra omgivelsene [Taylor 95]. Dette kravet må også kunne oppfylles av et system basert på distribuert objektteknologi. Siden grensesnittene i systemet er spesifisert i et eget språk vil det være forholdsvis lett å forandre disse. Spesielt viser det seg at det er trivielt å legge til nye grensensnitt og datatyper, siden dette i liten grad påvirker det som allerede er implementert.

Slik jeg ser det, vil forandringer opptre i to «soner»:

1. Inni et distribuert objekt
2. Mellom distribuerte objekter

En større forandringsprosess vil utvilsomt kunne påvirke begge disse sfærene, men effekten på ressursbruk vil være mest interessant å se på som følge av forandringer som har størst innflytelse på forhold mellom objektene, altså i sone 2 ovenfor. Dermed ikke sagt at interne forandringer, så som forbedringer og optimaliseringer inni et objekt eller en tjener, er uinteressante i et ressursbruk- og prosjektstyrings-perspektiv, men disse sidene representerer ikke noe nytt i og med innføring av distribuert objektteknologi. Derimot er det interessant å se hvordan forandinger i f.eks IDL-grensesnittene mellom CORBA-baserte objekter påvirker andre deler av prosjektet.

Mine erfaringer med dette problemet er at forandinger er forholdsvis lett å innføre underveis. I løsningsforslaget som Oddvar Kolset og jeg kom frem til i forbindelse med DSS-kurset nevnt tidligere var både kartdata og eiendomsinformasjon tenk ivaretatt av samme kapsel for enkelhets skyld. I den versjonen som foreligger her er de splittet opp i to deler, mer i tråd med virkeligheten på Plan- og bygningsetaten. Dette vil si at de to tjenerfunksjonene først delte grensesnitt i IDL-spesifikasjonen. Dette ble på et tidspunkt splittet opp i to forskjellige grensesnitt (indikert i Figur 6.7 på side 102) nettopp for å se hva dette førte til av merarbeid. Det totale arbeidet med å skille ut kode og lage et nytt tjenerprogram vil jeg anslå til omtrent tre timers arbeid. I Java-klienten begrenset arbeidet med dette seg i hovedsak til å deklarere en ny instans (`gabserver`) av en ny klasse (`_gabServerRef`) som ble generert med grensesnittkompilatoren. Ytterligere oppsplittinger vil formodentlig gå enda raskere.

Hvis nå kommunikasjonen i dette prosjektet hadde vært implementert med f.eks sockets/TLI (se side 56), ville jeg måtte lage en helt ny tjener, definere semantikken i datapakkene og laget kode som tar seg av alt dette. Jeg vil derfor anta at en slik løsning vil medføre adskillig mer arbeid, og det vil også innføre større usikkerhet med hensyn på planlegging og ressursbruk.

Heterogene plattformer

Et annet viktig poeng som prototypen er ment å illustrere er distribuerte systemer over heterogene plattformer. Det er å forvente at man i fremtidige føderasjoner må regne med at det ikke er mulig å hindre at forskjellige myndigheter og beslutningstakere inntar ulik policy hva angår valg av plattform, operativsystem, nettverk, applikasjonsportefølje etc. Applikasjoner som skal fungere på tvers av domenegrenser — som jo ikke er utenkelig med nye organisasjonsformer og nye driftsprosesser (se Figur 5.14 på side 85) — må kunne forholde seg til dette. Ett skritt i en slik nøytraliteteretning er å sørge for at de deler av edb-systemet som skal distribueres og brukes i uforutsigelige

omgivelser, dvs klientene, er plattformnøytrale. Én slik nøytral plattform er Java. Klienten i prototypen er som nevnt tidligere implementert som en Java «applet». Dette innebærer at den i prinsippet kan eksekveres i alle web-lesere som kan kjøre Java, eksempelvis NETSCAPE NAVIGATOR/COMMUCATOR eller MICROSOFT INTERNET EXPLORER. Dette medfører at programmet kan kjøres på flere UNIX-plattformer, APPLE MACINTOSH og WINDOWS95/NT.[22] Etterhvert vil også dedikerte nettverksdatamaskiner (NC'er — Network Computer) gjøre seg gjeldene, også maskiner som er mer optimalisert for Java.[23]

Java, C og C++

> *«Java: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language.»*
>
> *Fra «The Java language: A white Paper», sitert i [Flanagan 96][24]*

Et annet viktig poeng med gjennomføringen av dette prototypprosjektet har vært å få litt erfaring med Java og C/C++ — spesielt samspill mellom disse. Java er i stor grad inspirert av C++ som igjen er inspirert av Simula og andre objektorienterte språk. I Java er imidlertid endel uheldige konstruksjoner fjernet, slik at språket fremstår som «renere» og som et «vakkert» programmeringsspråk, i den grad ordene er dekkende for slike språk i det hele tatt. En sammenligning mellom Java og C/C++ finnes i [ibid]:15–47.

Java virker gjennomtenkt, og *er* et ganske vakkert språk. Mange av de fallgruber programmerere kan falle i med C/C++ er fjernet, som f.eks minnelekkasjer og rene pekere. I C/C++ er det lett å forårsake krasj ved å manipulere hukommelse på adresser som ikke burde refereres til. I C++ er `struct` og `class` deklarasjoner egentlig stort sett det samme, bortsett fra at klasser gir mer kontroll og innkapsling. I Java finnes bare `class`, slik som i Simula.

Imidlertid er det klart at ytelsen til Java-applikasjoner er et stort problem sammenlignet med applikasjoner skrevet i C/C++. Spesielt er dette merkbart på ikke helt nye maskiner. Under eksekvering vil en Java-applikasjon også bruke mye internminne. Siden Java er «dynamisk», vil en applikasjon, f.eks en applet, som består av mange klasser måtte laste ned hver eneste klasse fra tjeneren før det linkes og

---

[22] Men altså ikke iWindows 3.x. (se side 110)

[23] SUN har allerede lansert flere Java-prosessorbrikker med varierende ytelse. På den annen side kan det virke som Java's «plattformnøytralitet» viser seg å være et forsøk på å innføre en ny prosessorstandard med verdensherredømme, analogt med Intel's x86-hegemoni. Imidlertid virker konseptet med rene Java-maskiner mer gjennomtenkt enn x86/Windows, spesielt med hensyn på sikkerhet.

[24] Java ble tidligere også presentert som «Buzzword Compliant» på en av SUN's websider.

eksekveres.[25] Dette kan ta merkbart lang tid, over modem kan det gå flere minutter før programmet kan starte. For øyeblikket (midten av 1997) er 28800 baud modem dominerende iallefall i det private Internett-markedet. Slik som kartoppslag- og eiendomsprogrammet fremstår i dag, er det derfor lite brukervennlig for eksterne, private brukere av tjenesten slik den i utgangspunktet er tenkt. Under utviklingen har jeg testet programmet i NETSCAPE NAVIGATOR som har eksekvert på vekselvis SILICON GRAPHICS arbeidstasjoner og SUN ULTRASPARC nettverkstjenere, som gir et helt annet ytelsesnivå enn mer vanlige hjemmedatamaskiner. Imidlertid tror jeg at interne brukere av lignende systemer vil kunne oppnå brukbar ytelse. Dessuten tror jeg at maskinvareutviklingen med den progresjon den har i dag snart vil gjøre det mer attraktivt å ta i bruk større Java-applikasjoner enn det er praktisk mulig å gjøre i dag.

Sammendrag

Dette kapitlet har presentert en prototyp for distribuert kartoppslag, som kombinerer data fra Plan- og bygningsetatens kartdatabaser med register for eiendommer. Formålet med eksempelet er å prøve ut litt av det metodeverk som er presentert tidligere i oppgaven, nemlig DISGIS General Method som er inspirert av RM-ODP, OOram og SIMOD (En forløper for DISGIS). I tillegg er konkrete implementasjonsteknologier valgt, nemlig Java, C++ og CORBA med ORBIX som mellomvareprodukt.

Evalueringen underveis har vist at det er vanskelig å få prøvd ut metoder for utvikling av distribuerte systemer på et så lite eksempel, men eksemplet må være lite for å passe inn størrelsesmessig i en hovedoppgave. Helt klart er det imidlertid at kombinasjonen av RM-ODPs fem perspektiver og metodeverk som er sentrert rundt samarbeid mellom objekter eller roller (i dette tilfellet DISGIS), gir muligheter for fleksible modeller, hvor selve distribusjonen kan innføres sent i designfasen og senere endres etter behov, for eksempel som resultat av ny organisasjonsstruktur eller endrede driftsprosesser. Jeg har også observert at et distribuert system implementert med den teknologien og under de forutsentinger som er presentert ovenfor er forholdsvis lett å forandre, og dette er et meget viktig poeng hvis edb-systemene skal kunne holdes à jour med utviklingen i en organisasjon forøvrig.

[25] Flere klasser kan nå samles opp i én zip-fil og lastes ned på en mer effektiv måte, men det stiller litt krav til nettleseren.

7 Oppsummering

> *«The Distributed Enterprise is a new vision for business and the information technology that supports it. (…) business professionals are seeking systems that let them look at and work with their business in new ways that are defined by changing business opportunities instead of pre-conceived systems design.»*
>
> *[Shelton 94a]*

Hensikten med de foregående kapitlene har vært å belyse problemstillingen som ble definert på side 3 fra forskjellige ståsteder, både tekniske og ikke-tekniske. Støtte for distribusjon i vanlige systemutviklingsmetoder har vist seg som vanskelig å finne, men dette er utvilsom noe jeg tror vil bedre seg i tiden som kommer. Det kan imidlertid være på sin plass og spørre seg selv om systemutvikling i distribuerte miljøer egentlig trenger å være forskjellig fra utvikling i ikke-distribuerte sammenhenger. Mange ting tyder at dette er tilfelle. Men med de paradigmene som er presentert tidligere for realisering av distribuerte objektsystemer kan mye av de tilgjengelige ressursene i systemutviklingsprosjekter egnes til analyse av problemområdet og design av applikasjonslogikken istedenfor tekniske kommunikasjonsdetaljer, og slik fjerne kompliserende hindringer og usikkerhetsmomenter.

Metoder for distribuerte systemer

Noe av kjernen i denne hovedoppgaven er hvordan distribusjonen fremkommer. Hva er det som bestemmer hvordan distribusjonen skal bli? Det later til at det er tre faktorer som bidrar til dette:

- Eksplisitt distribusjon i organisasjonen, gjerne *etter* eventuelle organisasjonsmessige endringer, så som BPR.

- Bruk og deling av forretningsobjekter i f.eks en rollemodell. Dette kan bestemmes ved å bruke metoder inspirert av OOram/DISGIS.
- Fysisk distribusjon av eksisterende systemkomponenter. Hvis de eksisterende deler brukes som de er, ved f.eks screen-scrapere, terminalemulatorere eller andre grensesnitt, dikterer de i alle fall hvor de selv befinner seg, nemlig på den maskinen de alltid har vært.

Etter som forretningsprosessene endrer seg, må edb-systemet også kunne tilpasses disse endringene på en passelig måte. Hvis systemet er konstruert for endring og vedlikehold, f.eks ved å definere alle grensesnitt i CORBA, vil det være lettere å forandre og flytte objekter rundt som det måtte passe, og også forandre de forskjellige verktøy som benytter seg av disse objektene. Ved å beholde samme objektorienterte paradigme hele veien, og slik oppnå mest mulig sømløse faseoverganger i edb-systemets levetid, vil det være lettere å tilpasse det til stadig skiftende forretningsvilkår.

Modellering og bruk av eksisterende systemer

Likevel trengs en måte å modellere de eksisterende systemer sammen med de nye komponenter, og her gir i allefall [Taylor 95] noen retningslinjer til identifisering av hvilke komponenter og hvordan de kan behandles. I en ny objektmodell av driftsprosessene identifiseres hvordan de gamle edb-systemer kan brukes; hele eller i deler. Hvis de kan brukes hele kan de kapsles inn på en passelig måte, f.eks med terminalemulatorer, slik som beskrevet i Kapittel 2. Siden kan de erstattes, uten at dette trenger påvirke hvordan de nye systemer bruker dem. Hvis kun fragmenter av de eksisterende systemer er brukbare i den nye driftsprosessmodellen, kan disse skilles ut med kodeanalyse eller -slakt. Disse bitene danner så innmaten i den nye objektmodellen, som kan implementeres som et distribuert objektsystem, over en egnet infrastruktur. Disse kan så modelleres som grovkornete objekter, som tilbyr et sett grensesnitt utad.

Denne strategien passer bra med grensesnittparadigmet i CORBA eller DCOM, men i OOram er det mer usikkert hvordan dette bør håndteres. Siden grensesnittene her modelleres som tilhørende den aktive part, dvs klientens, kan dette by på problemer i modelleringen. Grensesnittene til eksisterende komponenter er ganske fastlåst, og bremser derfor kreativiteten som det innbys til, ved at klientroller kan «komme med ønsker» om hvilke meldinger tjenerroller skal godta (se side 84) som en innkapslet eksisterende systemkomponent ikke uten vider kan etterkomme. Grensensnittene fra de eksisterende systemkomponenter kan derfor bli spredt utover hele OOram-modellen, til

alle de roller som bruker dem. På den annen side kan denne måten å modellere samarbeide mellom objektene bidra til å identifisere hvilke deler av det eksisterende som kan beholdes som det er, og hva som må bygges om eller nyutvikles.

Uansett hvordan de eksisterende systemkomponenter tas i bruk, bør utviklerne se på hvordan det eksisterende systemet brukes i praksis og identifisere konsekvensene av av eventuelle lokale tilpasninger som er gjort, vaner og implisitt kunnskap. Dette bør gjøres ved revers- og forovermodellering i samarbeid med domeneekspertene. Ved gjenbruk av kode, er det utvilsomt en fordel at CORBA i utgangspunktet er språknøytralt. Dette letter tilkpasningen av grensesnittspesifikasjonen til de eksisterende kodefragmentene.

Veier til gjenbruk i systemutvikling

> *«(…) it took (…) under 14 developer years to upgrade WordPerfect from version 3 to version 4. However, it took 250 developer years to move the same product from version 5 to version 6.»*
>
> *[Orfali et al 96b]:29*

Det er som regel lettere å få aksept for systemutvikling som legger til rette for vedlikehold, i motsetning til systemutvikling for fremtidig gjenbruk hvor inntjeningsperspektivet kan være langsiktig [Haythorn 94]. Et brukbart kompromiss kan være utvikling og vedlikehold ved hjelp av gjenbruk av eksisterende programvare. Slik gjenbruk kan skje enten på modulnivå, slik at store deler av det eksisterende brukes om igjen uten rekompilering, linking etc — eller selve programteksten kan saumfares og parteres, slik som beskrevet i Kapittel 2. Den siste måten gir størst mulighet for senere gjenbruk, hvis de delene som identifiseres i eksisterende kode er generelle nok. Uansett hvilken granularitet som velges, finnes det flere måter å legge til rette for fremtidig gjenbruk, blant dem klassebiblioteker, komponenter og rammeverk. Disse er naturligvis svært beslektet, og den ene kan gjerne realiseres gjennom den annen. Imidlertid utgjøres graden gjenbrukbarhet i en konkret verktøykasse mer av designen enn hvilken kategori av de tre ovenfor verktøykassen tilhører.

Komponentbasert systemutvikling

> *«(…) one way to increase productivity (…) is to produce less software (…) while achieving the same functionality. This can be done by building the system out of reusable software components (…)»*
>
> *[Biggerstaff & Perlis 89]:296*

Én tilnærming til gjenbruk er komponentbasert systemutvikling. Akkurat som det er mulig å kjøpe komponenter rett fra hyllene i en

elektronikkbutikk for å bygge en datamaskin, tenker man seg at det skulle vært mulig å kjøpe programvarekomponenter, som så kan kobles sammen for å oppnå ny funksjonalitet. Det ligger i dette at komponentene, som kan sees på som svarte bokser, altså skal kunne kobles sammen på måter som de som lagde dem kanskje ikke hadde tenkt på på forhånd. I så måte er det nærliggende også å kalle utskilte delkomponenter av eksisterende systemer som brukes om igjen på nye måter for nettopp — komponenter.

Komponenter vil, i følge [Orfali et al 96b], kunne benyttes av utviklere med forskjellige forutsetninger og behov. Ressursbrukere og driftspersonell kan tilpasse applikasjonsporteføljen i sin organisasjon, utviklere bygger systemer fra grunnen av til oppdragsgivere kan sette sammen komponenter, og markedsløsninger kan spesialbygges, f.eks «tekstbehandling for advokater». Komponentbasert systemutvikling spås en stor fremtid innen systemutvikling, men dette gjenstår selvsagt å se.

Rammeverk av forretningobjekter

I motsetning til komponenter, som skal være så generelle som mulig og kunne settes sammen på nye uforusigbare måter, er rammeverk større konstruksjoner med klare predefinerte formål. Rammeverk følger gjerne «Hollywood-prinsippet». Når berømte filmprodusenter får en telefon fra en håpefull filmmanuskriptforfatter eller skuespiller som vil bli berømt i filmverdenen, avslutter produsenten samtalen med: *«Don't call us! We call you!»* Det er altså rammeverket som tar initiativet til handling. Det er rammeverket som styrer tingenes tilstand, og kaller brukerens prosedyrer, funksjoner eller objekter etter behov. Systemutviklerens oppgave er å bestemme hvordan det skal reageres på henvendelser fra rammeverket, og spesialtilpasse rammeverkets oppførsel. Dette representerer således en annen tilnærming til gjenbruk enn både klassebiblioteker og API'er. I sistnevnte tilbys funksjoner og tjenester som utvikleren selv må kalle i riktig rekkefølge.[1] Istedenfor er det systemutviklerens oppgave å skreddersy komponenter som så plasseres under kontroll av et rammeverk.

Eksempler på bruk av for rammeverk for brukergrenesnittprogrammeringer er BORLAND C++, JAVA WORKSHOP og andre integrerte programmeringsmiljøer. Om ikke miljøet tilbyr ferdige rammeverk, så kan det gjerne generere rammeverket ut fra visse kriterier, og la utvikleren ved hjelp av redefinering av virtuelle funksjoner fange opp hendelser fra f.eks vindussystemet, eksempelvis musklikk.

Tilsvarende kan man tenke seg rammeverk for andre applikasjonsaspekter, f.eks rammeverk av forretningsobjekter. Et eksempel (fra [Orfali et al 96b]): en pakke forretningsobjekter som modellerer *biler* kan brukes til bilutleie-, salg-, og verkstedsystemer, i kombinasjon

[1] Slik som sockets og TLI (se side 56).

med rammeverk for behandling av kunder, kontrakter, fakturaer etc. Det hele kan «limes sammen» med en objektinfrastruktur, som f.eks er CORBA-basert.

Veien videre

I løpet av den tiden jeg har brukt på denne oppgaven, har jeg funnet noen tema som det kunne vært interessant å følge opp, og som f.eks kunne vært tips til tema for fremtidige hovedoppgaver i systemarbeid:

- OO-modellering: forhold mellom tilbud og etterspørsel, som identifisert i avsnitt om OOram's grensesnittdefinisjoner på side 84.
- Utforske mulighet for å standardisere rammeverk av forretningsobjekter for offentlig forvaltning. En kan undre seg over hvorfor offentlig forvaltning generelt ikke er tatt med under spesifisering av CORBAs vertikale markedsfasisliteter. Kan dette komme av at de faller inn under andre bransjer som får sine egne rammeverk, så som *Finance*, *Law*, *Medical* etc? Det er nærliggende å forestille seg en standardisering i kontrollintegrasjonsdimensjonen i brødristermodellen (se Figur 3.5 på side 40), samt deler av dataintegrasjonen, i tillegg til presentasjonsintegrasjonen. Sistnevnte er også et krav i KOARK [KOARK 95]. Hvis det stilles standardiserte krav til disse tre dimensjonene vil dette gi klare rammebetingelser for produksjon av prosessintegrasjonskomponenter — dvs forskjellige verktøy, f.eks saksbehandlingssystemer — som kan passes inn i mellom disse tre. Den slående likheten mellom brødristere og vanlige PC'er med spor for utvidelseskort med standard lengder og høyder illustrerer denne tanken.[1]
- Kunne norske standarder for arkiv- og saksbehanlingssystemer, som beskrevet i f.eks KOARK, vært tjent med mer spesifike krav til distribusjonsmuligheter og objektmodeller og bruk av f.eks CORBA som infrastruktur?
- Hvordan påvirker infrastrukturer for distribusjon grad av kreativitet i systemutvikling? Med stadig flere tjenester tilgjengelig via Internett burde muligheter for å nyttiggjøre seg disse på nye og innovative måter øke.

Jeg tror det er lurt å følge spesielt godt med på disse feltene:

- OPEN/MeNtOr-prosjektet
- Hvem vinner av CORBA/DCE/DCOM/NetscapeONE/…? Vil disse kunne leve og fungere sammen i fremtiden?

[1] Faktisk så hadde noen tidlige PC-modeller et lokk rett over utvidelsessporene, så det skulle være enkelt å sette inn og ta ut kretskort.

- Hvordan blir det med Network Computere (NC) og Javamaskiner? Vil de kunne true PC-hegemoniet?

Med de muligheter som etterhvert gjør seg gjeldende for sømløs objektorientert systemutvikling og integrasjon ikke bare i brukergrensesnitt, men også med eksisterende systemer, tror jeg det er mulig å realisere prosjekter med større sjanse for suksess enn tidligere, fordi flere usikkerhetsfaktorer synes å være ryddet av veien. Mer ressurser kan brukes på selve applikasjonen og støtte for driftsprosessene, og mindre på tekniske detaljer på lavere nivåer. Med mer systematisk gjenbruk av distribuerte rammeverk av forretningsobjekter vil edb-systemer kunne utvikles raskere og forhåpentlig mer i samsvar med budsjettene. Dette gjenstår selvsagt å se.

A Referanser

[Andersen *et al* 95] Niels Erik Andersen, Finn Kensing, Monika Lassen, Jette Lundin, Lars Mathiassen, Andreas Munk-Madsen, Pål Sørgaard: *«Profesionel Systemudvikling; Erfaringer, muligheder og handling»*, Teknisk Forlag AS, Danmark 1995

[Arnesen 94] Svein A. Arnesen: «Hvordan kan gamle og nye applikasjoner forenes? Kan klient/tjener være en løsning?», *Klient/tjener arkitektur og distribuerte systemer*, Geilo 18.–19. april 1994

[Berre 96] Arne-Jørgen Berre: *«ISO/TC 211 WG4 N032: Services»*, Arbeidsdokument i forbindelse med ISO/TC 211 WG4, Geospatial Services, `http://www.statkart.no/isotc211/wg4/wg-n032.html`

[Berre et al 96] Arne-Jørgen Berre, Jan Øyvind Aagedal, António Rita Silva: *«SIMOD - An ODP-extended Role-Modeling Methodoly for Distributed Objects»*, SINTEF 1996

[Biffl *et al* 96] Stefan Biffl, Thomas Grechenig, Stephan Oberpfalzer: «Engineering an 'Open' client/server-platform for distributed austrian alpine road-pricing system in 240 days — Case study and experience report», *Proceedings of the 18th International Conference on Software Engineering, March 25-29, 1996, Berlin, Germany*

[Biggerstaff & Perlis 89] Ted J. Biggerstaff, Alan J. Perlis (ed.): *«Software Reusability Volume I: Concepts and Models»*, ACM Press, New York 1989

[Birrell & Nelson 83] Andrew D. Birrell, Bruce Jay Nelson: *«Implementing remote procedure calls»*, XEROX PARC CSL-83-7, December 1983

[Bjerknes et al 91] Gro Bjerknes, Tone Bratteteig, Kristin Braa, Larl Kautz, Jens Kaasbøll, Leikny Øgrim: *«Project FIRE: Functional Integration through REdesign»*, FIRE Report No. 1, Dep. of Informatics, University of Oslo 1991

[Booch 94] Grady Booch: *«Object-oriented analysis and design With applications (second edition)»*, The Benjamin/Cummings Publishing Company, inc. California 1994

[Borges 73] Jorge Luis Borges: *«Other Inquisitions (1937–1952)»*, Souvenir Press (Educational & Academic) Ltd, London 1973

[Brando 94] Thomas J. Brando: *«DOMIS Implementation of CTAPS Functionality Using Orbix»*, MITRE Document MP 94B-287, The MITRE Corporation, Bedford, Mass. USA, December 1994

[Brando 95] Thomas J. Brando: *«Comparing DCE and CORBA»*, MITRE Document MP 95B-93, MITRE Corporation, Bedford Mass. USA, March 1995

[Brown & Kindel 96] Nat Brown, Charlie Kindel: *«Distributed Component Object Model Protocol — DCOM/1.0»*, Microsoft Corporation, November 1996

[Brockschmidt 96] Kraig Brockschmidt: *«What OLE is really about»*, `http://www.microsoft.com/oledev/olecom/aboutole.htm`

[Brown 93] Lesley Brown (ed): *«The new shorter oxford english dictionary»*, Volume 1, Clarendon Press, Oxford 1993

[Braa et al 92] Kristin Braa, Tone Bratteteig, Jens Kaasbøll, Leikny Øgrim: «ENtry to the FIRE project», FIRE Report No. 2, Department of Informatics, University of Oslo, 1992

[Bråten 81] Stein Bråten: *«Modeller av menneske og samfunn — Bro mellom teori og erfaring fra sosiologi og sosialpsykologi»*, Universitetsforlaget, Oslo 1981

[Bråten 92] Stein Bråten: *«Dialogens vilkår i datasamfunnet — Essays om modellmonopol og meningshorisont i organisasjons- og informasjonssammenheng»*, Universitetsforlaget, Oslo 1992

[Burton et al 84] F. W. Burton, V. J. Kollias, J. G. Kollias: «Consistency in Point-in-Polygon tests», *The Computer Journal*, vol. 27 no. 4, 1984

[Carlson *et al* 96] Andrew C. Carlson, William R. Brook, Christopher L. F. Haynes: «Experiences with distributed objects», *AT&T Technical Journal*, March/April 1996

[Chauvet 95] Jean-Marie Chauvet: «The culture of object development», *Objects in Europe* July–August 1995 (vol. 2, iss. 4)

[COM 95] *«The Component Object Model Specification — Draft version 0.9»*, Microsoft Corporation and Digital Equipment Corporation Oct. 1995 `ftp://ftp.microsoft.com/developr/drg/Ole-info/COMSpecification/COM_Spec_PS.ZIP`

[Comer & Stevens 93] Douglas E. Comer, David L. Stevens: *«Internetworking with TCP/IP Vol. III: Client-server programming and applications — BSD Socket version»*, Prentice-Hall, New Jersey 1993

[Cox 90] Brad J. Cox: «There *is* a silver bullet», *BYTE* October 1990

[Cook & Daniels 94a] Steve Cook, John Daniels: «Object-oriented methods and the great object myth», *Objects in Europe* Autumn 1994 (vol 1, iss 4)

[Cook & Daniels 94b] Steve Cook, John Daniels: *«Designing object systems: Object-oriented modelling with Syntropy»*, Prentice Hall, New York 1994

[Crowcroft 96] Jon Crowcroft: «*Open distributed systems*», UCL Press Limited, London , England 1996

[Daniels 96] John Daniels: «ORBs at your service? Selecting an architecture for distributed systems», *Object expert* 1996 vol. 1 iss. 4

[Davenport 93] Thomas H. Davenport: «*Process Innovation — Reengineering work through information technology*», Harvard Businesss School Press, Mass. USA 1993

[Dietrich *et al* 89] Walter C. Dietrich, Lee R. Nackman, Franklin Gracer: «Saving a legacy with objects», *Proceedings of OOPSLA '89, Object-oriented programming: Systems, Languages and Applications, New Orleans 1989, SIGPLAN Notices,* vol. 24 iss. 10

[McDermid 91] John McDermid (ed): «*Software Engineer's Reference Book*», Butterworth-Heinemann Ltd., Oxford, England 1991

[Eckerson 95] Wayne Eckerson: «Searching for the middle ground», *Business Communications Review*, september 1995

[Elwahidi & Merlo 95] Abdu R. Elwahidi, Ettore Merlo: «Generating user interfaces from specifications produced by a reverse engineering process», *Preceedings of the second working conference on reverse engineering, Toronto Juli 1995*, IEEE Computer Siciety Press 1995

[Dix et al 93] Alan Dix, Janet Finlay, Gregory Abowd, Russel Beale: «*Human-computer interaction*», Prentice-Hall International (UK) Limited, Hertfordshire, England 1993

[Dock 92] Patti Dock: «OO-methods — Research or Ready?», *Hotline on Object-oriented technology*, vol.3 no.9 1992

[Easterby-Smith et al 91] Mark Easterby-Smith, Richard Thorpe, Andy Lowe: «*Management Research — An introduction*», SAGE Publications, London 1991

[Eckerson 95] Wayne Eckerson: «Searching for the middle ground», *Business communications review*, september 1995

[ECMA 93] «*Reference Model for Frameworks of Software Engineering Environments, 3rd Edition*», ECMA Technical Report 55, Juni 1993 (Publisert i samarbeid med NIST, da som NIST Special Publication 500-211)

[Ellis & Stroustrup 90] Margaret A. Ellis, Bjarne Stroustrup: «*The Annotated C++ Reference Manual*», Addison-Wesely Publishing Company 1990

[Elmasri & Navathe 94] Ramez Elmasri, Shamkant B. Navathe: «*Fundamentals of database systems 2nd edition*», Benjamin/Cummings, California USA, 1994

[Flanagan 96] David Flanagan: «*Java in a Nutshell*», O'Reilly & Associates, Inc. Sebastopol, USA 1996

[Fowler 97] Martin Fowler: «*Analysis patterns: reusable object models*», Addison-Wesley, USA, 1997

[Frost 94] Steve Frost: «From SSADM to OOPS: is SSADM capaple of supporting the next generation of client/server systems?», *Objects in Europe*, Autumn 1994 vol. 1, iss. 4

[Gall *et al* 95] Harald Gall, René Klösh, Roland Mittermeir: «Object-oriented re-architecturing», *Proceedings of ESEC '95, 5th European Software Engineering Conference*, Sitges, Spania sept 1995

[Gamma *et al* 95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: «*Design Patterns - Elements of reusable Object-Oriented software*», Addison Wesley Publishing Company, Mass. USA 1995

[GEOLOK 95] Dokumentasjon til GEOLOK Prosjektet; SINTEF 1995

[Goldberg & Robson 83] Adele Goldberg, David Robson: *«Smalltalk-80 — The language and its implementation»*, Addison-Wesley 1983

[Goldkuhl 94] Göran Goldkuhl: *«Några problem vid datadriven strukturering av informationssystem»*, Research report nr LiTH-IDA-R-94-53, Universitetet och Tekniska högskolan i Linköping 1994

[Goldkuhl 96] Göran Goldkuhl: *«Från IRM och VBS til PAKS: Process-, aktivets-, och komponentbaserad systemstrukturering»*, Research report nr LiTH-IDA-R-96-08, Universitetet och Tekniska högskolan i Linköping 1996

[Goldkuhl *et al* 93] Göran Goldkuhl, Karin Petterson, Owen Eriksson: *«Hur studera realisering och konsekvenser av strategibaserade informationssystem»,* Research report nr LiTH-IDA-R-93-19, Universitetet och Tekniska högskolan i Linköping 1993

[Graham 94] Ian Graham: *«Object Oriented Methods»*, Addison-Wesley Publishing Company 1994

[Greenbaum *et al* 96] Joan Greenbaum, Jens Kaasbøll, Olse Smørdal og Leikny Øgrim: *«En langsiktig studie av Plan- og bygningsetatens overordnede behov for IT»*, forskningsrapport, Institutt for Informatikk, Universitetet i Oslo 1996

[Guttman 84] Antonin Guttman: «R-Trees: A dynamic index structure for spatial searching», *Proceedings of SIGMOD '84*, Boston, June 18–21 1984

[Halpin 96] Terry Halpin: «Business rules and Object-role modeling», *Database Programming & Design*, October 1996

[Halsall 95] Fred Halsall: *«Data communications, Computer network and Open systems»*, Fourth edition, Addison-Wesley USA 1995

[Halvorsen 97] Øyvind Halvorsen: *«Sikker objektmigrering i CORBA»*, Hovedfagsoppgave, Institutt for Informatikk, Oslo 1997

[Hammer & Champy 93] Michael Hammer, James Champy: *«Reengineering the corporation: a manifesto for business revolution»*, Harper Business, New York 1993

[Hammer 96] Michael Hammer: *«Beyond reengineering: How the process-centred organization is changing our work and our lives»*, Harper Collins New York, 1996

[Hannemyr 92] Gisle Hannemyr: *«Åpne systemer — teknologi, strategi og praksis»*, Universitetsforlaget, Oslo 1992

[Harmon & Morrisey 96] Paul Harmon, William Morrisey: *«The object technology casebook»*, John Wiley & Sons inc. New York 1996

[Haythorn 94] Wayne Haythorn: «What is object-oriented design?», *Journal of Object-Oriented Programming*, mars-april 1994

[Henderson-Sellers & Edwards 93] Brian Henderson-Sellers, Julian Edwards: «The fountain model for object-oriented system development», *Object Magazine* vol 3 no 2 1993 (finn denne!)

[Henderson-Sellers & Edwards 95] Brian Henderson-Sellers, Julian Edwards: *«Booktwo of object-oriented knowledge: The working Object»*, Prentice Hall, inc. 1994

[Henderson-Sellers & Graham 96] Brian Henderson-Sellers, Ian Graham: «OPEN: Toward method convergence?», *IEEE Computer*, april 1996

[Hetland 97] Thor Henning Hetland: «*MODS: A role-based Method for analysis and design of object-oriented disitributed systems*», Siv.Ing. diplomoppgave, NTNU, Januar 1997

[Holst 79] Anna Holst: *«An outline of a neighbourhood data bank — Scope, content, use»*, Tekniska Högskolan i Stockholm, Almquist & Wiksell International 1979

[Hoffner 96] Tommy Hoffner: *«Evaluation and comparison of Program slicing tools»* Technical Report LiTH-IDA-R-95-0, Universitetet och Tekniska högskolan i Linköping 1994

[Iden 95] Jon Iden: *«Business process reengineering in Norway: what do professional reengineering consultants say?»*, Reports in information science no. 38, Universitetet i Bergen 1995

[Illiaifar *et al* 95] Sam Illiaifar, Sree Nilakanta, G.M. Prabu: «Technology imperatives of BPR an their effect on Organizational Decision support», *Proceedings of the 28th Annual Hawaii international Conference on System Sciences 1995*, IEEE press 1995 (?)

[Jacobson 87] Ivar Jacobson: «Object oriented development in an industrial environment», *Proceedings og OOPSLA '87 — Object-oriented programming systems, languages and applications October 4-8, 1987, Orlando, Florida*, SIGPLAN Notices vol. 22 nr. 12, ACM Press New York 1987

[Jacobson & Lindström 91] Ivar Jacobson, Fredrik Lindström: «Re-engineering of old systems to an object-oriented architecture», *Proceedings og OOPSLA '91 Object-oriented programming: Systems, Languages and Applications, Phoenix 1991, SIGPLAN Notices*, Vol 26 iss 11

[Jacobson *et al* 92] Ivar Jacobson, Magnus Christerson, Patrick Jonsson, Gunnar Övergaard: «*Object-oriented software engineering: A use case driven approach»,* Addison-Wesley, 1992

[Jacobson et al 94] Ivar Jacobson, Maria Ericsson, Agneta Jacobson: *«The object advantage: business process reengineering with object technology»,* Addison-Wesley, 1994

[Jain & Purao 91] Hermant K. Jain, Sandeep Purao: «Distributed application development», *Information & Management* 1991 vol. 20 pp. 247–255

[Jeffery 96] Brian Jeffery: «The return of COBOL», *Software economics Letter* vol. 5 no. 6, 1996

[Kappel *et al* 94] G. Kappel, S. Preishuber, E. Pröll, S. Rausch-Schott, W. Retschitzegger, R. Wagner, G. Gierlinger: «COMan - Coexistence of object-oriented and relational technology», *Proceedings of 13th Entity Relationship Approach - ER '94; Business modelling and re-engineering*, Manchester, England 1994

[Killingberg 94] Audun Killingberg: «*Reverse engineering — Fata morgana eller realitet?*», Cand.Scient-avhandling i informatikk, Universitetet i Oslo 1994

[Kirkerud 89] Bjørn Kirkeud: *«Object-oriented programming with SIMULA»*, Addison-Wesley 1989

[Knudsen et al 83] Trygve Knudsen, Alf Sommerfelt, Harald Noreng: «*Norsk Riksmålsordbok*», Kunnskapsforlaget, Oslo 1983

[KOARK 95] «*KOARK – Kommunal standard for edb-baserte sak-/arkivsystemer*», Kommunenes Sentralforbund 1995

[Konstantas 96] D. Konstantas: «Migration of legacy applications to a CORBA platform: a case study», *Distributed platforms: Proceedings of the IFIP/IEEE International-Conference on Distributed Platforms Dresden 1996*, Chapman & Hall London 1996

[Kythe 96] Dave K. Kythe: «The promise of distrubuted business components», *AT&T Technical Journal*, March/April 1996

[Kaasbøll 96] Jens Kaasbøll: «*Aspects of object-oriented modelling; Concepts for analysis and guidelines for design*», doktorgradsavhandling, Research report no. 214, Institutt for Informatikk, Universitetet i Oslo april 1996

[Kaasbøll 96b] Jens Kaasbøll: «Many small improvements of computer systems adding up to negative effects», *konferansebidrag NOKOBIT '96 Norsk konferanse om Organisasjoners bruk av IT*, Kristiansand 20–21 juni 1996

[Kaasbøll 96c] Jens Kaasbøll: «Building information architecture», *konferansebidrag til VITS Höstkonferens*, Högskolan i Borås 19.–21. november 1996

[[Krogdahl 96] Stein Krogdahl: «*Kompendium til IN-115 våren 1997*», Kompendium nr. 65, Institutt for Informatikk, Universitetet i Oslo 1996

[Lalonde 94] Wilf Lalonde: «*Discvering Smalltalk*», The Benjamim/Cummings Publishing Company, California 1994

[Landrø & Wangensteen 86] Marit Ingebjørg Landrø (ed), Boye Wangensteen (ed): «*Bokmålsordboka — definisjons- og rettskrivningsordbok*», Universitetsforlaget AS 1986

[Levine et al 92] John R. Levine, Tony Mason, Doug Brown: «*lex & yacc*», 2nd edition, O'Reilly & Associates, Inc, USA 1992

[Loomis 94] Mary E. S. Loomis: «Hitting the relational wall», *Journal of Object-Oriented Programming*, januar 1994

[Low et al 96] G. C. Low, G. Rasmussen, B. Henderson-Sellers: «Incorporation of distributed computing concerns into object-oriented methodologies», *Journal of Object-Oriented Programming*, June 1996

[Madsen 93] Ole Lehrman Madsen, Birger Møller-Pedersen, Kristen Nygaard: «*Object-oriented programming in the BETA programming language*», ACM Press, Wokingham, England 1993

[Martin et al 91] Bruce E. Martin, Claus H. Pedersen, James Bedord-Roberts: «An object-based taxonomy of distributed computer systems», *IEEE Computer*, August 1991

[Mathiassen et al 93] Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, Jan Stage: «*Objektorienteret Analyse*», Marko, Aalborg 1993

[Mathiassen et al 95] Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, Jan Stage: «*Objektorienteret Design*», Marko, Aalborg 1995

[Mock 93] Marcus U. Mock: «DCE++: Distributing C++-objects using OSF DCE», *Proceedings og International DCE Workshop, Karlsruhe, Germany Oct. 1993* Springer Verlag 1993

[Moldeklev 90] Kjersti Moldeklev: *«TCP/IP or OSI: Evaluation of functionality and performance»*, Lecture TF F21/90, Teledirektoratets Forskningsavdeling 1990

[Mullins 94] Craig S. Mullins: «The Great Debate», *Byte* April 1994

[Narasimhan *et al* 94] Badri Narasimhan, Shamkant B. Navathe, Sundaresan Jayaraman: «On mapping ER and relational models into OO schemas», Entity-Relationship Approach - ER '93, *Proceedings of the 12th International Conference on the Entity-Relationship Approach*, Arlington, Texas, USA, December 1993.

[Ning *et al* 94] Jim Q. Ning, Andre Engberts, W. Kozaczynski: «Automated support for Legacy code understanding», *Communications of the ACM*, May 1994, vol. 37 no. 5

[Nilsson 95] Anders G. Nilsson: «*Utvekling av metoder för systemarbete - ett historisk perspektiv»,* Research report nr LiTH-IDA-R-95-13, Universitetet och Tekniska högskolan i Linköping 1993

[Nordbeck & Rystedt 67] Stig Norbeck, Bengt Rystedt: «Computer cartography — point-in-polygon programs», *BIT* vol. 7 1967 pp 39–64

[Normann & Ressem 94] Ragnar Normann, Jan Erik Ressem: *«Kompendium 60 for IN113 — Dataorientert systemutvikling og databaser»*, Institutt for Informatikk, Universitetet i Oslo 1994

[Oldevik 96] Jon Oldevik: *«DISGIS General Method»*, ESPRIT Project 22.084, SINTEF 1996

[OOram 96a] Taskon A/S: *«Product description for OOram Professional 4.0»*, Taskon 1996 `http://www.sn.no/taskon/download/ooram40.ps`

[OOram 96b] Taskon A/S: *«OOram Professional: A method guide for real time/telecommunication system development»*, Taskon A/S 1996 `http://www.sn.no/taskon/download/guide.ps`

[ORBIX 95] «*Orbix Programmer's Guide*», Release 1.3, IONA Technologies 1995

[Orfali *et al* 96a] Robert Orfali, Dan Harkeu, Jeri Edwards: «*The essential Client/Server Survival Guide*», John Wiley & Sons inc. New York 1996

[Orfali *et al* 96b] Robert Orfali, Dan Harkeu, Jeri Edwards: «*The essential Distributed Objects Survival Guide*», John Wiley & Sons inc. New York 1996

[OSF/DCE 92] Open Software Foundation: *«Introduction to OSF DCE»*, Open Software Foundation, Cambridge, USA 1992

[Panagopoulou et al 96] Georgia Panagopoulou, Spiros Sirmakessis, Athanasios Tsakalidis: «A new approach to spatial data structures: Evaluation and redefinitions of their properties», *Geographical Information* vol 1 1996 pp 34–44

[Pettersson 94] Karin Petterson: *«Consequences of an IRM-based system development - experiences from a case study»,* Research report nr LiTH-IDA-R-94-05, Universitetet och Tekniska högskolan i Linköping 1994

[Plowiec 96] Krzysztof Plowiec: «*ODP viewpoint on DCE based Generic Object Services*», Hovedoppgave, Institutt for Informatikk, Universitetet i Oslo august 1996

[Rasmussen et al 96] G. Rasmussen, B. Henderson-Sellers, G. C. Low: «Extending the MOSES methodology to distributed systems», *Journal of Object-Oriented Programming*, July/August 1996

[Raymond 95] Kerry Raymond: *«Reference Model of Open Distributed Processing (RM-ODP): Introduction»*, `http://www.dstc.edu.au/AU/research_news/odp/ref_model/papers.html/icodp95.ps.gz`

[Reenskaug *et al* 96] Trygve Reenskaug, Per Wold, Odd Arild Lehne: *«Working with objects — The OOram software engineering method»*, Mannings Publications Co. 1996

[Ressem 95] Jan Erik Ressem: «*Hvor kommer alle objektene fra?»*, Cand.Scient- avhandling i informatikk, Universitetet i Oslo 1995

[RM-ODP 1] ISO/IEC DIS 10746-1 | ITU-T X.901, *«Reference Model of Open Distributed Processing — Part 1: Overview and Guide to use»*, juni 1995

[RM-ODP 2] ISO/IEC DIS 10746-2 | ITU-T X.902, *«Reference Model of Open Distributed Processing — Part 2: Foundations»*

[RM-ODP 3] ISO/IEC DIS 10746-3 | ITU-T X.903, *«Reference Model of Open Distributed Processing — Part 3: Architecture», 1995*

[Rosenberg 94] Doug Rosenberg: «Modeling client-server systems — No single methodology adresses all the necessary concerns», *Object Magazine*, Vol. 4 no. 1 March-April 1994

[Roussopoulos et al 95] Nick Roussopoulos, Stephen Kelley, Frédéric Vincent: «Nearest neighbor queries», *SIGMOD Record*, 1995 iss. 2, ACM Press, New York 1995

[Rudd 94] Anthony Rudd: «*C++ Complete — A reference and Tutorial to the proposed C++ standard*», John Wiley & Sons USA 1994

[Rösch 95] Martin Rösch: «Distributed objects are real», *Objects in Europe* Winter 1995

[Saarinen & Heikkilä 90] Timo Saarinen og Jukka Heikkilä: «Methods and tools for developing new information systems and enhancing or replacing the old ones», *Proceedings of the 13th IRIS Conference*, 12–15. august 1990

[Schmidt 92a] Douglas C. Schmidt: «Systems Programming with C++ wrappers — encapsulating interprocess communication mecanisms with object-oriented interfaces», C++ Report, Sepember/October 1992 `ftp://wuarchive.wustl.edu/languages/c++/ACE/ACE-documentation/C++-wrappers.ps.gz`

[Schmidt 92b] Douglas C. Schmidt: «IPC SAP — A family of object-oriented interfaces for local and remote interprocess communication», C++ Report, November/December 1992 `ftp://wuarchive.wustl.edu/languages/c++/ACE/ACE-documentation/IPC_SAP-92.ps.gz`

[[Schulz 95] Frederick Schulz: *«Open Systems Environment (OSE): Architectural Framework for Information Infrastructure»*, NIST Special Publication 500-323, U.S. Government Printing Office, Washington 1995

[Schwaderer 92] W. David Schwaderer: *«C programmer's guide to NetBIOS, IPX, and SPX»*, Sams Publishing, USA 1992

[SELECT 96] *«SELECT Enterprise — model-driven development toolset»*, brosjyremateriell og demonstrasjonsnotater

[Shaw *et al* 96] Mary Shaw, David Garlan: *«Software Architecture»*, Prentice-Hall New Jersey 1996

[Shelton 94a] Robert E. Shelton: «Object-oriented business engineering: delivering the distributed enterprise», *Proceedings of Object Expo Europe September 20–30 1994, London*, SIGS Conferences New York 1994

[Shelton 94b] Robert E. Shelton: «The distributed enterprise», *Journal of Object-oriented Porgramming*, vol. 7 iss. 2 1994

[Skagestein 91] Gerhard Skagestein: *«Data i fokus – Dataorientert systemutvikling i teori og praksis»*, Universitetsforlaget 1991

[Skagestein 94] Gerhard Skagestein: «Hvordan lage en klient/tjener-basert applikasjon ut fra et datamodelleringsutgangspunkt?», *Klient/tjener arkitektur og distribuerte systemer*, Geilo 18.–19. april 1994

[Smørdal 96] Ole Smørdal: «Soft Objects Analysis — A modelling approach for analysis of interdependent work practices», *Proceedings of Third international conference on object-oriented information systems OOIS '96*, London 1996

[SOSI 95] *«Samordnet Opplegg for Stedfestet Informasjon versjon 2.2»*, Statens Kartverk 1995

[Strand 96] Ragnhild Strand: *«Vedlikehold av edb-systemer og reverse engineering — En kartlegging av situasjonen i seks norske organisasjoner»*, Hovedfagsoppgave, Institutt for Informatikk, Universitetet i Oslo 1996

[Stubbs & Webre 89] Daniel F. Stubbs, Neil W. Webre: *«Data structures with abstract data types and Pascal»*, 2nd edition, Brooks/Cole Publishing Company, California 1989

[Tanenbaum 87] Andrew S. Tanenbaum: *«Operating systems: Design and implementation»*, Prentice-Hall, New Jersey 1987

[Tanenbaum 90] Andrew S. Tanenbaum: *«Structured computer organization»*, Third edition, Prentice-Hall, New Jersey 1990

[Taylor 95] David A. Taylor: *«Business engineering with object technology»*, John Wiley & Sons 1995

[Tepfenhart & Cusick 97] William M. Tepfenhart, James J. Cusick: «A unified object topology», *IEEE Software* vol. 14 no. 1, Jan/Feb 1997

[Veldwijk *et al* 94] R. J. Veldwijk, M. Boogard, E. R. K. Spoor: «Assessing the software crisis: Why information systems are beyond control», *Information Scienes* 1981 pp 103-116

[Webster 83] *«The Webster Dictionary of the English language»*, Publishers United Guild, New York 1983.

[Øgrim 92] Leikny Øgrim: *«Vedlikehold eller redesign, et spørsmål om organisering?»*, FIRE rapport nr. 3, Institutt for Informatikk, Universitetet i Oslo, 1992

[Øgrim 93] Leikny Øgrim: *«Ledelse av systemutviklingsprosjekter — En dialektisk tilnærming»*, Dr.Scient-avhandling, Research Report nr 183, Institutt for Informatikk, Universitetet i Oslo 1993

[Aagedal et al 97] Jan-Øyvind Aagedal, Arne-Jørgen Berre, Alistair Cockburn, Jon Oldevik, Trygve Reenskaug, Georges-Pierre Reich, Rebecca Wirfs-Brock: *«The OOram Meta-model — combining role models, interfaces and classes to support system centric and program centric modeling — A proposal to OMG OA&D RFP-1»*, Version 1.0, Taskon A/S, Reich Technologies, Humans and technology, 1997

B Ordforklaringer

ActiveX	Microsofts teknologi for objektkommunikasjon. Tidl. OLE (Object Linking and embedding).
AUIDL	Abstract User Interface Design Language.
BPR	Business Process Rengineering — totalfornyelse av en organisasjons driftsprosesser.
BSD Unix	Berkeley Software Distribution — familiegren av UNIX-varianter.
bro	boks som kobler mellom to eller flere lokalnettsegmenter.
COM	Component Object Model — Microsofts objektmodell for objektkommunikasjon
CORBA	Common Object Request Broker Architecture.
demon	(daemon) Bakgrunnsprosess i **UNIX** miljøer, f.eks skriverkøhåndtering (**BSD Unix**: `lpd(1)` — BSD Line Printer Daemon)
DCE	Distributed Computing Environment — standard for distribuerte systemer. Bestpr av bl.a sikkerhetsmekanismer og fjernprosedyrekall (se **RPC**) m.m.
DCOM	Distributed Component Object Model — Microsofts standard for distribuerte objekter (se **COM**).
DEC	Digital Equipment Corporation.
FIRE	Functional Integration through REdesign; forskningsprosjekt ved systemarbeidsgruppa ved institutt for Informatikk, UiO.
FooAM	Forward object-oriented Application Model.
hub	kobler sammen to eller flere Ethernettsegmenter, istedenfor busskabel.
IDL	Interface Definition Language — språk for å definere grensesnitt for distribuerte objekter. Forkjellige språk er definert for **CORBA**, **DCE** og **DCOM**.
IIOP	Internet Inter-Orb Protocol — felles kommunkasjonsprotokoll for CORBA-baserte objektmeglere.

IPX/SPX	Internetwork Packet Exchange / Sequenced Packet Exchange — nettverksprotokoll, brukes bl.a. i NOVELL NETWARE.
ISA-buss	16 bit ekpansjonsbuss, også kalt AT-buss. Standard buss for utvidelser i IBM-kompatible PC'er.
ITU-T	International Telecommunications Union.
MS-DOS	Microsoft Disk Operating System
NetBIOS	Network Basic Input/Ouput — nettverksprotokoll brukt på IBM PC.
ODL	Object Description Language
OMA	Object Management Architecture.
OMG	Object Management Group — konsortsium med over 700 selskaper som har som mål å standardisere infrastrukturer for distribuerte objetteknologi (se **CORBA**).
OORam	Object-Oriented Rôle analysis and modeling — objektorientert analyse- og designmetode utviklet ved Universitetet i Oslo, SINTEF og Taskon AS.
OSF	Open Systems Foundation.
OSI	Open Systems Interconnection (Referansemodell, se side 37).
RM-ODP	Reference model for Open Distributed Processing (se side 42).
RPC	Remote Procedure Call — fjernprosedyrekall; kall på prosedyre utføres av en (muligens) annen maskin. Forklart mer inngående på side 57.
RooAM	Reversely generated object-oriented Application Model.
SQL	Structured Query Language
søkesti	Liste av filkataloger som gjennomsøkes når et program skal eksekveres
TCP/IP	Transport Control Protocol/Internet Protocol — forbindelsesorientert kommunikasjon basert. Brukes på Internett.
TLI	Transport Layer Interface — funksjonsbibliotek for datakommunikasjon, brukes gjerne på UNIX System V-derivater.
UNIX	Flerbruker, nettverksbasert operativsystem, dominerende i akademiske miljøer. Kraftig, men med høy begynnerterskel. UNIX er et registrert varemerke for AT&T.
X-Window System	Nettverksbasert vindussystem som (som oftest) eksekveres i UNIX miljø.

C Kildekode til eksempel-applikasjonen

Som nevnt i Kapittel 6 har jeg laget en distribuert eksempelapplikasjon i Java og C++. Den viktigste kildekoden for prosjektet er presentert i dette tillegget, forhåpentlig til nytte for spesielt interesserte. Verdt å merke seg er hvor få linjer i koden som faktisk behandler distribusjon og kommunikasjon. Dette tar jeg til inntekt for bruk av distribuert objektteknologi med objektmeglere.

Figur 7.1 viser sammenhengen mellom de forskjellige kildefilene og de eksekverbare binærfilene som produseres. Noen av kildefilene er automatisk generert fra andre filer, og disse er ikke tatt med i dette tillegget.

Filene fra prosjektet som er tatt med her er:

- IDL-grensesnitt:

  `gis.idl` (side 135) — idl-spesifikasjonen av systemet. Genererer

  `gisS.cc` —

  `gisC.cc` —

- GAB-tjeneren:

  `gabserver.cc` (side 138)

  `gab_i.h`, **`gab_i.cc`** (side 136) headerfil og implementasjon av GAB-objektet

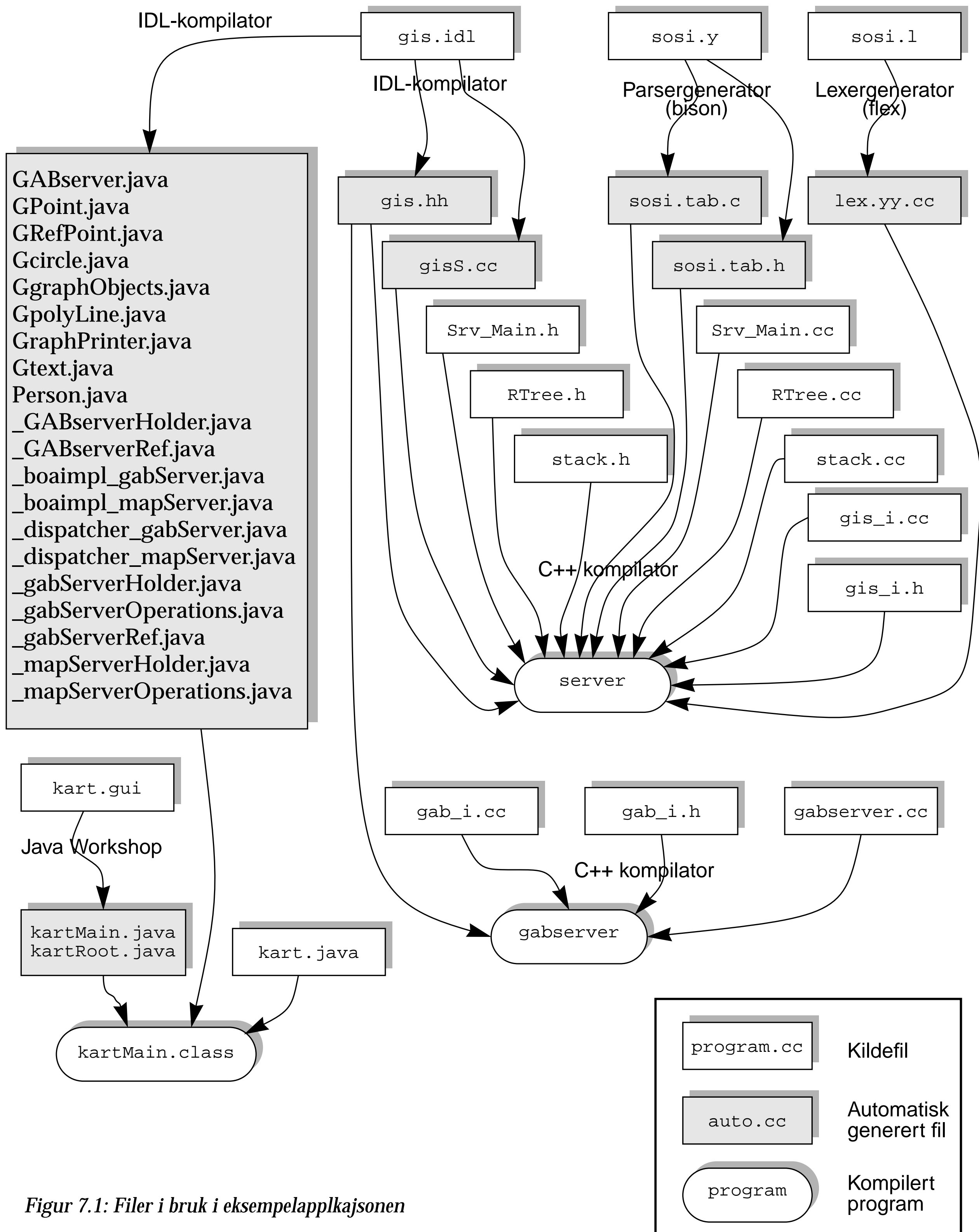
- Karttjeneren

  `sosiparser.l`, **`sosiparser.y`** — **`flex(1)`**- og **`bison(1)`**-kilde for leksikalsk analysator og parser for SOSI-parsering (side 141)

  `RTree.h`, **`RTree.cc`** — (side 146) kildekode for R-tre indekseringsstrukturen.

  `stack.h`, **`stack.cc`** — (side 174) kildekode for en enkel stakkstruktur, brukes av parseren.

  `mapserver.h`, **`mapserver.cc`** — hovedfilen til karttjeneren (inneholder **`main()`**-funksjonen) (side 178).



Figur 7.1: Filer i bruk i eksempelapplkajsonen

gis_i.h, **gis_i.cc** — headerfil og kildekode for objektimplementasjonen (side 136).
dbm.h, **dbm.cc** — Databaseadministrasjonsverktøy (side 181)

- Klienten:

  kart.gui — kildekode for brukergrensesnittet til klienten, generert med Java Workshop (side 183).

  kart.java — kildekode for klienten (side 187).

Grensesnittspesifikasjon

gis.idl

```
     // IDL
     struct Person {
         string Name;
         string Address1;
 5       string Address2;
         string Address3;
     };

     struct SOSIPoint {
10           long north;
             long east;
     };

     struct SOSICircle{
15           SOSIPoint theSOSIRefPoint;
              long SOSIDescription;
              SOSIPoint Gcentre;
              float Gradius;
     };
20
     struct SOSIText{
              SOSIPoint theSOSIRefPoint;
              long SOSIDescription;
              string theGText;
25            SOSIPoint Gposition;
              long SOSISize;
              float SOSIAngle;
     };

30   struct  SOSIRefPoint{
              SOSIPoint theSOSIRefPoint;
              long SOSIDescription;
     };

35   struct SOSIPolyLine{
              sequence<SOSIPoint> joints;
              SOSIPoint theSOSIRefPoint;
              long SOSIDescription;
     };
40
     struct SOSIGraphObjects{
         sequence<SOSICircle> SOSICircles;
         sequence<SOSIPolyLine> SOSIPolyLines;
```

```
        sequence<SOSIText> SOSITexts;
45      sequence<SOSIRefPoint> SOSIRefPoints;
    };


    interface mapServer{
50          SOSIGraphObjects getRegion(in SOSIPoint NW, in SOSIPoint SE);
            SOSIGraphObjects getOrigoRegion();
    };

    interface gabServer{
55      exception notFoundException{
        string reason;
        };
        Person findOwner(in SOSIPoint building);
        SOSIPoint findLocation(in string address);
60      string findAddress(in SOSIPoint location);
    };
```

gis_i.h

```
    #ifndef gab_ih
    #define gab_ih
    #include "gis.hh"
    #include <ndbm.h>
5   #include "dbm.h"
    #include <fcntl.h>

    class gab_i: public gabServerBOAImpl{
    private:
10    char tekst[]="hallo, der!";
    public:
      //constructor
        gab_i();

15    virtual char * findAddress(const SOSIPoint &location, CORBA::Environment
                  &IT_env=CORBA::default_environment);
      virtual SOSIPoint findLocation (const char * address, CORBA::Environment
                    &IT_env=CORBA::default_environment);
      virtual Person findOwner(const SOSIPoint &location, CORBA::Environment
20                &IT_env=CORBA::default_environment);
    };
    #endif
```

gis_i.cc

```
    #include <stream.h>
    #include <fstream.h>
    #include <malloc.h>
    #include "gis_i.h"
5   #include "mapserver.h"
```

```
        extern SOSIGraphObjects *theObjects; //mer innhold
        gis_i::gis_i(){
        theObjects=(SOSIGraphObjects*)malloc(sizeof(SOSIGraphObjects));
   10   }



        SOSIGraphObjects gis_i::getRegion (const SOSIPoint& NW, const SOSIPoint& SE,
   15                   CORBA::Environment &IT_env) {
        SOSIGraphObjects theObjects2;;

        ObjectRecord * O;
        Rect searchRect;
   20   register int polyLines_count=0;
        register int circles_count=0;
        register int texts_count=0;
        register int refPoints_count=0;
        searchRect.boundary[0]=(float)NW.east;
   25   searchRect.boundary[1]=(float)NW.north;
        searchRect.boundary[2]=(float)SE.east;
        searchRect.boundary[3]=(float)SE.north;
        theObjects=new SOSIGraphObjects;
        //R->search->resetList();
   30   R->searchRectangle(searchRect);


        theObjects->SOSIPolyLines._buffer = new SOSIPolyLine[R->search->statis-
tics.SOSIPolyLines];
        theObjects->SOSICircles._buffer = new SOSICircle[R->search->statistics.SOSICir-
cles];
   35   theObjects->SOSITexts._buffer = new SOSIText[R->search->statistics.SOSITexts];
        theObjects->SOSIRefPoints._buffer = new SOSIRefPoint[R->search->statistics.SO-
SIRefPoints];

        R->search->newTraverse();

   40   // copying data from searchlist to return package
        for (int i=0;i<R->search->statistics.total;i++){
          O=R->search->nextItem();
          if (O)
          switch (O->type){
   45         case ObjectRecord::POLYLINE:

            theObjects->SOSIPolyLines[polyLines_count].SOSIDescription=
              O->thePolyLine->SOSIDescription;
            theObjects->SOSIPolyLines[polyLines_count].theSOSIRefPoint=
   50         O->thePolyLine->theSOSIRefPoint;
            cout << theObjects->SOSIPolyLines[polyLines_count].theSOSIRefPoint.east
                 << ", "
                 << theObjects->SOSIPolyLines[polyLines_count].theSOSIRefPoint.north
                 << endl;
   55       theObjects->SOSIPolyLines[polyLines_count++].joints=O->thePolyLine->joints;
            //  cout << " polyLines_count: " << polyLines_count-1 << endl;
            break;

              case ObjectRecord::CIRCLE:
   60
            break;

              case ObjectRecord::TEXT:

   65       theObjects->SOSITexts[texts_count].SOSIDescription=
              O->theText->SOSIDescription;
            theObjects->SOSITexts[texts_count].theSOSIRefPoint=O->theText->theSOSIRef-
Point;
```

```
            theObjects->SOSITexts[texts_count].theGText=O->theText->theGText;
             theObjects->SOSITexts[texts_count++].Gposition=O->theText->Gposition;
  70        break;

              case ObjectRecord::REFPOINT:
            theObjects->SOSIRefPoints[refPoints_count].SOSIDescription=
              O->theRefPoint->SOSIDescription;
  75        theObjects->SOSIRefPoints[refPoints_count].theSOSIRefPoint=
              O->theRefPoint->theSOSIRefPoint;

            break;
              }
  80
        }
        // setting the actual sizes of the package:
        theObjects->SOSIPolyLines._length=theObjects->SOSIPolyLines._maximum=R->search-
>statistics.SOSIPolyLines;
        theObjects->SOSICircles._length=theObjects->SOSICircles._maximum=R->search-
>statistics.SOSICircles;
  85    theObjects->SOSITexts._length=theObjects->SOSITexts._maximum=R->search->statis-
tics.SOSITexts;
        theObjects->SOSIRefPoints._length=theObjects->SOSIRefPoints._maximum=R->search-
>statistics.SOSIRefPoints;

        // cleaning up

  90    //R->search->resetList();
        return *theObjects;

        }

  95    SOSIGraphObjects gis_i:: getOrigoRegion (CORBA::Environment &IT_env) {
        SOSIGraphObjects theObjects;
        logg << "getOrigoRegion invocation...\n";
        //theObjects=new SOSIGraphObjects;
        return theObjects;
 100    }
```

GAB-tjener

gabserver.cc

```
        #include "gis.hh"
        #include "gab_i.h"
        #include <stream.h>
        #include <fstream.h>
   5    #include <stdio.h>
        #include <malloc.h>

        int main(int argc, char *argv[]) {
        gab_i myGab;
  10      TRY {
            // tell Orbix that we have completed the server's initialisation:
            cout << "Registering gabServer...\n";
```

```
          CORBA::Orbix.impl_is_ready("gabServer",CORBA::Orbix.INFINITE_TIMEOUT,IT_X);
          cout << "done.\n";
15      }
        CATCHANY {
          // an error occured calling impl_is_ready() - output the error.
          cout << IT_X;
        }
20      ENDTRY

        // impl_is_ready() returns only when Orbix times-out an idle server
        // (or an error occurs).
        cout << "gabServer exiting" << endl;
25
        return 0;

      }
```

gab_i.h

```
      #ifndef gab_ih
      #define gab_ih
      #include "gis.hh"
      #include <ndbm.h>
 5    #include "dbm.h"
      #include <fcntl.h>

      class gab_i: public gabServerBOAImpl{
      private:
10      char tekst[]="hallo, der!";
      public:
        //constructor
            gab_i();

15      virtual char * findAddress(const SOSIPoint &location, CORBA::Environment
                     &IT_env=CORBA::default_environment);
        virtual SOSIPoint findLocation (const char * address, CORBA::Environment
                       &IT_env=CORBA::default_environment);
        virtual Person findOwner(const SOSIPoint &location, CORBA::Environment
20                   &IT_env=CORBA::default_environment);
      };
      #endif
```

gab_i.cc

```
      //  implementasjon av GAB-tjener

      #include <stream.h>
      #include <fstream.h>
 5    #include <malloc.h>
      #include "gab_i.h"
      #include <string.h>

      gab_i::gab_i(){
```

```
10    }

      char * gab_i::findAddress(const SOSIPoint &location, CORBA::Environment
                 &IT_env=CORBA::default_environment){
        // returns the address associated with the location
15
      return tekst;
      }

      Person gab_i::findOwner(const SOSIPoint &location, CORBA::Environment
20               &IT_env=CORBA::default_environment){
        Person *P=new Person;
        DBM* owner_db;
        datum loc, owner;
        loc.dptr=(char*)new SOSIPoint;  // preparing a new coordinate
25      loc.dptr=memcpy(loc.dptr, (void*)&location, sizeof(SOSIPoint));
        loc.dsize=sizeof(SOSIPoint);
        owner_db=dbm_open(OWNER_DB,  O_RDWR | O_CREAT, 0660); // opening database
        owner=dbm_fetch(owner_db, loc);                       // fetching data
        dbm_close(owner_db);                                  // closing database
30      P->Name=new char[owner.dsize+1];
        P->Name[owner.dsize]='\0';
        strncpy(P->Name, owner.dptr, owner.dsize);
        return *P; // done
      };
35
      SOSIPoint gab_i:: findLocation (const char * address,
                        CORBA::Environment &IT_env) {
        gabServer::notFoundException *e;
        SOSIPoint *P;
40      Coordinate *G;
        char *p;
        DBM * address_db;
        datum Address, location;
        P=new SOSIPoint();
45      Address.dptr=strdup(address);                    // preparing key
        Address.dsize=strlen(address);                   // ditto
        address_db=dbm_open(ADDRESS_DB,  O_RDWR, 0660);  // opening database
        location=dbm_fetch(address_db, Address);         // fetching data
        if (location.dptr==NULL) {                       // raising Exception
50        gabServer::notFoundException *e = new gabServer::notFoundException;
          char *why = "Address not found";
          e->reason= new char[strlen(why)+1];
          strcpy(e->reason, why);
          IT_env=e;
55      } else {
        p=location.dptr;
        G=new Coordinate;
        G=(Coordinate*)memcpy(G, p, sizeof(Coordinate)); // aligning struct
        P->north=G->north;                               // copying contents
60      P->east=G->east;                                 // ditto
        dbm_close(address_db);                           // close database
        delete G;
        }
        return *P;
65    }
```

Karttjeneren

sosilexer.l

```
/* simple SOSI lexical analyzer */

%option noyywrap
%option 8bit
%{
#include <stdio.h>
#include <stdlib.h>
#include "sosi.tab.h"
#include <math.h>
#define TOK(name) { return name;}
extern int lineno, level;

warning (char *s, char* S){
  printf("Warning at line %d: %s %s\n", lineno, s, S);
}

%}

%x IGNORE
%x STRING
%x HODEM

ignore_line    \.\.BYGGTYP
sosinivaa      \.\.SOSI\-NIVÅ
tegnsett       \.\.TEGNSETT
transpar       \.\.TRANSPAR
koordsys       \.\.\.KOORDSYS
origo          \.\.\.ORIGO\-NØ
enhet          \.\.\.ENHET
omraade        \.\.OMRÅDE
min            \.\.\.MIN\-NØ
max            \.\.\.MAX\-NØ
kkvalitet      \.\.\.KVALITET
ws          [ \t]+
enkeltekst      [^\"\' \t\n]+
hode            \.HODE
slutt           \.SLUTT
kurve           \.KURVE
linje           \.LINJE
tekst           \.TEKST
ltema           \.\.LTEMA
ptema           \.\.PTEMA
ttema           \.\.TTEMA
streng          \.\.STRENG
koord3d         \.\.NØH
koord2d         \.\.NØ
punkt           \.PUNKT
ekode           \.\.EKODE
arkode          \.\.ARKODE
nl      \n
heltall         [\-]?[0-9]+
dato            \.\.DATO
kval            \.\.KVALITET
komm            \.\.KOMM
flyttall        ("-")?{heltall}[EeDd][+-]?{heltall}|("-")?{heltall}"."{hel-
tall}[EeDd][+-]?{heltall}|("-")?{heltall}"."{heltall}
teksten          \"[^\"\n"]*[\"\n]|\'[^\'\n\']*[\'\n]
concat           \&
```

```
    %%
    <STRING>{ws}        ;
60  {ws}                ;
    {slutt}         {return SLUTT;}
    {tegnsett}       {BEGIN STRING; return TEGNSETT;}
    {transpar}       TOK(TRANSPAR)
    {koordsys}       {BEGIN STRING; return KOORDSYS;}
65  {origo}          TOK(ORIGO)
    {enhet}          TOK(ENHET)
    {omraade}        TOK(OMRAADE)
    {min}            TOK(MIN)
    {max}            TOK(MAX)
70  {sosinivaa}      TOK(SOSINIVAA)
    {ignore_line}   {BEGIN IGNORE;}
    {hode}          TOK(HODE)
    {linje}         TOK(LINJE)
    {kurve}         TOK(LINJE)
75  {tekst}         TOK(TEKST)
    {punkt}         TOK(PUNKT)
    {ekode}         TOK(EKODE)
    {arkode}        TOK(ARKODE)
    {ltema}         TOK(LTEMA)
80  {ptema}         TOK(PTEMA)
    {ttema}         TOK(TTEMA)
    {streng}        {BEGIN (STRING); return STRENG;}
    {koord3d}       TOK(KOORD3D)
    {koord2d}       TOK(KOORD2D)
85  {dato}          TOK(DATO)
    {komm}          TOK(KOMM)
    {kval}          TOK(KVALITET)
    {concat}        {BEGIN STRING; return KONKAT;}
    <STRING>{teksten} {
90    BEGIN INITIAL;
      yylval.string = strdup((char*)(yytext+1));  /* skip open quote */
      yylval.string[yyleng-2] = ' '; /* remove close quote*/
      return TEKSTEN;
    }
95
    <STRING>{enkeltekst} {
      BEGIN (INITIAL);
      yylval.string=strdup(yytext);
      return TEKSTEN;
100 }
    {nl}        {lineno++;}
    <IGNORE>{nl}    {BEGIN INITIAL;}
    {heltall}       {yylval.intval=atoi(yytext); return HELTALL;}
    {kkvalitet}     {BEGIN IGNORE;}
105 <IGNORE>{heltall}     {}
    {flyttall}      {yylval.floatval=atof(yytext); return FLYTTALL;}
    <IGNORE>{flyttall}    {}
    <IGNORE>{enkeltekst} ;
    <*>\!.*\n       {lineno++;} /* comment */
110 .               {return yytext[0]; }
```

sosiparser.y

```
/* simple SOSI parser*/
%{
```

```
    #define YYERROR_VERBOSE
    #include <stdio.h>
 5  #define MAXPARS 10
    #define MAXSYMS 100
    #include "sosi.h"
    #include "gis.hh"
    #include <stream.h>
10  #include "stack.h"
    #include <malloc.h>
    #include <stdlib.h>
    #include "RTree.h"
    #include "mapserver.h"
15  extern warning(char*, char*);
    extern yylex();
    extern int lineno, yyleng;
    extern char* yytext;
    extern RTree * R;
20  yyerror (char *s){
      printf("Error at line %d: %s\n", lineno, s);
      printf("%s\n", yytext);
    }

25  SOSIPolyLine * cur_polyline;
    SOSIText * cur_text;
    SOSIPoint * cur_point;

    Stack stack;
30  Stackelement *S;
    char temabuffer[32];
    int level, i, n;
    Rect box;
    SOSIPoint * P;
35  float enhet=1.0;
    int origo_x=0, origo_y=0;
    %}

    %union {
40    char *string;
      int intval;
      float floatval;
      struct symtab *symp;
    }
45
    %token <intval> HELTALL
    %token <floatval> FLYTTALL
    %token LINJE KURVE LTEMA PTEMA TTEMA PUNKT DATO KOMM EKODE ARKODE
    %token TEKST STRENG HODE SLUTT KOORD2D KOORD3D TRANSPAR KVALITET
50  %token TEGNSETT KOORDSYS ORIGO ENHET OMRAADE MIN MAX SOSINIVAA KONKAT
    %token <string> TEKSTEN
    %type <string> tekster
    %type <intval> taller

55  %%
    sosifil:    HODE hode elementer SLUTT {
    }
              ;

60  hode:        {warning("Empty file header","");}
                | TRANSPAR transpar hode
                | TEGNSETT TEKSTEN hode
                | OMRAADE omraade hode
                | SOSINIVAA HELTALL hode
65              ;

    omraade:
```

```
                | MIN HELTALL HELTALL omraade
                | MAX HELTALL HELTALL omraade
70              ;

    transpar:
                | KOORDSYS TEKSTEN transpar
                | ORIGO HELTALL HELTALL { origo_x = $2; origo_y =$3;} transpar
75              | ENHET FLYTTALL { enhet = $2;cout << enhet << endl;} transpar
                ;

    elementer:   element
               | elementer element
80             ;

    element:    KURVE  kurve
              | KURVE  HELTALL ':' kurve
              | LINJE ny_linje linje save_linje
85            | LINJE ny_linje HELTALL ':' linje save_linje
              | PUNKT refpunkt
              | PUNKT HELTALL ':' refpunkt
              | TEKST ny_tekst tekst save_tekst
              | TEKST HELTALL ':' ny_tekst tekst save_tekst
90            ;

    ny_tekst: {
    cur_text = new SOSIText;
    }
95
    ny_linje: {
                stack.clear();
                cur_polyline = new SOSIPolyLine; }
            ;
100 save_tekst: {
      cur_text->theSOSIRefPoint.east = cur_text->Gposition.east;
      cur_text->theSOSIRefPoint.north = cur_text->Gposition.north;
      RTree::boundingBox(&box, cur_text);
      R->InsertObject(&box, ObjectRecord::TEXT,
105           (GgraphObject*) cur_text, lineno);
    }
    ;

    save_linje: {
110   cur_polyline->theSOSIRefPoint.east = cur_polyline->joints._buffer[0].east;
      cur_polyline->theSOSIRefPoint.north = cur_polyline->joints._buffer[0].north;
      RTree::boundingBox(&box, cur_polyline);
      R->InsertObject(&box, ObjectRecord::POLYLINE,
              (GgraphObject*) cur_polyline, lineno);
115 }
    ;

    linje:
           | linje linjedel
120        ;

    linjedel: LTEMA HELTALL {
      cur_polyline->SOSIDescription=$2;
    }
125        | DATO HELTALL
           | KOMM HELTALL
           | KVALITET taller
           | punkter  {
        i=stack.getLength();
130     cur_polyline->joints._buffer=new SOSIPoint[i];
        for (n=0;n<i;n++) {
          S=stack.pop();
```

```
          cur_polyline->joints._buffer[n].east=S->east;
          cur_polyline->joints._buffer[n].north=S->north;
135     }
        cur_polyline->joints._maximum=i;
        cur_polyline->joints._length=i;
          }
         ;
140
tekst:
          | tekst tekstdel
          ;

145 tekstdel:  TTEMA HELTALL {cur_text->SOSIDescription=$2;}
          | DATO HELTALL
          | KOMM HELTALL
          | KVALITET taller
          | STRENG tekster
150       | punkter {

        do {S=stack.pop();} while (stack.getLength()>0);
        cur_point=new SOSIPoint;
        cur_point->east=S->east;
155     cur_point->north=S->north;
        cur_text->Gposition=*cur_point;
          }
          ;

160 tekster:   TEKSTEN {cur_text->theGText=$1;}
    | TEKSTEN KONKAT TEKSTEN
            ;

kurve:
165       | kurve kurvedel
          ;

kurvedel:   LTEMA HELTALL
          | DATO HELTALL
170       | KOMM HELTALL
          | KVALITET taller
          | punkter
          ;

175 refpunkt:
          | refpunkt punktdel
          ;

punktdel:  PTEMA HELTALL
180       | DATO HELTALL
          | KOMM HELTALL
          | KVALITET taller
          | EKODE HELTALL
          | ARKODE HELTALL
185       | punkt
          ;

taller:   HELTALL
          | HELTALL taller
190       ;

punkter: KOORD2D punkt2liste
          | KOORD3D punkt3liste
          ;
195
punkt2liste: HELTALL HELTALL {
  stack.push($2, $1);
```

```
        }
                 |   punkt2liste HELTALL HELTALL {
200              stack.push($3, $2);
            }
                 ;
        punkt3liste: HELTALL HELTALL HELTALL {
            stack.push($2, $1);
205     }
                 |   punkt3liste HELTALL HELTALL HELTALL {
                 stack.push($3, $2); }
                 ;

210     punkt: KOORD2D HELTALL HELTALL {
          cur_point=new SOSIPoint;
          cur_point->east=$3;
          cur_point->north=$2;
        }
215            | KOORD3D HELTALL HELTALL HELTALL {
          cur_point=new SOSIPoint;
          cur_point->east=$3;
          cur_point->north=$2;
        }
220
               ;
        %%
```

RTree.h

```
        // Header file based on index.h, split_l.h and split_q.h by Daniel Green
        //
        // C++ conversion by Kristoffer Moe 1997

  5     #ifndef _RTREE_
        #define _RTREE_

        #include "gis.hh"

 10     /* PGSIZE is normally the natural page size of the machine */
        #define PGSIZE  1024
        #define NUMDIMS 2   /* number of dimensions */

        typedef float RectReal;
 15

        /*-----------------------------------------------------------------------------
        | Global definitions.
        -----------------------------------------------------------------------------*/
 20
        #ifndef TRUE
        #define TRUE 1
        #endif
        #ifndef FALSE
 25     #define FALSE 0
        #endif

        #define NUMSIDES 2*NUMDIMS
```

```
30    // Global types:

      struct ObjectRecord{
        enum OBJECTTYPE {POLYLINE, CIRCLE, TEXT, REFPOINT} type;
        int lineno;
35      union{
          SOSIPolyLine * thePolyLine;
          SOSICircle * theCircle;
          SOSIText * theText;
          SOSIRefPoint * theRefPoint;
40      };
      };


      struct Rect
45    {
          RectReal boundary[NUMSIDES]; /* xmin,ymin,...,xmax,ymax,... */
      };

      struct Node;
50
      struct Branch
      {
          struct Rect rect;
          struct Node *child;
55    };

      /* branching factor of a node */
      #define NODECARD (int)((PGSIZE-(2*sizeof(int))) / sizeof(struct Branch))

60    /* balance criterion for node splitting */
      /* NOTE: can be changed if needed. */
      #define MinFill (NODECARD / 2)

      struct Node
65    {
          int count;
          int level; /* 0 is leaf, others positive */
          struct Branch branch[NODECARD];
      };
70
      struct ListNode
      {
          struct ListNode *next;
          struct Node *node;
75    };


      struct SearchListNode
      {
80        struct SearchListNode *next;
          struct ObjectRecord *content;
      };

      class SearchList{
85    public:
        SearchList();
        void addListNode(ObjectRecord *);
        void resetList();
        void newTraverse();
90      struct ObjectRecord * nextItem();
        struct {
          int SOSIPolyLines;
          int SOSICircles;
          int SOSITexts;
```

```
95          int SOSIRefPoints;
            int total;
          } statistics;

        private:
100       SearchListNode *first, *last, idle, *current_node;
          int length;
          int current_count;
        };

105     struct PartitionVars
        {
            int partition[NODECARD+1];
            int taken[NODECARD+1];
            int count[2];
110         struct Rect cover[2];
            RectReal area[2];
        };

        class RTree {
115     public:
          // methods
          RTree();
          ~RTree();
          int InsertObject(struct Rect *, ObjectRecord::OBJECTTYPE,
120                struct GgraphObject *, int=0);

          static Rect * boundingBox(struct Rect*, struct SOSIPolyLine*);
          static Rect * boundingBox(struct Rect*, struct SOSICircle*);
          static Rect * boundingBox(struct Rect*, struct SOSIText*);
125       static Rect * boundingBox(struct Rect*, struct SOSIRefPoint*);
          // variables
          SearchList * search;
          int searchRectangle(Rect);

130     private:
        #define METHODS 1
          // Variables:
            // Result list:
            SearchListNode * currentListNode;
135
            // Node splitting variables
            struct Branch BranchBuf[NODECARD+1];
            struct Rect CoverSplit;
            RectReal CoverSplitArea;
140         struct PartitionVars Partitions[METHODS];

            // The R-Tree:
            Node * Root;

145         // Methods:
            SearchListNode * AddListNode(ObjectRecord*);

            int AddBranch(struct Branch *, struct Node *, struct Node **);
            void Classify(int, int, struct PartitionVars *);
150         int Contained(struct Rect *, struct Rect *);
            int DeleteRect(struct Rect*, int, struct Node**);
            int DeleteRect2(struct Rect *, int, struct Node *, struct ListNode **);
            void DisconnectBranch(struct Node *, int);
            void FreeListNode(struct ListNode *);
155         void FreeNode(struct Node *);
            void GetBranches(struct Node *, struct Branch *);
            void InitBranch(struct Branch *);
            void InitNode(struct Node*);
            void InitPVars(struct PartitionVars *);
```

```
160         void InitRect(struct Rect*);
            int InsertRect(struct Rect*, int, struct Node**, int depth);
            int InsertRect2(struct Rect *, int, struct Node *, struct Node **, int);
            struct ListNode * NewListNode();
            void LoadNodes(struct Node *, struct Node *, struct PartitionVars *);
165         void MethodZero(struct PartitionVars *);
            SearchListNode * NewList();
            struct Node * NewIndex();
            struct Node * NewNode();
            int Overlap(struct Rect*, struct Rect*);
170         int PickBranch(struct Rect *, struct Node *);
            void PickSeeds(struct PartitionVars *);
            void Pigeonhole(struct PartitionVars *);
            void PrintBranch(struct Branch *, int);
            void PrintNode(struct Node *, int);
175         void PrintPVars(struct PartitionVars *);
            void PrintRect(struct Rect*, int);
            void RandomRect(struct Rect*);
            void ReInsert(struct Node *, struct ListNode **);
            struct Rect CombineRect(struct Rect*, struct Rect*);
180         struct Rect NodeCover(struct Node *);
            struct Rect NullRect();
            RectReal RectArea(struct Rect*);
            RectReal RectSphericalVolume(struct Rect *R);
            float RectSurfaceArea(struct Rect *);
185         RectReal RectVolume(struct Rect *R);
            int Search(struct Node *, struct Node *);
            void SearchRect(struct Rect *, struct Rect *);
            void SplitNode(struct Node*, struct Branch*, struct Node**);
            void TabIn(int);
190         void TraverseList(struct SearchListNode*);
            int main(int, char **);
            void print_volume(int, double);

        };
195
        #endif
        // _RTREE_
```

RTree.cc

```
        // C++ file based on index.c, node.c, rect.c, sphvol.c, split_l.c split_q.c
        // by Daniel Green
        //
        // C++ conversion by Kristoffer Moe 1997
  5
        #include "assert.h"
        #include <float.h>
        #include <malloc.h>
        #include <math.h>
 10     #include <stdio.h>
        #include <stdlib.h>
        #include <stream.h>
        #include "RTree.h"
        #include "gis.hh"
 15     #define BIG_NUM (FLT_MAX/4.0)
        #define Undefined(x) ((x)->boundary[0] > (x)->boundary[NUMDIMS])
        #define MIN(a, b) ((a) < (b) ? (a) : (b))
```

```
        #define MAX(a, b) ((a) > (b) ? (a) : (b))
20
        // constructor
        RTree::RTree(){

          Root = NewIndex();
25        search=new SearchList();
        }

        // public services:

30      // Insert an object in the structure
        int RTree::InsertObject(Rect * boundingBox,
                    ObjectRecord::OBJECTTYPE objectType, struct GgraphObject* G, int
line=0)
        {
          ObjectRecord * O;
35
          O=(ObjectRecord*) malloc(sizeof(ObjectRecord));
          O->type= objectType;
          switch (O->type){
          case ObjectRecord::POLYLINE:
40          O -> thePolyLine = (SOSIPolyLine*)G;
            break;

          case ObjectRecord::CIRCLE:
            O -> theCircle = (SOSICircle*)G;
45          break;

          case ObjectRecord::TEXT:
            O -> theText = (SOSIText*)G;
            break;
50
          case ObjectRecord::REFPOINT:
            O -> theRefPoint = (SOSIRefPoint*)G;
            break;
          }
55        O -> lineno=line;
          InsertRect(boundingBox, (int) O, &Root, 0);
          return 0;
        }

60
        // Make a new index, empty.  Consists of a single node.
        //
        struct Node * RTree::NewIndex()
        {
65          struct Node *x;
            x = NewNode();
            x->level = 0; /* leaf */
            return x;
        }
70

        //////////////////////////////////////////////////////////////////////////
        // Search list methods
        // Make a new search list
75
        SearchList::SearchList()
        {
          length=0;
          current_count=0;
80        first=last=current_node=0;
          idle.next=NULL;
```

```
      idle.content=NULL;
    }

85  // Reset the list for new search
    void SearchList::resetList(){
      current_node=&idle;
      current_count=0;
      first=last=NULL;
90    statistics.SOSIPolyLines=0;
      statistics.SOSICircles=0;
      statistics.SOSITexts=0;
      statistics.SOSIRefPoints=0;
      statistics.total=0;
95  }

    // Add a node to the list
    void SearchList::addListNode(ObjectRecord * O)
    {
100   if (!first) { // this is the first call
        first=&idle;
        last=first;
      }
      else {
105     if ((last->next)==NULL) { // is this the true last element
          // append a node at the end of the list
               last->next=(struct SearchListNode*)malloc(sizeof(struct SearchListN-
ode));
           last->next->next=NULL;
           //last->next=new SearchListNode;
110       //      last=last->next;

        }
        last=last->next;
      }
115
      last->content=O;
      length++;
      // gathering some statistics...
          statistics.total++;
120
          switch (last->content->type){
          case ObjectRecord::POLYLINE:
        statistics.SOSIPolyLines++;
        break;
125
          case ObjectRecord::CIRCLE:
        statistics.SOSICircles++;
        break;

130       case ObjectRecord::TEXT:
        statistics.SOSITexts++;
        break;

          case ObjectRecord::REFPOINT:
135     statistics.SOSIRefPoints++;
        break;
          }
    }

140 // Make list ready for new traversing
    void SearchList::newTraverse(){
      current_node=first;
      current_count=0;
    }
145
```

```
      // get next item from the searchlist in a traverse operation
      ObjectRecord * SearchList::nextItem()
      {
        ObjectRecord *O;
150       if (current_count <= length
          && current_node != NULL
          && current_node->content != NULL) {
            O=current_node->content;
            current_node=current_node->next;
155         current_count++;
            return O;
          }
          cout << "endtrav\n";
          return NULL;
160   }

      int RTree::searchRectangle(Rect rect){
        if (!search) search=new SearchList();
        search->resetList();
165
        return Search(Root, (Node*) &rect);
      }

      // Search in an index tree or subtree for all data retangles that
170   // overlap the argument rectangle.
      // Returns the number of qualifying data rects.
      //
      int RTree::Search(struct Node *N, struct Node *R)
      {
175       register struct Node *n = N;
          register struct Rect *r = (struct Rect*)R;
          register int hitCount = 0;
          register int i;

180       assert(n);
          assert(n->level >= 0);
          assert(r);

          if (n->level > 0) /* this is an internal node in the tree */
185       {
              for (i=0; i<NODECARD; i++)
                  if (n->branch[i].child && Overlap(r,&n->branch[i].rect))
                      hitCount += Search(n->branch[i].child, R);
          }
190       else /* this is a leaf node */
          {
              for (i=0; i<NODECARD; i++)
                if (n->branch[i].child && Overlap(r,&n->branch[i].rect)) {
                      hitCount++;
195                   search->addListNode((ObjectRecord*) n->branch[i].child);
                }

          }
          return hitCount;
200   }



      // Inserts a new data rectangle into the index structure.
205   // Recursively descends tree, propagates splits back up.
      // Returns 0 if node was not split.  Old node updated.
      // If node was split, returns 1 and sets the pointer pointed to by
      // new_node to point to the new node.  Old node updated to become one of two.
      // The level argument specifies the number of steps up from the leaf
210   // level to insert; e.g. a data rectangle goes in at level = 0.
```

```
     //
     int RTree::InsertRect2(struct Rect *r,
             int tid, struct Node *n, struct Node **new_node, int level)
     {
215  /*
         register struct Rect *r = R;
         register int tid = Tid;
         register struct Node *n = N, **new_node = New_node;
         register int level = Level;
220  */

         register int i;
         struct Branch b;
         struct Node *n2;
225
         assert(r && n && new_node);
         assert(level >= 0 && level <= n->level);

         // Still above level for insertion, go down tree recursively
230      //
         if (n->level > level)
         {
             i = PickBranch(r, n);
             if (!InsertRect2(r, tid, n->branch[i].child, &n2, level))
235          {
                 // child was not split
                 //
                 n->branch[i].rect =
                     CombineRect(r,&(n->branch[i].rect));
240              return 0;
             }
             else    // child was split
             {
                 n->branch[i].rect = NodeCover(n->branch[i].child);
245              b.child = n2;
                 b.rect = NodeCover(n2);
                 return AddBranch(&b, n, new_node);
             }
         }
250
         // Have reached level for insertion. Add rect, split if necessary
         //
         else if (n->level == level)
         {
255          b.rect = *r;
             b.child = (struct Node *) tid;
             /* child field of leaves contains tid of data record */
             return AddBranch(&b, n, new_node);
         }
260      else
         {
             /* Not supposed to happen */
             assert (FALSE);
             return 0;
265      }
     }



270  // Insert a data rectangle into an index structure.
     // InsertRect provides for splitting the root;
     // returns 1 if root was split, 0 if it was not.
     // The level argument specifies the number of steps up from the leaf
     // level to insert; e.g. a data rectangle goes in at level = 0.
275  // InsertRect2 does the recursion.
```

```
         //
         int RTree::InsertRect(struct Rect *R, int Tid, struct Node **Root, int Level)
         {
             register struct Rect *r = R;
280          register int tid = Tid;
             register struct Node **root = Root;
             register int level = Level;
             register int i;
             register struct Node *newroot;
285          struct Node *newnode;
             struct Branch b;
             int result;

             assert(r && root);
290          assert(level >= 0 && level <= (*root)->level);
             for (i=0; i<NUMDIMS; i++)
                 assert(r->boundary[i] <= r->boundary[NUMDIMS+i]);

             if (InsertRect2(r, tid, *root, &newnode, level))  /* root split */
295          {
                 newroot = NewNode();  /* grow a new root, & tree taller */
                 newroot->level = (*root)->level + 1;
                 b.rect = NodeCover(*root);
                 b.child = *root;
300              AddBranch(&b, newroot, NULL);
                 b.rect = NodeCover(newnode);
                 b.child = newnode;
                 AddBranch(&b, newroot, NULL);
                 *root = newroot;
305              result = 1;
             }
             else
                 result = 0;

310          return result;
         }



315
         // Allocate space for a node in the list used in DeletRect to
         // store Nodes that are too empty.
         //
         struct ListNode * RTree::NewListNode()
320      {
             return (struct ListNode *) malloc(sizeof(struct ListNode));
             //return new ListNode;
         }

325
         void RTree::FreeListNode(struct ListNode *p)
         {
             //free(p);
             delete(p);
330      }



         // Add a node to the reinsertion list.  All its branches will later
335      // be reinserted into the index structure.
         //
         void RTree::ReInsert(struct Node *n, struct ListNode **ee)
         {
             register struct ListNode *l;
340
```

```
        l = NewListNode();
        l->node = n;
        l->next = *ee;
        *ee = l;
345 }


    // Delete a rectangle from non-root part of an index structure.
    // Called by DeleteRect.  Descends tree recursively,
350 // merges branches on the way back up.
    //
    int
    RTree::DeleteRect2(struct Rect *R, int Tid, struct Node *N, struct ListNode **Ee)
    {
355     register struct Rect *r = R;
        register int tid = Tid;
        register struct Node *n = N;
        register struct ListNode **ee = Ee;
        register int i;
360
        assert(r && n && ee);
        assert(tid >= 0);
        assert(n->level >= 0);

365     if (n->level > 0)  // not a leaf node
        {
            for (i = 0; i < NODECARD; i++)
            {
            if (n->branch[i].child && Overlap(r, &(n->branch[i].rect)))
370         {
                if (!DeleteRect2(r, tid, n->branch[i].child, ee))
                {
                    if (n->branch[i].child->count >= MinFill)
                        n->branch[i].rect = NodeCover(
375                         n->branch[i].child);
                    else
                    {
                        // not enough entries in child,
                        // eliminate child node
380                     //
                        ReInsert(n->branch[i].child, ee);
                        DisconnectBranch(n, i);
                    }
                    return 0;
385             }
            }
            }
            return 1;
        }
390     else  // a leaf node
        {
            for (i = 0; i < NODECARD; i++)
            {
                if (n->branch[i].child &&
395                 n->branch[i].child == (struct Node *) tid)
                {
                    DisconnectBranch(n, i);
                    return 0;
                }
400         }
            return 1;
        }
    }

405
```

```
    // Delete a data rectangle from an index structure.
    // Pass in a pointer to a Rect, the tid of the record, ptr to ptr to root node.
    // Returns 1 if record not found, 0 if success.
410 // DeleteRect provides for eliminating the root.
    //
    int RTree::DeleteRect(struct Rect *R, int Tid, struct Node**Nn)
    {
        register struct Rect *r = R;
415     register int tid = Tid;
        register struct Node **nn = Nn;
        register int i;
        register struct Node *tmp_node_ptr;
        struct ListNode *reInsertList = NULL;
420     register struct ListNode *e;

        assert(r && nn);
        assert(*nn);
        assert(tid >= 0);
425
        if (!DeleteRect2(r, tid, *nn, &reInsertList))
        {
            /* found and deleted a data item */

430         /* reinsert any branches from eliminated nodes */
            while (reInsertList)
            {
                tmp_node_ptr = reInsertList->node;
                for (i = 0; i < NODECARD; i++)
435             {
                    if (tmp_node_ptr->branch[i].child)
                    {
                        InsertRect(
                            &(tmp_node_ptr->branch[i].rect),
440                         (int) tmp_node_ptr->branch[i].child,
                            nn,
                            tmp_node_ptr->level);
                    }
                }
445             e = reInsertList;
                reInsertList = reInsertList->next;
                FreeNode(e->node);
                FreeListNode(e);
            }
450
            /* check for redundant root (not leaf, 1 child) and eliminate
            */
            if ((*nn)->count == 1 && (*nn)->level > 0)
            {
455             for (i = 0; i < NODECARD; i++)
                {
                    tmp_node_ptr = (*nn)->branch[i].child;
                    if(tmp_node_ptr)
                        break;
460             }
                assert(tmp_node_ptr);
                FreeNode(*nn);
                *nn = tmp_node_ptr;
            }
465         return 0;
        }
        else
        {
            return 1;
470     }
```

```
    }


    // Initialize one branch cell in a node.
475 //
    void RTree::InitBranch(struct Branch *b)
    {
        InitRect(&(b->rect));
        b->child = NULL;
480 }



    // Initialize a Node structure.
485 //
    void RTree::InitNode(struct Node *N)
    {
        register struct Node *n = N;
        register int i;
490     n->count = 0;
        n->level = -1;
        for (i = 0; i < NODECARD; i++)
            InitBranch(&(n->branch[i]));
    }
495


    // Make a new node and initialize to have all branch cells empty.
    //
500 struct Node * RTree::NewNode()
    {
        register struct Node *n;

        //n = new Node;
505     n = (struct Node*)malloc(sizeof(struct Node));
        assert(n);
        InitNode(n);
        return n;
    }
510

    void RTree::FreeNode(struct Node *p)
    {
        assert(p);
515     delete p;
        //free(p);
    }


520 // Find the smallest rectangle that includes all rectangles in
    // branches of a node.
    //
    struct Rect RTree::NodeCover(struct Node *N)
    {
525     register struct Node *n = N;
        register int i, first_time=1;
        struct Rect r;
        assert(n);

530     InitRect(&r);
        for (i = 0; i < NODECARD; i++)
            if (n->branch[i].child)
            {
                if (first_time)
535             {
```

```
                        r = n->branch[i].rect;
                        first_time = 0;
                    }
                    else
540                     r = CombineRect(&r, &(n->branch[i].rect));
                }
        return r;
    }

545


    // Pick a branch.  Pick the one that will need the smallest increase
    // in area to accomodate the new rectangle.  This will result in the
    // least total area for the covering rectangles in the current node.
550 // In case of a tie, pick the one which was smaller before, to get
    // the best resolution when searching.
    //
    int RTree::PickBranch(struct Rect *R, struct Node *N)
    {
555     register struct Rect *r = R;
        register struct Node *n = N;
        register struct Rect *rr;
        register int i, first_time=1;
        RectReal increase, bestIncr=(RectReal)-1, area, bestArea;
560     int best;
        struct Rect tmp_rect;
        assert(r && n);

        for (i=0; i<NODECARD; i++)
565     {
            if (n->branch[i].child)
            {
                rr = &n->branch[i].rect;
                area = RectSphericalVolume(rr);
570             tmp_rect = CombineRect(r, rr);
                increase = RectSphericalVolume(&tmp_rect) - area;
                if (increase < bestIncr || first_time)
                {
                    best = i;
575                 bestArea = area;
                    bestIncr = increase;
                    first_time = 0;
                }
                else if (increase == bestIncr && area < bestArea)
580             {
                    best = i;
                    bestArea = area;
                    bestIncr = increase;
                }
585         }
        }
        return best;
    }

590


    // Add a branch to a node.  Split the node if necessary.
    // Returns 0 if node not split.  Old node updated.
    // Returns 1 if node split, sets *new_node to address of new node.
595 // Old node updated, becomes one of two.
    //
    int RTree::AddBranch(struct Branch *B, struct Node *N, struct Node **New_node)
    {
        register struct Branch *b = B;
600     register struct Node *n = N;
```

```
        register struct Node **new_node = New_node;
        register int i;

        assert(b);
605     assert(n);

        if (n->count < NODECARD)  /* split won't be necessary */
        {
            for (i = 0; i < NODECARD; i++)  /* find empty branch */
610         {
                if (n->branch[i].child == NULL)
                {
                    n->branch[i] = *b;
                    n->count++;
615                 break;
                }
            }
            assert(i<NODECARD);
            return 0;
620     }
        else
        {
            assert(new_node);
            SplitNode(n, b, new_node);
625         return 1;
        }
    }


630
    // Disconnect a dependent node.
    //
    void RTree::DisconnectBranch(struct Node *n, int i)
    {
635     assert(n && i>=0 && i<NODECARD);
        assert(n->branch[i].child);

        InitBranch(&(n->branch[i]));
        n->count--;
640 }


    /*-----------------------------------------------------------------------------
    | Initialize a rectangle to have all 0 coordinates.
645 -----------------------------------------------------------------------------*/
    void RTree::InitRect(struct Rect *R)
    {
        register struct Rect *r = R;
        register int i;
650     for (i=0; i<NUMSIDES; i++)
            r->boundary[i] = (RectReal)0;
    }


655 /*-----------------------------------------------------------------------------
    | Return a rect whose first low side is higher than its opposite side -
    | interpreted as an undefined rect.
    -----------------------------------------------------------------------------*/
    struct Rect RTree::NullRect()
660 {
        struct Rect r;
        register int i;

        r.boundary[0] = (RectReal)1;
665     r.boundary[NUMDIMS] = (RectReal)-1;
```

```
    for (i=1; i<NUMDIMS; i++)
        r.boundary[i] = r.boundary[i+NUMDIMS] = (RectReal)0;
    return r;
}
670

/*-----------------------------------------------------------------------------
| Fill in the boundaries for a random search rectangle.
| Pass in a pointer to a rect that contains all the data,
675 | and a pointer to the rect to be filled in.
| Generated rect is centered randomly anywhere in the data area,
| and has size from 0 to the size of the data area in each dimension,
| i.e. search rect can stick out beyond data area.
-----------------------------------------------------------------------------*/
680 void RTree::SearchRect(struct Rect *Search, struct Rect *Data)
{
    register struct Rect *search = Search, *data = Data;
    register int i, j;
    register RectReal size, center;
685
    assert(search);
    assert(data);

    for (i=0; i<NUMDIMS; i++)
690 {
        j = i + NUMDIMS;  // index for high side boundary
        if (data->boundary[i] > -BIG_NUM &&
            data->boundary[j] < BIG_NUM)
        {
695         size = (drand48() * (data->boundary[j] -
                     data->boundary[i] + 1)) / 2;
            center = data->boundary[i] + drand48() *
                (data->boundary[j] - data->boundary[i] + 1);
            search->boundary[i] = center - size/2;
700         search->boundary[j] = center + size/2;
        }
        else  // some open boundary, search entire dimension
        {
            search->boundary[i] = -BIG_NUM;
705         search->boundary[j] = BIG_NUM;
        }
    }
}
/*-----------------------------------------------------------------------------
710 | Calculate the n-dimensional volume of a rectangle
-----------------------------------------------------------------------------*/
RectReal RTree::RectVolume(struct Rect *R)
{
    register struct Rect *r = R;
715 register int i;
    register RectReal volume = (RectReal)1;

    assert(r);
    if (Undefined(r))
720     return (RectReal)0;

    for(i=0; i<NUMDIMS; i++)
        volume *= r->boundary[i+NUMDIMS] - r->boundary[i];
    assert(volume >= 0.0);
725 return volume;
}

/*-----------------------------------------------------------------------------
| Calculate the n-dimensional volume of the bounding sphere of a rectangle
730 -----------------------------------------------------------------------------*/
```

```
      /*
       * The volumes of the unit spheres for each dimension.
       * Generated by sphvol.c
       */
735   const double UnitSphereVolumes[] = {
          0.000000,  /* dimension   0 */
          2.000000,  /* dimension   1 */
          3.141593,  /* dimension   2 */
          4.188790,  /* dimension   3 */
740       4.934802,  /* dimension   4 */
          5.263789,  /* dimension   5 */
          5.167713,  /* dimension   6 */
          4.724766,  /* dimension   7 */
          4.058712,  /* dimension   8 */
745       3.298509,  /* dimension   9 */
          2.550164,  /* dimension  10 */
          1.884104,  /* dimension  11 */
          1.335263,  /* dimension  12 */
          0.910629,  /* dimension  13 */
750       0.599265,  /* dimension  14 */
          0.381443,  /* dimension  15 */
          0.235331,  /* dimension  16 */
          0.140981,  /* dimension  17 */
          0.082146,  /* dimension  18 */
755       0.046622,  /* dimension  19 */
          0.025807,  /* dimension  20 */
      };


760   #define UnitSphereVolume UnitSphereVolumes[NUMDIMS]

      #if 0
      /*
       * A fast approximation to the volume of the bounding sphere for the
765    * given Rect. By Paul B.
       */
      RectReal RTree::RectSphericalVolume(struct Rect *R)
      {
          register struct Rect *r = R;
770       register int i;
          RectReal maxsize=(RectReal)0, c_size;

          assert(r);
          if (Undefined(r))
775           return (RectReal)0;
          for (i=0; i<NUMDIMS; i++) {
              c_size = r->boundary[i+NUMDIMS] - r->boundary[i];
              if (c_size > maxsize)
                  maxsize = c_size;
780       }
          return (RectReal)(pow(maxsize/2, NUMDIMS) * UnitSphereVolume);
      }
      #endif

785   /*
       * The exact volume of the bounding sphere for the given Rect.
       */
      RectReal RTree::RectSphericalVolume(struct Rect *R)
      {
790       register struct Rect *r = R;
          register int i;
          register double sum_of_squares=0, radius;

          assert(r);
795       if (Undefined(r))
```

```
            return (RectReal)0;
        for (i=0; i<NUMDIMS; i++) {
            double half_extent =
                (r->boundary[i+NUMDIMS] - r->boundary[i]) / 2;
800         sum_of_squares += half_extent * half_extent;
        }
        radius = sqrt(sum_of_squares);
        return (RectReal)(pow(radius, NUMDIMS) * UnitSphereVolume);
    }
805

    /*-----------------------------------------------------------------------------
    | Calculate the n-dimensional surface area of a rectangle
    -----------------------------------------------------------------------------*/
810 RectReal RTree::RectSurfaceArea(struct Rect *R)
    {
        register struct Rect *r = R;
        register int i, j;
        register RectReal sum = (RectReal)0;
815
        assert(r);
        if (Undefined(r))
            return (RectReal)0;

820     for (i=0; i<NUMDIMS; i++) {
            RectReal face_area = (RectReal)1;
            for (j=0; j<NUMDIMS; j++)
                /* exclude i extent from product in this dimension */
                if(i != j) {
825                 RectReal j_extent =
                        r->boundary[j+NUMDIMS] - r->boundary[j];
                    face_area *= j_extent;
                }
            sum += face_area;
830     }
        return 2 * sum;
    }


835
    /*-----------------------------------------------------------------------------
    | Combine two rectangles, make one that includes both.
    -----------------------------------------------------------------------------*/
    struct Rect RTree::CombineRect(struct Rect *R, struct Rect *Rr)
840 {
        register struct Rect *r = R, *rr = Rr;
        register int i, j;
        struct Rect new_rect;
        assert(r && rr);
845
        if (Undefined(r))
            return *rr;

        if (Undefined(rr))
850         return *r;

        for (i = 0; i < NUMDIMS; i++)
        {
            new_rect.boundary[i] = MIN(r->boundary[i], rr->boundary[i]);
855         j = i + NUMDIMS;
            new_rect.boundary[j] = MAX(r->boundary[j], rr->boundary[j]);
        }
        return new_rect;
    }
860
```

```
     /*-----------------------------------------------------------------------------
     | Decide whether two rectangles overlap.
     -----------------------------------------------------------------------------*/
865  int RTree::Overlap(struct Rect *R, struct Rect *S)
     {
         register struct Rect *r = R, *s = S;
         register int i, j;
         assert(r && s);
870
         for (i=0; i<NUMDIMS; i++)
         {
             j = i + NUMDIMS;  /* index for high sides */
             if (r->boundary[i] > s->boundary[j] ||
875              s->boundary[i] > r->boundary[j])
             {
                 return FALSE;
             }
         }
880      return TRUE;
     }


     /*-----------------------------------------------------------------------------
885  | Decide whether rectangle r is contained in rectangle s.
     -----------------------------------------------------------------------------*/
     int RTree::Contained(struct Rect *R, struct Rect *S)
     {
         register struct Rect *r = R, *s = S;
890      register int i, j, result;
         assert((int)r && (int)s);

         // undefined rect is contained in any other
         //
895      if (Undefined(r))
             return TRUE;

         // no rect (except an undefined one) is contained in an undef rect
         //
900      if (Undefined(s))
             return FALSE;

         result = TRUE;
         for (i = 0; i < NUMDIMS; i++)
905      {
             j = i + NUMDIMS;  /* index for high sides */
             result = result
                 && r->boundary[i] >= s->boundary[i]
                 && r->boundary[j] <= s->boundary[j];
910      }
         return result;
     }

     /*
915   *                    SPHERE VOLUME
      *                    by Daniel Green
      *
      * Calculates and prints the volumes of the unit hyperspheres for
      * dimensions zero through the given value, or 9 by default.
920   * Prints in the form of a C array of double called sphere_volumes.
      *
      * From the recurance formula in "Regular Polytopes" by Coxteter as follows:
      *
      * Let S(n) be the n-1 dimensional surface area of an n dimensional sphere.
925   * Then S(n+2) = (2 * Pi * S(n)) / n, and the volume of the sphere is
```

```
         * simply Sn/n. The formula for the expanded expression is described
         * by the code below.
         *
         * Multiply the output volumes by R^n to get the volume of an n
930      * dimensional sphere of radius R.
         */

        #ifndef PI
        #   define PI 3.141592654
935     #endif
        #ifndef ODD
        #   define ODD(a) ((a) & 1)
        #endif
        #ifndef EVEN
940     #   define EVEN(a) (!ODD(a))
        #endif

        void RTree::print_volume(int dimension, double volume)
        {
945         printf("\t%.6f,  /* dimension %3d */\n", volume, dimension);
        }


        int RTree::main(int argc, char *argv[])
950     {
            int dim, max_dims=9;
            double products_of_odds=1, products_of_evens=1;

            if(2 == argc)
955             max_dims = atoi(argv[1]);

            printf("double sphere_volumes[] = {\n");
            print_volume(0, 0.0);
            print_volume(1, 2.0);
960         for(dim=2; dim<max_dims+1; dim++) {
                double numerator, denominator;
                numerator = pow(2, ceil(dim/2.)) * pow(PI, floor(dim/2.));
                if(EVEN(dim))
                    denominator = products_of_evens *= dim;
965             else
                    denominator = products_of_odds *= dim;
                print_volume(dim, numerator / denominator);
            }

970         printf("};\n");
            return 0;
        }

        #ifdef LINEAR_INSERT
975     /*-----------------------------------------------------------------------------
        | Load branch buffer with branches from full node plus the extra branch.
        -----------------------------------------------------------------------------*/
        void RTree::GetBranches(struct Node *N, struct Branch *B)
        {
980         register struct Node *n = N;
            register struct Branch *b = B;
            register int i;

            assert(n);
985         assert(b);

            /* load the branch buffer */
            for (i=0; i<NODECARD; i++)
            {
990             assert(n->branch[i].child);  /* every entry should be full */
```

```
            BranchBuf[i] = n->branch[i];
        }
        BranchBuf[NODECARD] = *b;

 995    /* calculate rect containing all in the set */
        CoverSplit = BranchBuf[0].rect;
        for (i=1; i<NODECARD+1; i++)
        {
            CoverSplit = CombineRect(&CoverSplit, &BranchBuf[i].rect);
1000    }

        InitNode(n);
    }

1005

    /*-----------------------------------------------------------------------------
    | Initialize a PartitionVars structure.
    -----------------------------------------------------------------------------*/
1010 void RTree::InitPVars(struct PartitionVars *P)
    {
        register struct PartitionVars *p = P;
        register int i;
        assert(p);
1015
        p->count[0] = p->count[1] = 0;
        for (i=0; i<NODECARD+1; i++)
        {
            p->taken[i] = FALSE;
1020        p->partition[i] = -1;
        }
    }


1025
    /*-----------------------------------------------------------------------------
    | Put a branch in one of the groups.
    -----------------------------------------------------------------------------*/
    void RTree::Classify(int i, int group, struct PartitionVars *p)
1030 {
        assert(p);
        assert(!p->taken[i]);

        p->partition[i] = group;
1035    p->taken[i] = TRUE;

        if (p->count[group] == 0)
            p->cover[group] = BranchBuf[i].rect;
        else
1040        p->cover[group] = CombineRect(&BranchBuf[i].rect,
                        &p->cover[group]);
        p->area[group] = RectSphericalVolume(&p->cover[group]);
        p->count[group]++;
    }
1045


    /*-----------------------------------------------------------------------------
    | Pick two rects from set to be the first elements of the two groups.
1050 | Pick the two that are separated most along any dimension, or overlap least.
    | Distance for separation or overlap is measured modulo the width of the
    | space covered by the entire set along that dimension.
    -----------------------------------------------------------------------------*/
    void RTree::PickSeeds(struct PartitionVars *P)
1055 {
```

```
        register struct PartitionVars *p = P;
        register int i, dim, high;
        register struct Rect *r, *rlow, *rhigh;
        register float w, separation, bestSep;
1060    RectReal width[NUMDIMS];
        int leastUpper[NUMDIMS], greatestLower[NUMDIMS];
        int seed0, seed1;
        assert(p);

1065    for (dim=0; dim<NUMDIMS; dim++)
        {
            high = dim + NUMDIMS;

            /* find the rectangles farthest out in each direction
1070         * along this dimens */
            greatestLower[dim] = leastUpper[dim] = 0;
            for (i=1; i<NODECARD+1; i++)
            {
                r = &BranchBuf[i].rect;
1075            if (r->boundary[dim] >
                    BranchBuf[greatestLower[dim]].rect.boundary[dim])
                {
                    greatestLower[dim] = i;
                }
1080            if (r->boundary[high] <
                    BranchBuf[leastUpper[dim]].rect.boundary[high])
                {
                    leastUpper[dim] = i;
                }
1085        }

            /* find width of the whole collection along this dimension */
            width[dim] = CoverSplit.boundary[high] -
                     CoverSplit.boundary[dim];
1090    }

        /* pick the best separation dimension and the two seed rects */
        for (dim=0; dim<NUMDIMS; dim++)
        {
1095        high = dim + NUMDIMS;

            /* divisor for normalizing by width */
            assert(width[dim] >= 0);
            if (width[dim] == 0)
1100            w = (RectReal)1;
            else
                w = width[dim];

            rlow = &BranchBuf[leastUpper[dim]].rect;
1105        rhigh = &BranchBuf[greatestLower[dim]].rect;
            if (dim == 0)
            {
                seed0 = leastUpper[0];
                seed1 = greatestLower[0];
1110            separation = bestSep =
                    (rhigh->boundary[0] -
                     rlow->boundary[NUMDIMS]) / w;
            }
            else
1115        {
                separation =
                    (rhigh->boundary[dim] -
                    rlow->boundary[dim+NUMDIMS]) / w;
                if (separation > bestSep)
1120            {
```

```
                    seed0 = leastUpper[dim];
                    seed1 = greatestLower[dim];
                    bestSep = separation;
                }
1125        }
        }

        if (seed0 != seed1)
        {
1130        Classify(seed0, 0, p);
            Classify(seed1, 1, p);
        }
    }

1135

    /*-----------------------------------------------------------------------------
    | Put each rect that is not already in a group into a group.
    | Process one rect at a time, using the following hierarchy of criteria.
1140 | In case of a tie, go to the next test.
    | 1) If one group already has the max number of elements that will allow
    | the minimum fill for the other group, put r in the other.
    | 2) Put r in the group whose cover will expand less.  This automatically
    | takes care of the case where one group cover contains r.
1145 | 3) Put r in the group whose cover will be smaller.  This takes care of the
    | case where r is contained in both covers.
    | 4) Put r in the group with fewer elements.
    | 5) Put in group 1 (arbitrary).
    |
1150 | Also update the covers for both groups.
    -----------------------------------------------------------------------------*/
    void RTree::Pigeonhole(struct PartitionVars *P)
    {
        register struct PartitionVars *p = P;
1155    struct Rect newCover[2];
        register int i, group;
        RectReal newArea[2], increase[2];

        for (i=0; i<NODECARD+1; i++)
1160    {
            if (!p->taken[i])
            {
                /* if one group too full, put rect in the other */
                if (p->count[0] >= NODECARD+1-MinFill)
1165            {
                    Classify(i, 1, p);
                    continue;
                }
                else if (p->count[1] >= NODECARD+1-MinFill)
1170            {
                    Classify(i, 0, p);
                    continue;
                }

1175            /* find areas of the two groups' old and new covers */
                for (group=0; group<2; group++)
                {
                    if (p->count[group]>0)
                        newCover[group] = CombineRect(
1180                        &BranchBuf[i].rect,
                            &p->cover[group]);
                    else
                        newCover[group] = BranchBuf[i].rect;
                    newArea[group] = RectSphericalVolume(
1185                            &newCover[group]);
```

```
                    increase[group] = newArea[group]-p->area[group];
                }

                /* put rect in group whose cover will expand less */
1190            if (increase[0] < increase[1])
                    Classify(i, 0, p);
                else if (increase[1] < increase[0])
                    Classify(i, 1, p);

1195            /* put rect in group that will have a smaller cover */
                else if (p->area[0] < p->area[1])
                    Classify(i, 0, p);
                else if (p->area[1] < p->area[0])
                    Classify(i, 1, p);
1200
                /* put rect in group with fewer elements */
                else if (p->count[0] < p->count[1])
                    Classify(i, 0, p);
                else
1205                Classify(i, 1, p);
            }
        }
        assert(p->count[0] + p->count[1] == NODECARD + 1);
    }
1210


    /*-----------------------------------------------------------------------------
    | Method 0 for finding a partition:
1215 | First find two seeds, one for each group, well separated.
    | Then put other rects in whichever group will be smallest after addition.
    -----------------------------------------------------------------------------*/
    void RTree::MethodZero(struct PartitionVars *p)
    {
1220    InitPVars(p);
        PickSeeds(p);
        Pigeonhole(p);
    }

1225



    /*-----------------------------------------------------------------------------
    | Copy branches from the buffer into two nodes according to the partition.
1230 -----------------------------------------------------------------------------*/
    void RTree::LoadNodes(struct Node *N, struct Node *Q,
                struct PartitionVars *P)
    {
        register struct Node *n = N, *q = Q;
1235    register struct PartitionVars *p = P;
        register int i;
        assert(n);
        assert(q);
        assert(p);
1240
        for (i=0; i<NODECARD+1; i++)
        {
            if (p->partition[i] == 0)
                AddBranch(&BranchBuf[i], n, NULL);
1245        else if (p->partition[i] == 1)
                AddBranch(&BranchBuf[i], q, NULL);
            else
                assert(FALSE);
        }
1250 }
```

```
        /*-----------------------------------------------------------------------
1255    | Split a node.
        | Divides the nodes branches and the extra one between two nodes.
        | Old node is one of the new ones, and one really new one is created.
        -----------------------------------------------------------------------*/
        void RTree::SplitNode(struct Node *n, struct Branch *b, struct Node **nn)
1260    {
            register struct PartitionVars *p;
            register int level;
            RectReal area;

1265        assert(n);
            assert(b);

            /* load all the branches into a buffer, initialize old node */
            level = n->level;
1270        GetBranches(n, b);

            /* find partition */
            p = &Partitions[0];
            MethodZero(p);
1275
            /* record how good the split was for statistics */
            area = p->area[0] + p->area[1];

            /* put branches from buffer in 2 nodes according to chosen partition */
1280        *nn = NewNode();
            (*nn)->level = n->level = level;
            LoadNodes(n, *nn, p);
            assert(n->count + (*nn)->count == NODECARD+1);
        }
1285
        #else // Quad-method

        /*-----------------------------------------------------------------------
        | Load branch buffer with branches from full node plus the extra branch.
1290    -----------------------------------------------------------------------*/
        void RTree::GetBranches(struct Node *n, struct Branch *b)
        {
            register int i;

1295        assert(n);
            assert(b);

            /* load the branch buffer */
            for (i=0; i<NODECARD; i++)
1300        {
                assert(n->branch[i].child); /* n should have every entry full */
                BranchBuf[i] = n->branch[i];
            }
            BranchBuf[NODECARD] = *b;
1305
            /* calculate rect containing all in the set */
            CoverSplit = BranchBuf[0].rect;
            for (i=1; i<NODECARD+1; i++)
            {
1310            CoverSplit = CombineRect(&CoverSplit, &BranchBuf[i].rect);
            }
            CoverSplitArea = RectSphericalVolume(&CoverSplit);

            InitNode(n);
1315    }
```

```
1320    /*-----------------------------------------------------------------------------
        | Put a branch in one of the groups.
        -----------------------------------------------------------------------------*/
        void RTree::Classify(int i, int group, struct PartitionVars *p)
        {
1325        assert(p);
            assert(!p->taken[i]);

            p->partition[i] = group;
            p->taken[i] = TRUE;
1330
            if (p->count[group] == 0)
                p->cover[group] = BranchBuf[i].rect;
            else
                p->cover[group] =
1335                CombineRect(&BranchBuf[i].rect, &p->cover[group]);
            p->area[group] = RectSphericalVolume(&p->cover[group]);
            p->count[group]++;
        }

1340



        /*-----------------------------------------------------------------------------
        | Pick two rects from set to be the first elements of the two groups.
1345    | Pick the two that waste the most area if covered by a single rectangle.
        -----------------------------------------------------------------------------*/
        void RTree::PickSeeds(struct PartitionVars *p)
        {
            register int i, j, seed0, seed1;
1350        RectReal worst, waste, area[NODECARD+1];

            for (i=0; i<NODECARD+1; i++)
                area[i] = RectSphericalVolume(&BranchBuf[i].rect);

1355        worst = -CoverSplitArea - 1;
            for (i=0; i<NODECARD; i++)
            {
                for (j=i+1; j<NODECARD+1; j++)
                {
1360                struct Rect one_rect = CombineRect(
                                &BranchBuf[i].rect,
                                &BranchBuf[j].rect);
                    waste = RectSphericalVolume(&one_rect) -
                            area[i] - area[j];
1365                if (waste > worst)
                    {
                        worst = waste;
                        seed0 = i;
                        seed1 = j;
1370                }
                }
            }
            Classify(seed0, 0, p);
            Classify(seed1, 1, p);
1375    }




1380    /*-----------------------------------------------------------------------------
```

```
      | Copy branches from the buffer into two nodes according to the partition.
      -----------------------------------------------------------------------------*/
      void RTree::LoadNodes(struct Node *n, struct Node *q,
                  struct PartitionVars *p)
1385  {
          register int i;
          assert(n);
          assert(q);
          assert(p);
1390
          for (i=0; i<NODECARD+1; i++)
          {
              assert(p->partition[i] == 0 || p->partition[i] == 1);
              if (p->partition[i] == 0)
1395              AddBranch(&BranchBuf[i], n, NULL);
              else if (p->partition[i] == 1)
                  AddBranch(&BranchBuf[i], q, NULL);
          }
      }
1400




      /*-----------------------------------------------------------------------------
1405  | Initialize a PartitionVars structure.
      -----------------------------------------------------------------------------*/
      void RTree::InitPVars(struct PartitionVars *p)
      {
          register int i;
1410      assert(p);

          p->count[0] = p->count[1] = 0;
          p->cover[0] = p->cover[1] = NullRect();
          p->area[0] = p->area[1] = (RectReal)0;
1415      for (i=0; i<NODECARD+1; i++)
          {
              p->taken[i] = FALSE;
              p->partition[i] = -1;
          }
1420  }




1425  /*-----------------------------------------------------------------------------
      | Method #0 for choosing a partition:
      | As the seeds for the two groups, pick the two rects that would waste the
      | most area if covered by a single rectangle, i.e. evidently the worst pair
      | to have in the same group.
1430  | Of the remaining, one at a time is chosen to be put in one of the two groups.
      | The one chosen is the one with the greatest difference in area expansion
      | depending on which group - the rect most strongly attracted to one group
      | and repelled from the other.
      | If one group gets too full (more would force other group to violate min
1435  | fill requirement) then other group gets the rest.
      | These last are the ones that can go in either group most easily.
      -----------------------------------------------------------------------------*/
      void RTree::MethodZero(struct PartitionVars *p)
      {
1440      register int i;
          RectReal biggestDiff;
          register int group, chosen, betterGroup;
          assert(p);

1445      InitPVars(p);
```

```
            PickSeeds(p);

            while (p->count[0] + p->count[1] < NODECARD + 1
                && p->count[0] < NODECARD + 1 - MinFill
1450            && p->count[1] < NODECARD + 1 - MinFill)
            {
                biggestDiff = (RectReal)-1.;
                for (i=0; i<NODECARD+1; i++)
                {
1455                if (!p->taken[i])
                    {
                        struct Rect *r, rect_0, rect_1;
                        RectReal growth0, growth1, diff;

1460                    r = &BranchBuf[i].rect;
                        rect_0 = CombineRect(r, &p->cover[0]);
                        rect_1 = CombineRect(r, &p->cover[1]);
                        growth0 = RectSphericalVolume(
                                &rect_0)-p->area[0];
1465                    growth1 = RectSphericalVolume(
                                &rect_1)-p->area[1];
                        diff = growth1 - growth0;
                        if (diff >= 0)
                            group = 0;
1470                    else
                        {
                            group = 1;
                            diff = -diff;
                        }
1475
                        if (diff > biggestDiff)
                        {
                            biggestDiff = diff;
                            chosen = i;
1480                        betterGroup = group;
                        }
                        else if (diff==biggestDiff &&
                             p->count[group]<p->count[betterGroup])
                        {
1485                        chosen = i;
                            betterGroup = group;
                        }
                    }
                }
1490            Classify(chosen, betterGroup, p);
            }

            /* if one group too full, put remaining rects in the other */
            if (p->count[0] + p->count[1] < NODECARD + 1)
1495        {
                if (p->count[0] >= NODECARD + 1 - MinFill)
                    group = 1;
                else
                    group = 0;
1500            for (i=0; i<NODECARD+1; i++)
                {
                    if (!p->taken[i])
                        Classify(i, group, p);
                }
1505        }

            assert(p->count[0] + p->count[1] == NODECARD+1);
            assert(p->count[0] >= MinFill && p->count[1] >= MinFill);
        }
1510
```

```
        #endif // LINEAR_METHOD

        /*-----------------------------------------------------------------------
        | Split a node.
1515    | Divides the nodes branches and the extra one between two nodes.
        | Old node is one of the new ones, and one really new one is created.
        | Tries more than one method for choosing a partition, uses best result.
        -----------------------------------------------------------------------*/
        extern void RTree::SplitNode(struct Node *n, struct Branch *b, struct Node **nn)
1520    {
            register struct PartitionVars *p;
            register int level;

            assert(n);
1525        assert(b);

            /* load all the branches into a buffer, initialize old node */
            level = n->level;
            GetBranches(n, b);
1530
            /* find partition */
            p = &Partitions[0];
            MethodZero(p);

1535        /*
             * put branches from buffer into 2 nodes
             * according to chosen partition
             */
            *nn = NewNode();
1540        (*nn)->level = n->level = level;
            LoadNodes(n, *nn, p);
            assert(n->count+(*nn)->count == NODECARD+1);
        }

1545    struct Rect * RTree::boundingBox(struct Rect *R, struct SOSIPolyLine * G)
        {
          // calculates a rectangle that covers the SOSIPolyLine G
          register i;
          register joints=0;
1550      register float east;
          register float south;
          register float west;
          register float north;
          if (G->joints._length){
1555        west = (float)G->joints._buffer[0].east;
            south= (float)G->joints._buffer[0].north;
            east = (float)G->joints._buffer[0].east;
            north= (float)G->joints._buffer[0].north;
          }
1560      for (i=0;i<G->joints._length;i++){
            west=MIN(west, G->joints._buffer[i].east);
            south=MIN(south, G->joints._buffer[i].north);
            east=MAX(east,  G->joints._buffer[i].east);
            north=MAX(north,  G->joints._buffer[i].north);
1565      }
          // chech the Refreence point as well...
          west=MIN(west, G->theSOSIRefPoint.east);
          south=MIN(south, G->theSOSIRefPoint.north);
          north=MAX(north,  G->theSOSIRefPoint.north);
1570      east=MAX(east,  G->theSOSIRefPoint.east);

          R->boundary[0]=west;
          R->boundary[1]=south;
          R->boundary[2]=east;
1575      R->boundary[3]=north;
```

```
    return R;
  }

  static struct Rect * boundingBox(struct Rect *R, struct SOSICircle * G)
1580  {
    // calculates a rectangle that covers the SOSICircle G
    R->boundary[0]=MIN(G->Gcentre.east - G->Gradius, G->theSOSIRefPoint.east);
    R->boundary[1]=MIN(G->Gcentre.north - G->Gradius, G->theSOSIRefPoint.north);
    R->boundary[2]=MAX(G->Gcentre.east + G->Gradius, G->theSOSIRefPoint.east);
1585    R->boundary[3]=MAX(G->Gcentre.north + G->Gradius, G->theSOSIRefPoint.north);
    return R;
  }

  struct Rect * RTree::boundingBox(struct Rect *R, struct SOSIText * G)
1590  {
    // calculates a rectangle that covers the SOSIText G
  R->boundary[0]=G->Gposition.east;
  R->boundary[1]=G->Gposition.north;
  R->boundary[2]=R->boundary[0]+1000;
1595  R->boundary[3]=R->boundary[1]+1000;
   return R;
  }

  static struct Rect * boundingBox(struct Rect *R, struct SOSIRefPoint * G)
1600  {
    // calculates a rectangle that covers the SOSIRefPoint G

  }

1605
```

stack.h

```
  struct Stackelement{
    int east;
    int north;
    Stackelement * next;
5  };

  class Stack{

  public:
10    Stack();
    int push(int, int);
    Stackelement * pop();
    int getLength();
    void clear();
15    ~Stack();

  private:
    int length;
    int max;
20    Stackelement * theStack;
    Stackelement * top;
    Stackelement * freelist;
  };
```

stack.cc

```
// A simple dynamic stack
#include "stack.h"
#include <stream.h>

// constructor
Stack::Stack(){
  length=0;
  theStack = 0;
  top=theStack;
  freelist=0;
};

// destructor
Stack::~Stack(){
while (pop());
}

void Stack::clear(){
while (pop());
}

int Stack::push(int E, int N){
if (freelist) {
  top=freelist;
  freelist = freelist->next;
}  else top = new Stackelement;
  length++;
  top->east=E;
  top->north=N;
  if (theStack)
    top->next=theStack;
  theStack=top;
  return 0;
}

Stackelement * Stack::pop(){
  Stackelement * P;
  if (theStack && length) {
    P=theStack;
    top=theStack->next;
    theStack->next=freelist;
    freelist = theStack;
    theStack=top;
    length--;
    return P;
  }
  return (void *)0;
}

int Stack::getLength(){
  return length;
}
```

gis_i.h

```
        // gis_i.h
        #ifndef gis_ih
        #define gis_ih

   5    #include "gis.hh"
        #include "mapserver.h"
        #include "RTree.h"

        extern ofstream logg;
  10
        class gis_i:public mapServerBOAImpl {
        public:
          RTree * R;
          gis_i::gis_i();
  15
          virtual SOSIGraphObjects getRegion (const SOSIPoint& NW, const SOSIPoint& SE,
CORBA::Environment &IT_env=CORBA::default_environment);
          virtual SOSIGraphObjects getOrigoRegion (CORBA::Environment &IT_env=COR-
BA::default_environment);
        };
        #endif
  20
```

gis_i.cc

```
        #include <stream.h>
        #include <fstream.h>
        #include <malloc.h>
        #include "gis_i.h"
   5    #include "mapserver.h"

        extern SOSIGraphObjects *theObjects; //mer innhold
        gis_i::gis_i(){
        theObjects=(SOSIGraphObjects*)malloc(sizeof(SOSIGraphObjects));
  10    }



        SOSIGraphObjects gis_i::getRegion (const SOSIPoint& NW, const SOSIPoint& SE,
  15                    CORBA::Environment &IT_env) {
        SOSIGraphObjects theObjects2;;

        ObjectRecord * O;
        Rect searchRect;
  20    register int polyLines_count=0;
        register int circles_count=0;
        register int texts_count=0;
        register int refPoints_count=0;
        searchRect.boundary[0]=(float)NW.east;
  25    searchRect.boundary[1]=(float)NW.north;
        searchRect.boundary[2]=(float)SE.east;
        searchRect.boundary[3]=(float)SE.north;
        theObjects=new SOSIGraphObjects;
        //R->search->resetList();
  30    R->searchRectangle(searchRect);


        theObjects->SOSIPolyLines._buffer = new SOSIPolyLine[R->search->statis-
```

```
tics.SOSIPolyLines];
        theObjects->SOSICircles._buffer = new SOSICircle[R->search->statistics.SOSICir-
cles];
  35    theObjects->SOSITexts._buffer = new SOSIText[R->search->statistics.SOSITexts];
        theObjects->SOSIRefPoints._buffer = new SOSIRefPoint[R->search->statistics.SO-
SIRefPoints];

        R->search->newTraverse();

  40    // copying data from searchlist to return package
        for (int i=0;i<R->search->statistics.total;i++){
          O=R->search->nextItem();
          if (O)
          switch (O->type){
  45          case ObjectRecord::POLYLINE:

            theObjects->SOSIPolyLines[polyLines_count].SOSIDescription=
              O->thePolyLine->SOSIDescription;
            theObjects->SOSIPolyLines[polyLines_count].theSOSIRefPoint=
  50          O->thePolyLine->theSOSIRefPoint;
            cout << theObjects->SOSIPolyLines[polyLines_count].theSOSIRefPoint.east
                 << ", "
                 << theObjects->SOSIPolyLines[polyLines_count].theSOSIRefPoint.north
                 << endl;
  55        theObjects->SOSIPolyLines[polyLines_count++].joints=O->thePolyLine->joints;
            //  cout << " polyLines_count: " << polyLines_count-1 << endl;
            break;

              case ObjectRecord::CIRCLE:
  60
            break;

              case ObjectRecord::TEXT:

  65        theObjects->SOSITexts[texts_count].SOSIDescription=
              O->theText->SOSIDescription;
            theObjects->SOSITexts[texts_count].theSOSIRefPoint=O->theText->theSOSIRef-
Point;
            theObjects->SOSITexts[texts_count].theGText=O->theText->theGText;
             theObjects->SOSITexts[texts_count++].Gposition=O->theText->Gposition;
  70        break;

              case ObjectRecord::REFPOINT:
            theObjects->SOSIRefPoints[refPoints_count].SOSIDescription=
              O->theRefPoint->SOSIDescription;
  75        theObjects->SOSIRefPoints[refPoints_count].theSOSIRefPoint=
              O->theRefPoint->theSOSIRefPoint;

            break;
              }
  80
        }
        // setting the actual sizes of the package:
        theObjects->SOSIPolyLines._length=theObjects->SOSIPolyLines._maximum=R->search-
>statistics.SOSIPolyLines;
        theObjects->SOSICircles._length=theObjects->SOSICircles._maximum=R->search-
>statistics.SOSICircles;
  85    theObjects->SOSITexts._length=theObjects->SOSITexts._maximum=R->search->statis-
tics.SOSITexts;
        theObjects->SOSIRefPoints._length=theObjects->SOSIRefPoints._maximum=R->search-
>statistics.SOSIRefPoints;

        // cleaning up

  90    //R->search->resetList();
```

```
        return *theObjects;

        }

95      SOSIGraphObjects gis_i:: getOrigoRegion (CORBA::Environment &IT_env) {
        SOSIGraphObjects theObjects;
        logg << "getOrigoRegion invocation...\n";
        //theObjects=new SOSIGraphObjects;
        return theObjects;
100     }
```

mapserver.h

```
        #ifndef _SRVMAIN_
        #define _SRVMAIN_

        #include <fstream.h>
5
        struct Rect* boundingBox(struct Rect*, struct SOSIPolyLine*);
        struct Rect* boundingBox(struct Rect*, struct SOSIText*);
        struct Rect* boundingBox(struct Rect*, struct SOSICircle*);
        struct Rect* boundingBox(struct Rect*, struct SOSIRefPoint*);
10
        #endif
```

mapserver.cc

```
        // mapserver.cc

        #include <stream.h>
        #include <fstream.h>
5       #include "gis_i.h"
        #include "mapserver.h"
        #include "sosi.h"
        #include "sosi.tab.h"
        #include "getopt.h"
10      #include <stdio.h>
        #include <malloc.h>
        #include "dbm.h"
        #include <sys/types.h>
        #include <sys/stat.h>
15      #include <sys/fcntl.h>

        #ifndef MIN
        #define MIN(a, b) ((a) < (b) ? (a) : (b))
        #endif
20      #ifndef MAX
        #define MAX(a, b) ((a) > (b) ? (a) : (b))
        #endif

        int lineno;
```

```
25    extern FILE * yyin;
      extern int level;
      extern int yyparse();
      RTree * R;
      SOSIGraphObjects *theObjects; //mer innhold
30    ofstream logg("/hom/kristom/.jws/cpp-server/server.log", ios::app);

      char *
      basename (char * name)
      {
35      const char *base = name;

        while (*name)
          {
            if (*name++ == '/')
40            {
                base = name;
              }
          }
        return (char *) base;
45    }

      usage (char * path){
         cout << "Usage: " << basename(path) << " [-d] [-h] [-i] {<filename>}+ " << endl
              << "       -d reads address/representation data into database " << endl
50           << "      -i interactive mode - SOSI data is entered from stdin" << endl
              << "       -h this message" << endl
              << "       <filename> refers to files in SOSI format" << endl ;
          exit(0);

55    }

      read_database(){
      Coordinate G,*P;
      char *p;
60    long North, East;
      DBM * owner_db;
      datum location;
      cout << "Reading address database..." << endl;
      owner_db=dbm_open(OWNER_DB,  O_RDWR, 0660);
65      for (location=dbm_firstkey(owner_db);
             location.dptr !=NULL;
             location=dbm_nextkey(owner_db)) {

            p=location.dptr;
70          memcpy(&G, p, sizeof(Coordinate));
            cout << G.north << ", " << G.east << endl;
      };
          dbm_close(owner_db);
          cout << "done." << endl;
75    }


      int main(int argc, char *argv[]) {
        gis_i myGis;
80      int interactive=FALSE;
        int merge=FALSE;
        if (argc==1) usage(argv[0]);
        R = new RTree();
        //  myGis.gis_i(R);
85      logg << "Srv_Main.cc: main()\n";
        logg.flush();
        while(1){
          int this_option_optind = optind ? optind : 1;
          char c=getopt(argc, argv, "dhi");
```

```
 90       if (c==EOF)
            break;
          switch(c)
            {
            case 'd':
 95         cout << "Merging database..." << endl;
            merge=TRUE;
            break;

            case 'h':
100       usage(argv[0]);
          break;

            case 'i':
          cout << "Interactive mode, end with ^D..." << endl;
105       interactive=TRUE;
          break;
            }
        }
        if (interactive) yyparse();
110     if (optind < argc)
          {
            while (optind < argc){
              cout << "Reading " << argv[optind] << "..." << endl;
          if ((yyin = fopen(argv[optind++], "r")) == NULL) {
115         perror (argv[optind-1]);
            exit(1);
          }

          yyparse();
120       cout << "OK." << endl;
            }
            printf ("\n");

          } else {
125         cout << "No files to read?" << endl;
            exit(0);
          }
        if (merge) read_database();

130     myGis.R=R;
        TRY {
          // tell Orbix that we have completed the server's initialisation:
          cout << "Registering mapServer...\n";
          CORBA::Orbix.impl_is_ready("mapServer",CORBA::Orbix.INFINITE_TIMEOUT,IT_X);
135       cout << "done.\n";
        }
        CATCHANY {
          // an error occured calling impl_is_ready() - output the error.
          cout << IT_X;
140     }
        ENDTRY

        // impl_is_ready() returns only when Orbix times-out an idle server
        // (or an error occurs).
145     cout << "server exiting" << endl;

        return 0;

      }
150


      int xmain(int argc, char **argv){
      if (argc > 1 && (yyin = fopen(argv[1], "r")) == NULL) {
```

```
155       perror (argv[1]);
          exit(1);
        }
          cout << "main()" << endl;
          //  Test();
160       yyparse();
        }
```

Databasehåndteringsverktøyet

dbm.h

```
        #include <ndbm.h>
        #include <string.h>

        #define ADDRESS_DB "adresser"
  5     #define OWNER_DB "eiere"
        #define BUFSIZE 80

        typedef struct Coordinate {
          long  north;
 10       long  east;
        };
```

dbm.cc

```
        #include <sys/types.h>
        #include <sys/stat.h>
        #include <fcntl.h>
        #include <stdio.h>
  5     #include <stream.h>
        #include "dbm.h"

        int main(int argc, char** argv){
        Coordinate * G,*P;
 10     char *p;

        long North, East;
        DBM * address_db, * owner_db;
        datum address, location, owner;
 15     char buffer[BUFSIZE];
        char buffer2[BUFSIZE];
        address_db=dbm_open(ADDRESS_DB,  O_RDWR | O_CREAT, 0660);
        owner_db=dbm_open(OWNER_DB,  O_RDWR | O_CREAT, 0660);
        P=new Coordinate;
 20     if (argc >1) {
          if (!strcmp(argv[1], "-a" )){ // add an entry
```

```
         cout << "Address: ";
          cin.getline(buffer, BUFSIZE);
          address.dptr=strdup(buffer);
25        address.dsize=strlen(buffer);
          cout << "Owner: ";
          cin.getline(buffer, BUFSIZE);
          owner.dptr=strdup(buffer);
          owner.dsize=strlen(buffer);
30        cout << "Location (north):";
          cin >> North;
          cout << "Location  (east):";
          cin >> East;
          P->north=North;
35        P->east=East;
          location.dptr=(char*)P;
          location.dsize=sizeof(Coordinate);
          // updating database:
          dbm_store(address_db, address, location, DBM_INSERT);
40        dbm_store(owner_db, location, owner, DBM_INSERT);
          dbm_close(address_db);
          dbm_close(owner_db);
          cout << "Database updated!" << endl;
        }
45      else
          if  (!strcmp(argv[1], "-d" )){ // delete an entry
          cout << "Address: ";
          cin.getline(buffer, BUFSIZE);
          address.dptr=strdup(buffer);
50        address.dsize=strlen(buffer);
          location=dbm_fetch(address_db, address);
          if (location.dptr == NULL) {
            cout << "No match in database!" << endl;
            return -1;
55        };
          location=dbm_fetch(address_db, address);
          owner=dbm_fetch(owner_db, location);
          if (owner.dptr == NULL) {
            cout << "No owner match in database!" << endl;
60          return -1;
          };
          dbm_delete(address_db, address);
          dbm_delete(owner_db, location);
          dbm_close(address_db);
65        dbm_close(owner_db);
          cout << buffer << " deleted from database!" << endl;
          }

      }
70    else { // display database contents
        for (address=dbm_firstkey(address_db);
             address.dptr !=NULL;
             address=dbm_nextkey(address_db)) {
          location=dbm_fetch(address_db, address);
75        memset(buffer, 0,  BUFSIZE);
          strncpy(buffer, address.dptr, address.dsize);
          p=location.dptr;
          G=new Coordinate;
          G=(Coordinate*)memcpy(G, p, sizeof(Coordinate));
80        North=G->north;
          East=G->east;
          owner=dbm_fetch(owner_db, location);
          memset(buffer2, 0,  BUFSIZE);
          strncpy(buffer2, owner.dptr, owner.dsize);
85        cout << "Address: " << buffer << endl
           << "    Owner: " << buffer2 << endl
```

```
        << "    Location (North/East): "
        << (long) North << ", "
        << (long) East << endl << endl;
90    }
  }
}
```

Klienten

kart.gui

```
  GUI DESCRIPTION VERSION 6
  Root root1 {
    java.awt.Dimension appletSize width=928;height=657
    java.lang.Boolean willGenerateGroup true
5   java.lang.String groupType Panel
    java.lang.String generateClass kart
    child list {
      GBPanelShadow kartpanel {
        [I rowHeights 14,14,14,14,14
10      GBC GBConstraints x=0;y=0;fill=both
        [I columnWidths 14,14,106,14,14,14,14,14
        [D rowWeights 0,0,0,0,0
        java.awt.Dimension layoutSize width=1257;height=784
        [D columnWeights 0,1,0,0,0,0,0,0
15      java.awt.Point layoutLocation x=29;y=121
        child list {
          VJCanvasShadow kartflate {
            GBC GBConstraints x=0;y=1;width=8;fill=both;ipadx=650;ipady=378
            java.awt.Color background white
20          [Lsunsoft.jws.visual.rt.type.Op; operations {
            }
          }
          java.awt.TextFieldShadow adressefelt {
            GBC GBConstraints x=3;y=2;width=5;fill=horizontal;ipadx=450
25          java.awt.Font font name=Dialog;style=plain;size=14
          }
          java.awt.LabelShadow ledetekst1 {
            GBC GBConstraints x=2;y=2;ipadx=52
            java.lang.String text Adresse
30          sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
            sunsoft.jws.visual.rt.type.AnchorEnum anchor east
            java.awt.Font font name=Dialog;style=bold;size=14
          }
          java.awt.LabelShadow servertekst {
35          GBC GBConstraints x=0;y=2;width=2;fill=horizontal;ipadx=78
            java.lang.String text ""
            java.awt.Color foreground red
            sunsoft.jws.visual.rt.type.AlignmentEnum alignment left
            sunsoft.jws.visual.rt.type.AnchorEnum anchor west
40          java.awt.Font font name=Dialog;style=bold;size=14
          }
          java.awt.LabelShadow ledetekst2 {
```

```
                GBC GBConstraints x=2;y=4
                java.lang.String text "Målestokk 1:"
 45             sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
                sunsoft.jws.visual.rt.type.AnchorEnum anchor east
                java.awt.Font font name=Dialog;style=bold;size=14
              }
              java.awt.TextFieldShadow skaleringsfaktor {
 50             GBC GBConstraints x=3;y=4;fill=horizontal
                java.awt.Font font name=Dialog;style=plain;size=14
              }
              java.awt.ButtonShadow nabobutton {
                GBC GBConstraints x=6;y=4;ipadx=8
 55             java.lang.String text "Tegn nabolaget"
                java.awt.Font font name=Dialog;style=bold;size=14
              }
              java.awt.ButtonShadow tegnbutton {
                GBC GBConstraints x=5;y=4
 60             java.lang.String text "Tegn kartet"
                java.awt.Font font name=Dialog;style=bold;size=14
              }
              java.awt.ButtonShadow eierbutton {
                GBC GBConstraints x=7;y=4
 65             java.lang.String text "Vis eier"
                java.awt.Font font name=Dialog;style=bold;size=14
              }
              java.awt.ButtonShadow symbolbutton {
                GBC GBConstraints x=0;y=4
 70             java.lang.String text Tegnforklaring
                sunsoft.jws.visual.rt.type.AnchorEnum anchor west
                java.awt.Font font name=Dialog;style=bold;size=14
              }
            }
 75       }
          java.awt.DialogShadow eierdialog {
            java.lang.String title GAB-informasjon
            java.lang.Boolean resizable false
            java.lang.Boolean visible false
 80         java.awt.Dimension layoutSize width=654;height=529
            java.awt.Point layoutLocation x=411;y=216
            child list {
              GBPanelShadow gbpanel1 {
                [I rowHeights 14,14,14,14
 85             GBC GBConstraints x=0;y=0;fill=both
                [I columnWidths 14,14,14,14,14,14
                [D rowWeights 0,0,0,0
                [D columnWeights 0,0,0,0,0,0
                child list {
 90               java.awt.TextAreaShadow adresselinjer {
                    GBC GBConstraints x=1;y=3;width=5;ipadx=110
                    java.awt.Color background white
                    java.lang.Boolean editable false
                  }
 95               java.awt.TextFieldShadow eierfelt {
                    GBC GBConstraints x=1;y=1;width=5;fill=horizontal
                    java.awt.Color background white
                    java.lang.Boolean editable false
                  }
100               java.awt.LabelShadow label3 {
                    GBC GBConstraints x=0;y=1;fill=horizontal
                    java.lang.String text Navn
                    sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
                    sunsoft.jws.visual.rt.type.AnchorEnum anchor east
105                 java.awt.Font font name=Dialog;style=plain;size=14
                  }
                  java.awt.LabelShadow label4 {
```

```
                    GBC GBConstraints x=0;y=3
                    java.lang.String text Adresse
110                 sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
                    sunsoft.jws.visual.rt.type.AnchorEnum anchor northeast
                    java.awt.Font font name=Dialog;style=plain;size=14
                  }
                  LabelBarShadow labelbar1 {
115                 GBC GBConstraints x=0;y=0;width=6;fill=horizontal
                    java.lang.String text Eierinformasjon
                    java.awt.Font font name=Dialog;style=italic;size=16
                  }
                }
120           }
            }
          }
          java.awt.DialogShadow symbolDialog {
            java.lang.String title Tegnforklaring
125         java.lang.Boolean resizable false
            java.lang.Boolean visible false
            java.awt.Dimension layoutSize width=639;height=643
            java.awt.Point layoutLocation x=36;y=190
            child list {
130           GBPanelShadow gbpanel2 {
                [I rowHeights 14,14,14,14,14,14,14,14
                GBC GBConstraints x=0;y=0;fill=both
                [I columnWidths 14,14,14
                [D rowWeights 0,0,0,0,0,0,0,0,0
135             [D columnWeights 0,0,0
                child list {
                  LabelBarShadow labelbar2 {
                    GBC GBConstraints x=0;y=0;width=3;fill=horizontal
                    java.lang.String text Tegnforklaring
140                 java.awt.Font font name=Dialog;style=italic;size=16
                  }
                  VJCanvasShadow tegneflateBygning {
                    GBC GBConstraints x=1;y=1;width=2;ipadx=210
                    java.awt.Color background white
145                 java.lang.Integer minWidth 10
                    java.lang.Integer minHeight 70
                  }
                  VJCanvasShadow tegneflateGate {
                    GBC GBConstraints x=1;y=2;width=2;fill=horizontal
150                 java.awt.Color background white
                    java.lang.Integer minWidth 10
                    java.lang.Integer minHeight 50
                  }
                  VJCanvasShadow tegneflateTerreng {
155                 GBC GBConstraints x=1;y=3;width=2;fill=horizontal
                    java.awt.Color background white
                    java.lang.Integer minWidth 10
                    java.lang.Integer minHeight 50
                  }
160               VJCanvasShadow tegneflateGAB {
                    GBC GBConstraints x=1;y=4;width=2;fill=horizontal
                    java.awt.Color background white
                    java.lang.Integer minWidth 10
                    java.lang.Integer minHeight 30
165               }
                  java.awt.LabelShadow label1 {
                    GBC GBConstraints x=0;y=1
                    java.lang.String text Bygning
                    sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
170                 sunsoft.jws.visual.rt.type.AnchorEnum anchor northeast
                    java.awt.Font font name=Dialog;style=plain;size=16
                  }
```

```
                java.awt.LabelShadow label2 {
                  GBC GBConstraints x=0;y=2
175               java.lang.String text Gate/Vei
                  sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
                  sunsoft.jws.visual.rt.type.AnchorEnum anchor northeast
                  java.awt.Font font name=Dialog;style=plain;size=16
                }
180             java.awt.LabelShadow label5 {
                  GBC GBConstraints x=0;y=3
                  java.lang.String text Terreng
                  sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
                  sunsoft.jws.visual.rt.type.AnchorEnum anchor northeast
185               java.awt.Font font name=Dialog;style=plain;size=16
                }
                java.awt.LabelShadow label6 {
                  GBC GBConstraints x=0;y=4
                  java.lang.String text Gårds/br.nr.
190               sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
                  sunsoft.jws.visual.rt.type.AnchorEnum anchor northeast
                  java.awt.Font font name=Dialog;style=plain;size=16
                }
                java.awt.ButtonShadow lukkbutton {
195               GBC GBConstraints x=2;y=7
                  java.lang.String text Lukk
                  sunsoft.jws.visual.rt.type.AnchorEnum anchor east
                  java.awt.Font font name=Dialog;style=bold;size=14
                }
200             java.awt.LabelShadow label7 {
                  GBC GBConstraints x=0;y=5
                  java.lang.String text Grense
                  sunsoft.jws.visual.rt.type.AlignmentEnum alignment right
                  sunsoft.jws.visual.rt.type.AnchorEnum anchor northeast
205               java.awt.Font font name=Dialog;style=plain;size=16
                }
                VJCanvasShadow tegneflateGrense {
                  GBC GBConstraints x=1;y=5;width=2;ipadx=210
                  java.awt.Color background white
210               java.lang.Integer minWidth 10
                  java.lang.Integer minHeight 50
                }
              }
            }
215       }
        }
        java.awt.DialogShadow feilDialog {
          java.lang.String title Feilmelding
          java.lang.Boolean resizable false
220       java.awt.Dimension layoutSize width=400;height=300
          java.awt.Point layoutLocation x=172;y=156
          child list {
            GBPanelShadow gbpanel3 {
              [I rowHeights 14,14,14
225           GBC GBConstraints x=0;y=0;fill=both
              [I columnWidths 14
              [D rowWeights 0,0,0
              [D columnWeights 0
              child list {
230             java.awt.LabelShadow label8 {
                  GBC GBConstraints x=0;y=0
                  java.lang.String text "Adressen finnes ikke!"
                  java.awt.Font font name=Dialog;style=plain;size=14
                }
235             java.awt.ButtonShadow feilOKbutton {
                  GBC GBConstraints x=0;y=2
                  java.lang.String text OK
```

```
                    }
                  }
240             }
              }
            }
          }
        }
245
```

kart.java

```
        /**
         * kart.java
         */

 5      import sunsoft.jws.visual.rt.base.*;
        import sunsoft.jws.visual.rt.shadow.java.awt.*;
        import sunsoft.jws.visual.rt.awt.*;
        import java.awt.*;
        import _gabServer.notFoundException;
10      public class kart extends Group {

          private kartRoot gui;
          private VJCanvas kartflatebody;
          private VJCanvas tegneflateBygning;
15        private VJCanvas tegneflateTerreng;
          private VJCanvas tegneflateGate;
          private VJCanvas tegneflateGAB;

          private GBPanel kartpanelbody;
20
          public static SOSIGraphObjects gobjektliste;
          private Polygon[] huspolygoner;
          private SOSIPoint[] rep_punkter;
          private Polygon akthuspolygon;
25        private SOSIPoint akthusPunkt;
          private int antallhus;

          SOSIPoint gposisjon, gposisjonNW, gposisjonSE;
          SOSIPoint GABposisjon;
30        private int kartbredde, karthøyde;

          private float faktor1 = 1;
          private float faktor2 = (float)0.9;   // ... foreløpig.

35        public static _mapServerRef mapserver = null; // referanse til karttjener
          public static _gabServerRef gabserver = null; // referanse til GAB-tjener

          public kart() {
            addForwardedAttributes();
40        }

          protected Root initRoot() {
            /**
             * Initialize the gui components
45           */
            gui = new kartRoot(this);
            addAttributeForward(gui.getMainChild());
```

```
          return gui;
50      }

        protected void initGroup() {
          gui.tegnbutton.set("enabled", new Boolean(false));
          gui.nabobutton.set("enabled", new Boolean(false));
55        gui.eierbutton.set("enabled", new Boolean(false));
          logginn();
        }

        protected void showGroup() { }
60

        protected void hideGroup() { }

        protected void createGroup() { }
65
        protected void destroyGroup() { }

        protected void startGroup() {
          if (gobjektliste != null)  tegnkart(gobjektliste);
70        System.out.println("startGroup()");
        }

        protected void stopGroup() { }

75      protected Object getOnGroup(String key) {
          return super.getOnGroup(key);
        }

        protected void setOnGroup(String key, Object value) {
80        super.setOnGroup(key, value);
        }

        public boolean handleMessage(Message msg) {
          return super.handleMessage(msg);
85      }

        public boolean handleEvent(Message msg, Event evt) {
          return super.handleEvent(msg, evt);
        }
90
      public boolean gotFocus(Message msg, Event evt, Object what){
      if (gobjektliste != null)       tegnkart(gobjektliste);

      return true;
95    };
      public boolean mouseEnter(Message msg, Event evt, Object what){
      if (gobjektliste != null)       tegnkart(gobjektliste);

      return true;
100   };
      /***********************************************************************/
      /***********************************************************************/
      /*                   Metoder som håndterer events                      */
      /***********************************************************************/
105   /***********************************************************************/
      /* Metode som avgjør hvilken knapp som er trykket og kaller de         */
      /* metodene som skal utføre handlingene.                               */
      /***********************************************************************/
        public boolean action(Message msg, Event evt, Object what) {
110       String GABadresse;
          Person P;
          kartflatebody = (VJCanvas)gui.kartflate.getBody();
          kartbredde = kartflatebody.bounds().width;
```

```
        karthøyde = kartflatebody.bounds().height;
115
        if (msg.target == gui.tegnbutton)  {

          GABadresse = new String(lesfelter());

120       if (GABadresse.length() == 0)
            // Hvis ingen adresse oppgitt, tegn ut fra midten av området.
            // Faktor1 er satt slik at hele området får plass på kartflaten
            // når faktor2 = 1 (default).
            GABposisjon = new SOSIPoint((int)((kartbredde/2)*faktor1),
125           (int)((karthøyde/2)*faktor1));
          else
            GABposisjon = hentGABposisjon(GABadresse);
          if (GABposisjon !=null){
          gobjektliste = hentområde();
130
          tegnkart(gobjektliste);
          P=finnEier(GABposisjon);
          System.out.println(P.Name);
          System.out.println(P.Address1);
135       System.out.println(P.Address2);
          System.out.println(P.Address3);
          }
          else
            {
140           System.out.println("Fant ikke adresse...");
            }
        return true;
        }
        else if (msg.target == gui.nabobutton) {
145      // Tegner ut huset det er klikket på og dets nærmeste nabolag.
          gui.nabobutton.set("enabled", new Boolean(false));
          gui.eierbutton.set("enabled", new Boolean(false));
          finn_nabolag(akthuspolygon);
          hentområde();
150       tegnkart(gobjektliste);
        return true;
        }
        else if (msg.target == gui.eierbutton) {
          P=finnEier(GABposisjon);
155       gui.eierdialog.set("visible", new Boolean(true));
          gui.eierfelt.set("text", P.Name);
          gui.adresselinjer.set("text", P.Address1 + "\n"+ P.Address2 +
                                "\n"+ P.Address3 );
          System.out.println(P.Name);
160       System.out.println(P.Address1);
          System.out.println(P.Address2);
          System.out.println(P.Address3);
        return true;
        }
165     else if (msg.target == gui.symbolbutton) {
          gui.symbolDialog.set("visible", new Boolean(true));
          tegnForklaringer();
          return true;
        }
170     else if (msg.target == gui.lukkbutton) {
          gui.symbolDialog.set("visible", new Boolean(false));
          return true;
        }
        else if (msg.target == gui.feilOKbutton) {
175       gui.feilDialog.set("visible", new Boolean(false));
          return true;
        }
        return false;
```

```
        }
180
      /*****************************************************************/
      /*            Metode som tolker musklikk og kaller de             */
      /*             metodene som skal utføre handlingene.              */
      /*****************************************************************/
185     public boolean mouseUp(Message msg, Event evt, int x, int y) {
          if (msg.target == gui.kartflate)  {
            akthuspolygon = new Polygon();
            try {
            finnhus(x, y);
190         } catch (Fantikke e){
            }
            return true;
          }
          else
195       {
            return false;
          }
        }

200   /*****************************************************************/
      /*****************************************************************/
      /*           Metode som leser feltene i skjermbildet              */
      /*****************************************************************/
        private String lesfelter() {
205       String adresse;
          String faktortekst;

          adresse = new String(gui.adressefelt.get("text").toString());

210       faktor2 = (Float.valueOf(gui.skaleringsfaktor.get("text").to-
String())).floatValue();

      //  Returnerer gab-adressen, faktor2 har hele klassen som scop.
          return adresse;
        }
215

      /*****************************************************************/
      /*****************************************************************/
      /*        Metode som plukker ut de forskjellige objekttypene      */
220   /*                  fra objektlisten SOSIGraphObjects             */
      /*        og kaller tegnemetoder med de riktige parametrene.      */
      /*****************************************************************/
        private void tegnkart(SOSIGraphObjects gliste) {
          int senterx, sentery;
225       float radius;
          Color pennfarge, fyllfarge;
          Polygon nyttpolygon;

          Point polyrefpunkt;
230       int polydescription;

          Point[] huspolyrefpunkt;
          int[] huspolydescription;

235       Point tekststartpunkt;

          huspolygoner = new Polygon[1000];
          huspolyrefpunkt = new Point[1000];
          huspolydescription = new int[1000];
240       rep_punkter = new SOSIPoint[1000];
          antallhus = 0;
```

```
          kartflatebody = (VJCanvas)gui.kartflate.getBody();

245       kartbredde = kartflatebody.bounds().width;
          karthøyde = kartflatebody.bounds().height;

          kartflatebody.getGraphics().clearRect(0, 0, kartbredde, karthøyde);

250       for (int i = 0; i < gliste.SOSICircles.length; i++) {
            senterx = gliste.SOSICircles.buffer[i].Gcentre.east;
            sentery = gliste.SOSICircles.buffer[i].Gcentre.north;
            pennfarge = new Color(0,0,0);
            pennfarge = fargetolker(gliste.SOSICircles.buffer[i].SOSIDescription);
255         fyllfarge = new Color(0, 200, 0);
            radius = gliste.SOSICircles.buffer[i].Gradius;

            tegnsirkel(servtilkart_x((float)senterx, gposisjonNW.east),
                       servtilkart_y((float)sentery, gposisjonNW.north),
260                    pennfarge,
                       fyllfarge,
                       servtilkart(radius, 0),
                       kartflatebody.getGraphics());
          }
265       for (int i = 0; i < gliste.SOSIPolyLines.length; i++) {
            nyttpolygon = new Polygon();
            for (int j = 0; j < gliste.SOSIPolyLines.buffer[i].joints.length; j++) {
              nyttpolygon.addPoint(
                       servtilkart_x(gliste.SOSIPolyLines.buffer[i].joints.buff-
er[j].east,
270                                gposisjonNW.east),
                       servtilkart_y(gliste.SOSIPolyLines.buffer[i].joints.buff-
er[j].north,
                                   gposisjonNW.north));
            }
            polyrefpunkt = new Point(
275                servtilkart_x(gliste.SOSIPolyLines.buffer[i].theSOSIRefPoint.east,
                                   gposisjonNW.east),
                       servtilkart_y(gliste.SOSIPolyLines.buffer[i].theSOSIRef-
Point.north,
                                   gposisjonNW.north));
            polydescription = gliste.SOSIPolyLines.buffer[i].SOSIDescription;
280         pennfarge = fargetolker(polydescription);
            tegnpolygon(nyttpolygon, pennfarge, kartflatebody.getGraphics());

            if (polydescription/1000 == 5 || polydescription/1000 == 6) {
            // Polygoner som representerer hus er grønne.
285           huspolygoner[antallhus] = nyttpolygon;
              huspolyrefpunkt[antallhus] = polyrefpunkt;
              huspolydescription[antallhus] = polydescription;
              rep_punkter[antallhus] =
                gliste.SOSIPolyLines.buffer[i].theSOSIRefPoint;
290           antallhus++;
            }
          }
          for (int i = 0; i < gliste.SOSITexts.length; i++) {
            tekststartpunkt = new Point(
295                    servtilkart_x(gliste.SOSITexts.buffer[i].Gposition.east,
                                   gposisjonNW.east),
                       servtilkart_y(gliste.SOSITexts.buffer[i].Gposition.north,
                                   gposisjonNW.north));
            pennfarge = fargetolker(gliste.SOSITexts.buffer[i].SOSIDescription);
300         skrivtekst(gliste.SOSITexts.buffer[i].theGText,
                       pennfarge,
                       tekststartpunkt,
                       kartflatebody.getGraphics());
          }
```

```
305         for (int i = 0; i < gliste.SOSIRefPoints.length; i++){
            }
          }

        /*****************************************************************/
310     /*****************************************************************/
        /*               Metoder som kommuniserer med server            */
        /*****************************************************************/
        /*****************************************************************/
        /*               Oppretter forbindelse med serveren.            */
315     /*****************************************************************/
          private void logginn() {

            try {
              IE.Iona.Orbix2._CORBA.Orbix.setDiagnostics(2);
320           IE.Iona.Orbix2._CORBA.IT_BIND_USING_IIOP = false;
              mapserver = mapServer._bind(":mapServer","tyrfing.ifi.uio.no");
              gabserver = gabServer._bind(":gabServer","tyrfing.ifi.uio.no");
              gui.servertekst.set("text", "Oppkoblet!");
              gui.ledetekst1.set("text", "Adresse");
325           gui.ledetekst2.set("text", "Målestokk 1:");
              gui.tegnbutton.set("enabled", new Boolean(true));
              gui.adressefelt.set("text", new String("Trondheimsveien 4"));
              gui.skaleringsfaktor.set("text", new String("20"));
            }
330         catch (IE.Iona.Orbix2.CORBA.SystemException se) {
              gui.servertekst.set("text", "Ikke kontakt!");
              System.out.println ("Feil ved oppkobling til server.");
              System.out.println (se.toString ());
            }
335
          }

        /*****************************************************************/
        /*      Henter ut kartdata for et område rundt et angitt punkt.   */
340     /*****************************************************************/
          private SOSIGraphObjects hentområde()  {

            try {

345           gposisjonSE = new SOSIPoint();
              gposisjonNW = new SOSIPoint();

            kartflatebody = (VJCanvas)gui.kartflate.getBody();
            kartbredde = kartflatebody.bounds().width;
350         karthøyde = kartflatebody.bounds().height;

              gposisjonSE.east = (int)(GABposisjon.east + (kartbredde/
2)*faktor1*faktor2);
              gposisjonSE.north = (int)(GABposisjon.north + (karthøyde/
2)*faktor1*faktor2);
              gposisjonNW.east = (int)(GABposisjon.east - (kartbredde/
2)*faktor1*faktor2);
355           gposisjonNW.north = (int)(GABposisjon.north - (karthøyde/
2)*faktor1*faktor2);
              System.out.println("GABposisjon: " + GABposisjon.east + " " +
                                    GABposisjon.north);
              System.out.println("gposisjonSE: " + gposisjonSE.east + " " +
                                    gposisjonSE.north);
360           System.out.println("gposisjonNW: " + gposisjonNW.east + " " +
                                   gposisjonNW.north);
              gobjektliste = null;
              gobjektliste = new SOSIGraphObjects();
              gobjektliste = mapserver.getRegion(gposisjonNW, gposisjonSE);
365           return gobjektliste;
```

```
            }

          catch
            (IE.Iona.Orbix2.CORBA.SystemException se) {
370     //    Skriver feilmelding dersom det oppstår feil under hentingen
        //    av informasjon fra tjeneren
              System.out.println(se.toString());
              return null;
              }
375       }

        /****************************************************************/
        /*      Henter eier til bygning i GAB-registeret ut ifra koordinat    */
        /****************************************************************/
380
          private Person finnEier(SOSIPoint Posisjon){

            try {
            return gabserver.findOwner(Posisjon);
385         }
            catch
                (IE.Iona.Orbix2.CORBA.SystemException se) {
        //  Skriver feilmelding dersom det oppstår feil under hentingen
        //  av informasjon fra tjeneren
390             System.out.println(se.toString());
                return null;
            }

          }
395
        /****************************************************************/
        /*      Henter posisjon i GAB-registeret ut ifra oppgitt adresse.     */
        /****************************************************************/
          private SOSIPoint hentGABposisjon(String adresse)  {
400         try {
              GABposisjon = new SOSIPoint();
              GABposisjon = gabserver.findLocation(gui.adressefelt.get("text").to-
String());
              return GABposisjon;
            }
405         catch
                (IE.Iona.Orbix2.CORBA.SystemException se) {
        //  Skriver feilmelding dersom det oppstår feil under hentingen
        //  av informasjon fra tjeneren
                System.out.println(se.toString());
410           gui.feilDialog.set("visible", new Boolean(false));
                return null;
            }

          }
415
        /****************************************************************/
        /*                      Diverse hjelpemetoder                         */
        /****************************************************************/
        /****************************************************************/
420     /*          Oversetter fra TEMA-koder til Java-farger.              */
        /****************************************************************/
        private Color fargetolker(int fargekode) {
        Color farge;
        if (fargekode==-1) farge=Color.pink;
425     else
          if (fargekode/1000==2) //terreng
            farge = new Color(0,150,0);                  // Grønn
          else if (fargekode/1000==1)  // plandata
            farge = new Color(0,0,150);                // Blå
```

```
430       else if (fargekode/1000 == 5 || fargekode/1000==6) // bygninger
            farge = new Color(0,0,0);                  // sort
          else if (fargekode==3)
            farge = new Color(0,0,0);                    //
          else if (fargekode/1000==4) // div. grenser
435         farge = new Color(150,0,0);                  // Rød
          else if (fargekode==5)
            farge = new Color(0,0,0);                    //
          else if (fargekode==6)
            farge = new Color(0,0,0);                    //
440       else if (fargekode/1000==7) // div. vei
            farge = new Color(127, 127, 127);            // Grå
          else farge = new Color(0,0,0);               // Sort (default)
          return farge;
        }
445
        /*******************************************************************/
        /*      Omregning fra serverkoordinater til kartets koordinater:      */
        /*******************************************************************/
         private int servtilkart(float serverkoordinat, int fradrag) {
450         int kartkoordinat;
            serverkoordinat = serverkoordinat - fradrag;
            kartkoordinat = (int)(serverkoordinat/(faktor1*faktor2));
            return kartkoordinat;
        }
455
          private int servtilkart_x( float serverkoordinat, int fradrag) {
            int kartkoordinat;
            serverkoordinat = serverkoordinat - fradrag;
            kartkoordinat = (int)(serverkoordinat/(faktor1*faktor2));
460         return kartkoordinat;
        }
          private int servtilkart_y(float serverkoordinat, int fradrag) {
            int kartkoordinat;
            serverkoordinat = serverkoordinat - fradrag;
465         kartkoordinat = (int)(kartflatebody.bounds().height -  serverkoordinat/
(faktor1*faktor2));
            return kartkoordinat;
        }

        /*******************************************************************/
470     /*                            Tegne-metoder                          */
        /*******************************************************************/

          private void tegnsirkel(int sx, int sy, Color pcolor, Color fcolor, int radius,
Graphics g) {
            g.setColor(fcolor);
475         g.fillArc(sx - radius, sy - radius, 2*radius, 2*radius, 0, 360);
            g.setColor(pcolor);
            g.drawArc(sx - radius, sy - radius, 2*radius, 2*radius, 0, 360);
          }

480       private void tegnpolygon(Polygon polygonet, Color pcolor, Graphics g) {
            g.setColor(pcolor);
            g.drawPolygon(polygonet);
          }

485       private void skrivtekst(String tekst, Color farge, Point startpunkt, Graphics
g) {
            g.setColor(farge);
            g.drawString(tekst, startpunkt.x, startpunkt.y);
          }

490     /***************************************************************/
        /*              Finner nabolaget til et valgt hus                */
```

```
        /***************************************************************/
          private void finn_nabolag(Polygon huspolygon) {
            Rectangle husboks;
 495
            new Rectangle();
            husboks = akthuspolygon.getBoundingBox();
          // Finn senter i huspolygonet.
            GABposisjon.east = (int)(gposisjonNW.east+(husboks.x + (husboks.width/
2))*faktor1*faktor2);
 500        GABposisjon.north = (int)(gposisjonNW.north+(kartflatebody.bounds().height -
husboks.y + (husboks.height/2))*faktor1*faktor2);
          // Legg til passelig område rundt senteret. Gjøres enkelt ved å
          // sette faktor2. Kunne gjort noe mer avansert.
            //faktor2 = (float)0.2;

 505      }

        /***************************************************************/
        /*              Finner hvilket hus som er klikket på              */
        /***************************************************************/
 510      private int finnhus(int x, int y) throws Fantikke {
            int i;
            Graphics g;
            Polygon huspolygon;
            boolean ihus = false;
 515        huspolygon = new Polygon();

            for (i = 0; i < antallhus; i++) {
              if (huspolygoner[i].inside(x, y)) {
                akthuspolygon = new Polygon();
 520            akthuspolygon = huspolygoner[i];

                g=kartflatebody.getGraphics();
                g.setXORMode(Color.yellow); // setPaintMode();
                g.setColor(Color.darkGray);
 525            System.out.println(g.getColor().toString());
                g.fillPolygon(akthuspolygon);
                g.setPaintMode();
                ihus = true;
                gui.nabobutton.set("enabled", new Boolean(true));
 530            gui.eierbutton.set("enabled", new Boolean(true));
                try {
                  gui.adressefelt.set("text", gabserver.findAddress(rep_punkter[i]));
                }
                catch (IE.Iona.Orbix2.CORBA.SystemException se) {
 535                  //  Skriver feilmelding dersom det oppstår feil under hentingen
                      //  av informasjon fra tjeneren
                System.out.println(se.toString());
                }
              }
 540        }
            if (ihus){
            }
            else {
              throw new Fantikke();
 545        }
              return i; // Alt OK
          }

          private void tegnForklaringer(){
 550        // metode som tegner ut eksempler på hus, terengkurver osv
          Graphics g;
          Polygon P1;
            g=((VJCanvas)gui.tegneflateBygning.getBody()).getGraphics();
            g.setColor(fargetolker(5000)); // bygninger
```

```
555             P1=new Polygon();
            P1.addPoint(10,10);P1.addPoint(110,10);
            P1.addPoint(110,10);P1.addPoint(110,50);
            P1.addPoint(110,50);P1.addPoint(80,50);
            P1.addPoint(80,50);P1.addPoint(80,60);
560         P1.addPoint(80,60);P1.addPoint(60,60);
            P1.addPoint(60,60);P1.addPoint(60,50);
            P1.addPoint(60,50);P1.addPoint(10,50);
            P1.addPoint(10,50);P1.addPoint(10,10);
            g.drawPolygon(P1);
565         g.dispose();
            g=((VJCanvas)gui.tegneflateTerreng.getBody()).getGraphics();
            g.setColor(fargetolker(2000)); // terreng
            Polygon P2 = new Polygon();
            Polygon P3 = new Polygon();
570         Polygon P4 = new Polygon();
            P2.addPoint(14, 27);P2.addPoint(24, 30);P2.addPoint(30, 31);
            P2.addPoint(36, 31);P2.addPoint(42, 31);P2.addPoint(45, 31);
            P2.addPoint(49, 30);P2.addPoint(51, 29);P2.addPoint(52, 27);
            P2.addPoint(55, 26);P2.addPoint(56, 25);P2.addPoint(59, 22);
575         P2.addPoint(61, 20);P2.addPoint(64, 17);P2.addPoint(67, 16);
            P2.addPoint(70, 14);P2.addPoint(75, 14);P2.addPoint(81, 12);
            P2.addPoint(83, 9);P2.addPoint(84, 8);P2.addPoint(87, 5);
            P3.addPoint(7, 36);P3.addPoint(18, 40);P3.addPoint(20, 40);
            P3.addPoint(26, 42);P3.addPoint(32, 42);P3.addPoint(37, 39);
580         P3.addPoint(43, 39);P3.addPoint(49, 38);P3.addPoint(55, 38);
            P3.addPoint(59, 36);P3.addPoint(61, 31);P3.addPoint(62, 29);
            P3.addPoint(65, 24);P3.addPoint(70, 21); P3.addPoint(73, 20);
            P3.addPoint(78, 20);P3.addPoint(82, 20);P3.addPoint(87, 18);
            P3.addPoint(90, 16);P3.addPoint(92, 13);P3.addPoint(95, 12);
585         P3.addPoint(96, 8);P3.addPoint(97, 5);P3.addPoint(97, 4);
            P3.addPoint(98, 3);

            P4.addPoint(97, 20); P4.addPoint(95, 20);P4.addPoint(91, 25);
            P4.addPoint(84, 32);P4.addPoint(82, 35);P4.addPoint(80, 36);
590         P4.addPoint(79, 38);P4.addPoint(77, 39);P4.addPoint(75, 39);
           P4.addPoint(74, 39);P4.addPoint(73, 40);
            P4.addPoint(73, 40);
            g.drawPolygon(P2);g.drawPolygon(P3);g.drawPolygon(P4);
            g.dispose();
595         g=((VJCanvas)gui.tegneflateGAB.getBody()).getGraphics();
            g.setColor(fargetolker(4000));
            g.drawString("123/45", 20,20);
            g=((VJCanvas)gui.tegneflateGate.getBody()).getGraphics();
            g.setColor(fargetolker(7000));
600         Polygon P5=new Polygon();
            Polygon P6=new Polygon();
            P5.addPoint(10,10);P5.addPoint(150,10);P5.addPoint(155,5);
            P5.addPoint(155,0);P6.addPoint(170,0);P6.addPoint(170,5);
            P6.addPoint(175,10);P6.addPoint(200,10);
605         g.drawPolygon(P5);
            g.drawPolygon(P6);
            g.drawLine(10,40,200,40);
            g.dispose();
            g=((VJCanvas)gui.tegneflateGrense.getBody()).getGraphics();
610         g.setColor(fargetolker(4000));
            g.drawLine(10,10,100,10);g.drawLine(100,10,100,20);
            g.drawLine(100,20,200,30);g.drawLine(100,10,100,20);
            g.dispose();
          };
615     }

        class Fantikke extends _gabServer.notFoundException {}
```

D Register

A
AoC 18
Area of Concern 18

B
BSD sockets
  Se sockets
Bygningsetaten
  Se Plan- og bygningsetaten

C
COMan 33
CORBA 60–66
CORBAServices 65–66

D
DAK 17
Dataassistert konstruksjon 17
dialektisk 24
Distributed Computation Environment (DCE) 67

F
fjernprosedyrekall 57
FooAM 30–31

G
GAB-registeret 18

I
IDL
  Se Interface Definition Language
Interface Definition Language 62
IPX/SPX 56

L
legacy systems 25
  innkapsling 27

M
mission critical
  Se virksomhetskritisk

N
Network File System (NFS) 58

O
Object Management Group (OMG) 59
Object Request Broker
  Se Objektmegler
Object Request Broker (ORB) 61
Objectory 73
Objektmegler 58
OOram Professional 77
Open Software Foundation (OSF) 67
OSI-modellen 61

P
perspektiv
  Area of Concern 76
  Grensesnitt 76
  Metodepe 76
  Prosess 76
  Rolleliste 76
  Samarbeid 76
  Scenario 76
  Semantisk 76
  Stimulus-respons 76
Plan- og bygningsetaten 17
Plan- og bygningsloven 24
program slicing 29

R
Rational Rose 73
Remote Procedure Call (RPC)
  Se fjernprosedyrekall
reverse engineering
  Se reverskonstruksjon

reverskonstruksjon 29
RooAM 30–31

S
Select Enterprise 73
sockets 55

T
Tilstandsdiagram 76
Transport Layer Interface (TLI) 55, 56

U
use cases 72
utkikkspunkter 76

V
viewpoints se utsiktpunkter
virksomhetskritisk 25
virksomhetskritiske systemer 25

W
WinSock 55
wrapper 56