NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERISTET
FAKULTET FOR ELEKTROTEKNIKK OG TELEKOMMUNIKASJON

HOVEDOPPGAVE

Objektorienterte datamodeller for geodata

Knut Aage Aksnes

17. januar 1997

NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

FAKULTET FOR ELEKTROTEKNIKK OG TELEKOMMUNIKASJON

HOVEDOPPGAVE

Kandidatens navn: Knut Aage Aksnes

Fag: Datateknikk

Oppgavens tittel (norsk): Objektorienterte datamodeller for geodata

Oppgavens tittel (engelsk): Object-Oriented Geodata Models

Oppgavens tekst:

Applikasjoner som lagrer og behandler geografisk informasjon (geodata) brukes på mange områder. Karakteristisk for slike applikasjoner er bruk av komplekse datastrukturer og spesielle krav til databasesystemer for lagring. Etterhvert som antallet applikasjoner øker, blir behovene for å kunne utveksle informasjon mellom disse større. Mangel på standarder for slik informasjonsutveksling gjør imidlertid dette vanskelig.

Oppgaven går ut på å studere standardiseringsarbeider med objektorienterte datamodeller for geodata, og mekanismer for utveksling av geografisk informasjon. For å kunne vurdere hvordan modellene kan brukes i praksis, skal modellene realiseres i to databasesystemer; et objektorientert databasesystem, og et objekt-relasjonsdatabasesystem. På grunnlag av dette vurderes hvordan ulike databaseteknologier egner seg for implementasjon av slike systemer.

Oppgaven gitt: 10. september 1996

Besvarelsen leveres innen: 17. januar 1997

Besvarelsen levert: 17. januar 1997

Utført ved: Fakultet for elektroteknikk og telekommunikasjon

Veileder: Kjetil Nørvåg

Trondheim, 17. januar 1997

Kjell Bratbergsengen
Faglærer

Summary

This diploma thesis represents the result of a project performed for the Database Systems Group at the Department of Computer Systems of Norwegian University of Science and Technology, Trondheim. The theme for the project was Object-Oriented Geodata Models.

Ongoing standardization efforts, led by the Open GIS Consortium (OGC) in USA, has been studied. The “Open Geodata Model” presented by the technical committee of was then used as the grounding for modelling my own, a bit limited geodata model. My model was then used as a basis for implementing an application, using two different database platforms. The Shore database system, developed at University of Wisconsin-Madison, was one of the databases that was used. Shore is an object-oriented database system. As the second database was PostgreSQL used, a relational database system with many object-oriented extensions. PostgreSQL was developed at University of California at Berkeley. The application was build using C++ as a programming language and with wxWindows, a platform-independent class library for building graphical user interfaces, and runs on a Unix-platform using SunOS 4.1.3 and X/Motif. wxWindows was developed at the University of Edinburgh.

Through the work with implementing this application some major differences between the two database systems was clearly pointed out.

- When using a relational database system in an object-oriented context, the conceptual gap between using SQL and the object-oriented language C++ evidently came true.
- Using the same kind of datastructures on Shore and PostgreSQL, and measuring elapsed time on operations on these structures showed that Shore was about 100 times faster than PostgreSQL. Retrieving 100 polygons that constitutes of two rings of 4 points each, used about 150 seconds on PostgreSQL, while Shore could do the same in about 1.5 second.
- The one object-oriented extension in PostgreSQL that showed to be of any value was inheritance. This made the mapping from an objectmodel to relational tables easier and more elegant. It also made the retrieval of information from the subclasses easier when creating the proper *select*-sentences.

A parser was written to convert geographic information from SOSI-files, a Norwegian standard for exchanging such informations, was also build. Several files with different kind of data was loaded into the Shore database. This showed that the geodata model was fully capable of storing the different kinds of geometries used on maps.

As a part of the Open Geodata Model, datastructures are defined that can hold all of the described geometries. These structures, called Well Known Structures (WKS), are prescribed for use when exchanging geographic information between different applica-

tions on different sites. The WKS were used internally in my application as method parameters and were as such used for bringing geometric information between the graphical device and the database. When collecting large amounts of data, all of which had to be returned as part of the same structure, very much time was spent building and parsing this structure. I assume then, that using these structures as part of a protocol for exchanging information it will evidently lead to large datavolumes sent out on the inter- or intranet, and there will be an obvious need for compression routines.

Forord

Denne rapporten oppsummerer det arbeidet som ble gjort høsten 1996 og tidlig i januar 1997 av Knut Aage Aksnes, som hoved oppgave ved Institutt for datateknikk og telematikk, Norges Teknisk-Naturvitenskapelig Universitet.

Oppgaven bestod i å sette seg inn i pågående standardiseringsarbeide for geografiske informasjonssystemer, utført av Open GIS Consortium (OGC)i USA. En objektorientert datamodell for geodata, modellert av OGS, skule brukes som utgangspunkt for en implementasjon på to ulike databaseplattformer. Databasesystemene som skulle benyttes var Shore, et objektorientert databasesystem, og Postgres95 som er en relasjonsdatabase utvidet med objektorienterte egenskaper.

Oppgaven ble foreslått av databasegruppa ved IDT, faglærer er Kjell Bratbergsengen, og veileder har vært Kjetil Nørvåg, IDT.

Under arbeidet med hovedoppgaven har jeg fått bistand fra flere hold, og vil her få rette en takk til disse. Ved Telenors Forsknings og Utviklingsavdeling på Lillehammer har Bård W. Tørustad og Stein Helge Riise gitt hjelp i praktiske programmeringstekniske problemer og ved å komme med innspill av ulike slag. Statens Kartverkskontor på Lillehammer stilte også velvillig opp med veiledning om SOSI-standarden og om referansesystemer brukt i Norge for stedfesting av geografisk informasjon. Mine tidligere arbeider i faget 45073 Databehandlingsprosjekt [28] og erfaringer derfra har også vært svært nyttige. Aller mest hjelp har jeg imidlertid fått av min veileder Kjetil Nørvåg, som har vært en aktiv støttespiller i hele prosjektperioden.

Trondheim 17. januar 1997
Knut Aage Aksnes

Kapittel 1 Innledning

1.1 Prosjektbakgrunn

Denne rapporten beskriver det arbeidet som er gjort som hovedoppgave på fagretning for databaseteknikk ved fakultet for elektroteknikk og teleteknikk, Norges Teknisk-Naturvitenskapelige Universitet høsten 1996. Oppgaven er foreslått av databasegruppa ved IDT, faglærer for oppgaven er Kjell Bratbergsengen, og veileder er Kjetil Nørvåg.

Oppgaven tar utgangspunkt i pågående standardiseringsarbeider med objektorienterte datamodeller for geodata, utført ved Open GIS Consortium i USA. Her har modeller for geodata blitt utviklet som en av flere standardiseringsprosjekter med sikte på å legge tilrette for åpne systemer for geografisk informasjon.

En forenklet utgave av denne modellen skulle implementeres på to ulike databasesystemer; et objektorientert og et objekt-relasjonsdatabasesystem.

Et databasesystem kalt Shore ble valgt som objektorientert databasesystem. Dette systemet er utviklet ved University of Wisconsin-Madison i USA, og er tilgjengelig uten kostnader. Dette systemet har jeg tidligere benyttet i en lignende oppgave som vårprosjekt ved fakultet.

Som objekt-relasjonsdatabasesystem ble valgt Postgres, som også er gratis programvare. Dette systemet er utviklet ved University of California at Berkeley, og har flere objektorienterte egenskaper.

Oppgaven ble dermed å sette seg inn i modellen spesifisert av Open GIS, og velge ut en avgrenset del av deres datamodell for implementasjon. Videre å sette seg inn i egenskapene til de to databasesystemene og implementere datamodellen på disse.

1.2 Organisering av arbeidet

Prosjektarbeidet er hovedsakelig utført ved min arbeidsplass hos Telenor FoU på Lillehammer. Kommunikasjon med veileder har foregått ved hjelp av elektronisk post og telefon, noe som har fungert bra.

Innledningsvis ble tiden brukt til litteraturstudier av spesifikasjoner for åpne geoprosess-systemer. Kunnskapene fra dette ble så brukt for å designe og implementere en liten applikasjon.

Programmering og uttesting av applikasjonen har pågått fra midten av november til tidlig i januar. Deretter har jeg arbeidet med prosjektrapporten.

1.3 Organisering av rapporten

Rapporten er organisert slik:

- I kapittel 2 gis en generell omtale av geografiske informasjonssystemer
- Kapittel 3 tar for seg spesifikasjonene for åpne geoprosess-systemer, og datamodellen som er utviklet av Open GIS.
- En omtale av hvilke objektorienterte egenskaper som er ønskelig i et databasesystem som skal håndtere geodata gjøres i kapittel 4.
- I kapittel 5 gjennomgås objektorienterte databaser, med omtale av begreper og utrykk som forbindes med slike databaser.
- Kapittel 6 omhandler Postgres, og hvilke egenskaper dette databasesystemer har.
- Avsluningsvis, i kapittel 7, blir applikasjonen som er implementert gjennomgått, med design og implementasjon.

I vedlegg til rapporten er kildekoden til applikasjonen tatt med.

Kapittel 2 Geografiske Informasjonssystemer (GIS)

2.1 Innledning

Et geografisk informasjonssystem (GIS) er et informasjonssystem designet for å arbeide med kartverk som er digitalt representert. GIS omfatter flere elementer. Hovedvekten her blir lagt på

- hvilke elementer GIS-systemer omfatter,
- krav til datalagring og -manipulering,
- behov for utveksling av informasjon mellom ulike GIS-systemer.

Geografisk informasjon som er digitalt representert benevnes ofte *geodata*. En database som inneholder geodata kalles en *geografisk database*. For å utføre planlegging og analyser hvor kartverksinformasjon ligger til grunn, må man kunne operere på dataene i den geografiske databasen. Dette kan være spørringer mot data, endringer ved inntegning av nye data på et kart, og fjerning/endring av bestående objekter.

Til planleggingsformål er det vanlig å arbeide med ulike *tematiske lag* av kartverk. I bunnen ligger det aktuelle geografiske området med et topografisk kart. På ett nytt lag tegnes f eks inn planlagte vann/kloakkfremføringer, mens et annet lag kan inneholde planlagte boligbyggingsområder. Flere slike lag med kartinformasjon legges så sammen for å vise et kart i en ønsket planleggingssituasjon.

Figur 1. Eksempel på lagdeling av kart

I en database kan slik lagdeling representeres enten ved å beholde en referanse til hvilket lag de ulike objektene i kartet hører til, eller ved at vising av kartinformasjon på skjerm eller papirutskrift omfatter spesifikke objekter.

2.2 Funksjonelle elementer i et GIS.

Elementene i et generelt GIS er innsamling og bearbeiding, lagring, manipulering og analyser av data, samt produktgenerering. Disse elementene gis en fyldigere omtale i det etterfølgende, basert på [6] og [19].

2.2.1 Datainnsamling og -bearbeiding

GIS inneholder informasjon om et kontinuerlig flerdimensjonalt rom. Men databaser er endelige og kan ikke representere kontinuerlig informasjon korrekt. Før lagring i en database må en derfor gjennom en diskretisering av de kontinuerlige data. Denne prosessen benevnes vanligvis som digitalisering i forbindelse med innsamling og registrering av geografisk informasjon. Typiske komponenter som brukes for innsamling av data er scannere, digitaliseringsbord, satelittfotografing mm. I tillegg er det ofte behov for manuelle registreringer av data.

Digitalisering av kontinuerlige data er en tilnærming av virkeligheten, og introduserer feil i registerte data. En feilkilde er bruken av et flerdimensjonalt rutenett over overflaten som skal digitaliseres som er ganske vanlig. Bare data som ligger i krysningspunkter i rutenettet registreres. Målefeil kan også forringe nøyaktigheten for de attributter som måles i et slikt koordinatpunkt. Ethvert gitt målepunkt på et målefoto skal korrespondere med er med et punkt i virkeligheten. Denne korrespondansen må vil bare bli tilnærmet riktig, slik at vi også her introduserer en feilkilde.

Etter digitaliseringen vil egenskapene som modelleres være representert ved koordinater. Disse koordinatene er gitt av det referansesystemet som benyttes. Et utall av slike referansesystemer finnes, og også innenfor Norge benyttes flere slike.

Innsamlede data beskriver i hovedsak to geografiske typer data; *egenskaper* og *fenomener*. Egenskaper er reelle entiteter, eller en abstraksjon av (en del av) den virkelige verden. Eksempler på egenskaper er bygninger, elver, veier, osv. Fenomener er abstrakte entiteter; f eks beskrivelsen av temperaturdistribusjon over et areal. For slike fenomener tilordnes vanligvis alle koordinatpunkter en verdi fra domenet for fenomenet som skal beskrives.

2.2.2 Datalagring.

Etter at data er samlet inn og digitalisert må de organiseres og lagres.

Mange GIS har en tematisk inndeling. I figur 1 er det vist eksempler på inndeling i tematiske lag. For en praktisk arbeidssituasjon vil gjerne et lag inneholde flere attributter som f eks planlagte veier og hus. For lagringsformål er det vanlig å splitte lagene ytterligere og la hvert lag representere verdien til *en* unik attributt i det geografiske rommet. Innen et slik lag kan man fritt splitte opp og dekomponere basert på en bestemt verdi eller et verdiområde av attributten. Dette kan illustreres slik: Et lag kan være basert på koordinatpunktenes høyde over havet. Laget kan da fritt deles opp i et

sublag som inneholder bare punkter med en gitt høyde over havet, eller som ligger innenfor en bestemt øvre og nedre grense. Det kan være nyttig å lagre de oppdelte lagene i stedet for hele samlingen av punkter for et gitt lag. Noen slike lag eksisterer kun for å bestemme attributter for det tematiske laget, og må da behandles forskjellig fra selve attributtene. I eksemplet med høydeangivelser kan en oppdeling bli gjort for å tegne høydekoter på et kart, og samlingen av punkter med en gitt høyde eksisterer kun for å bestemme arealet som skal tegnes innenfor en gitt kote.

To typer representasjoner eksisterer innen hvert lag; raster- og vektorangivelser.

- *Rasterdata* får en når en for et hvert koordinatpunkt i det rutenettet som benyttes tilordner verdier til relevante attributter.
- Med *vektordata* representeres geometriske objekter som punkter, linjer og polygoner. Eksempelvis lar en innsjøer og bebyggelse bli representert av polygoner, mens veier og elver blir representert med linjer (eventuelt som polygoner ved tegning med høy oppløsning).

Både raster- og vektordata kan benyttes for tredimensjonaler registreringer.

GIS data kan også inkludere temporale data. Eksempel på slike temporale data er utbredelsen av havoverflaten som varierer med flo og fjære, eller trafikkintensitet som i et trafikkovervåkingssystem der trafikken på veiene som overvåkes varierer over tid. Slike temporale data vil ofte ha stor interesse for sluttbruker.

2.2.3 Manipulering av data.

Geografiske informasjonssystemer brukes på mange områder. Miljøundersøkelser og -overvåking, byplanlegging, ruteplanlegging for transportselskaper er alle eksempler på applikasjonsområder der GIS brukes for beslutingsstøtte. Analyser som gjøres i slike systemer er oftest helt forskjellige fra tilsvarende analysere i konvensjonelle relasjonsdatabaser.

Operasjonene inkluderer

- geometriske operasjoner som avstand, arealer og volumer, snitt og union av arealer og volumer
- topologiske operasjoner, som korteste vei beregninger og utlegg av nettverk
- temporale operasjoner, der informasjonen som søkes knyttes opp til tidspunkter eller tidsintervall.

For et transportselskap vil det for eksempel være av stor interesse å beregne korteste vei for kjøreoppdrag, eller å beregne korteste kjøretid når en tar hensyn til rushtid ol. For et kabelselskap vil det være nyttig å få tegnet opp og kostnadsberegnet et kabelnettverk som må følge gitte beskrankninger.

2.2.4 Produktgenerering

Utdata fra et GIS må være vesentlig mer sofistikert enn hva tekstbaserte grensesnitt kan gi. Mange typer utstyr benyttes, slik som plottere av ulike slag, laserskrivere osv. Presentasjonen av utdata kan inkludere grafiske bilder basert på vektor- eller rasterdata, i

form av et eller flere kart, der hvert tematisk lag kan vises for seg. Visualisering i 3D er blitt ganske vanlig, likedan animasjoner for å vise hvordan temporale data endrer seg over tid. Produksjon og trykking av kart til ulike formål er naturligvis også viktig for de fleste GIS.

2.3 Krav til datalagring og -manipulering

Et GIS er en kompleks applikasjon der en bruker må kunne registrere, analysere og hente ut data av både til romlig og ikke-romlig art. I dette avsnittet blir behovene innen datamodellering, spørring og dataintegritet litt nærmere gjennomgått. Hovedvekten blir lagt på behov knyttet til manipulering og modellering av romlige data. Gjennomgangen er basert på [6].

2.3.1 Raster og vektor data

Som tidligere nevnt er det rasterdata eller vektordata som benyttes i GIS. Rasterdata er en tabell der hvert element representerer verdien til en attributt. Rasterbilder er typiske eksempler på slike rasterdata. I 2D kalles basisenheten vanligvis en pixel, mens en i 3D bruker benevelsen voxel. Datamengden som skal lagres blir vanligvis svært stor for rasterdata, eksempelvis vil et fargebilde i med 1024x1024 pixler og 16 bits representasjon av hvert pixel gi 16 MB. Typiske applikasjoner som benytter rasterdata er systemer som overvåker naturressurser og transportapplikasjoner, der bilder av veier brukes for å kontrollere og oppdatere kartverk. Et rasterbilde har som oftest behov for å lagre data om bildet som f eks kan være en beskrivelse av hvilke objekter som finnes. Operasjoner som utføres på bilder er f eks kantdetektering. Resultatet etter kantdetektering kan da lagres om rasterdata, eller etter ytterligere analysering som vektordata.

Vektorobjekter inkluderer punkt, linjer og polygoner som er representasjoner av objekter i den virkelige verden. Et punkt er en abstraksjon av en entitet med lav oppløsning, for eksempel kan en by representeres som et punkt på et kart i stor skala. Linjer er abstraksjoner for grenser eller ferdselsårer. Polygoner er abstraksjon for en region av en viss utstrekning og som kan representeres av tre eller flere linjer.

GIS bør kunne behandle både raster og vektormodeller.

2.3.2 Komplekse objekter

Oppdelingen av et geografisk område representer oftest komplekse objekter. Eksempelvis vil den administrative oppdelingen av Norge i fylker, kommuner, byer, tettsteder, valgkretser osv gi en svært kompleks struktur. Å modellere grensene for de ulike administrasjonsnivåene og samtidig kunne ta høyde for endringer av grenser er kort sagt ikke noen enkel sak. Grensene kan betraktes som et nettverk, på samme måte som veinettet på et kart representerer et nettverk. Slike nettverk bør kunne modelleres i GIS, likedan oppsplittinger av disse.

Modellering av nettverk som objekter er ikke vanlig i GIS i dag, men det forskes en del på området. I stedet blir nettverkene modellert ved deres basisobjekter, punkter og linjer og regioner. Dette kompliserer f eks beregning av korteste vei, og manipulering av nettverkene.

GIS-databaser må kunne håndtere komplekse objekter.

2.3.3 Temporale og dynamiske data

I tillegg til å ha en kompleks struktur vil også data i GIS kunne ha variasjoner over tid. Temporale data har ikke blitt brukt i GIS så lenge. Transportselskapers behov for planlegging av ruter, eller veimyndighetenes overvåking av veinettet, er eksempler på hvordan temporale variasjoner overvåkes og utnyttes i analysesammenheng.

Lignende eksempler har man i systemer som modellerer grenser for vann og havområder. Grensene for havområdene varierer som kjent med flo og fjære, og påvirkes også av en rekke andre forhold. Flommen på Østlandet sommeren 1995 viste jo også at vannområder i innlandet kan variere mye i utbredelse. For å sette inn flomtiltak vil modellering og analysering i et GIS kunne benyttes.

Mange GIS vil ha behov for å kunne lagre slike temporale data.

2.3.4 Utvidbarhet

GIS benyttes i en mengde forskjellige applikasjonsområder, som f eks økologi, miljøovervåking, byplanlegging og transport. Innenfor hvert område har man behov for å kunne defineres et sett av datatyper som modellerer objektene innefor området. Et GIS bør ha et basis typesystem som dekker kjerneobjektene. Men typesystemet må være utvidbart slik at applikasjonsspesifikke typer kan defineres.

2.3.5 Datakonsistens og kvalitet

GIS må sørge for at enhver transaksjon bringer databasen til en ny konstistent tilstand. Det betyr at GIS må inkludere egenskaper som bestandighet, mulighet for deling av data og inneholde integritetskontroller. Verd å merke seg er også at transaksjoner må kunne være langvarige. Digitalisering av et kart er eksempel på en transaksjon som kan ta lang tid.

Datakvalitet kan illustreres med analyseeksempler. Et veinett som er digitalisert ut fra rasterbilder vil ha flere tilnærminger og unøyaktigheter i forhold til den virkelige verden. Analyser som gjøres f eks på avstander mellom to punkt må derfor tolkes under hensyntagen til feiltoleransene for de registrerte data. Dermed er det også essensielt at GIS lagrer feiltoleransen for lagrede data.

2.3.6 Spørre- og analysemuligheter

GIS bør støtte spørring og analysering mot lagrede data på ulikt vis.

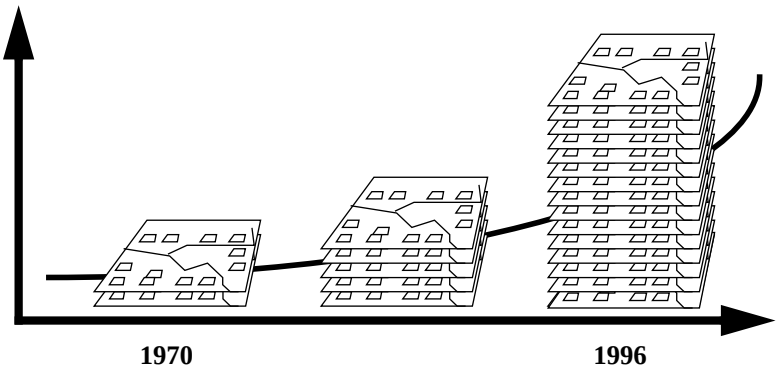
- Målinger. Beregning av lengder, araeler og volumer er fundamentale operasjoner i et GIS.
- Omfangsanalyser. Dette er analyser som går ut på å finne “interessante” soner rundt gitte objekter. Eksempler på slike analyser er flomanalyser; finn ut hvor mye landareal som blir oversvømmet dersom vannstanden stiger 1.5 meter.

- Bildebehandling. Bilder bør kunne behandles med tanke på å generere vektordata. Essensielt i så henseende er kantdeteksjon og lokalisering av arealer gjennom mønstergjenkjenning. Likedan er det interessant å kunne sammenligne registrert vektordata med nyere bildedata.
- Overflatemodellering. Basert på vektor eller rasterdata med 3D data registrert må en kunne konstruere 3-dimensjonale bilder.
- Nettverksanalyser. Nettverk er et viktig aspekt ved GIS. Operasjoner her kan være dynamisk segmentering og analyser basert på nettverk. Dynamisk segmentering har en dersom GIS kan beregne koordinater ut fra lineære referanser. Det kan for eksempel gjøres ved å bestemme et punkt på en linje ved interpolasjon.

2.4 Utveksling av informasjon mellom GIS-systemer

De første GIS-systemer så dagens lys for vel 25 år siden, og antallet slike systemer har økt hurtig, som illustrert i figur 2. Utbredelsen av datamaskiner forbedring av teknologi gjør dette stadig mer attraktivt. Mange ulike metoder for innsamling, lagring, prosessering og vising av geodata har blitt utviklet, ofte uavhengig av hverandre. Dette har gjort av de fleste systemer er proprietære og heller ikke kan utveksle informasjon direkte. Et utall formater har blitt utviklet for utveksling av informasjon, men til nå er ingen av disse egnet for kommunikasjon over det globale internettet som har blitt så populært de senere år.

Figur 2. Antall GIS-systemer vokser raskt



2.4.1 SOSI-standarden

I Norge har Statens Kartverk ansvaret for standarder og regelverk innen kart og oppmåling. De har også ansvaret for SOSI-standarden[20] (Samordnet Opplegg for Stedfeste Informasjon), som inneholder standardiserte beskrivelser av geometri og topologi, datakvalitet, koordinatsystemer, metadata i form av informasjon om eier, oppløsning på data, områdeavgrensning osv. Og ikke minst viktig i denne sammenheng, den omfatter også databeskrivelser for ulike datatyper og anvendelsesområder.

Figur 3. Eksempel på SOSI-fil

```
.HODE
..TEGNSETT ND7
..TRANSPAR
...KOORDSYS 1
...ORIGO-NØ 309900 -2800
...ENHET 0.010
..OMR]DE
...MIN-NØ 309900 -2800
...MAX-NØ 311000 -1100
..SOSI-VERSJON 2.0
..SOSI-NIV] 4
..DATO 1993039
..KVALITET 55 200 1
..PRODUSENT "GJØVIK INGENIØRHØGSKOLE"
.KURVE 1:
..LTEMA 7001
..DATO 19931015
..NØH
98190 91670 16770
98450 92070 16760
```

Ved hjelp av databeskrivelsen kan geodata beskrives presist og entydig, slik at utveksling av informasjon mellom systemer kan gjøres ved hjelp av SOSI-filer. Som lagringsformat for et system er den imidlertid ikke egnet. GIS-systemer vil som hovedregel ha sitt eget interne format, man kan lese inn SOSI-filer, og skrive egne data ut til filer i SOSI-format.

Som utvekslingsformat er SOSI-standarden meget utbredt i Norge. Gjennom samarbeide med andre organer ivaretas også samordning med internasjonale standardiseringsorganer på området.

2.4.2 DIMAN filoverføringsprotokoll

Som eksempel på andre, og mindre utbredt filformater nevnes DIMAN filoverføringsprotokoll, som benyttes internt i Telenor. Dette er en protokoll som er utviklet og anvendes mellom det PC-baserte grafiske programmet DIMAN og verktøyet DATRAN på IBM-stormaskin for overføring av grafiske bilder og kommandoer.

Figur 4. Eks på DIMAN metafil

```
J062GRF1Q6;G-GRAFR10;G-METAFILXZ15;GRAF-METAFILX-ZB2;68J10;1055647023
&OA0;52;0;32;0;52;0;32;
*T1DDXB1YB1G10APB0;0B0;32B52;32B52;0B0;0
*T2DDXB1YB1G10APB1;1B1;31B51;31B51;1B1;1
*T20DDXB1YB1G10APB0;0B0;10B10;10B10;0B0;0G17AQB1G20ATB5;5C4;Bgfh
G40APB0;1B2;1G41APB0;2B2;2G42APB0;3B2;3G43APB0;4B2;4G44APB0;6B2;6G45APB0;7
G46APB0;8B2;8G47APB0;9B2;9G48APB0;5B10;5
#T1F1PC0;0
#T2F2PC0;0
#T20F20PC1;1
#T20F21PC11;1
#T20F22PC21;1
#T20F23PC31;1
#T20F24PC41;1
#F29XB-8.5YB11.0PC107;-24
G1000ALB1
```

Som figur 3 og figur 4 vil vise er DIMAN-formatet vesentlig mer kompakt enn SOSI-formatet. I dette ligger også SOSI-formatets styrke og svakhet; filen kan lett leses av mennesker og forståes. Men dette medfører at filstørrelsen for blir stor og forholdsvis uhåndterlig. DIMAN-metafiler benyttes kun internt i Telenor og er svært lite utbredt.

Kapittel 3 Åpne GIS-systemer

3.1 Innledning

Som det fremgår i kapittel 2.4 er GIS-systemene som finnes *lukkede*, ved at utveksling av data mellom ulike systemer som hovedregel må skje ved filoverføring, med filformater i henhold til avtalte standarder. Det finnes ingen generelle mekanismer som gjør at brukere kan søke i fjerntliggende systemer etter geodata av interesse, og få tilgang til disse i sin egen applikasjon for videre prosessering.

Open GIS Consortium (OGC) er en amerikansk non-profitt organisasjon som arbeider for *åpne* geoprosesseringssystemer.

Gjennom bruk av internett, intranett og standardiserte løsninger for geografiske informasjonssystem arbeider OGC for at alle geoprosesserings-ressurser kan integreres [21]. For å muliggjøre dette utarbeides spesifikasjoner og standarder for geoprosesssystemer som kan samarbeide over internett. OGC har medlemmer fra leverandører, systemintegratører, akademiske miljøer, offentlige forvaltningsorganer og standardiseringsorganisasjoner. Fra Norge er SINTEF "associate member".

Arbeide med standarder og spesifikasjoner gjøres i ulike komiteer. OGC's tekniske komite har utarbeidet "*The OpenGIS Abstract Specification: An Object Model for Interoperable Geoprocessing*"[22]. Utgangspunktet for modellen som beskrives er å skape et rammeverk for systemutviklere til å lage åpne GIS-systemer. Brukere skal derigjennom kunne aksessere og behandle data fra ulike kilder over et åpent datanett. I dette ligger visjon om at en

- ved å enes om globale standarder for utveksling av informasjon, og
- etablere prosedyrer for hvordan informasjon kan ettersøkes og avgrenses,

på sikt kan utnytte det globale internett til utveksling av geodata like enkelt som man i dag utveksler informasjon via World Wide Web.

Gjennomgangen her av disse spesifikasjonene danner grunnlaget for den implementasjonen som gjøres i diplomoppgaven.

3.1.1 Modelleringsmetode

I sitt arbeide med spesifikasjoner har OGC's tekniske komite brukt en metode beskrevet av Cook og Daniels [24]. Dette er en metode som ligger nær opptil "Object Oriented Modelling and Design" beskrevet av Rumbaugh et al [25].

Sentralt i metoden står utarbeiding av tre modeller:

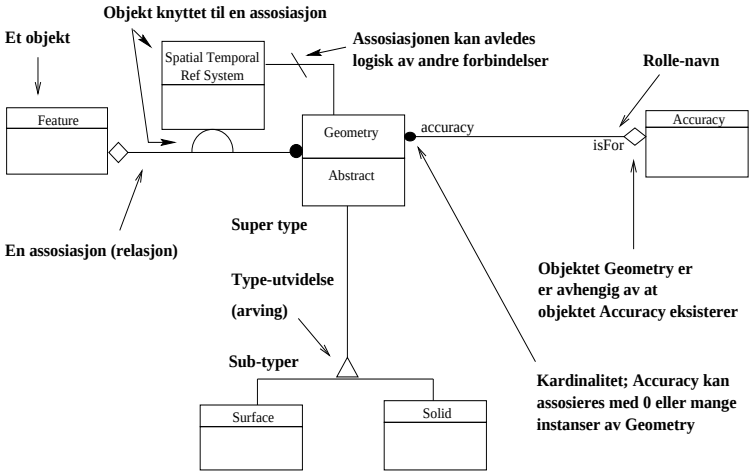
- *Essensiell (abstract) modell* - en modell av "fakta" bestående av virkelige objekter (entiteter, attributter og relasjoner) og hendelser. Dette er strukturen (kodifiserbar) i den virkelige verden slik den oppfattes av de som lager spesifikasjonene.
- *Spesifikasjonsmodell* - en generell modell av programvaren, hvilke tilstander den kan være i og hvordan den reagerer på hendelser, ved å endre tilstander og produsere responser (som også er hendelser). Modellen består av *ideell* programvare og *ideelle* hendelser.
- *Implementasjonsmodell* - modell av programvaren i gitte maskin-/programvareomgivelser og hvordan programvaren kommuniserer med sine omgivelser.

OGC har i "The OpenGIS Abstract Specification"[22] utarbeidet en essensiell modell og en spesifikasjonsmodell.

I det etterfølgende gjennomgås den essensielle modellen, som er grunnlaget for øvrig modellering. To spesifikasjonsmodeller er utarbeidet av OGC; "Open Geodata Specification Model "og "OpenGIS Services Specification Model". Geodatamodellen gjennomgås her. I denne gjennomgangen har jeg valgt å bruke de engelske benevnelsene, da det er vanskelig å finne gode og dekkende norske ord.

Diagrammene som brukes er en variant av Rumbaugh's OMT konvensjoner. De symboler som benyttes er vist i figur 5.

Figur 5. Basis notasjon for objektdiagrammer



3.2 Essensiell modell

3.2.1 Generelt om geografisk informasjon

Geografisk informasjon og systemer som prosesserer disse er allerede omtalt i kapittel 2. Her omtales nærmere ting som er relevant i forbindelse med den essensielle modellen.

Den essensielle modellen er en generalisering av geografiske informasjonssystemer og deres brukere. Geodata representeres ulikt i forskjellige systemer. I tillegg vil informa-

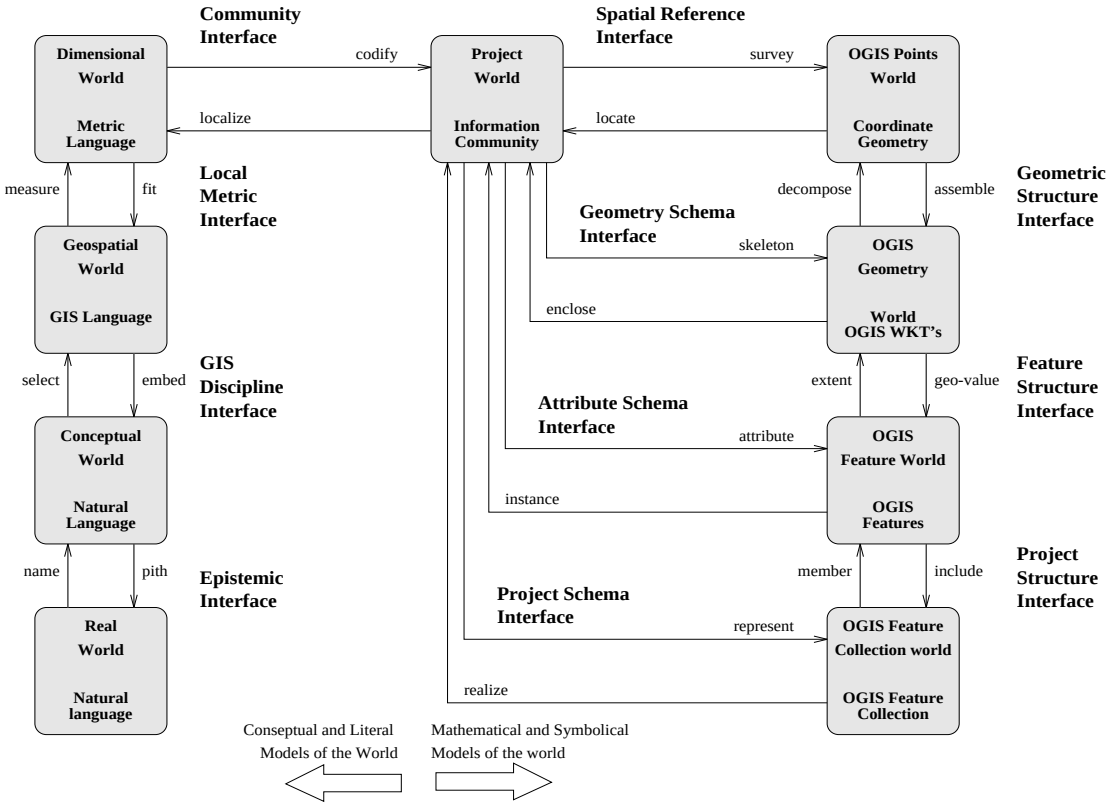
sjon bli tolket ulikt i ulike miljøer, avhengig av hva informasjonen skal utnyttes til. Ofte brukes ulike navn for samme egenskaper. Dette gjør det også vanskelig å benytte data innsamlet og registrert i ett miljø i et annet miljø. Åpne geodata-systemer må kunne gi informasjon om hvordan attributter skal forståes.

3.2.2 Geografisk informasjonssamfunn

OpenGIS introduserer begrepet Geographic Information Community (GIC). Begrepet dekker en samling av systemer og/eller individer som deler digital geografisk informasjon gjennom et felles sett av definisjoner og koderegler. En GIS-applikasjon og dens brukere er et eksempel på en GIC. En åpen GIS-applikasjon må ha mulighet til å annonsere sin eksistens, og det må finnes generelle oversettingsmekanismer slik at geografisk informasjon enkelt kan letes opp og utveksles mellom ulike applikasjoner og deres brukermiljøer.

Den essensielle (generelle) modellen for en GIS-applikasjon og dens brukere omfatter 9 abstraksjonsnivåer. Disse er vist i figur 6.

Figur 6. Essensiell modell; verdensbilde[1]



"*Real World*" (virkeligheten) er samlingen av alle fakta enten de er kjent for mennesker eller ikke. Bare en del av virkeligheten er kjent.

1. Hentet fra [22].

"*Conseptual World*" er vår verden av naturlige språk. Ting vi ser i virkeligheten setter vi navn på (abstraherer dem) og kommuniserer med hverandre ved å bruke disse navnene.

Den konseptuelle verden er ikke tilstrekkelig abstrakt for GIS. Kun et forenklet subsett av den konseptuelle verden er interessant, og dette kalles "*Geospatial World*" Språket som benyttes i *Geospatial World* er et GIS-disiplin språk.

Ved å ta i bruk måleutstyr kan Geospatial world abstraheres ytterligere til "*Dimensional World*", hvor alle fakta blir tilordnet måleverdier.

"*Project World*" er den verden som eksisterer innen en spesifikk GIS disiplin. Ulike brukermiljøer kan se de samme fakta på ulike måter, og dette vil reflekteres i deres informasjonssystemer.

Hver prosjektverden har sitt koordinatsystem. Med dette koordinatsystemet kan alle fakta i prosjektverden abstraheres og tilordnes punktverdier. Det betyr at alle egenskaper kan lokaliseres som et sett av punkter i "*OGIS Point World*".

Prosjektverdenens fakta kan også kategoriseres etter geometri, som gir opphav til en "*OGIS Geometry World*". Instanser av fakta er spesifisert ved et sett av attributter med verdier. Disse finnes i "*OGIS Feature World*". Samlingen av alle fakta som inngår i en prosjektverden er "*OGIS Feature Collection World*". Prosjektverden vil ha grensesnitt til *Point, Geometry, Feature* og *Feature Collection Worlds*.

3.2.3 Begrepsdefinisjoner

OpenGIS definerer et sett av begreper som skal danne grunnlag for en lik forståelse av fenomener som observeres og skal registreres. Begrepene som defineres er: Feature Instances, Geometry, Corners, Geometry Schema, Spatial Reference System, Feature Types, Property-Value Pairs, Attribute Schema og Project Schema. Disse strukturene er essensielle for å oppnå informasjonsdeling, og må benyttes. Strukturene omtales litt nærmere i det etterfølgende.

Feature Instances

"Feature" er basis enheten i geografisk informasjon. En GIS-applikasjon er fullstendig definert av hvilke Features den inneholder. Det kreves at alle fenomener og relasjoner mellom slike skal modelleres som Features eller som attributter til Features. Det skal også finnes eksplisitte regler for hvilke Features i *Geospatial World* som skal inkluderes i applikasjon. En instans av Feature identifiseres av en objektidentifikator (OID).

Feature Types

Features som inngår i en samling kan inndeles i typer, Feature Types.

Geometry

Features i Prosjektverden som har geografisk utbredelse, må kunne representeres med et sett av datastrukturer definert av OpenGIS. Typedeklarasjoner for disse kalles "Well-

Known Types" (WKT'er), mens forekomster av typene er "Well-Known Structures". Strukturene omfatter polygoner, linjer, polyhedroner osv. For hver Feature type må det finnes eksplisitte regler for hvilke strukturer som skal benyttes for å representere dets geometri. Strukturene skal inneholde tilstrekkelig informasjon til å kunne rekonstruere de Features de er en del av.

Corners

Det grunnleggende romlige konseptet er "Corner". Corner er et sted som definerer den romlige utbredelsen av de definerte fenomener. Alle datastrukturer som defineres av OpenGIS kan konstrueres av et endelig sett av Corners.

Geometry Schema

Reglene som implisitt følger av Geometry- og Corner-definisjonene kalles geometri-skjema (Geometry Schema). Eksempler på regler er

- for hver Feature Type, hvilken WKT den skal representeres av
- for hver WKS, hvilken Feature Instance den er en del av
- for hver Feature Instance, hvilke WKS'er den består av
- for hver WKS, en liste av Corners den er definert av
- for hver Corner, hvilken WKS den er en del av

Spatial Reference System

Hver prosjektverden lokaliserer sine punkter og geometrier i henhold et koordinatsystem, som kan være spesifikk for hvert prosjekt. For å muliggjøre informasjonsdeling må referanser gitt i ett koordinatsystem kunne omdannes til koordinater i andre koordinatsystemer. Ethvert referansesystem (Spatial Reference System) må da spesifiseres i forhold til andre kjente systemer, slik at det kan lages oversettingsmekanismer. (Se også kapittel 2.2.1 på side 4.)

Property-Value Pairs

Features skal, i tillegg til å kategoriseres etter type og hvilken datastruktur de modelleres med, beskrives med et definert sett av egenskapsnavn og verdityper for disse egenskapene (attributter og attributttype). I OpenGIS-terminologi kalles dette Property-Value Pairs.

Attribute Schema og Feature Schema

Definisjonene av alle Features med deres attributter og attributt-typer kalles et attributtskjema (Attribute Schema). Attributtskjema og geometri-skjema utgjør tilsammen Feature schema.

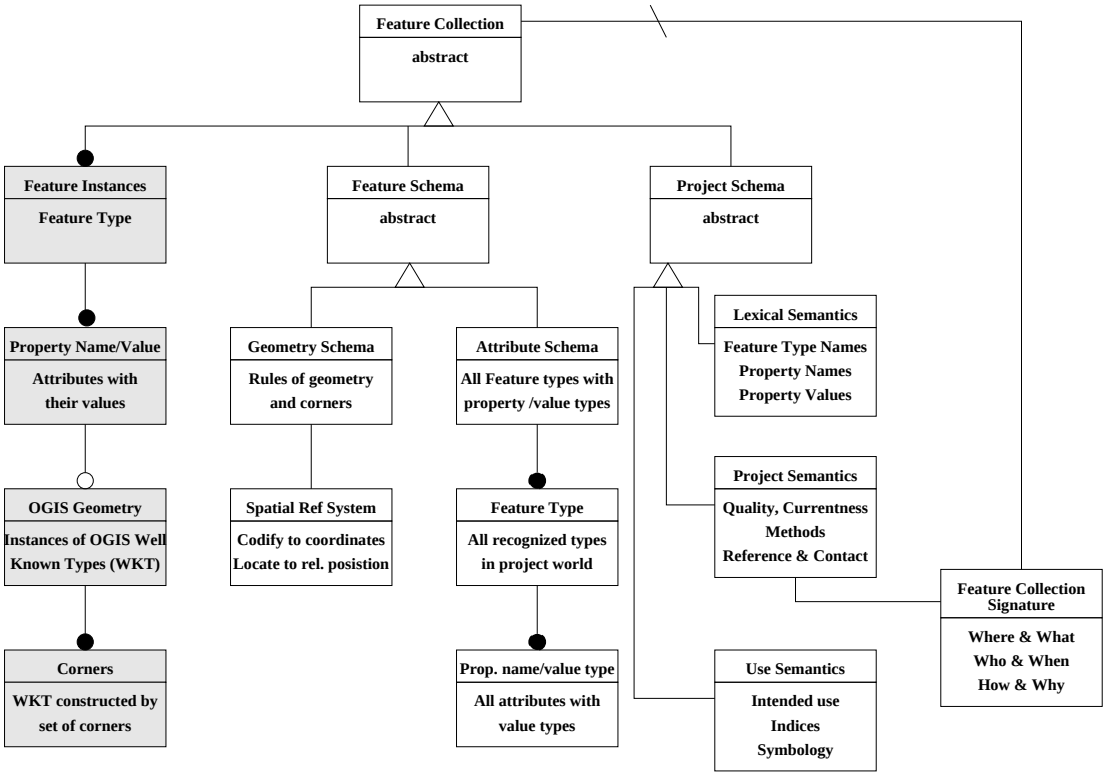
Project Schema

Basis enhet for utveksling av informasjon mellom ulike applikasjoner antas å være en samling av Features. Enhver samling av Features må bære med seg informasjon om samlingen (metadata) slik at den kan tolkes og utnyttes. Et Prosjektskjema (Project Schema) er formalisering av denne metainformasjonen. I figur 7 vises sammenhengen mellom en Feature Collection, de Features den inneholder og hvilke metadata som må inkluderes. Skyggelagte objekter er de som er nødvendig for å modellere Feature Instances i samlingen. Øvrige objekter inneholder metainformasjon, og benevnes i fellesskap metadata.

Prosjektskjemaet inneholder fire deler:

1. En entydig identifikasjon av samlingen
2. Leksikal semantikk; dvs navn på Feature-typer, attributtnavn og -typedefinisjoner
3. Prosjekt semantikk, som er beskrivelser av innsamlingsmetoder, nøyaktighetsangivelser referanser etc.
4. Bruks-semantikk, som er beskrivelser av formålet med systemet og dets samling av Features, bruk av indekser, bruk av symboler etc.

Figur 7. Prosjektskjemaets rolle i en samling av Features



3.3 OpenGIS datamodell for geodata

Datamodellen som beskrives her kalles “Open Geodata Model” (OGM), og er en objektmodell for de sentrale datatypene knyttet til geografisk informasjon.

Enheten for geografisk informasjon er “Feature”, slik denne er definert i foregående kapittel. Et Feature-objekt i programvare korresponderer til en reell eller abstrakt entitet i den virkelig verden. Attributtene til et slikt Feature-objekt beskriver målbare eller beskrivbare fenomener om denne entiteten. Semantikk og bruk av objektene i OGM er avledet fra korresponderende entiteter i den virkelig verden.

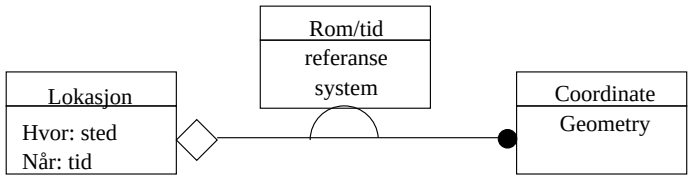
3.3.1 Lokasjon og referansesystemer

Entiteter og fenomener, modellert ved Feature-objekter har ingen mening dersom disse ikke kan stedfestes. En slik stedfesting, representert ved *sted* og *tid*, har ingen korresponderende programvare-entiteter. Stedfestingen er en del av grensesnittet mellom programvaren og den virkelige verden. *Sted* er en målbar del av den virkelige verden. *Tid* er et punkt, et interval eller samling av punkter og intervaller i det som vi oppfatter som tidsforløpet. Tid og sted kan måles og overvåkes, og deres koordinater i et rom/tid referansesystem kan avledes. I OGM-modellen er lokasjon et objekt som har både sted- og tid-attributter.

Et referansesystem er et hjelpemiddel til å tilordne verdier til en lokasjon. Et rom/tid referansesystem har både et romlig og et temporalt koordinatsystem, og som sådan kan det benyttes til å knytte en koordinatgeometri til lokasjoner i sted og tid. Sammenheng mellom Lokasjon, referansesystem og koordinatgeometri er modellert i figur 8.

Romlig dimensjon er antall koordinater som brukes for å beskrive sted og er vanligvis 0, 1, 2 eller 3, mens temporal dimensjon er antall koordinater som brukes for å beskrive tid og er vanligvis 0 eller 1.

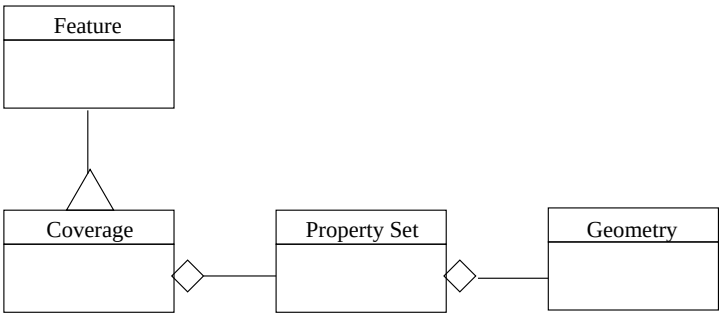
Figur 8. Lokasjon og geometri



3.3.2 Egenskaper og fenomener

Både egenskaper og fenomener er modellert i OGM. Fellesbegrepet for disse i OGM er basisenheten Feature. Feature er definert ved sine attributter og deres typedeklarasjoner (Property Set). Fenomener kalles i OGM for “Coverage”, mens Feature er synonymt med Feature. Sammenhengen mellom Feature, attributter (Property Set) og coverage er vist i figur 9.

Figur 9. Feature og Coverage



3.3.3 Geometri og koordinat-geometri

GIS-applikasjoner håndterer et vidt spekter av geometrier. Modellen tar sikte på å modellere geometrier uavhengig av hvordan applikasjoner vil representere disse ved visualisering.

Geometri defineres å være er kombinasjonen av en koordinatgeometri og et referanse system. Koordinatgeometri består av inntil fire deler:

1. En sekvens av koordinatpunkter fra det samme referansesystemet
2. En samling av andre geometrier som alle har samme referansesystem.
3. En oversettelsesalgoritme som bruker disse geometriene og koordinatpunktene til å 'konstruere' en geometrisk entitet som indirekte definerer omfanget av geometrien i tid og rom.
4. Et referansesystem som oversetter mellom koordinatgeometrien og lokasjon, og dermed gir geometrien en tolkning i den virkelige verden.

Koordinatene for geometri-type objekter skal representeres med OGM's definerte datastrukturer (WKS'er).

I figur 10 er OGM-modellen sammenfattet, og det henvises til denne for den videre gjennomgangen.

Punkt, og punktliste

Et punkt spesifiserer en lokasjon, og har dimensjon 0. En realisering kan være en liste av lengde- og breddegrads-koordinater gitt i av det referansesystemer som benyttes.

Kurve

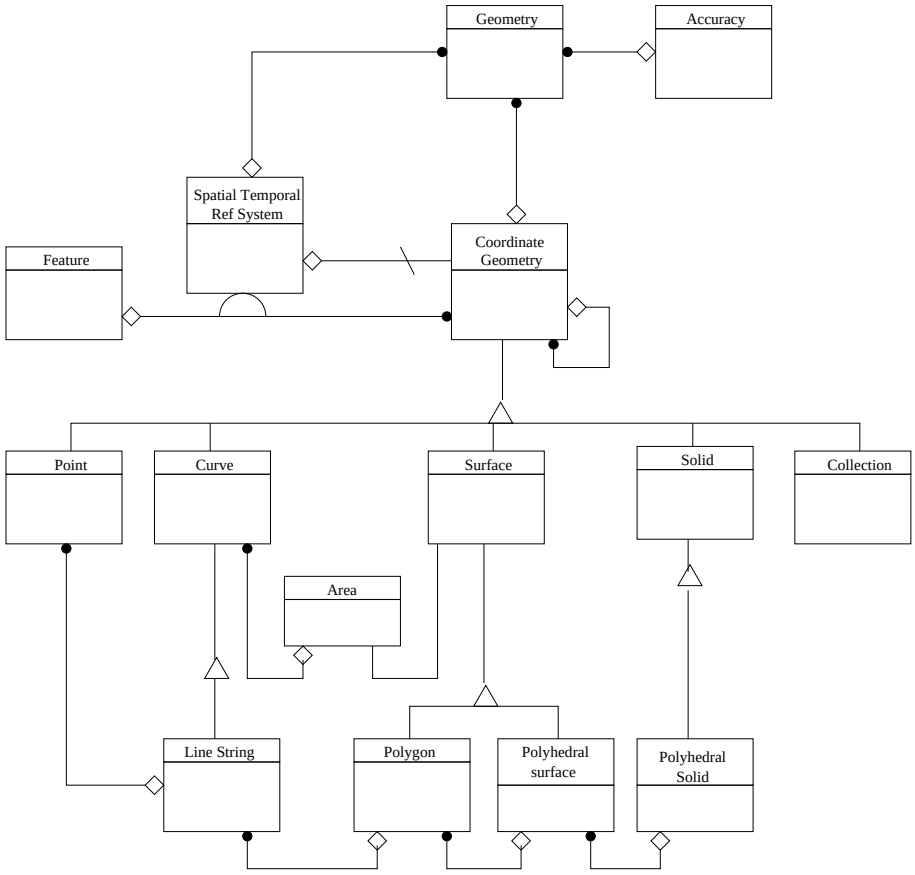
Kurver er endimensjonale geometrier, som vanligvis lagres som sekvenser av punkter. Kurve spesifiserer en hel familie av entiteter som omfatter linje segmenter, buer, splinekurver osv. Linjer er de enkleste, disse blir lineært interpolert mellom de lagrede punktene.

Følgende definisjon gjelder:

- Enkle kurver (Simple) - går ikke gjennom samme punkt to ganger
- lukkede kurver (Closed) - slutter hvor de begynner

- skalerte kurver (Scaled) - parameterisert ved et multippel av linjelengden

Figur 10. OGM Objekt modell, sammenfattet



Linje og Linje-streng

En linje er definert som en rett linje mellom to punkter. En Linje-streng er gitt av et sett av linjer, og spesifiseres ved et settet av linjenes punkter.

Overflater

Overflater er todimensjonale topologier. Strategien for å definere 2 dimensjonale objekter er å dekomponere disse i planare overflater. Overflater er “nestet” sammen langs grensekomponenter og former slik mer komplekse overflater. Hver “enkel” overflate assosieres med en “ekstern grense” der vi kan ta bort et vilkårlig antall hill ved å spesifisere interne grenser.

En overflate dekomponeres således i et sett av enkle overflater. Domene for en enkel overflate er et Areal, som igjen er omsluttet av et sett av enkle kurver.

Polygon, polyhedrisk overflate og polyhedron

Et polygon er en enkel overflate som er planar.

En polyhedrisk overflate er en overflate der alle del-overflater(fasetter) er polygoner. Selv om en polyhedrisk overflate der stykkevis planar, behøver den totalt sett ikke å være planar.

Et polyhedron er en lukket, enkel polyhedrisk overflate. Et polyhedron er ikke planar.

Legemer

Definisjonen av et solid legeme er:
En lukket overflate i tre rom, har et interiør som den skiller fra resten av rommet, dette er et solid legeme.

Hver overflate i et solid legemets yttergrense er lukket. Legemet er interiøret til det største omsluttende overflaten, der andre overflater fjernes som hull.

Collection

Geometri defineres rekursivt ved objektet Collection.

3.3.4 Transformasjoner

Transformasjoner omformer koordinat-geometri i et referansesystem til et annet. I en perfekt verden ville en slik transformasjon ivareta lokasjonen 100% nøyaktig. I virkeligheten vil det oppstå feil. Nøyaktigheten for en gitt transformasjon angis i et eget objekt Accuracy.

3.3.5 Well Known Structures

Datastrukturene som er definert av OpenGIS brukes for å realisere geometrien for objekter slik de er definer her. Disse er gjengitt i vedlegg A, i den formen de er brukt i implementasjonen i diplomoppgaven.

Kapittel 4 Databaser med objektorienterte egenskaper

4.1 Innledning

Fremveksten av mer komplekse applikasjoner som f eks GIS-systemer, har påvist at tradisjonelle relasjonsdatabaser har en del svakheter. Systemene har karakteristikker og krav til databasesystemer som er svært forskjellige fra de tradisjonelle applikasjonene. Dette gjelder blant annet innenfor områder som CAD/CAM (Computer Aided Design/ Computer Aided Manufacturing), bildebehandling og grafiske databaser, vitenskapelige databaser, geografiske informasjonssystemer og multimediadatabaser.

Disse ulikhetene er knyttet til

- mer komplekse strukturer for data,
- transaksjoner av lengre varighet,
- nye datatyper for å lagre bilder eller store mengder tekstinformasjon, og
- behov for egendefinerte operasjoner og datatyper for applikasjonen.

For å møte kravene fra disse applikasjonstypene har utviklingen gått parallelt på å utvide funksjonaliteten i relasjonsdatabaser (RDB), og utvikling av rene objektdatabaser (ODB).

Relasjonsdatabaser forbedres og nye opsjoner tilbys som møter kravene fra de nye applikasjonstypene i varierende grad; slike databaser blir kalt utvidede(exstended) relasjonsdatabaser (ERDB) eller objekt-relasjonsdatabaser (ORDB). SQL3 standarden for relasjonsdatabaser får mange objektorienterte mekanismer.

Objektdatabaser bygger på de samme konseptene som objektorienterte språk bygger på, og er lagringsmedium for objekter. Objektene kan bygges med så komplekse strukturer, og med de datatyper som tillates med det programmeringsspråket som benyttes.

Objektorienterte Databaser med objektorienterte egenskaper er fremdeles et aktivt forskningsområde. Det finnes ingen felles definisjon for en databasemodell, men en felles forståelse av hvilke egenskaper som forventes i et databasesystem eksisterer, og de facto standarder er i ferd med å bli etablert. En må forvente at utvidede relasjonsdatabaser og rene objektdatabaser fortsatt vil leve side om side, men etterhvert med tilnærmet lik funksjonalitet.

4.2 Objektorienterte begreper

Objektorientering (forkortet OO) har sin bakgrunn i objektorienterte programmeringsspråk. OO-konseptene har blitt tatt i bruk i innenfor områder som databaser, programvareutvikling, kunnskapsdatabaser, kunstig intelligens og i datamaskiner generelt.

OO-språk inndeles gjerne i *rene* og *hybride* OO-språk. *Rene* OO-språk er konstruert som objektorienterte språk, mens *hybride* språk inkorporerer objektorienterte konsepter i allerede eksisterende programmeringsspråk. Eksempel på førstnevnte er Smalltalk, mens C++ er eksempel på et hybrid OO-språk.

Et *objekt* kan ses på som en abstrakt maskin med en tilstand og en definerert *protokoll* som må benyttes for å kommunisere med objektet. Objektets *tilstand* beskrives av attributter som er lagret som en *innkapslet* del av memory. Protokoller implementeres typisk som et sett av *meldinger* (*operasjoner*) med typede *signaturer.* En melding kan sendes til et objekt for å få utført en aksjon, og kan medføre at objektets tilstand endres. Så lenge typene til meldingsparametrene stemmer overens med de som er angitt i signaturen, vil objekter som mottar medlingen utføre ønsket aksjon. En melding er implementert ved en *metode*, som er et stykke kode.

Objekters *struktur* er etter dette

- attributter som beskriver objektets tilstand,
- operasjoner definert for objektet, og
- relasjoner til andre objekter.

Strukturen kan være av vilkårlig kompleksitet for å kunne lagre all nødvendig informasjon som beskriver objektet.

Attributter kan *innkapsles* i objektet og er ikke nødvendigvis synlig for eksterne brukere. Også attributter kan være av vilkårlig kompleksitet.

Operasjoner defineres i to deler. I den ene delen defineres operasjonens navn og dens *argumenter (parametre),* dette kalles *signatur* eller *interface*. I den andre delen, som kalles metoden eller kroppen, spesifiseres implementasjon. Operasjoner aktiveres ved å sende en melding til et objekt, som inkluderer operasjonnavnet og parametre. Objektet eksekverer da metoden for den operasjonen. På denne måten kan et objekts interne struktur endres uten å påvirke eksterne program som bruker operasjonene. I noen OO-modeller kreves at alle operasjoner som skal kunne anvendes må være definert på forhånd, slik at objektet er fullstendig innkapslet.

Relasjoner mellom objekter representeres vanligvis ved at referanser til andre objekter lagres som en del av objektets struktur.

Objekter i programmeringsspråk eksisterer bare mens et program utføres. En objektorientert database gjør det mulig å lage objekter som kan være varige (persistente), og som dermed kan deles av flere program og applikasjoner.

4.3 Objektorienterte egenskaper ved databasesystemer

4.3.1 Objekt identitet

Databasesystemet må tildele unike identiteter til hvert uavhengige objekt lagret i databasen. Identiteten tildeles via en systemgenerert "*object identifier*" OID. En OID er ikke synlig for brukere, men er kun til internt bruk i databasen for å identifisere hvert objekt, og for å håndtere referanser mellom objekter.

En OID identifiserer objektet entydig i alle sammenhenger, og må være uforanderlig (*immutable*) slik at OID-verdien for et objekt aldri kan endres. Videre har vi at

- Hver OID må bare kunne brukes en gang, gjenbruk tillates ikke
- OID skal ikke være avhengig av noen av objektets attributtter, da disse kan endres
- OID bør ikke være avhengig av fysisk lagerplass for objektet. Noen systemer gjør det, da dette kan øke effektiviteten ved gjenfinning.
- OO-databaser må ha en mekanisme for å generere OID, som ikke kan endres

Enkle verdier slik som integer, boolske verdier, strenger etc, betraktes også som objekter, men disse behøver ikke egne OID'er. En slik tilnærming ville ha ført til generering av svært mange OID'er, og det er ikke praktisk i en reell database.

4.3.2 Type konstruktorer

Tidligere databasemodeller gir brukere et fast sett av innebygde datatyper og et lite sett av typekonstruktorer. Nye typer kan dannes med typekonstruktorene, men disse nye typene kan ikke ha andre operasjoner en de som allerede er definert for denne typen.

Typesystemet i database må være utvidbart slik at nye typer kan lages. Disse kan bestå av både strukturer og operasjoner. Biblioteker av typer lages, applikasjoner kan så bruke eller modifisere disse typene ved å lage subtyper av disse. Databasen må så sørge for at lagring og gjenfinning av objekter, også de som inneholder store mengder data, blir utført effektivt.

Typekonstruktorene brukes ved beskrivelsen av et databaseskjema. Typedefinisjonene bør også kunne benyttes til type sjekking.

4.3.3 Objekt struktur

Strukturen til objekter konstrueres ved å bruker typekonstruktorene i definisjonsspråket for den gitte databasen. Konstruktorene bør gi muligheter for å definere attributter som kan være

- *atomiske verdier*, som er en av basisverdiene fra systemet (integer, char etc),
- *tupler* av formen $<a_1:i_1, a_2:i_2, a_n:i_n>$, der hver a_i er et attributt navn og hver i_j er en OID,
- *sett*, som er ikke-ordnede lister av OID'er uten duplikater,
- *liste*, en ordnet liste av OID uten duplikater,

- *array*, en array av OID, eller
- *bag*, som er en ikke-ordnet liste av OID'er hvor duplikater er tillatt.

Ved å bruke typekonstruktorene rekursivt på flere nivåer må objekter med vilkårlig kompleksitet kunne konstrueres. Slike objekter inndeles vanligvis i strukturerte og ustrukturerte komplekse objekter, med følgende definisjon av begrepene:

- Et strukturert komplekst objekt genereres ved å bruke de tilgjengelige typekonstruktorene rekursivt på ulike nivåer.
- Et ustrukturert komplekst objekt er en datatype som krever mye lagringsplass, f eks bilder eller store tekstlige objekter.

Eksempler på ustrukturerte komplekse objekter er bitmap bilder, og lange tekststrenger, også kalt binære objekter. Objektene er ustrukturert i den forstand av databasen ikke vet hva strukturen er - bare applikasjonen som bruker dem kan tolke betydningen. Grunnen til at disse betraktes som komplekse er at de krever stor lagringsplass, og de er ikke en del av de standard datatypene som er tilgjengelige i tradisjonelle databaser. Det kan være nødvendig med egne cashing og buffrings-metoder for å aksessere slike objekter.

Alle operasjoner på ustrukturerte komplekse objekter må defineres i metoder for objektene. Eksempler på dette kan være et bilde som lagres som et bitmap, og at det defineres metoder for mønstergjenkjenning som kan kjøres mot bitmap delen av objektet.

Strukturerte komplekse objekter defineres som nevnt ved å bruke de typekonstruktorene som finnes i definisjonsspråket til databasen. Objektstrukturene blir da kjent for databasesystemet.

4.3.4 Relasjoner

Attributter som referer til andre objekter kalles referanser, og representerer relasjoner mellom objekttyper. Relasjoner kan være binære eller flerveis, og kan avbildes som fysiske linker eller som logiske linker. Fysisk link har vi når ett dataelement inneholder en referanse som angir lokaliseringen av et annet data element som lagres, mens en logisk link betyr at ett dataelement inneholder en verdi som identifiserer ett annet dataelement ved en av dens egenskaper.

Binære relasjoner kan være representert i en retning, eller den kan også ha en invers referanse som vedlikeholdes av databasesystemet.

Databasene bør kunne lagre relasjoner, og vedlikeholde inverse referanser.

4.3.5 Innkapsling

Innkapsling er ikke tilgjengelig i tradisjonelle relasjonsdatabaser. I en relasjonsdatabase kan f eks *select*, *insert*, *delete* og *modifisering* gjøres på alle relasjoner i databasen.

Innkapsling kan benyttes på databaseobjekter i en database. Som i OO-språk defineres et sett av operasjoner for en klasse av objekter ved å definere operasjoner som kan

benyttes mot objektet. Objektets interne struktur er skjult, og objektet kan bare aksesseres gjennom de forhåndsdefinerte operasjonene.

Eksterne brukere av et objekt har bare kjennskap til protokollen for objektet, karakterisert ved navn og argumenter til operasjonene (signaturene). Implementasjon av operasjonene og objektets attributter er skjult fra eksterne brukere.

Fullstendig innkapsling av objekter ansees for å være for strikt. Derfor deles vanligvis objektets struktur inn i en skjult og en synlig (åpen) del. Synlige attributter kan aksesseres direkte for lesing og oppdatering av eksterne objekter, eller av et programmeringsspråk. Skjulte attributter kan bare aksesseres gjennom predefinerte operasjoner.

Hovedårsaken til at fullstendig innkapsling ikke blir benyttet er at

- det fører til et stort antall operasjoner for å lese og oppdatere attributter.
- spørrespråk må kunne operere mot objektenes attributter, eller i alle fall utvalgte attributter.

I databasene bør det skilles mellom objektenes skjulte og synlige attributter, og spørrespråkene skal kun aksessere de synlig attributtene.

4.3.6 Definisjons- og programmeringsspråk

Som allerede nevnt konstrueres objekter ved å bruke typekonstruktorer i databasesystemets definisjonsspråk. Definisjonsspråket spesifiserer objektets struktur og protokoll, mens implementasjonen av operasjoner gjøres i et objektorientert programmeringsspråk. Programmeringsspråket som da benyttes blir på denne måten en del av definisjonsspråket.

Det er en fordel om definisjonsspråk og programmeringsspråk ligger nært hverandre i uttrykkskraft.

4.3.7 Multippel arving og selektiv arving

Multippel arving har vi når en subtype arver egenskaper fra to (eller flere) forskjellige supertyper. Selektiv arving har vi når en subtype bare arver noen av funksjonene til en supertype, mens andre funksjoner ikke arves. Selektiv arving støttes ikke av alle OO-systemer, heller ikke multippel arving.

Databasesystemene bør støtte multippel arving.

4.3.8 Versjoner og konfigurasjoner

I mange applikasjoner er det behov for å håndtere flere versjoner av det samme objekter [26]. Slik vil det f eks være tilfelle ved programvareutvikling, der en kan ha design moduler, kildekode, konfigurasjonsdata, testtilfeller etc for et system. Ved vedlikehold av systemet vil det typisk være behov for å håndtere flere versjoner av en kodemodul som skal endres. Det ligger vanligvis til en applikasjon å kunne sammenligne ulike versjoner, liste ulikheter mellom disse, om endringer er korrekte osv. Noen DBMS'er støtter bruk av flere versjoner av objekter, og lager og vedlikeholder en versjonsgraf som viser sammenheng mellom ulike versjoner.

For komplekse objekter der ulike versjoner er tillatt kan man ha en konfigurasjon som viser hvilke sammenstillinger av objekter som hører sammen og som er akseptable (konstistente). Ett konsistent sett av versjoner av objekter betegnes som en konfigurasjon.

En database bør ha støtte for håndtering av ulike versjoner av objekter, og støtte for konfigurering av konsistente sett av versjoner.

4.4 Generelle databaseegenskaper

Objektorienterte databaser skal først og fremst skal et datalagringssystem (DBMS), og som sådan inkludere de egenskaper og funksjonaliteter som finnes i moderne databaser. Slike egenskaper finnes normalt ikke i objektorienterte språk. En del slik egenskaper gjennomgås her.

4.4.1 Bestandighet

En DBMS gir en varig og stabil lagerplass., dvs data er tilgjengelige utover den prosessen som skaper dem. Data er beskyttet mot

- prosessfeil, dvs program som bruker DBMS avslutter unormalt,
- systemfeil, DBMS tåler krasj forårsaket av operativsystem, databasesystem eller hardware-feil
- mediafeil, dvs krasj i lagringsmedium for data, eks magnetisk disk.

For ¨å sikre bestandighet brukes logging tilbakerullingsmekanismer osv.

4.4.2 Deling av data

En DBMS tillater data å deles mellom flere applikasjoner og flere brukere. Her brukes samtidighetskontroll mekanismer som transaksjoner låsing etc. Atomicity støttes, dvs alt eller intet av en transaksjon oppdateres og partielle endringer er ikke synlig for andre brukere.

4.4.3 Tilfeldig størrelse

Adresseområdet for en DBMS er ikke begrenset av prosessorens begrensninger - f eks databasestørresles begrenses ikke av størrelsen på main-memory.

4.4.4 Integritetskontroller

En DBMS kan bidra til å sikre korrekthet og konsistens av data, ved å tilby mulighet for å legge inn beskrankninger. Slike beskrankninger kan være f eks domain-angivelser for simple attributter, nøkler - som angir at noen attributter identifiserer ett element unikt, referanse beskrankninger som sikrer at en referanse i ett element virkelig peker til ett annet element.

4.4.5 Autorisasjon

Aksesskontroll støttes av de aller fleste DBMS'er, og er mekanismer for å etablere eierskap på data, og for å selektere aksessrettigheter for disse.

4.4.6 Spørrespråk

For relasjonsdatabaser er SQL som spørrespråk velkjent. Situasjonen har ikke vært likedan for objektdatabaser, i det ikke har vært etablerte standarder på området. Med definisjonene som er gitt for OQL (Object Query Language) har en imidlertid også fått standarder for objektdatabaser.

Databaser bør ha et spørrespråk, SQL eller OQL, med grensesnitt mot et programmeringsspråk, og med muligheter for direkte interaktive ad-hoc spørringer.

4.4.7 Separate skjema

De fleste DBMS'er har et sentralt skjema for databasen, som er en katalog over typene definert i databasen og navnene til objektene deklarert av disse typene. Ulike program og brukere deler denne meta-informasjonen. Dette i motsetning til programmeringspråk, der hvert program har sine egne definisjoner av typer og variable.

4.4.8 Database administrasjon

Et spesielt grensesnitt for databaseadministrator for å utføre administrative funksjoner, som f eks reorganisering, statistikk innhenting, overvåking av databruk på grunn av sikkerhet og til avregningsformål, brukeradministrasjon og arkivering med kopiering eller fjerning av deler av databasen.

Databasesystemet bør kunne støtte trinnvis utvikling av en applikasjon, der kompleks datastruktur kan implementeres underveis, og tas i bruk for allerede registrerte objekter.

4.4.9 Rapportgenerering

Mange leverandører inkluderer eller selger verktøy for å generere skjema og rapporter utfra høynivå spesifikasjoner.

4.4.10 Distribuerte databaser

Senere databaseapplikasjoner støtter distribuerte databaser over flere maskiner som kan være geografisk adskilte. Distribuerte databaser begrunnes med forbedret ytelse og økt tilgjengelighet.

4.5 Standardiseringsarbeider[1]

Gjennom åttiårene ble det gjort mye eksperimentelt og teoretisk arbeide, men man var ikke enige om definisjonen av en objektorientert database. Et tidlig arbeide for å skape en slik felles forståelse var “The Object-Oriented Database System Manifesto”[3]. Denne ble umiddelbart brukt som en slags sjekkliste av databaseleverandører.

Dette databasemanifestet var et forsøk på å definere et objektorientert databasesystem ved å beskrive hovedegenskaper og karakteristika et slik system må ha for å kunne kalles et objektorientert system. Forfatterne av manifestet var en del ledende forskere på området.

“Third-generation database system manifesto”[4] ble utgitt av “Committee for Advanced DBMS Functions” Dette manifestet beskriver egenskaper et “tredje generasjons databasesystem” må ha. Hierarkiske og nettverksdatabaser refereres da til som første generasjon, relasjonsdatabaser kalles andre generasjons databasesystem. Dette dokumentet argumenterer i hovedsak for å utvide relsajonsdatabasesystemer til en tredje generasjons database, framfor å lage nye objektsystemer.

4.5.1 Objektdatabaser

Det arbeidet som faktisk har blitt en *de facto* standard for leverandører av objektdatabaser ble gjort av “The Object Management Group”. Boken “The Object Database Standard: ODMG-93” [5] ble utgitt i 1993. Denne har siden blitt revidert en gang, og det kom i en ny utgave i 1996.

ODMG-93 standarden er sammensatt av to deler:

- Rammeverk: En del av ODMG-93 som er felles for alle programmeringsspråk. Denne definerer en arkitektur, en objekt-/datamodell, et definisjonsspråk og et deklarativt spørrespråk.
- Språkbindinger: ODMG-93 beskriver i hovedsak arkitektur for programmeringsspråk. En språkbinding må defineres for hvert programmeringsspråk der ODMG-93 skal benyttes. ODMG-93 inkluderer en C++ og en Smalltalk binding.

Det felles rammeverket består av tre komponenter:

- Objekt modellen: en felles datamodell som skal støttes av objektorienterte databaser. Et subsett av objektmodellen sikrer at samme database kan benyttes av program skrevet i forskjellige programmeringsspråk.
- Objekt definisjonsspråk (ODL), som er en syntaks for objekt modellen. Skjema for en applikasjon kan defineres med ODL, og kan etterpå bli oversatt til deklarasjoner for det programmeringsspråket som skal benyttes.
- *Object Query Language* (OQL), er deklarativt språk for spørring mot databaseobjekter. OQL kan benyttes av sluttbrukere eler fra et programmeringsspråk.

1. Basert på [16]

4.5.2 Utvidete relasjonsdatabaser

Også for relasjonsdatabaser arbeides det med endringer i standarder som gjør at disse vil tilby objektorienterte egenskaper. Dette forventes å gjenspeile seg i SQL3-standarden ved å inkludere en utvidet datamodell og prosedurale muligheter. På denne måten vil relasjondatabaser kunne representere mer komplekse data og gjøre SQL til et mer fullverdig programmeringsspråk. Dermed vil også forskjellen mellom objektorienterte og relasjonsdatabaser bli mindre. SQL3 standarden er i hovedsak ferdig, men det gjentår å få innarbeidet denne i kommersielle databasesystemer.

Kapittel 5 Databasesystemet Shore

5.1 Innledning

Databasesystemet Shore ble valgt som objektdatabase for implementasjonen i denne oppgaven. Det skyldes først og fremst at det er gratis programvare, men også at systemet inneholder flere spennende muligheter. I dette kapitlet gjøres en kort gjennomgang av Shore, med en avsluttende oppsummering av egenskaper, sammenholdt med de krav som er listet i kapittel 4.

Shore (Scalable Heterogenous Object REpository)[9] er en objektdatabase utviklet ved universitetet i Wisconsin[1] for UNIX-plattform. Systemet gir mulighet både for objektlagring og lagring av filer som skal kunne aksesseres av andre UNIX-baserte verktøy. Betaversjoner har vært tilgjengelig lenge, og høsten 1996 ble første offisielle versjon av Shore 1.0 gitt ut.

Shore er konstruert for å være uavhengig av språket applikasjoner mot databasen skrives i, noe som realiseres gjennom språkbindinger. Fra en definisjon av persistente data skal Shore's kompilator kunne generere data- og metodeskjeletter for forskjellige programmeringsspråk, se figur 13. Versjon 1 av Shore har imidlertid bare kompilator for C++.

Kommersielle objektdatabaser leveres i dag med spørrespråk, enten slik det er definert av ODMG med OQL, eller en dialekt av denne. Dette gjør databasen mer komplette, ved at man får med det beste fra to verdener. Dersom en objektdatabase med spørrespråk hadde vært tilgjengelig ved prosjektets oppstart ville denne databasen ha blitt brukt i stedet for Shore. Shore-systemet har ikke noe slikt spørresspråk, slik at søking i databasen må gjøres med andre mekanismer.

Databasegruppen ved fakultetet har eveluert diverse kommersielle OODB'er, og har nå gått til innkjøp av systemet O2. Dette systemet støtter ODMG-standarden fullt ut.

5.2 Arkitektur

Shore støtter distribuerte databaser over flere geografisk adskilte maskiner. Dette gjøres gjennom en tjener-tjener arkitektur hvor flere klienter kan jobbe mot flere tjenere[2]. Når Shore kjører gjør den altså det som et sett av kommuniserende prosesser, Shore tjenere,

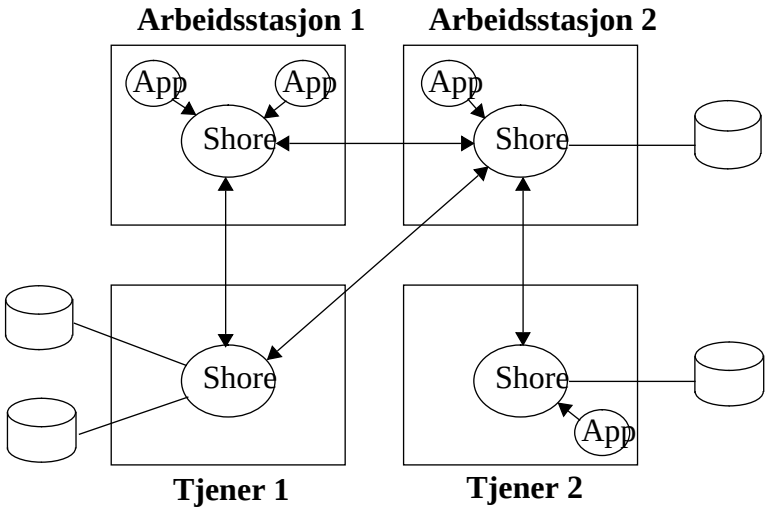
1. Både programvare og dokumentasjon er tilgjengelig over WWW (URL: http://www.cs.wisc.edu/shore/)
2. Versjon 1.0 støtter kun kjøring mot en enkel tjener

hvor hver tjener inneholder persistente objekter og kjørbar kode. Arkitekturen er illustrert i figur 11.

Applikasjonsprosesser merket “App” i figuren manipulerer objekter. Slike prosesser får bare lov til å manipulere objekter den har autorisasjon for. Dermed kan den ikke ødelegge metadata slik som indekser og databasetabeller.

En tjener har flere roller. Den jobber primært med sider (pages) allokert fra diskvolumer, som de lagrer i et sidehurtigbuffer. Disse sidene kan både komme fra lokale volumer og fra andre tjenere hvis en klient ber om et objekt som ikke er lagret lokalt. Versjon 1.0 støtter imidlertid ikke slike fjerne tjenere. Tjenere er også ansvarlig for å sende objekter til klienter når de ber om dette. Klienter sender slike forespørsler gjennom RPC (*Remote Procedure Call*) kall. Til slutt er tjeneren ansvarlig for parallellitetskontroll og gjenervervelse (recovery), for transaksjonshåndtering og for å erverve låser på objekter på vegne av sine klienter. Eieren av hver side (den tjenerprosessen som administrerer den) er ansvarlig for å behandle forespørsler om låsing av objekter, samtidig med at den logger og utfører forandringer.

Figur 11. Arkitekturen i Shore



Denne arkitekturen gir stor fleksibilitet. Ved at Shore prosesser både kan opptre som tjenere og som klienter, kan data distribueres for optimal ytelse. Når en Shore-prosess eier en side opptrer den som tjener for forespørsler mot den siden, når den videresender en forespørsel til en annen tjener opptrer den som klient. Fra en applikasjons synspunkt vil lokalisering av tjenerprosessene (og sider) være transparent, da den kan aksessere persistente data på samme måte enten de er lagret lokalt eller et annet sted.

5.3 Transaksjonshåndtering

En Shore tjener er ansvarlig for å utføre transaksjoner for sine lokale klienter. Hvis en transaksjon inneholder modifiserte data som eies av flere tjenere, utføres en to-fase oppdateringsprotokoll mellom de involverte tjenerne. Transaksjoner som bare medører oppdatering av lokale data utføres direkte. Shore versjon 1.0 støtter ikke nøstede transaksjoner,

Hver tjener har dessuten ansvar for å opprettholde en loggfil for de sidene den har ansvar for og utfører transaksjons-tilbakerulling (*rollback*) og gjenervervelse på objekter i disse sidene. Hvis en tjener lagrer et objekt fra en annen tjener, oppretter den en midlertidig logg for objektet som oversendes eier-tjeneren ved en eventuell tilbakerulling.

5.4 Objekter

Alle persistente dataentiteter i Shore er objekter, og de kan være enkle eller komplekse. Et objekts interne struktur og metodesett er bestemt av objektets type, og kan fastslås gjennom en referanse til et typeobjekt som inneholder informasjon om hvordan objektet skal tolkes. Typeobjektene opprettes av Shore's prekompilator, se figur 13.

Shore identifiserer objekter unikt gjennom objektidentitet (OID) som tilordnes av systemet.

For å støtte lagring av mer spesielle informasjonsentiteter som kan forandre størrelse slik som strenger og tabeller, tillater Shore at et objekt utvides med en *heap* av variabel størrelse. Denne realiseres som logisk sammenhengende blokker på sekundærlager.

Shore tillater dessuten at objekter inneholder referanser, både til verdier og til andre objekter. Dette omtales også i forbindelse med gjennomgangen av Shore's datadefinisjonsspråk - SDL.

5.5 Filsystem egenskaper

Shore kombinerer teknikker fra filsystemteknologi i tillegg til objektorientert databaseteknologi. Motivasjonen for dette er at eksisterende applikasjoner lett skal kunne benytte seg av objekter i en Shore-database gjennom Unix filsystemkall.

Vurdert som et filsystem filbyr Shore to hovedfasiliteter. Et Unix-lignende navnerom tilbys for lagring av objekter, hvor alle objekter kan nås, enten direkte eller indirekte fra en rotunde. Dernest er det laget et sette med mekanismer som tillater andre Unix-applikasjoner å aksessere Shore-objekter gjennom Unix filsystemkall.

5.5.1 Objekt navnerom

Shore gir et navnerom med trestruktur, slik vi kjenner det fra Unix filsystem. Systemets rotkatalog er utgangspunkt for å lage strukturer med kataloger og "filer". Velkjente filsystembegreper som kataloger, symbolske linker og regulære filer utvides med *kryssreferanser*, *pool*, *moduler* og *typeobjekter*, som alle kan opprettes gjennom Shore-definerte funksjonskall i en applikasjon.

Et katalogobjekt i Shore minner mye om en Unix katalog. Konsepter som stinavn, underkatalog, harde- og symbolske linker og rotkatalog er alle definert slik de er definert i Unix. Som i Unix er en katalog et sett av tupler <navn, OID>, der OID kan referere til en et hvilket som helst annet Shore objekt. Verd å merkes seg er kanskje at i denne sammenhengen regnes katalog, linker, filer, pool'er etc som objekter, slik at en OID kan referere til en ny katalog, til en pool osv.

Objekter som opprettes i Shore deles opp i *registrerte* og *anonyme* objekter.

Kataloger og de persistente objekter som opprettes i dem er registrerte objekter. Disse har et sett av systemegenskaper som eierskap, aksessrettigheter og tidsmerker, slik vi kjenner det fra Unix-filsystem. Shore har derfor mulighet for sikkerhetskontroll for dataene i databasen basert på aksessrettigheter for registrerte objekter.

Alle objekter i databasen har ikke behov for f eks aksesskontroll definert pr objekt. For å gi mulighet for slike objekter er det innført en ny type (registrert) objekt som kalles *pool*. Objekter som opprettes i en pool kalles *anonyme* objekter. Anonyme objekter deler systemegenskaper med pool'en de tilhører. Anonyme objekter har ikke stinavn, og må aksesseres gjennom sin OID.

Alle typer objekter kan opprettes i en katalog, som registrert objekt, eller i en pool som et anonymt objekt. I en typisk Shore database vil hovedmengden av objekter bli opprettet som anonyme objekter.

Sletteregler er noe ulike for registrerte og anonyme objekter, idet anonyme objekter kan slettes når som helst, mens registrerte objekter bare kan slettes når de er tomme. Dette medfører bl a at en OID-referanse kan peke på et anonymt objekt som ikke eksisterer.

Tre andre typer objekter introduseres også av Shore; *moduler*, *typeobjekter* og *kryssreferanser.* Moduler og typeobjekter er identiske med hhv pool og anonyme objekter, men har en noe annen slettesemantikk. Kryssreferanser er identiske med en soft-link.

Figur 12. Navnerom i Shore

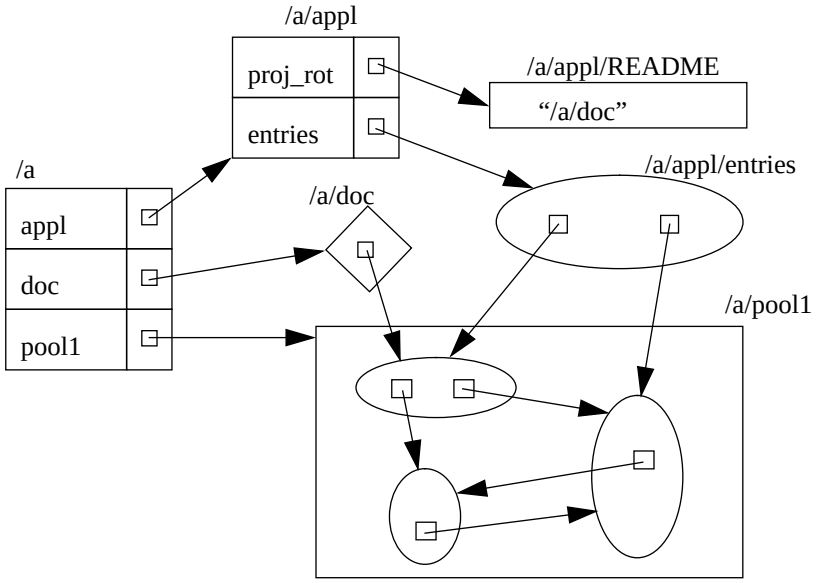


Figure 12, "Navnerom i Shore," on page 34 illustrerer dette. Under Shore sin rotkatalog er det opprettet en katalog "a". Denne inneholder katalogen "appl", kryssreferansen "doc" og pool'en "pool1". I katalogen /a/appl finnes en symbolsk link navngitt README og som peker på /a/doc, og et registrert objekt "entries". Objektet "entries" inneholder pekere til medlemmer i pool1. "/a/doc" er en kryssreferanse som peker til et anonymt objekt i pool1. Dette anonyme objektet kan da aksesseres gjennom /a/appl/README eller gjennom /a/doc gjennom Unix filsystemkall.

5.5.2 Støtte til Unix baserte verktøy

I enkelte situasjoner kan det være gunstig å kunne aksessere registrerte objekter i Shore med Unix-applikasjoner.

For applikasjoner som kan rekompileres tilbys et “Unix compatibility” grensesnitt. Objekter som inneholder attributter av typen *text* kan da aksesseres gjennom modifiserte Unix systemkall (som f eks open, read etc), slik at det er objektenes *text*felter som blir aksessert av applikasjonen.

For applikasjoner som ikke kan rekompileres har man konstruert en NFS-filserver, slik at hele subtreet for Shore kan monteres i et eksisterende Unix-filsystem. Versjonen av Shore som er benyttet her er ikke montert i et Unix-filsystem.

5.6 Databasefunksjoner

Shore inneholder et sett av funksjoner for kommunikasjon med databasen. Disse kan grupperes innenfor områder som transaksjonskontroll, oppretting og sletting av kataloger og vanlige objekter, navigering i navnerom etc. Likedan tilbys indeksmekanismer i form av B-trær, med et sett av tilhørende funksjoner, scanningsmekanismer for gjennomløping av alle objekter i pool’er og kataloger.

Det finnes foreløpig ikke noe tilhørende spørrespråk mot databasen, slik at all gjenfinning og eventuell selektering av objekter må gjøres ved hjelp av traversering av indekser eller ved scanning.

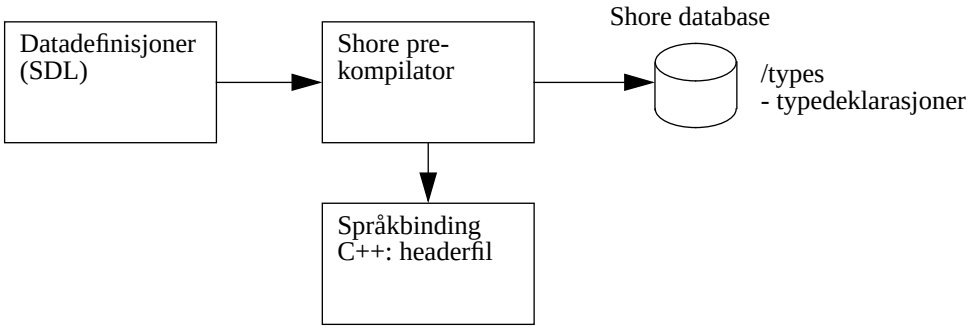
Databaseadministratorfunksjoner som tilbys er minimalt, og baseres på et bruk av Tcl-shell med et begrenset kommandosett. Det finnes ingen funksjonalitet for skjemamigrering.

5.7 SDL - Shore Data Language

Shore sitt datadefinisjonsspråk kalles SDL - Shore Data Language [10], og skal være et språkuavhengig definisjonsspråk. SDL er utviklet paralellelt med ODMG-93 sitt objektdefinisjonsspråket ODL [5]. At arbeidet har skjedd parallelt har medført at språkene har mange likhetstrekk, men de avviker en del fra hverandre.

Gjennom SDL spesifiseres de persistente objekttypene som databasen skal administrere, og metoder for disse objektene. En datadefinisjon i SDL blir så parset av SDL-kompilatoren som oppretter typedefinisjonene i databasen. Basert på disse definisjonene genereres det datadeklarasjoner for objektene i det språket kompilatoren er bygget for. For øyeblikket er det som tidligere nevnt C++. Sammenhengen mellom definisjoner, typer i databasen og srpåkbindingen med generering av headerfil for C++ er vist i figur 13.

Figur 13. Prekompilering i Shore



Alle objekter er forekomster av typen *interface*. Interface kan ha attributter, metoder og relasjoner. Attributter kan være primitive, f eks av type integer, character og real, eller sammensatte. Vanlige typekonstruktorer tilbys; enumerasjon, strukturer, tabeller og *referanser* som brukes for å angi relasjoner. I tillegg har Shore konstruktorer for sett, lister og sekvenser. Gjennom typekonstruktorene kan binære relasjoner av ulik kardinalitet tilbys. Binære relasjoner som defineres som *inverse* vedlikeholdes av Shore. Attributter av typen *text* brukes spesielt for lagring av data som skal kunne aksesseres av andre Unix-applikasjoner gjennom Unix filsystemkall.

5.8 Oppsummering av Shore's egenskaper

5.8.1 Objektorienterte egenskaper

Shore tildeler unike OID'er til alle objekter som opprettes. OID'en kan ikke endres av programmerere og den er uavhengig av objektenes attributter . På programmeringsnivå benyttes den logiske OID'en, som er uavhengig av lagringsplass.

Typesystemet er utvidbart, og gjør det mulig å beskrive objekter med vilkårlig kompleksitet. Beskrivelsene gjøres i SDL, et språk som ligger nær opptil standarden som er definert for objektdatabaser. Typebeskrivelsene lagres i databasen, og benyttes til typesjekking ved eksekvering. Etter prekompilering og generering av typedeklarasjoner i programmeringsspråket vil disse også bli benyttet til typesjekking ved kompilering.

I typesystemet inngår de vanlige atomiske verdier (integer, char etc), utvidet med tekststrenger med variabel størrelse. Videre kan sett, bag, list og arrays av OID'er defineres som attributter. Ustrukturerte komplekse objekter kan også deklareres.

Binære relasjoner mellom objekter kan være representert i en retning eller med inverse referanser, og vil i såfall holdes oppdatert av Shore.

Objekter deklareres med en privat og en offentlig (public) del, og støtter dermed innkapsling. Protokollen for objekter defineres i den offentlige delen av deklarasjonen, og objektene kan bare aksesseres gjennom disse. Shore støtter ikke overlagring av metoder.

Definisjonsspråket SDL er svært likt C++ i syntaks og semantikk, og kan ses på som en utvidelse av C++. Metodene implementeres i C++. Dermed har en ikke noe konseptuelt gap mellom deklarasjonsspråk og programmeringsspråk.

Arving tillates med multippel eller enkel arving, mens selektiv arving (bare enkelte funksjoner arves) ikke støttes.

Det finnes ingen mekanismer for håndtering av flere versjoner av objekter, og det blir opp til programmerer å håndtere dette. Konfigurasjoner har dermed heller ikke noen støtte i Shore.

5.8.2 Generelle databaseegenskaper

Shore inkluderer de vanligste databaseegenskaper som bestandighet, datadeling og tilfeldig størrelse ved hjelp av logging, transaksjoner og mekanismer for tilbakerulling og gjenervervelse (recovery). Lange transaksjoner støttes, men ikke nøstede transaksjoner. Domeneangivelser for attributter kan ikke angis.

Adgangskontroll tilbys som i Unix med mulighet til å skille tilgang på bruker, gruppe og alle. Adgang kan skilles på objektnivå for registrerte objekter, og på poolnivå for anonyme objekter.

Shore har ikke noe spørrespråk. Alle objektene må søkes etter ved å benytte scanningmekanismer eller egendefinerte indekser, og selekteres ved egendefinerte funksjoner. All aksess til objekter må gjøres gjennom de definerte protokollene.

I prekompileringsprosessen genereres typedeklarasjonene som lagres i et sentralt skjema i databasen. Typedeklarasjonene er deretter tilgjengelig for alle, og “eies” ikke bare av programmet som har deklarert typene.

For databaseadministrasjon finnes ingen verktøy inkludert i versjon 1.0. Dermed må all funksjonalitet for reorganisering, statistikker, overvåking, brukeradministrasjon osv programmeres spesielt for hvert program eller man må selv lage biblioteker med slike rutiner. Det samme gjelder rapportgenereringsverktøy. Mangelen på slike administrative verktøy vanskeliggjør også trinnvis utvikling av applikasjoner, der objektdefinisjonene endres underveis. Instanser av objekter som har fått endret sin deklarasjon må enten slettes, eller man må programmere egne rutiner for kopiering fra gammel til ny versjon.

Spesifikasjonene for Shore lover mye med hensyn til distribusjon av databaser over flere maskiner, jfr kapittel 5.2. Nå er imidlertid ikke den omtalte arkitekturen implementert i versjon 1.0, slik at det for øyeblikket ikke finnes noen mulighet for å distribuere databaser. Tilgjengeligheten til Shore er nå slik at applikasjon og database må kjøres på samme maskin. Dersom brukere skal kunne benytte databaser definert i Shore over et nettverk, må applikasjonen ha en klient-tjener arkitektur, der tjeneren må kjøres på samme maskin som Shore.

5.8.3 Vurdering av Shore

Shore har de fleste objektorienterte egenskaper som er nødvendig for komplekse applikasjoner. Med sitt definisjonsspråk SDL og grensesnitt til C++ har det en lav terskel for C++ programmerere med Unix-kompetanse.

Mangelen på administrative verktøy gjør imidlertid at systemet må betraktes som et databasesystem som kan benyttes som grunnlag for å lage en databaseserver. Etter å ha laget en server, kan man tilby denne sammen med Shore for generell applikasjonsutvikling. Databaseserveren bør da inkludere verktøy for databaseadministrasjon, rapportgeneratorer, spørrespåk, implementasjon av flere typer indekser osv.

Shore er imidlertid glimrende å kunne benytte for opplæring i bruk av objektdatabaser.

Kapittel 6 Databasesystemet Postgres

6.1 Innledning

Som utvidet relasjonsdatabasesystem ble Postgres valgt. I likhet med Shore er dette gratis programvare uten restriksjoner på bruken av systemet. Og som med Shore har også PostgreSQL løsninger som ikke er vanlig i kommersielle databasesystemer. Gjennomgangen av systemet her er basert på [27]. Avslutningsvis vurderes PostgreSQL opp mot egenskapene listet i kapittel 4.

Postgres er opprinnelig navnet på et prosjekt ved University of California at Berkeley, som utviklet databasesystemet Postgres i perioden 1986 - 93. Dette var en relasjonsdatabase, men med et avansert "rule system". Flere versjoner av systemet ble laget i denne perioden, med fokus på portabilitet og pålitelighet. "Rule-systemet" ble forbedret underveis, og støtte for multiple lagringsadministratorer ble introdusert. Det opprinnelige prosjektet ble avsluttet med versjon 4.2. Nevnes må også at Postgres ble kommersialisert av selskapet Illustra Information Technologies under navnet Illustra, senere endret til Informix.

Et nytt prosjekt ble så startet opp med Postgres som grunnlag. Dette resulterte i systemet Postgres95 som ble frigitt i september 1995. Flere utgaver har kommet siden da, i denne oppgaven er Versjon 1.0.8 benyttet. Versjon 1.0.9 er tilgjengelig, og ytterligere en ny versjon ventes i løpet av januar 1997. Navnet på systemet har nylig blitt endret til PostgreSQL. Systemet kan kjøres under flere operativsystemer på UNIX-plattform.

I denne rapporten omtales PostgreSQL som Postgres.

6.2 Arkitektur

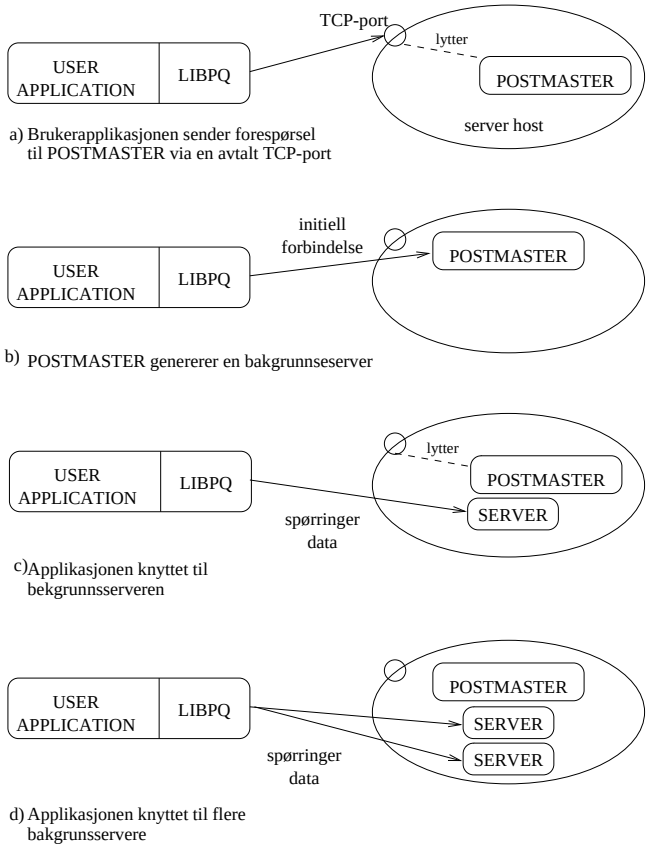
Postgres benytter en enkel "en-prosess pr bruker" klient/tjener modell. En Postgres sesjon består av følgende samarbeidende UNIX-prosesser:

- en demon-prosess, kalt *postmaster,*
- brukerens applikasjon (f eks *psql*-programmet), og
- databaseserver (en eller flere) som kjøres i bakgrunnen, Postgres-prosessen.

Dette er også illustrert i figur 14. En *postmaster* administrerer et sett av databaser på en maskin (en *host*). Applikasjoner som vil aksessere en database gjør et kall til LIBPQ-biblioteket. En forespørsel blir da sendt over nettverk til *postmaster* (a), som igjen starter opp en ny bakgrunnsserver-prosess og knytter applikasjonen til denne bakgrunnsprosessen (b). Applikasjonen kommuniserer da med bakgrunnsprosessen uten å ha forbindelse med *postmaster* (c). En applikasjon kan via LIBPQ opprette forbindelse til

flere bakgrunnsprosesser samtidig (d). *Postmaster* og bakgrunnsprosesser må kjøres på samme maskin, mens applikasjoner kan kjøres på andre maskiner.

Figur 14. Hvordan en forbindelse blir etablert



Det er ingen støtte i Postgres for distribusjon av databaser over flere maskiner. Applikasjoner har bare tilgang til da databaser den har autorisasjon for. Bare “superuser” eller brukere som er autorisert for det kan endre i Postgres’ metadata.

Bakgrunnsprosessene jobber mot sider (pages) som hentes fra diskvolumer via *Postgres* til rammer (frames) i en av flere buffre de er tildelt av postmaster ved oppstart. Rammer i denne bufferpoolen kan splittes i mindre sider, gitt av størrelsen på de klasser som lagres i buffrene. Tjeneren har også ansvaret gjenoppretting (recovery) og transaksjonhåndtering.

Postgres støtter ikke nøstede transaksjoner, og er heller ikke forberedt for langvarige transjaksjoner.

Med Postgres følger *psql*; en programpakke som gjør det mulig å arbeide interaktivt mot databaser med skriving, editering og eksekvering av SQL-kommandoer.

6.3 Avansert SQL-funksjonalitet

Postgres støtter et subset av SQL-92. De viktigste manglene i så måte er

- ingen støtte for primærnøkler
- ingen støtte for nøstede subspørringer

- ikke mulig med HAVING i forbindelse med GROUP BY

SQL er imidlertid utvidet med mulighet for brukerdefinerte typer, funksjoner og arving. Postgres tar i bruk objektorientert terminologi, der grunnkonseptene er:

- en klasse er en navngitt samling av objektinstanser. Hver instans i klassen har det samme settet av navngitte attributter, og hver attributt er av en spesifisert type.
- Hver instans har en unik objektidentifikator (OID) som er unik innenfor en installasjon.
- Tabeller i henhold til SQL syntaks er identisk med klassebegrepet.
- Rader i henhold til SQL syntaks er identisk med en instans.

6.3.1 Arving

Klasser defineres med SQL deklarasjonen; CREATE TABLE. Denne er utvidet slik at det er mulig å deklarere at en tabell (B) arver en annen tabell (A). Da vil instanser av B arve alle attributter fra A, og tabellen som defineres for A får alle felter i A og B.

```
CREATE TABLE A (
  attr1   text,
  attr2   float,
  attr3   int
);
CREATE TABLE B (
  attr4   char2
) INHERITS (A);
```

For select, update, delete osv gjelder da syntaksen som i eksemplet under ved operasjoner mot A og B.

```
SELECT attr1, attr2
FROM A
WHERE attr3 > 12345;
```

Her gjelder søket bare i klassen A.

```
SELECT attr1, attr2
FROM A*
WHERE attr3 > 12345;
```

I dette tilfellet gjelder søket for klassen A og alle klasser som arver A.

6.3.2 Time Travel

Postgres har støtte for hva som kalles *time travel*. Med denne muligheten kan man gjøre historiske spørringer, og få historisk oversikt over hvilke attributtverdier en instans har hatt. Syntaksen for select er utvidet slik

```
SELECT attr1, attr2
FROM A['Start-tidspkt','Slutt-tidspkt']
WHERE.........;
```

6.3.3 Ikke atomiske verdier

Vanlige relasjonsdatabaser opererer med atomiske verdier for attributtene i en rad. I Postgres kan imidlertid attributtene inneholde subverdier som kan aksesseres med spørrespråket. Det betyr at man f eks kan definere attributter som er arrays av basistypene.

6.3.4 Utvidbarhet

Postgres lagrer informasjon om databaser, tabeller, og kolonner i systemkataloger i likhet med andre relasjonsdatabaser. Men Postgres lagrer i tillegg opplysninger om definerte typer, funksjoner, aksessmetoder osv. Disse systemkatalogene er tilgjengelige, og kan oppdateres av autoriserte brukere. Ettersom Postgres baserer sine interne operasjoner på informasjonen i disse klassen, betyr dette at Postgres kan utvides av brukerne.

6.3.5 Funksjoner

Funksjoner kan defineres og inkluderes i Postgres-serveren. Funksjonene må defineres i SQL eller i C-kode. Funksjoner skrevet i SQL kan inneholde en vilkårlig samling av SQL-setninger, men må avsluttes med en *select* som returnerer en verdi i samsvar med definisjonen. Funksjoner som skrives i C må kompileres og linkes spesielt for å bli tilgjengelig for dynamisk innlasting i serveren. C-funksjoner kan operere og returnere på typer med fast eller variabel lengde, og kan bruke verdi- eller peker overføring i funksjonskallet (pass-by-value eller pass-by-reference).

6.3.6 Typedefinisjoner

Typer av vilkårlig kompleksitet kan defineres i Postgres. Ved definisjon av typer må det defineres en *input* og en *output* funksjon som skal assosieres med typen. Disse funksjonene definerer hvordan den deklarerte typen ser ut i strenger, slik de opptrer i kommunikasjon med bruker. I tillegg kan man definere egne funksjoner for typen.

For egendeklarerte typer kan det også deklareres operatorer. Disse operatorene kan overlagres, og Postgres vil eventuelt kreve en typespesifikasjon for operandene dersom det ikke kan bestemmes entydig hvilken operator som skal benyttes.

Egendefinderte typer kan også deklareres som sekundærindekser, men det finnes ingen enkel kommandoer som gjør dette. En forholdsvis omfattende programmering må gjøres for å deklarere operatorer, aksessmetoder, kostnadsfunksjoner osv.

6.4 Programmeringsgrensesnitt

6.4.1 LIBPQ

LIBPQ er programmeringsgrensesnittet mot Postgres. Dette er et sett av biblioteksrutiner som gjør at klienter kan sende forespørsler til bakgrunnsserveren og ta imot resultater fra spørringene. Grensesnittet er laget for C-programmering.

Gjennom LIBPQ er funksjoner for å

- etablere og avslutte en forbindelse
- starte og avslutte en transaksjon
- oversende for eksekvering SQL-setninger, og lese resultatene fra disse
- et sett med funksjoner for aksessering av Large Objects

6.4.2 LIBPQ++

Et enkelt grensesnitt for C++ programmering er tilgjengelig i et eget bibliotek. Et sett av klasser er definert, med metoder som ved kall til LIBPQ-funksjoner gi samme funksjonalitet men i en C++ innpakning.

6.5 Postgres Rule System

Postgres' "production rule" system er konseptuelt enkelt. Man deklarerer funksjoner som skal utføres ved spesifiserte aksjoner mot en klasse eller en attributt i en klasse. Aksjon refererer da til *select*, *update*, *delete* eller *insert*.

Et eksempel illustrerer dette:

```
CREATE RULE eksempel
ON UPDATE ARBTAKER.loenn WHERE CURRENT.navn = "Knut
Aage"
DO UPDATE ARBTAKER (loenn = NEW.loenn * 2);
```

Ved oppdatering av Knut Aage's lønn vil denne bli doblet i forhold til hva som er spesifisert i den opprinnelige *update*-kommandoen.

Versjon 1.0.8 som benyttes har imidlertid ikke et stabil rule-system, og i dokumentasjonen advares det sterk mot å benytte dette.

6.6 Oppsummering av Postgres' egenskaper

6.6.1 Objektorienterte egenskaper

Postgres tildeler unike OID'er til alle objekter som opprettes. OID'en kan ikke endres av programmerere og den er uavhengig av objektenes attributter. OID'ene er tilgjengelig på programmeringsnivå og kan dermed benyttes til navigering i databasen.

Typesystemet er utvidbart, og gjør det mulig å beskrive objekter med vilkårlig kompleksitet. Beskrivelsene gjøres dels SQL, og dels i C for å deklarere *in*- og *out*-funksjoner for typenes utseende i strenger. Dette avviker sterkt fra standarden som er definert for objektdatabaser. Typebeskrivelsene lagres i systemkataloger i databasen, og benyttes til typesjekking ved eksekvering av SQL-setninger mot Postgres-server. I programmene vil SQL-setningene bli pakket inn som strenger, og innholdet i disse vil ikke bli typesjekket ved kompilering.

I typesystemet inngår de vanlige atomiske verdier (integer, char etc), utvidet med tekststrenger med variabel størrelse. Ustrukturerte komplekse objekter kan også deklareres (Large Objects). Postgres har ikke predefinerte typer for sett, liste, eller bag, men kan

ha arrays av OID'er som attributt. Det er allikevel mer nærliggende å realisere disse typene i egne klasser (dvs tabeller) og ivareta eventuelle duplikater ved programmering, eventuelt ved rules eller definerte funksjoner. Men Postgres opererer uansett ikke med ordnede klasser, og vil ikke kunne operere med en ordnet liste av OID'er uten duplikater.

Binære relasjoner med inverse referanser mellom objekter realiseres i form av egne klasser. Binære relasjoner i en retning i realiseres ved å definere en OID som attributt. Postgres iveretar ikke oppdatering av inverse referanser, dette må eventuelt ordnes ved bruk av å definere funksjoner eller rules som ivaretar dette.

Objekter deklareres kun med en offentlig (public) del, og støtter dermed ikke innkapsling. Metoder for klasser kan ikke deklareres, men bruk av funksjoner kan til en viss grad erstatte dette. Postgres støtter bruk av overlagrede funksjoner og operatorer.

Definisjonsspråket SQL er deklarativt, mens programmeringsspråkene som benyttes er imperative. Det forblir et skarpt skille mellom instanser av objekter i databasen og instanser av objekter i programmeringsspråket, der verdier må etterspørres og deretter kopieres inn i en objektinstans i programmet.

Arving tillates med enkel arving, mens selektiv arving ikke støttes.

Det finnes ingen mekanismer for håndtering av flere versjoner av objekter, og det blir opp til programmerer å håndtere dette. Konfigurasjoner har dermed heller ikke noen støtte i Postgres.

6.6.2 Generelle databaseegenskaper

Postgres inkluderer de vanligste databaseegenskaper som bestandighet, datadeling og tilfeldig størrelse ved hjelp av logging, transaksjoner og mekanismer for tilbakerulling og gjenervervelse (recovery). Lange transaksjoner støttes ikke, heller ikke nøstede transaksjoner. Domeneangivelser for attributter kan ikke angis.

Adgangskontroll tilbys som i Unix med mulighet til å skille tilgang på bruker, gruppe og alle. Adgang kan skilles på databasenivå. Det er mulig å installere Postgres med et utvidet adgangskontrollsystem.

Spørrespråket for Postgres er et subsett av SQL92, med utvidelser. Det gjør det forholdsvis enkelt å aksessere klasser og objekter.

Typedeklarasjonene gjøres ved definisjoner med SQL-syntaks, sammen med et sett av funksjoner for å operere mot de nye typene (minimum *in*- og *out*-funksjoner). Typedeklarasjonene er deretter tilgjengelig for alle, og "eies" ikke bare av programmet som har deklarert typene.

For databaseadministrasjon finnes et begrenset sett av verktøy inkludert. Ved hjelp av dette biblioteket av funksjoner kan reorganisering utføres for Postgres-definerte typer. Egendefinerte typer må ivaretas ved egenutviklede rutiner. Rapporteringsverktøy er ikke inkludert i Postgres.

Trinnvis utvikling i Postgres blir omtrent som i Shore. Instanser av klasser som får endret sine definisjoner må håndteres spesielt dersom de har egendefinerte typer inkludert

Postgres har ikke mulighet for distribuerte databaser. Applikasjoner kan kjøres på andre maskiner enn Postgres, og vil da kunne etablere forbindelse over TCP ved å bruke det medfølgende libpq-biblioteket.

6.6.3 Vurdering av PostgreSQL

Terskelen for å ta i bruk Postgres er svært lav for programmerere med Unix-kompetanse og med erfaring i bruk av SQL. For å utnytte mulighetene for egendefinerte typer, funksjoner og rules kreves imidlertid god kompetanse i C-programmering.

Postgres har en objektorienterte egenskap som er av vesentlig verdi; arving. De øvrige egenskapene har så stor kompleksitet ved implementasjon at de vanskelig kan utnyttes i et objektorientert programmiljø. Som eksempel vil en applikasjon basert på C++ og med bruk av egendefinerte typer, funksjoner og rules gi programmerer følgende å forholde seg til:

- et sett av C++ programmer som ved hjelp av libpq++ kommuniserer med databaseserveren
- et sett av datadefinisjoner, funksjoner og rules skrevet i SQL
- et sett av funksjoner skrevet i C

Dette synes å gi en mer kompleks og fragmentert systemmiljø enn hva man f eks får i Shore.

Postgres' styrke ligger i at man ved de utvidete funksjonene er i stand til å håndtere mer komplekse datastrukturer enn tradisjonelle relasjonsdatabaser. Videre vil en ved rule-systemet kunne flytte en del av konsistenskontrollen inn i databasen. Bruk av funksjoner vil også kunne utnyttes for å flytte mer komplekse operasjoner fra programmet og til databasen.

Heller ikke Postgres har administrative verktøy tilgjengelig for databaseadministrasjon. Men til opplæringsformål er systemet den godt egnet.

Kapittel 7 Implementasjon av GIS

7.1 Innledning

Hoveddelen av arbeidet med oppgaven har gått med til å designe og implementere en enkel GIS-applikasjon. Applikasjonen bygger på datamodellen utviklet av OpenGIS, og er implementert med to databaseplattformer; Postgres og Shore.

I dette kapitlet beskrives denne applikasjonen, som har fått navnet GO etter initialer i oppgavens tittel: Objektorienterte datamodeller for GeOdata.

I en overordnet beskrivelse av systemet gis en kort omtale av funksjonaliteten i systemet. De verktøy som er valgt og benyttes er omtalt kort, før systemets design beskrives generelt uten tanke på databaseplattform. Designet er så videreført til implementasjonen mot de to databaseplattformene.

7.2 Overordnet systembeskrivelse

GO omfatter to hovedskjermbilder, kalt *GO Hovedpanel* og *GO Kartpanel*. I Hovedpanelet ligger all funksjonalitet i form av nedtrekksmenyer, sprett-opp vinduer, valgknapper, utfyllingsfelter og et tekstområde hvor meldinger fra programmet skrives ut.

Fra hovedpanelet kan man

- Velge databaseplattform; Shore eller Postgres.
- Generere et antall polygoner og lagre disse i den valgte databasen.
- Lese inn en fil i SOSI-format, og lagre objektene fra denne i valgt database.
- Velge og hente objekter i database, ut fra referansesystem, temakode og utsnitt av geografisk område.
- Zoome ut/inn på de tegnede objektene.
- Lagre meldinger fra programmet som en loggfil.
- Slette data i en database.

I kartpanelet blir objekter som hentes fra database tegnet ut. Det er ingen annen funksjonalitet knyttet til kartpanelet.

7.3 Verktøyvalg

7.3.1 Generelt

I og med at databasesystemene Shore og Postgres skulle brukes, var en hel del verktøyvalg implisitt bestemt. Programmeringsspråket som må benyttes er C++, sammen med gcc-kompilator versjon 2.7.2. Systemet må også kjøres på Unix arbeidsstasjon. Disse kjøres med operativsystemet SunOS 4.1.3 og med Motif/X som vindushåndteringssystem.

For brukergrensesnitt var det åpent for flere muligheter, men jeg valgte å bruke wxWindows, et klassebibliotek for C++ som tilbyr GUI (Graphical User Interface). Dette har jeg erfaring med fra tidligere prosjektarbeide.

7.3.2 wxWindows

wxWindows er et klassebibliotek for C++ for grafiske brukergrensesnitt. Klassebiblioteket er utviklet for å kunne benyttes for flere plattformer, som for øyeblikket er Open Look (XView) og Motif på Unix plattform, og MS Windows og Windows NT på PC-plattform. Det inneholder ca 60 klasser, med 650 funksjoner.

wxWindows er utviklet ved Artificial Intelligence Applications Institute, University of Edinburgh, opprinnelig for internt bruk, men er nå allment tilgjengelig[1] i form av kildekode og oppsett for kompilering på ulike plattformer og med ulike kompilatorer. En fyldig dokumentasjon [11], [12]. Dette gjør at terskelen for å ta i bruk klasssebiblioteket er lav.

Den viktigste egenskapen ved biblioteket er at en ut fra samme kildekode kan generere sin applikasjon for såpass ulike plattformer som Motif og Windows NT. GO applikasjonen er imidlertid knyttet til bruk av Shore og Postgres databasesystem, og er derfor ikke portabelt til Windows miljø. Det har dermed ikke vært aktuelt å lage en applikasjon for flere plattformer i dette prosjektet.

7.4 Design

7.4.1 Mål for applikasjonen

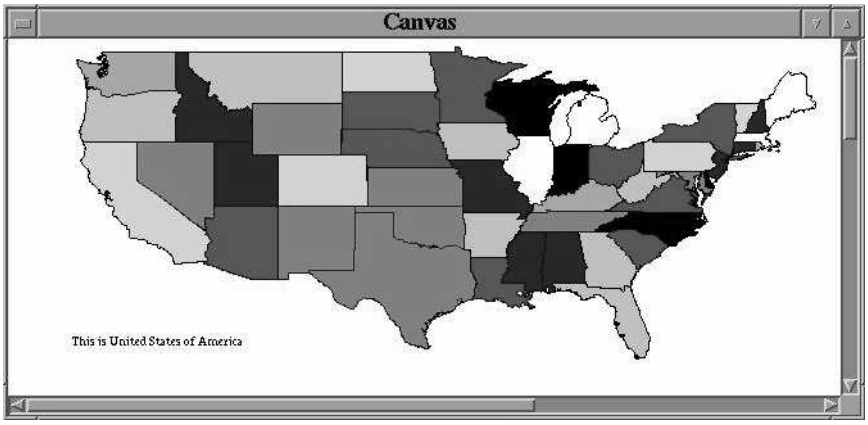
Målsetting ved design og implementasjon har vært følgende:

1. Å modellere en database for geodata, basert på OpenGIS sin “Object Model for Interoperable Geoprocessing”, og implementere denne på ulike databaseplattformer. Plattformene skulle representere relasjonsdatabaser, relasjonsdatabaser med objektorienterte egenskaper og objektdatabaser. Til dette skulle hhv Sybase, Postgres og Shore benyttes. Implementasjon på Sybase-plattformen er ikke gjort på grunn av tidsnød.
2. Utføre målinger av tidsforbruk ved operasjoner mot databasene og kunne logge disse for senere sammenligninger av resultatene.

1. Tilgjengelig via FTP (ftp.aiai.ed.ac.uk) og WWW (URL: http://www.aiai.ed.ac.uk/~jacs/wxwin.html)

3. Å kunne utføre en simulering av innhenting av informasjon over et nettverk, der datastrukturene definert av OpenGIS (WKS'er) benyttes for overføring av informasjon.
4. Å kunne visualisere geodata hentet fra database.
5. Kunne generere testdata som gir like betingelser for målinger mot databasene.
6. Kunne laste inn testdata fra utenforliggende filer.
7. Lage en enkel applikasjon som omfatter all funksjonalitet i hht til målsettingene.

Figur 15. Kart basert på DIMAN-metafil; USA



7.4.2 Valg av kildefiler for testdata

For å kunne teste databasene med reelle data, ble flere kilder vurdert.

DIMAN-metafiler, omtalt i kapittel 2.4.2, var lett tilgjengelige og ble derfor undersøkt først. En parser for dette filformatet ble laget og utprøvet, i figur 15 er et eksempel på uttegning av data fra en fil med dette formatet. Det er imidlertid ikke geografisk stedfestet informasjon i filene, koordinater angis i forhold til origo i et vindu. Disse filene ble derfor vurdert som lite egnet for testformål.

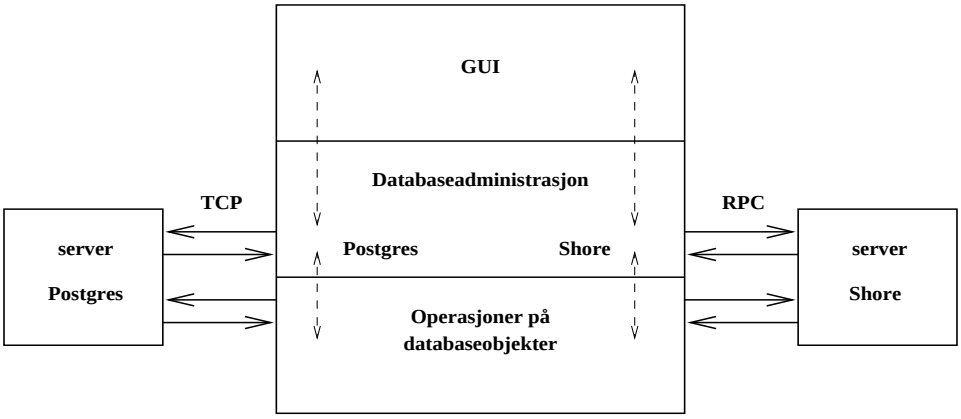
SOSI-formatet (se kapittel 2.4.1) ble deretter vurdert. Statens Kartverks avdeling på Lillehammer ble kontaktet og stilte velvillig opp med dokumentasjon og veiledning. En del testfiler ble skaffet til veie, og disse ble da brukt som grunnlag for implementasjon av en parser for filer i SOSI-format.

7.4.3 Arkitektur

GO har en konvensjonell klient-tjener arkitektur, og opptrer som klient i forhold til databaseserverne. Kommunikasjonen mot Shore går via RPC (Remote Procedure Call), mens kommunikasjon mot Postgres går via TCP. GO må kjøres på samme maskin som databaseserveren for Shore kjører, mens Postgres-serveren om nødvendig kan kjøres på en annen maskin.

Internt er GO splittet i 3 lag. Øverste lag er GUI, og omfatter all funksjonalitet mot bruker. Midterste lag håndterer etablering/aslutning av forbindelse med databaseservere samt start og avslutning av transaksjoner. På nederste lag finnes realisering av metoder for persistente objekter. Dette er illustrert i figur 16.

Figur 16. Arkitektur for applikasjonen GO



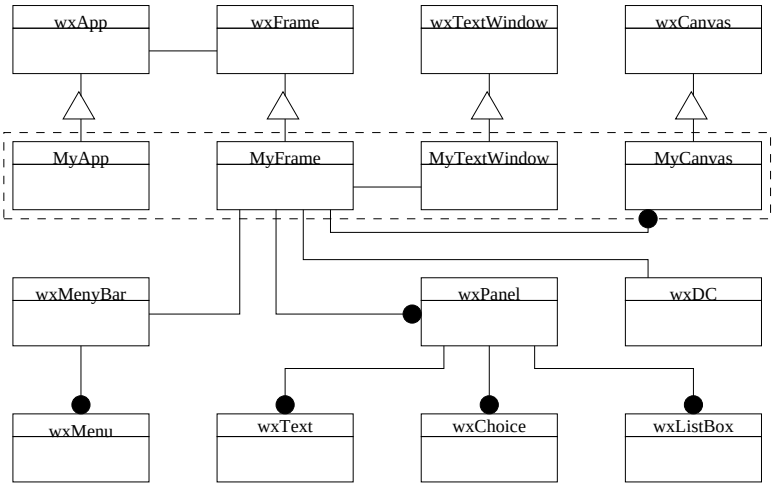
Kommunikasjonen internt i GO går vertikalt gjennom lagene[1]. Datastrukturene definert av OpenGIS - de såkalte Well Known Structures - benyttes gjennom disse lagene, for å kommuniserer geometrien til objektene som behandles.

7.4.4 Brukergrensesnitt

I figur 17 er vist objektdiagram for brukergrensesnittet. Objektene inne i stiplet boks er de som defineres i denne applikasjonen, mens resten av objektene er hentet fra wxWindows klassebiblioteket. Bare hovedelementene er tatt med, flere andre klasser fra wxWindows brukes til ymse formål.

MyApp objektet er eier av hele applikasjonen. Definisjoner av hvilke vinduer som hører til applikasjonen og lay-out for disse, gjøres ved initiering av dette objektet. I GO defineres her “GO Hovedpanel” og “GO Kartpanel”.

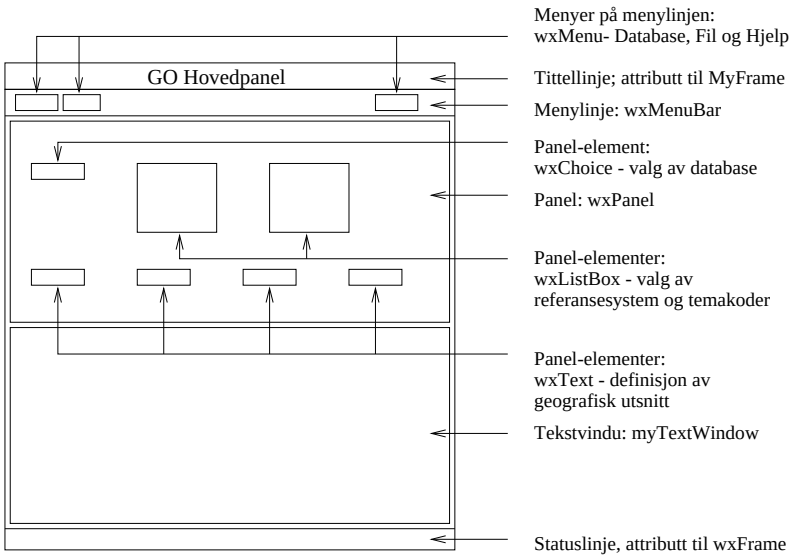
Figur 17. Objektdiagram for GUI-delen av applikasjonen



1. I forsøk på å effektivisere henting/vising av data er det gjort noen unntak i implementasjonen mot Shore-plattformen.

MyFrame objektet har attributter og metoder for vinduet “GO Hovedpanel”. Attributtene omfatter pekere til menylinje, panel, tekstvindu og statuslinje. MyFrame er også *eier* av objektet MyCanvas som representerer “GO Kartpanel”. Sammenhengen mellom objektene er visualisert i figur 18.

Figur 18. Sammenheng mellom objekter og skjermbilde-elementer



MyFrame

All funksjonalitet knyttet til bruk av

- knapp for valg av database som er en instans av wxChoice, og knyttet til objektet wxPanel,
- valg av referansesystemer og temakoder, instanser av wxListBox og knyttet til wxPanel, og
- definisjon av geografisk utsnitt, instanser av wxText knyttet til wxPanel

er globale callback-funksjoner. Ledetekster i panelet er objekter som er attributter til wxPanel.

Funksjonalitet knyttet til menyvalg (en instans av wxMenyBar og instanser wxMenu) er metoder i objektet wxFrame.

Visualisering av geodata

For tegning til MyCanvas brukes et objekt wxDC (Device context), som er en generalisering av utskriftsmedier. Kontekst settes til MyCanvas, og ved hjelp av wxDC-metoder skrives og tegnes så til “GO Kartpanel”. Samme kode kan benyttes til å skrive/ tegne til andre medier, f eks til postscript-skriver, men det gjøres ikke her. Geometri som skal tegnes overleveres i form av WKS’en “Geometry” (se kapittel 7.5.1).

Måling av tidsforbruk og logging

I myFrame benyttes *tidtaker*-objekter for å måle forbrukt tid for alle operasjoner mot databaser. Forbrukt tid skrives ut i tekstpanelet.

Simulering av aksess til fjerne databaser

Simulering gjøres ved oppkall av en metode i databaseadministrasjonslaget for innhenting av en samling geometriobjekter. Denne metoden skal returnere alle objekter knyttet til et referansesystem i form av WKS'en Geometry. Referansesystem gis som parameter til denne metoden. Oppkopling mot database, innhenting av hvilken informasjon som er tilgjengelig og fortolkning av denne, selektering av data og nedkopling av forbindelse inngår ikke i simuleringen.

Selektering av data fra database

Utvalg fra database gjøres ved å velge temakoder. Registrering av slike gjøres i metoder i brukergrensesnittet. Ved henting at data bruker temakode "Alle" for å spesifisere at simuleringsmetoden skal benyttes, mens alle andre temakoder gir innhenting av data uten at simuleringshensyn tas. Innhentet informasjon returneres imidlertid som WKS'en Geometry.

Øvrig funksjonalitet

Funksjonalitet for generering av testdata, innlesing av SOSI-filer, og åpning / lukking av forbindelse mot databaser realiseres ved oppkall av metoder i underliggende lag.

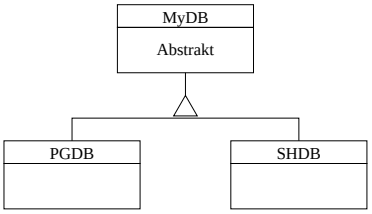
7.4.5 Databaseadministrasjon

Databaseplattformer er representerert av objekter i applikasjonen. Dette er vist i figur 19. Objektet MyDB generaliserer databaseobjekter, og inneholder virtuelle metoder for

- initiering av forbindelse mot databaseserver
- generering av testdata
- hente data i database, simulerer fjernaksess
- slette data,
- innlasting av SOSI-fil
- lukking av forbindelse til databaseserver
- hente data om referansesystemer i databasen til panelet
- hente data om geografisk avgrensning av data knyttet til et referansesystem til panelet
- henting av geodata fra databaser basert på temakoder

Det instansieres aldri noe objekt av typen MyDB (derfor er objektet angitt som "Abstrakt" i figuren).

Figur 19. Objektdiagram for databaseadministrasjon i GO



Objektene PGDB og SHDB realiserer de virtuelle metodene definert for MyDB. I GUI-delen av appliksjonen er databasen det opereres på representert ved en peker til MyDB. Denne settes så til å peke på den subtypen som representerer den valgte databaseplattformen. På denne måten oppnåes at grensesnittet mellom databaselaget og GUI-delen av applikasjonen blir likt, uansett hvilken plattform som velges.

7.4.6 Geodata

Objektdiagrammet for persistente data er modellert med utgangspunkt i geodatamodellen utviklet av OpenGIS, som er omtalt i kapittel 3.3. Den forenklede datamodellen er vist i figur 20. Ett nytt objekt introduseres i forhold til OpenGIS-modellen; “Query Manager”.

Query Manager

Query Manager skal i prinsippet administrere fjerntliggende applikasjoners aksess til databasen, og burde som sådan ha metoder som kan parse en inngående forespørsel, velge ut data fra databasen og returnere disse i form av WKS’er. Både metadata om databasen og geodata er viktige i en slik sammenheng.
Her begrenses dette imidlertid til en metode som henter alle data i databasen knyttet til et oppgitt referansesystem (getGeometry(inout Geometry)).

Feature

Inneholder attributter for navn, beskrivelse, temakode, og metoder for initiering av lagring av attributter og tilhørende geometri. Har inverse relasjoner til Geometric_Property og invers egenrelasjon som attributter.

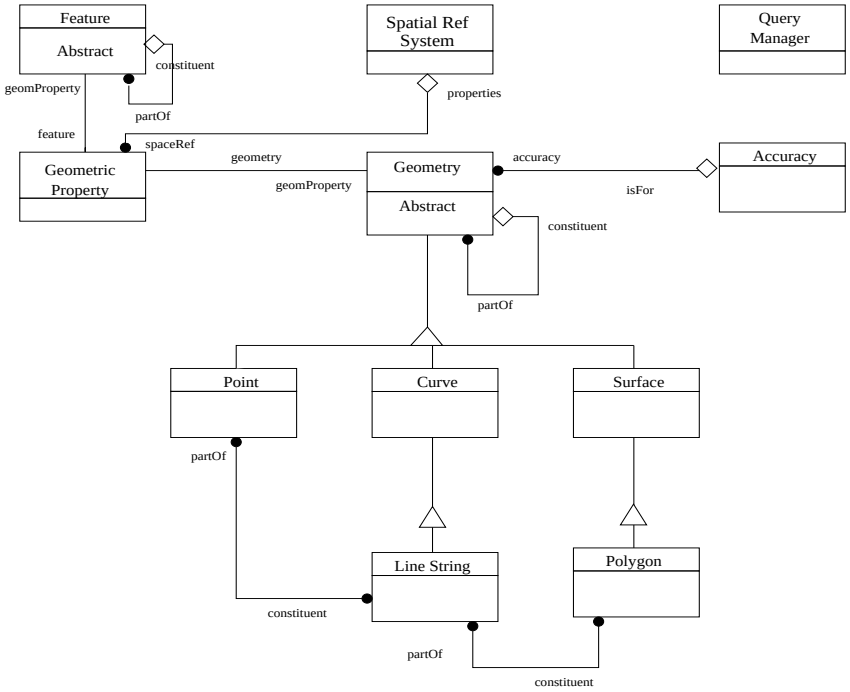
Accuracy

Har attributter for nøyaktighet, målemetode og invers relasjon til Geometry.

Geometric_Property

Har attributter som ivaretar inverse relasjoner til Feature, Spatial_Ref_System. En metode som lagrer geometri (createGeometry(in CoordinateGeometry)).

Figur 20. Objektdiagram for geodata



Spatial_Ref_System

Har attributter for (unikt) navn, lineær måleenhet (meter), vinkelmåleenhet (grader), primærmeridian, ellipsoide, geografisk koordinatsystem (GCS), koordinat-transformasjon (CT), sone, geografisk område avgrenset av soneangivelse; sonens origokoordinater, maks og minverdier for nord/øst-angivelser, projisert koordinatsystem (PC), temporalt system (dag-mnd-år angivelser), romdimensjon (2), tidsdimensjon (1) og total dimensjon. Må ha metoder for lesing/oppdatering av alle attributter. Har også invers relasjon til Geometric_Property som attributt.

Geometry

Har attributter for koordinatdimensjon, boolsk angivelse av Planar/Closed/Simple, dimensjon for geometrien (0=punkt, 1=kurve, 2=overflate, 3=legeme), tolkning (brukes for å angi f eks spline-type ved uttegning av geometrien). Invers egenrelasjon til Geometry, og invers relasjon til Geometric_Property.
Må ha metoder for å returnere hvilket referansesystem geometrien tilhører, lesing/oppdatering av attributter, henting av koordinater for geometrien (getCoordinateGeometry(inout CoordinateGeometry)).

Point

Arver Geometry-objektet. Har attributter for koordinatene til et punkt og boolsk angivelse av om punktet er en del av flere linjesegmenter. Må ha metoder for lagring av punkt gitt ved WKS's strukturen Coordinate, eller gitt ved intern representasjon. Videre en metode for henting av koordinatverdier som returnereres som WKS Coordinate (getCoordinateGeometry(inout CoordinateGeometry)).
Har også invers relasjon til Line_String.

Curve

Arver Geometry-objektet. Brukes for å representere kurver som angis på andre måter enn ved en sekvens av punkter, kan f eks være sirkel, ellipser ol. Blir ikke brukt her.

Surface

Arver Geometry-objektet. Er også "arvet" fra OpenGIS-spesifikasjonen, og har ingen spesiell funksjon her.

Line_String

Arver Curve-objektet. Har invers relasjoner til Point som representerer punktene Line_String består av, invers relasjon til Polygon som har informasjon om hvilke polygoner Line_String er en del av. Metoder omfatter lagring av Line_String og Ring, og henting av koordinatgeometrien til objektet (getCoordinateGeometry(inout CoordinateGeometry)).

Polygon

Arver Surface-objektet. Har invers relasjon til Line_String for å ivareta informasjon om hvilke ringer som danner polygonet. Metoder omfatter lagring av polygon og uthenting av koordinatgeometri for polygonet (getCoordinateGeometry(inout CoordinateGeometry)).

7.5 Implementasjon

7.5.1 Well Known Structures

Datastrukturene som er definert av OpenGIS og kalt WKS (Well Known Structures) benyttes i utstrakt grad i applikasjonen. Strukturbeskrivelsene er omskrevet til Shore SDL-syntaks, og vist i vedlegg A. Strukturene blir alment tilgjengelig i applikasjonen etter at disse er kjørt gjennom Shore's prekompilator.

Kort oppsummert benyttes følgende strukturer:

- Coordinate; er en sekvens av 4 elementer, som beskriver nord- og øst-koordinat, angivelse av høyde over havet og en tidsangivelse for når koordinatene er registrert. De 4 elementene er alle definert av OpenGIS å være *long integer* eller *double integer*. I GO er disse definert som *long integer*.
- LineString; er en sekvens av *Coordinate*'s
- Ring; som også er en sekvens av *Coordinate*'s. En Ring er lukket, i motsetning til en LineString.
- Polygon; er en sekvens av Ringer.
- CoordGeom; koordinatgeometri, som er en union av alle geometritypene som er deklarert. I GO vil dette være en union av Coordinate, LineString, Ring og Polygon.
- CoordGeomCollection; som er en sekvens av CoordGeom, og som sådan representerer en samling av geometriobjekter.

- Geometry; en struktur av CoordGeom og SpatialReferenceName. Geometry-strukturen vil representere en samling av geometriobjekter som alle er knyttet til samme referansesystem.

Prefikset **C_** er gjennomgående brukt for variabelnavn som representerer disse strukturene. Ved deklarasjoner er følgende norm brukt:

```
C_pt - Coordinate
C_ls - LineString
C_ri - Ring
C_pg - Polygon
C_cg - CoordGeom
C_cgc - CoordGeomCollection
C_gm - Geometry
```

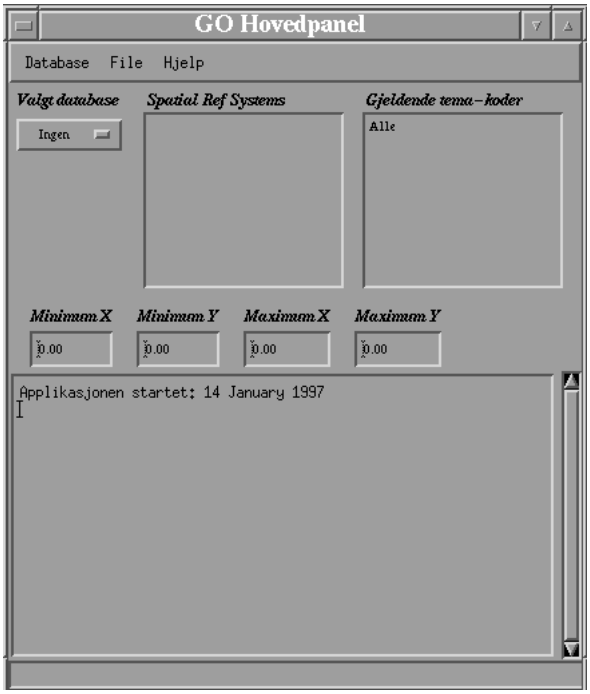
Det ble fort klart at det var nødvendig med en slik navnekonvensjon i programmene, da det ellers ble vanskelig å skille mellom variable som representerer objekter med forsjellig struktur, f eks Polygon representert som WKS og Polygon representert ved et persistent objekt.

Prekompilatoren i Shore definerer et sett med funksjoner som kan benyttes mot sekvenser og unioner, og som gjør det enkelt å ta i bruk slike strukturer. Sekvenser og unioner slik OpenGIS har definert disse er ellers ikke en del av standard typesett for C++.

7.5.2 Brukergrensesnitt

Basert på objektdiagrammet i figur 17, er klassene som tilhører brukergrensesnittet implementert i filene GO.h og GO.C. Disse er gjengitt i hhv vedlegg D og vedlegg E.

Figur 21. GO Hovedpanel ved oppstart

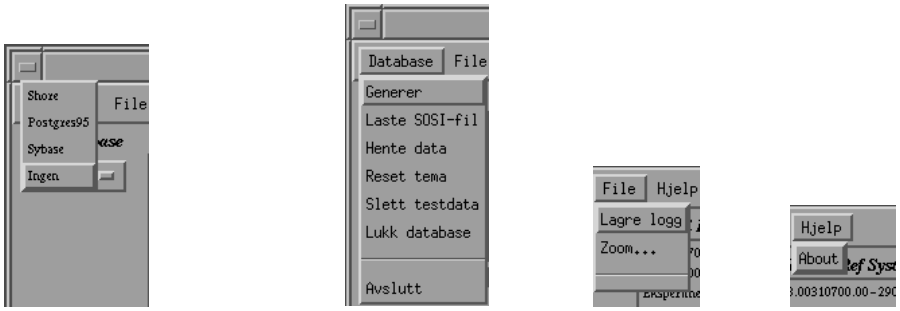


Ved oppstart av applikasjonen blir “GO Hovedpanel” og et tomt “GO Kartpanel” vist på skjermen. I figur 21 er en skjermbildekopi av “GO Hovedpanel” gjengitt. De ulike

menyer og databasevalg er vist i figur 22. Først velges databaseplattform, med knappvalg 22a. Øvrige felt i vunduet blir da fylt ut med data hentet fra databasen som er valgt.

Deretter vil alle valg i nedtrekksmenyene være tilgjengelig; 22b, 22c og 22d.

Figur 22. Menyer og knappevalg i GO Hovedpanel



a) Valg av database-plattform.

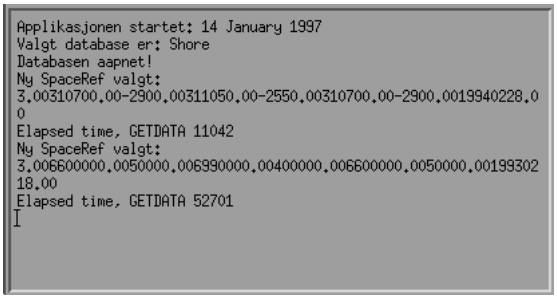
b) Nedtrekksmeny "Database"

c) Nedtrekksmeny "File"

d) "Hjelp" nedtrekksmeny

Meldinger fra applikasjonen skrives ut i tekstvinduet. Eksempel på slike meldinger er vist i figur 23.

Figur 23. Meldinger i tekstvinduet i GO Hovedpanel

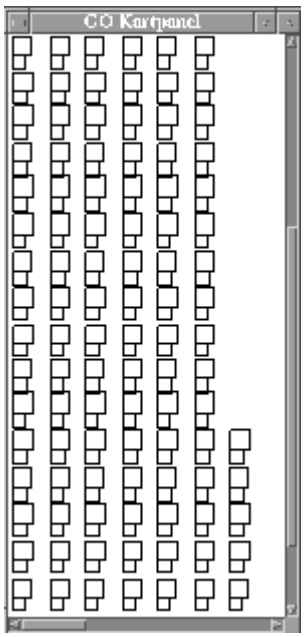


Bruk av opsjonen "Generer" medfører at et antall polygoner blir generert og lagret i databasen. Disse blir knyttet til et fiktivt referansesystem kalt "Experimental". Antallet spesifiseres i et sprett-opp vindu.

Lagrede data hentes fra databasen ved valget "Hente data". Data selekteres da ut fra innholdet i referansesystemboksen, temakodeboksen og min/max verdier for X og Y koordinatene. I figur 24 er vist 101 genererte polygoner, etter henting og fremvising i "GO Kartpanel".

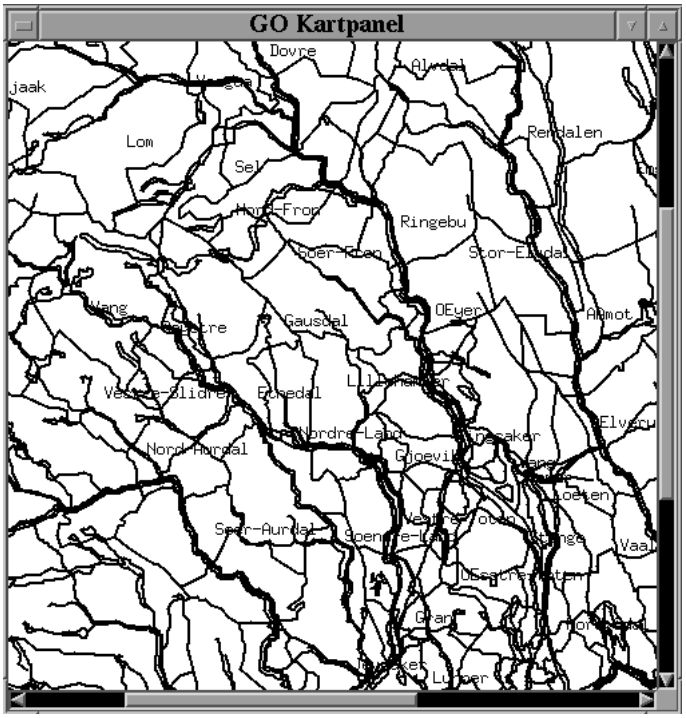
Innlasting av filer i SOSI-format kan gjøres med valget "Laste SOSI-fil". Hvilken fil som skal leses spesifiseres i et sprett-opp vindu. Etter innlesing kan dette vises i kartpanelet slik det er gjort i figur 25.

Figur 24. Genererte polygoner



Det er også mulig å zoome inn eller ut på innholdet i kartpanelet, ved valg “Zoom” i File-menyen. Zoom-faktor spesifiseres i et sprett-opp vindu.

Figur 25. Utsnitt av Hedmark og Oppland



WKS’er brukes internt i metodene helt til geometrien skal tegnes. Ved uttegning blir disse parset, og kopiert inn i lister av punkter. Punktene er instanser av wxPoint-klassen. I et wxPoint-objekt er et punkt definert av sin x-og y-koordinat gitt som float. Ett eksempel:

Et polygon er representert som en sekvens av ringer, som igjen er en sekvens av

punkter.

Ved uttegning (som skjer i metoden MyFrame::DrawGeometry) transformeres hver ring til en liste av punkter. Punktverdiene i strukturer av typen Coordinate kopieres til wxPoint x/y-koordinater og må samtidig endres fra type *long* til *float*.

7.5.3 Databaseadministrasjon

Objektdiagrammet i figur 19 er realisert i filene

- MyDB.h (vedlegg F) og MyDB.C (vedlegg G)
- SHDB. h (vedlegg H) og SHDB.C (vedlegg I)
- PGDB.h (vedlegg J) og PGDB.C (vedlegg K)

Det er bare implementasjonene i SHDB.C og i PGDB.C som blir brukt av applikasjonen. Disse realiserer databasespesifikke funksjoner for etablering av forbindelse til databasene (InitDB()) og lukking av forbindelse (CloseDB(). For øvrige operasjoner mot databasene påbegynnes en transaksjonsblokk i metodene her, før metoder mot persistente objekter eksekveres. Ved avslutning utføres commit av transaksjonen.

I Shore er bruk av lange transaksjoner støttet, slik at det ikke byr på noe problem å sette start transaksjon f eks ved generering av et ukjent antall polygoner. Mot Postgres-serveren fører imidlertid dette til svært store loggfiler, og lang responstid for commit av en transaksjon. For Postgres-implementasjonen er derfor start-transaksjon/commit flyttet inn i metoden som genererer polygonene, slik at en transaksjon omfatter ett polygon. For et driftssystem måtte en vurdere om en slik flytting av transaksjonsgrenser var akseptabelt i tilfelle en unormal stans under kjøring. I GO spiller dette imidlertid ingen rolle.

Forøvrig var design-tanken at lagdelingen som er vist i figur 16 skulle gjennomføres strikt, og at metodene her kun skulle brukes for å sette start/slutt av transaksjonsblokker. Det viste seg etterhvert å være svært lite praktisk, slik at flere av metodene her også inneholder kall til objektmetoder, kopiering av verdier og logikk. Av tidsmessige årsaker er ikke alle metodene implementert for Postgres.

7.5.4 Geodata i Shore-database

Persistente objekter er deklarert i henhold til objektdiagrammet i figur 20, med noen små endringer. Deklarasjone er skrevet i SDL, og vist i vedlegg B. Alle objektnavn er prefixet med **SH_** i deklarasjonene. Endringene består i følgende:

Relasjonen mellom Geometry og Accuracy er tegnet som en invers binær relasjon. Denne er omgjort til enveis relasjon, realisert ved at referanse til Accuracy er en attributt i Geometry-objektet.

I Shore må programmerer definere et objekt som eier av de indekser man ønsker å benytte. Dette objektet, kalt SH_Indices kommer derfor i tillegg til objektene i diagrammet. Indekser er realisert i Shore som B-tre, unik eller ikke-unik. Det er her lagt indeks på Punkt (unik), referansesystemnavn (unik), serienummer (unik) - en indeks som benyttes i forbindelse med innlesing av SOSI-filer, temaindeks (ikke-unik) og geo-

metritype (ikke-unik). R-tre indekser er ikke tilgjengelig i denne versjon av Shore, men en slik indeks kunne ellers ha vært aktuell å bruke for Punkt.

De metoder som er omtalt i kapittel 7.4.6 er implementert, i tillegg kommer metoder nødvendig for indeksobjektet; innlegging, spørring mot og fjerning av indekser. Implementasjonen er gjengitt i vedlegg L, vedlegg M og vedlegg N.

WKS'er benyttes også her som parametre til ulike metoder. Ved lagring blir en geometri mottat, parset og lagret som hhv Polygon, LineString av type Ring eller LineString, og Point.

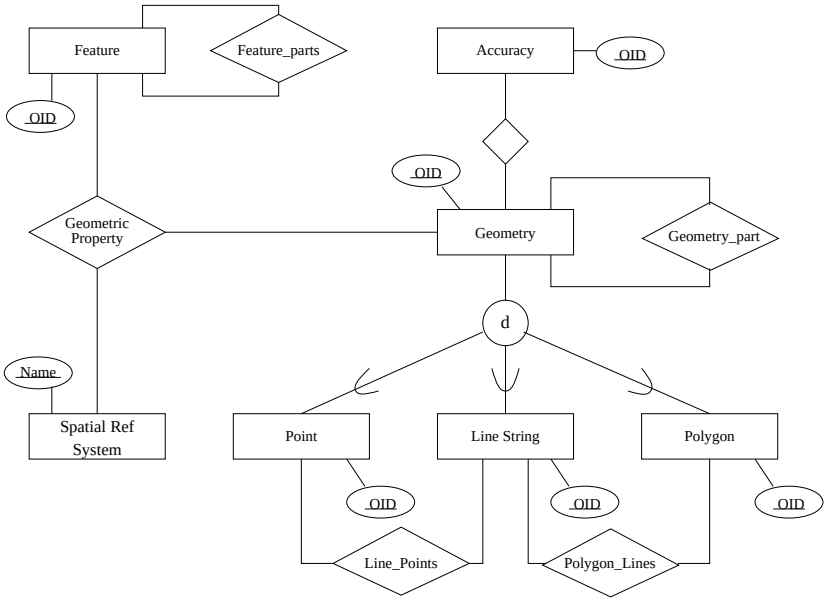
Ved deklarering av pekere til persistente objekter er følgende normer brukt;

```
aFeature - peker til SH_Feature
aSpaceRef - peker til SH_Spatial_Ref_System
aGeomP - peker til SH_Geometric_Property
aGeometry - peker til SH_Geometry
anAccuracy - peker til SH_Accuracy
aPoint - peker til SH_Point
aLineString - peker til SH_Line_String
aPolygon - peker til SH_Polygon
```

7.5.5 Geodata i Postgres-database

For å realisere en database i Postgres er objektdiagrammet i figur 20 transformert til ER-diagrammet vist i figur 26. Som det vil fremgå er det gjort en del forenklinger. Objektene *"Curve"* og *"Surface"* er tatt bort. *Geometry* er spesialisert i klassene *Point*, *LineString* og *Polygon*.

Figur 26. ER-diagram for geodata i Postgres-database



I figuren er nøkkelattributter tatt med. Det er bare objektet *Spatial Ref System* som har en attributt som nøkkel. Postgres tildeler imidlertid alle tupler en unik OID, og denne brukes om identifikator for alle øvrige klasser.

Tabellene som er definert ut fra figur 26 er vist i vedlegg C. En treveis relasjon “Geometric Property” realiseres som en tabell hvor OID fra de tre klassene er kolonner. Klassene Point, LineString og Polygon arver alle attributter fra Geometry. Relasjonen mellom Accuracy og Geometry realiseres ved at OID for Accuracy er en attributt i Geometry. Relasjonene Line_Points og Polygong_Lines er begge realisert som tabeller med OID fra de respektive klassene som attributter.

En funksjon er definert i SQL; denne brukes for å oppdatere i tabellen Geometric Property.

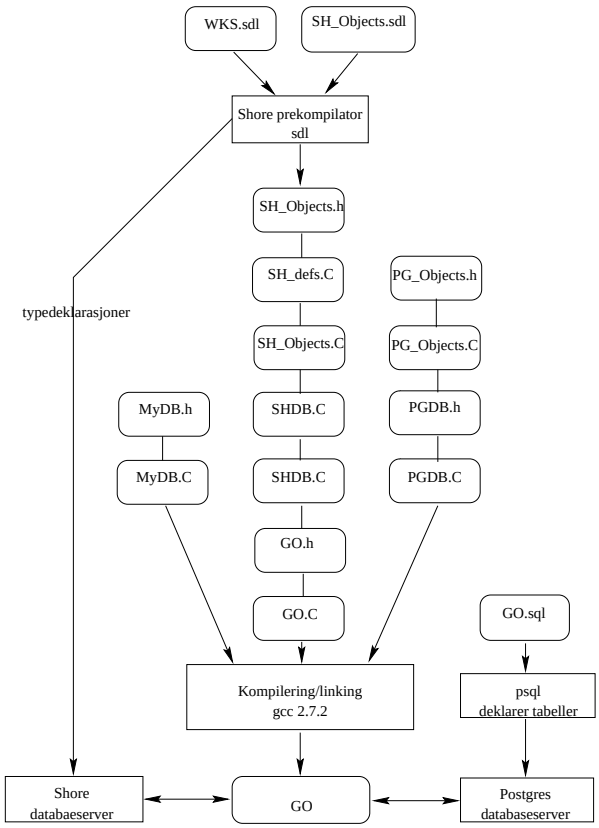
Et ikke-persistent objekt PG_Query_Manager brukes for å ivareta tilsvarende funksjonalitet som SH_Query_Manager ved simulering av fjern-aksess til databasen. Oppdatering mot database og henting av verdier ivaretas forøvrig av funksjonene definert i vedlegg O, og implementert som vist i vedlegg P.

Det er deklarert tre sekundærindekser i et forsøk på å forbedre svarstidene ved henting av data fra databasen. Også her er det brukt B-tre indekser, selv om Postgres har støtte for R-tre indekser.

7.5.6 Moduloversikt

En samlet oversikt over filene i applikasjonen, og hvilke prosesser som benyttes for å få en kjørbar applikasjon er vist i figur 27.

Figur 27. Oversikt over filer i applikasjonen



7.6 Erfaringer

7.6.1 Brukergensesnittet

Bruk av wxWindows klassebibliotek har vært uten nevneverdige problemer. Underveis i arbeidet med applikasjonen har jeg valgt å fjerne en del av klassene fra det oppkompilerte biblioteket, for å få ned størrelsen på den eksekverbare modulen. Ved ulike "triks" ved kompilering av dette biblioteket ble størrelsen på GO redusert fra 11.5 MB til ca 6 MB. Av dette svarer wxWindows for ca 2 MB, Shore for ca 3 MB og GO applikasjonen og Postgres for ca 0.5 MB hver.

Bruk av ulike typer for tegning av punkt (float) og lagring (long integer) og den typekonverteringen som dermed må gjøres for alle punkter som skal tegnes, påvirker nok responstidene en del. En gjennomgående endring i hele applikasjonen for å endre koordinater til type float har det imidlertid ikke blitt tid til, slik at det måleresultater som påviser eventuelle endringer i responstider ikke er tilgjengelig.

String-håndteringen i klassebiblioteket inneholder noen feil, som medfører "Segmentation fault" av og til. Noen slike feil gjenstår, mens andre har blitt rettet opp underveis. Generelt er inntrykket av kvaliteten og funksjonaliteten som finnes i wxWindows bra.

7.6.2 Bruk av OpenGIS datamodell

Datamodellen ser ut til å være godt gjennomarbeidet og vil kunne brukes som utgangspunkt for konstruksjon av databaser for mangeartede GIS-applikasjoner. I et objektorientert miljø vil de definerte klassene lett kunne utvides med arvinger av egendefinerte klasser.

Definisjonen av Polygon som en sekvens av Ringer, dvs lukkede arealer, avviker fra SOSI-standardens datamodell, der Polygoner defineres som en sekvens av linjesegmenter. Dette medførte at innlesing av SOSI-filer måtte implementeres med andre metoder enn de som var naturlig ved å benytte de definerte WKS'ene.

7.6.3 Shore

Implementasjon av datamodellen og de metodene som ble skissert i designfasen forløp uten alt for store problemer. Bruk av SDL til å definere objekter og deres grensesnitt med påfølgende prekompilering innfører dog et ekstra nivå i utviklingsprosessen. Bruk av prekompilatoren betydde lite for tidforbruk ved oppkompilering av nye versjoner. Kompilering av SH_defs.C som genererer kode for metodene definert i sdl-modulen var imidlertid svært tidkrevende, vanligvis rundt 10 min.

I GUI-delen av applikasjonen er det lagt inn en del tidmålere slik at forbrukt klokketid ved kall til databaselagene i applikasjonen kan måles. I tabell 1 er resultater ved operasjoner mot Shore gjengitt.

Tabell 1. Responstider målt mot Shore-databasen

Polygoner	Create (ms)	Get (Warm) (ms)	Get (Cold) (ms)	Delete (ms)
0	-	343, 366, 353, 346	430, 392	3179
10	7050, 6197, 7101, 6646, 6680, 9745	145, 146*2, 147, 148*2, 150*4, 151	438, 506	5682, 5418, 7645, 5693, 5565
Snitt (10)	7237	148	472	6001
100	47009, 46920, 45966, 46210, 40681	1090, 1123, 1141, 1143, 1144, 1149, 1150, 1152*2, 1153*2, 1156, 1157, 1170	1419, 1456, 1678, 1742, 1849, 2090, 1804, 1834	29565, 35285, 36083, 30322
Snitt (100)	45357	1145	1734	32814
200	84142	2825, 2844, 2851, 2824, 2887, 2942, 2902, 3014, 3099	3583, 5149	63562, 64572
Snitt (200)	84142	2910	4366	64067

Tallene er antall millisekunder som brukes på

- start transaksjon
- operasjoner mot persistente objekter, generering av WKS'er
- commit transaksjon
- uttegning av objektene ved traversering av WKS'ene

Det sier seg selv at med et så begrenset antall målinger vil disse bare antyde den reelle responstiden. Skillet mellom henting av data når disse allerede finnes i bufre (warm) i shared memory og når disse må hentes inn i forbindelse lesing av data er allikevel markant.

Målingene vil ikke avdekke hvor tidforbruket er stort eller lite. Min antakelse er at storparten av tiden går med til å pakke objektene og relasjonene inn i WKS'er.

En måling, der resultatene er gjengitt i tabell 2, bekrefter dette. Ved henting av data med metodekallet getData("Alle") vil alle geometrier som finnes i database knyttet til et gitt referansesystem bli hentet ved navigering gjennom relasjonene i databasen. Geometriobjektene blir hele tiden lagt til WKS'en Geometry, som tilslutt vil inneholde en sekvens av alle geometrier. Denne returneres GUI-delen for uttegning. Målingene er som i forrige tabell, inkludert uttegning av disse geometriene.

Tabell 2. Responstid for ulike aksessmetoder i Shore

getData("alle")		getData(alle temakoder)	
cold	**warm**	**cold**	**warm**
149513	134061, 130788	49175, 47709, 48026, 48242	57201, 59697

Ved henting av data med metodekallet getData(alle temakoder), vil programmet ved bruk av indekser lete opp alle Featureobjekter for hver av temakodene, hente geome-

trien for hvert featureobjekt ved traversering, lage en Geometry-struktur for denne, og deretter tegne denne ut, før neste featureobjekt hentes og behandles. Ved å oppgi samtlige temakoder som finnes i databasen, vil alle geometriobjekter bli hentet og tegnet. Tabellen viser tidforbruket ved denne formen for henting av data, og viser vesentlig kortere responstid. Målingene i tabell 2 er utført mot et vesentlig større antall objekter enn i tabell 1. Utvalget her betår av 1343 Feature-objekter, 116 Polygon-objekter, 1174 LineString-objekter og 22165 Point-objekter, og representerer et utsnitt av Hedmark og oppland. Dette er uttegnet i figur 25.

Bruk av metodekallet getData(“alle”) er primært en simulering av aksess til en fjerntliggende database, og bruk av WKS’en Geometry kreves som standard for utveksling av informasjon ved slik aksess. For en slik aksess ville i virkeligheten også transporttiden for overføring av data komme i tillegg. Behandlingstiden internt i databasen vil da ofte være av mindre betydning. I en applikasjon som brukes interaktivt, og hele tiden har behov for henting/skriving av data vil naturligvis slike svarstider være uakseptabelt. I slik interaksjon bør man derfor ikke bruke WKS’er.

Min lagdeling og bruk av WKS’er forhindrer direkte utnyttelse av persistente objekter i f eks GO.hovedprogram, og dermed den mest effektive måten bruke objektdatabasen på. Ved programmering i C++ burde det i prinsippet ikke være noen forskjell i måten persistente objekter og temporære objekter brukes. I praksis vil det allikevel være en vesentlig ulikhet:

- All bruk av metoder for persistente objekter må pakkes inn i en transaksjon. Transaksjoner kan være lange og omfatte bruk av mange metoder, men dette vil uansett skille persistente objekter fra temporære objekter.

Dette medfører en del merarbeide for å teste på utfallet av start/slutt for transaksjoner. En gjennomgripende endring i applikasjonen for å bruke objektdatabasen mer effektivt har det ikke vært tid til.

7.6.4 Postgres

Ved oppstart av implementering av datamodellen på Postgresdatabasen var et utgangspunkt å bruke funksjoner i databasen i størst mulig utstrekning. Flere av metodene knyttet til de ulike klassene ble forsøkt implementert som SQL-funksjoner. Testing av disse funksjonene viste imidlertid at dette ikke var gjennomførbart. Forsøk på å definert WKS’ene som datatyper i Postgres er kanskje mulig, men det har ikke vært forsøkt. Slik det nå er implementert er Postgres brukt nesten som en vanlig relasjonsdatabase. Arving er brukt, og i tillegg er *en* funksjon definert og tatt i bruk.

For å få punkter i en linje i rett sekvens benyttes et sekvensnummer som et attributt i Line_Points. Denne brukes da som sorteringskriterium ved *select* mot tabellen. Tilsvarende gjøres i tabellen Polygon_Lines, for å få linjesegmenter i rett rekkefølge for et polygon. Tilsvarende målinger som for Shore-plattformen er utført for oppretting, henting og sletting av plolygoner. Postgresserver og applikasjon kjøres på samme maskin, for å gjøre resultatene mest mulig sammenlignbare. Resultater fra målinger er vist i tabell 3.

Tabell 3. Responstider målt mot Postgres-databasen

Polygoner	Create	Get (Warm)	Get (Cold)	Delete
10 a)	75609	119074	2427805	
10 b)	70111	11597, 11211	12921, 12384, 12614, 12114	12114
Snitt (10b)	70111	11401	12508	12114
100	641536, 666455, 677791	229336, 211456, 210840	332014, 315549, 285883, 287455, 287794, 286377	
100 c)		144614, 144360, 144596	153827,	
Snitt(100) c)	661927	217211 144523	299179 153827	

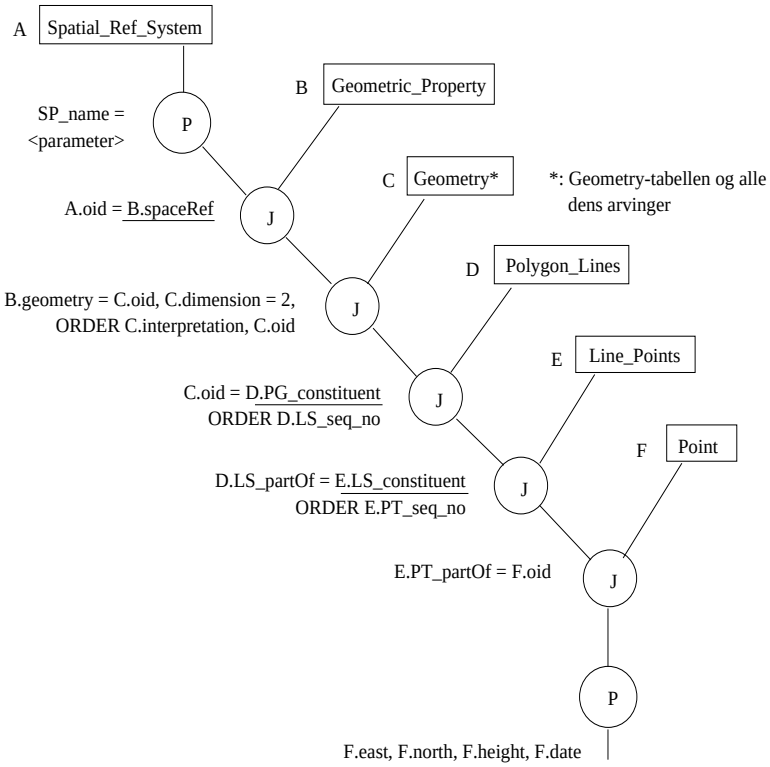
Resultater merket a), er fra første versjon av hente-rutinen. I denne ble hentingen pakket inn i en lang transaksjon. Det enorme tidforbruket viste seg å være knyttet til Postgres sin oppdatering av logger ved commit av transaksjonen. Programmet navigerte i dette tilfellet i tabellene ved hjelp av OID'er, omtrent som ved navigering i en objektdatabase.

Hente-rutinen ble så skrevet om til å hente alle data med en *cursor*-definisjon inneholdende en *select* setning (b).Samtidig ble funksjonaliteten splittet opp flere korte transaksjoner. En skisse av hvilke *join*- og *projeksjoner* som blir gjort i *select*-setningen er vist i figur 28.

Responstiden ble forbedret, men fremdeles svært lang. Utførelse av *select* står for nær halvparten av tidforbruket,mens opprydding i loggfiler ved commit står for resten. Tidforbruk til generering av WKS'er og uttegning tar en forsvinnende liten del av tiden.

I et forsøk på å redusere responstiden ble feltene som er understreket i figur 28 indeksert. Resultatet ble en halvering av responstiden (c), men fremdeles er denne ca 100 ganger større enn for tilsvarende hent-operasjon mot Shore.

Figur 28. Graf for *select* alle geometrier knyttet til ett referansesystem



Selv om Postgres har noen objektorienterte egenskaper falt mye av objektorienteringen bort i denne delen av implementasjonen. Forsøk på å beholde de samme klasser og metoder som var definert i Shore-implementasjonen falt heller ikke heldig ut, og resultatet ble funksjoner som opererte på tabellene i Postgres.

På grunn av de lange responstidene ble ikke innlasting av SOSI-filer implementert for Postgres-plattformen.

7.7 Konklusjoner

Arbeidet med GO-applikasjonen har vært svært lærerikt. Ved å implementere datamodellen på to så ulike plattformer har ulikhetene i alle deler av arbeidet med å få en operativ database klart blitt demonstrert; definisjon av innhold, operasjoner mot databasen, databaseadministrasjon i fbm viderutvikling av datamodellen osv.

Når arbeidet ellers foregår i et objektorient miljø vil bruk av en relasjonsdatabase medføre et stort konseptuelt gap mellom tabeller i database og objekter slik disse betraktes i programmiljøet. Dette synes også å være tilfellet med relasjonsdatabaser med objektorienterte egenskaper.

Komplekse objekter, med relasjoner mot et ordnet sett av andre komplekse objekter er vanskelig å realisere i tabeller. Realisering på tradisjonelt vis med tabeller som ivaretar relasjonene gir i såfall problemer med effektiv og rask henting og lagring av data. For komplekse objekter og programmering i et objektorientert språk/miljø vil jeg derfor anbefale bruk av objektdatabaser ut fra de erfaringer som er gjort i dette prosjektet.

Referanser og støttelitteratur

[1] Trondheim Kommune og Statens Kartverk: “Trondheimskartet”, 1991 utgave.

[2] R. Conradi, Ed: “Dr.ing course 45942 Object-oriented Systems”, Course Compendium, spring 1996.

[3] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik: “The Object-Oriented Database System Manifesto”, DOOD’89, Dec’89, Kyoto

[4] M. Stonebraker et al.: “Third-Generation Data Base System Manifesto”, ACM SIGMOD Record 19, 3,September 1990.

[5] R. G. G. Catell, Ed: “The Object Database Standard: ODMG-93”, Morgan Kaufmann Publishers, San Mateo, California, 1993.

[6] M. Coyle, S. Shekhar. D.R.Liu, S. Sarkar: Experiences with Object Data Models in Geographic Information Systems”, Submitted to the Communications of the ACM (Special Issue on Object Oriented Experiences), 1995, (TR95-10)

[7] S. B. Zdonik, D. Maier, Eds: Readings in Object-Oriented Database Systems, page 1 through 32; “Fundamentals of Object-Oriented Databases”, The Morgan Kaufmann Series in Data Management Systems Morgan Kaufman Publishers Inc., San Mateo, DCA, 1990.
ISBN 1-55860-000-0, ISSN 1046-1698

[8] R. G. G. Cattell: “Object Data Management: object-oriented and extended relational database systems”, Addison-Wesley Publishing Company, Inc. 1994 ISBN 0-201-54748-1

[9] The Shore Project Group: “An overview of Shore”. Technical report. Computer Sciences Department, University of Wisconsin - Madison. September 15. 1995

[10] The Shore Project Group: “Shore Data Language Reference Manual”, Technical report. Computer Sciences Department, University of Wisconsin - Madison. September 15. 1995

[11] J. Smart: “User manual for wxWindows: a portable C++ GUI toolkot”, Artificial Intelligence Applications Institute, University of Edinburg, EH1 1HN, August 1995.

[12] J. Smart. “Reference Manual for wxWindows: a portable C++ GUI toolkot”, Artificial Intelligence Applications Institute, University of Edinburgh, EH1 1HN, July 1995

[13] P.A. Aamot: “OBSERVE - OBjektorientert database i ShorE foR VidEostar”, Hovedoppgave. Universitetet i Trondheim, Norges Tekniske høgskole, Fakultet for elektro- og datateknikk, Institutt for Datateknikk og Telematikk. 21. desember 1995.

[14] R Grønmo og Ø. Riise: “Automatisk oppdatering av digitale sjøkart”, Hovedoppgave i databehandling, Institutt for informatikk, Universitetet i Oslo. November 1995.

[15] R. Elmashri, S.B Navathe: “Fundamentals of Database Systems”, Second edition, The Benjamin/Cummings Publishing Company, Inc. 1994.
ISBN 0-8053-1753-8

[16] K. Aa. Aksnes: “Object-Oriented Database. Some experiences in GIS applications”. Artikkel innlevert i faget Objektorienterte Systemer, våren 1996.

[17] Draft Base Document - OGIS Prosject Document 94-025R1. The Open Geodata Interoperability Specification, October 1994.

[18] R. Laurini, D. Thompson: Fundamentals of spatial information systems.Academic Press Limited. Fourth printing 1995.
ISBN 0-12-438380-7

[19] J. Star, J. Esties: Geografic Information Systems, An introduction. Prentice-Hall inc. 1990
ISBN 0-13-351123-5

[20] Statens Kartverk: Samordnet Opplegg for Stedfestet informasjon. Versjon 2.2, Juni 1995.

[21] Open GIS Consortium, Inc: The OpenGIS™ Guide, Introduction to Interoperable Geoprocessing. OGIS TC Document 96-001.
(Tilgjenglig fra http://www.opengis.org)

[22] Open GIS Consortium, Inc: The OpenGIS Abstract Specification: An Object Model for Interoperable Geoprocessing, Rev 1. OpenGIS Project Document Number 95-015R1.
(Tilgjenglig fra http://www.opengis.org)

[23] Televerkets Forskningsinstitutt Rapport TF R 47/92; DIMAN dialogmekanisme og kommunikasjonsprotokoller, av Håkon Solbakken
ISBN 82-423-0236-7

[24] Cook, Steve and Daniels, John. Designing Object Systems: Object-Oriented Modelling with Syntropy. Prentice Hall, London 1994.

[25] Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; and Lorenson, William: Object Oriented Modelling and Design. Prentice Hall, Englewood Cliffs, NJ, 1991.

[26] Bratsberg, Svein Erik: Evolution and Integration of Classes in Object-Oriented Databases. Doktor ingeiøravhandling 1993:55, Norges Tekniske Høyskole,Institutt for datateknikk og telematikk, Trondheim. IDT-rapport 1993:2

[27] University of California at Berkely: The POSTGRES95 User Manual, Version 1.0 (September 5, 1995)
(Tilgjengelig fra http://logical.thought.net/postgres95/)

[28] K Aa Aksnes: Objektorienterte dqatabaser. Rapport fra databehandlingsprosjekt, våren 1996, Institutt for datateknikk og telematikk, Norges teknisk-naturvitenskapelige universitet.

Vedlegg A “Well Known Structures”; WKS.sdl

```
// ************************************************************
// File name: WKS.sdl
// Project: GO
// Copyright: IDT, NTNU
// Language used: SDL (Shore Data Language)
// Short description: OpenGIS Well-Known Structures definition
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 21.11.96 Created
// ************************************************************
// For the first version of the specification, the following
// simplifying assumptions vill be made for geometry, and in
// particular for the WKS for geometry:
// All supported geometry will be linear. That is to say, all
// interpolation between data points in WKS geometry will be linearly
// interpolated.
// All coordinates tuples are homogenous in type. Further, these types
// will be restricted to either 32 bit integer (long integers) or 64
// bit IEEE floating point (double precision).
// All of the WKS described below are assumed to be within the scope
// of the WKS module {...}

module WKS
{
  export all;

  // COORDINATES
  // The use of 4 as a maximum dimension will allow the WKS to be
  // expanded to temporal at a later date. The intentions is to use
  // the last dimension to represent time if needed. Space is at most
  // 3 dimensional
  const long MaxDim=4;
  typedef sequence<long,MaxDim> LongBSeq;
  typedef sequence<double,MaxDim> DoubleBSeq;
  // The B is for bounded.

  enum CoordinateTypeTag { TypeDouble, TypeLong };
  // Following union declaration won’t compile with the
  // struct Envelope definition (later)
  //  union Coordinate
  //    switch (CoordinateTypeTag CoordinateType)
  //    {
  //      caseTypeDouble:

  //        DoubleBSeq d_coord;
  //      caseTypeLong:
  //        LongBSeq l_coord;
  //    };
  typedef LongBSeq Coordinate; // Done to get around the sdl-compiler.

  // LINE STRING
  //Lines are 2 point line strings and do not require a separate WKS.
```

```
typedef sequence<Coordinate> LineString;
// Assertions:
// Each line must have at least 2 distinct vertexes, each
// represented by a coordinate in the seqquence.
// Degenerated one point lines should use point structures.
// The coordinate types and ranks are homogeneous (for this and all
// other WKS) - all the bounded sequences within a geometric
// structure are the same length and the same type.
// Sequential vertexes should be distinct. Repeated (sequential)
// vertexes are ignored and discouraged with extreme prejudice.
// A line string is linearly interpolated between sequential
// vertices.
// A line string (with 4 or more vertices) may be self-intersecting.


// RING
typedef sequence<Coordinate>Ring;
// There is concern over whether one should have a ring structure
// (or type) distinct from a line string structure. This should be
// addressed during implementation.
// Assertions:
// Rings are closed. Creators of the WKS must not repeat the first
// vertex (at the end) since the definition of ring forces closure.
// Rings are simple - are not self intersecting.
// Sequential vertexes are distinct. Repeated (sequential) vertexes
// are ignored and discouraged with extreme prejudice.
// Each Ring must have at least 3 distinct vertexes.
// Rings need not be planar - no assumption about bounding surfaces
// are made. Surfaces are defined by polygons or by faceted
// surfaces.
// Rings are not knotted. This means that they can be spanned by a
// simple faceted surface that is topologically equivalent to a
// planar disc (the interior of a circle).

// POLYGONES
typedef sequence<Ring> Polygon;
// Assertions:
// 3D polygons are planar. Note that we need an accuracy tolerance
// in order to determine whether a set of points is planar.
// Each ring in a polygon is simple (non-self intersecting). Rings
// in a single polygon may intersect one another.
// Polygons represent planar (2D) surfaces. A point is interior
// (contained in) to the polygon if:
// 1) it is coplanar with the boundary rings and
// 2) within that plane, it is interior to an odd number of
// bounding rings (see the Jordan curve theorem).  A point on any of
// the bounding rings is also contained in the polygon if it is
// adjacent to the polygons interior (polygons are closed as
// topological entities). Note that this definition permits
// disconnected polygons (islands) and holes, and islands within
// holes nested to any level.
// These assertions are to arrive at a generally accepted canonical
// form for polygons. Any set of closed, coplanar line strings can
// be used to delineate a set of polygons. The process consists of
// several steps. Each closed line string can be decomposed into
// simple line strings by breaking the sting at the point of
// intersection and knotting it back into two separate line strings
// which are now tangent to one another at the old intersection
// point. A similar operation can be performed to eliminate
// non-tangent line intersections. Rings that share full line
// segments can be merged into a single ring by elimination of the
// common segment.  Orientation can be accomplished after the
// interior of the polygon had been determined.
```

```
// TRIANGLES
const long TriangleDim=3;
typedef sequence<Coordinate,TriangleDim> Triangle;
// Assertions:
// Triangles have 3 distinct points. These points are not
// collinear. Note: the points of a triangle uniquely define a plane
// (in any dimensional space)
// Triangles represent planar (2D) surfaces. A point is interior to
// the triangle if 1) it is coplanar with the vertices and 2) within
// that plane, it is interior to the triangle. A point on any of the
// bounding lines is also on the surface
// There is the same debate over distinguishing a triangle from a
// polygon as there is over distinguishing a line string and a
// ring. An additional argument in favor of distinguishing triangles
// is that arrays are more efficient than sequences.

// POLYHEDRAL SURFACE
typedef sequence<Polygon> PolyhedralSurface;
// Assertions:
// Polyhedral surfaces are the union of the surfaces defioned by the
// polygons. Each polygonal face must be connected (although it may
// have holes)
// Surfaces are connected. This means that any two poins on the
// surface can be chained together by a sequence of polygons in the
// surface that each share a common edge with the next polygon in
// the sequence.

// Surfaces selfintesext only at th boundary edges of polygons.

// POLYHEDRON
typedef sequence<Polygon> Polyhedron;
// Assertions:
// All assertions for polyhedral surfaces apply to polyhedra.
// Polyhedra are simple closed surfaces. They are topologically
// equivalent to spheres, tori, or multiple-handled tori. They ar
// the analogy to rings (which are topologically equivalent to
// circles). Surface rings need not have a ‘no-knot, no-twist’
// condition since simple polyhedra cannot be knotted in less than 4
// dimensions (Kline bottle).
// This implies that polyhedron have an inside and an outside.

// POLYHEDRAL SOLID
typedef sequence<Polyhedron> PolyhedralSolid;
// Assertions:
// A point is interior to the solid if it’s interior to an odd
// number of bounding rings (see the Jordan Curve theorem for 3
// dimensions). A point on any of the bounding rings is also on the
// solid if it is adjacent to the solid’s interior (solids are
// closed as topological entities).

// ENVELOPE
struct Envelope
{
  Coordinate min;
  Coordinate max;
};
// Envelopes are defined as the minimum and maximum coordinates for
// enclosed coordinate geometries. The minimum coordinate is defined
// as the tuple of mimimum coordinate values in each direction. The
// maximum coordinate is defined as the tuple of maximum coordinate
// values in each dimension.

// COORDINATE GEOMETRY ENUMERATION, TYPE AND COLLECTION
```

```
     enum  CoordGeomTypeTag  {  CoordCoordinateType,  CoordLineStringType,
CoordRingType,
  CoordPolygonType, CoordTriangleType,
  CoordPolyhedralSurfaceType, CoordPolyhedronType,
  CoordPolyhedralSolidType, CoordEnvelopeType,
  CoordGeomCollectionType };
  union CoordGeom
    switch (CoordGeomTypeTag CoordGeomType)
    {
      case CoordCoordinateType:
Coordinate point;
      case CoordLineStringType:
LineString line_string;
      case CoordRingType:
Ring ring;
      case CoordPolygonType:
Polygon polygon;
      case CoordTriangleType:
Triangle triangle;
      case CoordPolyhedralSurfaceType:
PolyhedralSurface surface;
      case CoordPolyhedronType:
Polyhedron polyhedron;
      case CoordPolyhedralSolidType:
PolyhedralSolid solid;
      case CoordEnvelopeType:
Envelope envelope;
      case CoordGeomCollectionType:
CoordGeomCollection cgc;
    };
  typedef sequence<CoordGeom> CoordGeomCollection;
  // The enumeration of the coordinate geometry types is used to
  // create a union type, and the collection type for coordinate based
  // geometry WKS.

  // SPATIAL REFERENCESYSTEM NAME
  typedef string SpatialReferenceName;

  // GEOMETRY WKS
  struct Geometry
  {
    CoordGeom coordinate_geometry;
    SpatialReferenceName reference_system_name;
  };
  // The geometry structure associates a collection of coordinate
  // geometry with a specific coordinate system (spatial reference
  // system). Similar externalization of the reference system sould be
  // uses on feature collections WKS

  // COORDINATE VECTORS
  typedef Coordinate Vector;
  // These vector definitons will be used to determine grid structures
  // that are regular in each dimension. We will assume that the
  // standard vector geometry holds for accosiated types.

  // GRID GEOMETRY
  struct GridGeometry
  {
    Coordinate Origin;
    sequence <Vector, MaxDim> Offset;
    sequence <long, Maxdim> size;
  };
  enum SeqForm {BandSequential, BandInterleaf, Morton };
  // Assertions
```

```
// The number of offset vectors is no more than the dimension of the
// origin coordinate.
// The dimension of the offset vectors is the same as the dimension
// of the origin coordinate.
// The number of offset vectors is the same as the number of size
// integers.
// One way to correspont the grid structure to the values would be
// to give them corresponding to the offsets, which is band
// sequential form. Some usages prefer to alternate row
// directions. Other prefer a quadtree like bit interleave of Morton
// code order. We could fix on band sequential which is tha most
// natural, but just in case, we can allows other fomrs byt tagging
// them.

// OPENGIS VALUE
typedef long OpenGISValue; // necessery to define this on my own....

// GRID COVERAGES
struct GridCoverage
{
  GridGeometry gridGeometry;
  SeqForm StorageFormat;
  string Interpretation;
  sequence<OpenGISValue> value;
};
// Since cell coverages and post values use the same essential
// structure, ther is no reason to separate them.  It is allowed for
// an interpoltion and interpretation algorithm string to act as a
// differentiator between methods to distribute values across the
// geometry. It would be reasonable to change this to an enumerated
// list before the final specification. Note that since the value is
// left as a general root, when stored functions WKS are defined, we
// can replace the value by a patch function that maps points within
// the cell to different values. This perfectly describes the
// standard patch mechanism used in surface spline functions or
// compression algorithms.

// POINT SET COVERAGES
struct PointCoverage
{
  sequence<Coordinate> point;
  sequence<OpenGISValue> value;
};
// A coverage is defined by simply listing points in paralell to
// values. Alternately, this parallell listing could be done as a
// sequence of point value tuples, but this would confuse the
// geometry to feature/coverage parallel.

// LINE STRING COVERAGE
struct LineStringCoverage
{
  LineString point;
  sequence<OpenGISValue> value;
  string Interpretation;
};
// As a line string is a sequence of points, the structure above can
// be utilized to create coverages whose domain is a single line
// string, given an opportunity to define the interpretation of that
// coverage on the linear links between the line strings vertexes.

// TIN COVERAGE
struct TINCoverage
{
  PointCoverage samples;
```

```
  sequence<Triangle> TIN;
  string Interpretation;
};
// Given a point coverage, we can define a coverage for a TIN from
// the points and values at the triangle vertexes. Again, the number
// of interpretation schemes is almost uncountable. The assumption
// here is that the corners of the triangles in the TIN are the
// points in the point cluster geometry in the point coverage.

// Assertions:
// The TIN is a polyhedral surface.
// The point set consist of the corners of the triangles in the TIN.

// POLYGON COVERAGES
struct PolygonCoverage
{
  sequence<Polygon> polygons;
  sequence<OpenGISValue> value;
  string Interpretation;
};
// Assertions:
// The constituents polygons are mutually exlusive.
// The set of polygons are planar.

// POLYHEDRAL SURFACE COVERAGES
struct PolyhedralSurfaceCoverage
{
  sequence<PolyhedralSurface> surface;
  sequence<OpenGISValue> value;
  string Interpretation;
};
// Assertions:
// The constituents Polyhedral Surface are mutually exlusive.

// POLYHEDRAL SOLID COVERAGE
struct PolyhedralSolidCoverage
{
  sequence<PolyhedralSolid> solid;
  sequence<OpenGISValue> value;
  string Interpretation;
};
//Assertions:
// The constituent solids are mutually exlusive.

// IDENTITY : assumed to be an object-identifier
typedef ref<any> Identity; // Type LOID: Logical object identifier

// TOPOLOGICAL SURFACES
typedef Identity NodeId;
typedef Identity EdgeId;
typedef Identity FaceId;
struct Node
{
  Identity NodeId;
  Coordinate position;
};
struct IsoNode
{
  Identity NodeId;
  Coordinate position;
  FaceId ContainingFace;
};
struct Edge
{
```

```
    Identity EdgeId;
    NodeId StartNode;
    NodeId EndNode;
    FaceId LeftFace;
    FaceId RightFace;
    EdgeId NextEdgeLeft;
    boolean NextEdgeLeftOrientation;
    EdgeId NextEdgeRight;
    boolean NextEdgeRightOrientation;
    LineString PositiveEdge;
  };
  struct PlanarFace
  {
    Identity FaceId;
  };
  struct SurfaceFace
  {
    Identity FaceId;
    PolyhedralSurface position;
  };
  struct PlanarTopology
  {
    sequence<Node> NodeList;
    sequence<Edge> EdgeList;
    sequence<PlanarFace> FaceList;
  };
  struct SurfaceTopology
  {
    sequence<Node> NodeList;
    sequence<Edge> EdgeList;
    sequence<SurfaceFace> FaceList;
  };
  enum Topology2DTypeTag { PlanarTopologyType, SurfaceTopologyType };
  union Topology2D
    switch (Topology2DTypeTag Topology2DType)
    {
      case PlanarTopologyType:
PlanarTopology planar;
      case SurfaceTopologyType:
SurfaceTopology surface;
    };
  // The standard DIGEST (MiniTopo or Winged Edge) topological model is used
  // here. Since the 3D structures are not restricted to 2 1/2D, there may
  // be folds in the topological 3D surfaces. We normally retain a planar
  // consistent topology (you can fold, but you can’t cut and paste). The
  // structure her is valid for any surface representable using faceted
  // patches.
  // The final structures (topological surface) are assumed to be complete
  // in that the intratopological pointers are confined to a single
  // surface. The mechanism for identity is not defined, but we can assume
  // that they are the integer offsets in the sequences by default, until we
  // clarify how identity is to be handled generally.
  // Assertion:
  // The topological relations are consistent with the geometry. No edge
  // crosses another. They meet only at common nodes. Surfaces do not
  // intersect except at common edges on theri boundary. They meet only at
  // common edges. Isolated nodes are on and interior to their indicated
  // faces. This assertion applies to the structures within a single 2D
  // topology.
  // The next edge left is the next edge in the positive
  // (counterclockwise) direction for the face on the left of the current
  // edge. The next edge left orientation is TRUE (positive) if its
  // internal line string orientation is the same as that for the edge and
  // FALSE (negative) otherwise. The next edge right is the next edge in
```

```
// the positive direction for the face on the right of the current
// edge. The next edge right orientation is TRUE (positive) if its
// internal line string orientation is the same as that for the face and
// FALSE otherwise.

// TOPOLOGICAL SURFACE COVERAGES
struct TopologySurfaceCoverage
{
  Topology2D Topology;
  sequence<OpenGISValue> NodeValues;
  sequence<OpenGISValue> EdgeValues;
  sequence<OpenGISValue> FaceValues;
  string Interpretation;
};
// Again the paralell listings is used to define values for each of
// the pimitives. Most current applications vould NULL all but the
// face values, giving us an exlusive polygon coverage.

//TOPOLOGICAL SOLIDS
typedef Node SolidNode;
struct SolidEdge
{
  Identity EdgeId;
  NodeId StartNode;
  NodeId EndNode;
  sequence<FaceId> Faces;
  sequence<boolean> EdgeToFaceOrientation;
  sequence<EdgeId> NextEdges;
  sequence<boolean> NextEdgeOrientations;
  LineString PositiveEdge;
};
// The structures for suffices are paralell in 3D for solids. The
// biggest note here is the usual left and right face for the edge
// become a set of surfaces (sometimes referred to as pencil).
// Assertions:
// The orientation of the interior surfaces is consistent with the edge
// to face orientations above.
struct SolidFace
{
  Identity FaceId;
  PolyhedralSurface position;
  SolidNode TopSolid;
  SolidNode BottomSolid;
};
struct TopoSolid
{
  Identity SolidId;
};
struct SolidTopology
{
  sequence<SolidNode> NodeList;
  sequence<SolidEdge> EdgeList;
  sequence<SolidFace> FaceList;
  sequence<TopoSolid> SolidList;
};

// TOPOLOGICAL SOLID COVERAGES
struct TopologySolidCoverage
{
  SolidTopology Topology;
  sequence<OpenGISValue> NodeValues;
  sequence<OpenGISValue> EdgeValues;
  sequence<OpenGISValue> FaceValues;
  sequence<OpenGISValue> SolidValues;
```

```
    string Interpretation;
  };
  // General 3D covarges are handles in the same manner as used above for
  // other coverages.

  // COVERAGE WELL KNOWN STRUCTURES
  enum CoverageTypeTag { GridType, PointType, TINType, PolygonType,
 PolyhedralSurfaceType, PolyhedralSolidType,
 TopologySurfaceType, TopologySolidType,
 CollectionType};

  union CoordinateCoverage
    switch (CoverageTypeTag CoverageType)
    {
      case GridType:
GridCoverage grid;
      case PointType:
PointCoverage point;
      case TINType:
TINCoverage tin;
      case PolygonType:
PolygonCoverage polygon;
      case PolyhedralSurfaceType:
PolyhedralSurfaceCoverage surface;
      case PolyhedralSolidType:
PolyhedralSolidCoverage solid;
      case TopologySurfaceType:
TopologySurfaceCoverage topoSurface;
      case TopologySolidType:
TopologySolidCoverage topoSolid;
      case CollectionType:
CollectionCoverage collection;
    };
  typedef sequence<CoordinateCoverage> CollectionCoverage;
  // Previosly defined mechanisms for representing coverages for a
  // variety of different geometry types are aggregated here, and the
  // concept of coverage is extended to include mixed geometric types.
  struct Coverage
  {
    CoordinateCoverage coverage;
    SpatialReferenceName reference_system_name;
  };

  // This defines coverages based upon the coordinate systems. To
  // extend this to coverages that will understand the spatial
  // reference system semantics, the coordinate base coverage must be
  // unambigously associated to a specific spatial reference system.

} // End of module WKS definition
```

Vedlegg B SDL Datadefinisjoner; SH_Objects.sdl

```
// ****************************************************************
// File name: SH_Objects.sdl
// Project: GO
// Copyright: IDT, NTNU
// Language used: SDL (Shore Data Language)
// Short description: Datatype definitions for all objects
// to be stored in the database
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 02.11.96 Created
// ****************************************************************
module SH_Objects {
  import “WKS”; // Importing OpenGIS Well-Known Structures
  struct Distance {
    long meters;
    long time;
  };

  struct A_Point {
    long east;
    long north;
    long height;
    long date;
  };

  enum InterpretationType { GO_Arc,
    GO_Ellipse,
    GO_Icon,
    GO_Line,
    GO_Lines,
    GO_Polygon,
    GO_Point,
    GO_Rectangle,
    GO_RoundedRectangle,
    GO_Spline,
    GO_Text };

  typedef sequence<ref<any> > Refs_any; // Might be useful
  //  forward declaration
  interface SH_Query_Manager;
  interface SH_Feature;
  interface SH_Accuracy;
  interface SH_Geometric_Property;
  interface SH_Spatial_Ref_System;
  interface SH_Geometry;
  interface SH_Point;
  interface SH_Curve;
  interface SH_Surface;
  interface SH_Line_String;
  interface SH_Polygon;
  // ---------------------------------------------------------------
  // Interface name: SH_Query_Manager
```

```
  // Description: link between GO-application and Shore db
  //
  // ------------------------------------------------------------
  interface SH_Query_Manager {
  private:
    attribute ref<SH_Indices> inx;
    void createRing(in lref<Coordinate> inC_pt_zero,
     inout Ring inC_ri_theRing,
     in long inEfz, in long inNfz,
    in long inHigh, in long inBroad);
  public:
    void initialize(); // Constructor
    void finalize(in lref<char> inSpaceRef);   // "ALL" or a SpaceRef
    long loadTestFig(in long inNo); // Creates inNo double-polygons
    ref<SH_Indices> getIndex() const;// Return the inx attribute
    void getGeometry(inout Geometry inC_gm) const; // First version;
  };
  // ------------------------------------------------------------
  // Interface name: SH_Feature
  // Description:
  //
  // ------------------------------------------------------------
  interface SH_Feature {
  public:
    attribute string Name;
    attribute string Description;
    attribute long Type;
    attribute string someStuff;
    void initialize(in lref<char> inName, in lref<char> inDescription,
    in long inType, in lref<char> inSomeStuff);
    void initWKS(in lref<char> inName, in lref<char> inDescription,
in long inType, in lref<char> inSomeStuff,
in lref<Geometry> inC_gm, in ref<SH_Accuracy> inAccuracyRef);
    void finalize() const;
    long getType() const;
    relationship ref<SH_Geometric_Property> geomProperty inverse feature;
    relationship ref<SH_Feature> partOf inverse constituent;
    relationship set<SH_Feature> constituent inverse partOf;
  };
  // ------------------------------------------------------------
  // Interface name: SH_Accuracy
  // Description:
  //
  // ------------------------------------------------------------
  interface SH_Accuracy {
  private:
    attribute long Accuracy;
    attribute long MeasureMethod;
  public:
    void initialize(in long inAccuracy,in long inMeasureMethod);
    long getAccuracy() const;
  };
  // ------------------------------------------------------------
  // Interface name: SH_Geometric_Property
  // Description:
  //
  // ------------------------------------------------------------
  interface SH_Geometric_Property {
  private:
    //    attributes
  public:
    void finalize() const; // Destructor
    // Creates the correct Geometry:
    void createGeometry(in lref<CoordGeom> inC_cg,
```

```
in ref<SH_Accuracy> inAccuracyRef);
    relationship ref<SH_Feature> feature inverse geomProperty;
    relationship ref<SH_Spatial_Ref_System> spaceRef inverse properties;
    relationship ref<SH_Geometry> geometry inverse geomProperty;
  };
  // ---------------------------------------------------------------
  // Interface name: SH_Spatial_Ref_System
  // Description: Finds locations in real world, given geometry
  //                  (place/time)
  //                        A very limited version is implemented
  // ---------------------------------------------------------------
  interface SH_Spatial_Ref_System {
  private:
    attribute string Name;           // A unique name
    attribute float  Linear_unit;    // meter*unit
    attribute string Angular_unit;   // Degree
    attribute string Prime_meridian; // Greenwich
    attribute string Ellipsoide;     // GauzzKrugers
    attribute string GCS;            // Geographic Coordinate System: WGS84
    attribute string CT;             // Coordinate Transformation
    attribute long   Zone;           //
    attribute long   OrigoN;         // Coordinates of origo
    attribute long   OrigoOE;
    attribute long   MaxN;           // Max-values of coordinates
    attribute long   MaxOE;
    attribute long   MinN;           // Min-values of coordinates
    attribute long   MinOE;
     attribute string PC; // Projected Coordinate System: WGS/Europeen datum
ED50
    attribute string Temporal_system;// Using a discrete temporal system
    attribute long   SpaceDimension; // Topological dimensions of the spatial
    attribute long   TimeDimension;  // This is 1
    attribute long   TotalDimension; // The sum of space- and timeDimension

  public:
    void initialize(in lref<char> inName);     // Set attributes
    void finalize() const; // Destructor
    long spaceDimension() const; // Returns the spaceDimension value
    long timeDimension() const; // Returns the timeDimension value
    long totalDimension() const; // Returns the totalDimension value
    void setLinear_unit(in float inLinear_unit);
    void setZone(in long inZone);
    void setOrigo(in long inOrigoN, in long inOrigoOE);
    void setMax(in long inMaxN, in long inMaxOE);
    void setMin(in long inMinN, in long inMinOE);
    float getLinear_unit() const;
    long getZone() const;
    void getOrigo(inout long inOrigoN, inout long inOrigoOE) const;
    void getMax(inout long inMaxN, inout long inMaxOE) const;
    void getMin(inout long inMinN, inout long inMinOE) const;
    lref<char> metaData(in lref<char> name) const; // Returns the given meta-
data
    relationship set<SH_Geometric_Property> properties inverse spaceRef;
    // Some other possibilities
    // Geodesic distance between two points:
    // Distance distance(in ref<SH_Point> inaPoint1,
    //          in ref<SH_Point> inaPoint2) const;
    // Generate a point at given distance in given direction from given point:
    // ref<SH_Point> pointAtDistance(in ref<SH_Point> inaPoint,
    //                                in long inDirection,
    //   in Distance inDistance);
    // Generate geodesic curve, given distance, direction and a point:
    // void geodesic(Point, Direction, Distance)
    // Generates shortest geodesic curve from point 1 to point 2
```

```
    // void geodesic(in ref<SH_Point> point1,
    //        in ref<SH_Point> point2) const;
  };
  // ------------------------------------------------------------
  // Interface name: SH_Geometry
  // Description:
  //
  // ------------------------------------------------------------
  interface SH_Geometry {
  protected:
    attribute long CoordDimension; // used 4; North, East, Height and Date
    attribute boolean Planar, Closed, Simple;
    attribute ref<SH_Accuracy> accuracyRef;
    attribute long Dimension; // The actual dimension of current geometry
    // meaning 0 : Point
    //         1 : Curve
    //         2 : Surface
    //         3 : Solid
    attribute InterpretationType Interpretation;
    // Use Interpretation when splines, ellipses(circles),
    // rounded rectangeles etc are to be drawn from set of points.
  public:
    ref<SH_Spatial_Ref_System> referenceSystem() const;
    void getCoordinateGeometry(inout CoordGeom inC_cg) const;
    ref<SH_Accuracy> getAccuracyRef() const;
    void setAccuracyRef(in ref<SH_Accuracy> inAccuracyRef);
    relationship ref<SH_Geometry> partOf inverse constituent;
    relationship set<SH_Geometry> constituent inverse partOf;
    relationship ref<SH_Geometric_Property> geomProperty inverse geometry;
    //Other possibilities considered:
    // lref<Geometry> Export() const; // Returns geometry structure and refsys
    // lref<Envelope> minBoundBox() const; // Returns minimum bounding box
    // boolean equal(in lref<Geometry> inC_gm) const; // Determines
    //  whether or not the passed geometry is spatially equal to the geometry
    //  No tolerance considerations are taken here!
    // Distance distance(in lref<Geometry> inC_gm) const; // Returns
    //  spatial distance between two objects. Defined to be the shortest dis-
tance
    //  between any two points chosen from the two objects, within same Space-
RefSys
    // lref<Geometry> bufferZone(in Distance inDistance); // Returns a geome-
try
     //  that represents all points whose distance is less or equal to the
given
    //  distance, within same SpaceRefSys
    // lref<Geometry> boundary(); // Will return the closure of the
    //  combinatorial boundary.
    // boolean planar() const; // Returns TRUE if the current object is planar
    //  (always true, as I'll only use 2D
    // boolean closed() const; // Returns TRUE if the current object is closed
    //  i.e. curve w/identical endpoints, surface that bounds a solid...
    // boolean simple() const; // TRUE if current objekt has no anomalous geo-
metric
    //  points, such as self intersection or self tangency.
    // boolean connected() const; // TRUE if current object is connected. That
is if
     //  any two points in the set can be joined by a curve of points in the
set.
    // boolean contains(in lref<Geometry> inC_gm) const; // TRUE if
    //  passed geometry is spatially within the current object
    // boolean isIn(in lref<Geometry> inC_gm) const; // TRUE if current
    //  object is spatially within passed geometry.
  };
  // ------------------------------------------------------------
```

```
  // Interface name: SH_Point
  // Description:
  //
  // ------------------------------------------------------------
  interface SH_Point : public SH_Geometry {
  private:
    attribute A_Point Point; // The actual point; sequence<long,MaxDim>
  public:
     attribute boolean KP; // True if this point is common to several line
strings
    void initialize(in lref<A_Point> inPt,
    in ref<SH_Accuracy> inAccuracyRef);
    void initWKS(in lref<Coordinate> inPt,
    in ref<SH_Accuracy> inAccuracyRef);
    void finalize(); // Destructor
    void getCoordinateGeometry(inout CoordGeom inC_cg) const;
    relationship set<SH_Line_String> partOf inverse constituent;
    // Other possibilities considered:
    // lref<Geometry> Export() const; // Returns geometry structure and refsys
    // lref<Envelope> minBoundBox() const; // Returns minimum bounding box
    // boolean equal(in lref<Geometry> inC_gm) const; // Determines
    //  whether or not the passed geometry is spatially equal to the geometry
    //  No tolerance considerations are taken here!
    // Distance distance(in lref<Geometry> inC_gm) const; // Returns
    //  spatial distance between two objects. Defined to be the shortest dis-
tance
    //  between any two points chosen from the two objects, within same Space-
RefSys
    // lref<Geometry> bufferZone(in Distance inDistance); // Returns a geome-
try
     //  that represents all points whose distance is less or equal to the
given
    //   distance, within same SpaceRefSys
    // lref<Geometry> boundary(); // Will return the closure of the
    //  combinatorial boundary.
    // boolean closed() const; // Returns TRUE if the current object is closed
    //  i.e. curve w/identical endpoints, surface that bounds a solid...
    // boolean simple() const; // TRUE if current objekt has no anomalous geo-
metric
    //  points, such as self intersection or self tangency.
    // boolean connected() const; // TRUE if current object is connected. That
is if
     //  any two points in the set can be joined by a curve of points in the
set.
    // boolean contains(in lref<Geometry> inC_gm) const; // TRUE if
    //  passed geometry is spatially within the current object
    // boolean isIn(in lref<Geometry> inC_gm) const; // TRUE if current
    //  object is spatially within passed geometry.
  };
  // ------------------------------------------------------------
  // Interface name: SH_Curve
  // Description:
  //(Abstract at the moment)
  // ------------------------------------------------------------
  interface SH_Curve : public SH_Geometry {
  private:
  public:
    //    void initialize(); // Constructor
    //    void finalize();   // Destructor
    //    Distance length(in lref<Coordinate> point1,
    //     in lref<Coordinate> point2) const;
    //    length() returns the distance between two points in the curve domain
    //    void getCoordinateGeometry(inout CoordGeom inC_cg) const;
    //    // the actual geometry used.
```

```
     //     lref<Geometry> Export() const; // Returns geometry structure and
refsys
    //    lref<Envelope> minBoundBox() const; // Returns minimum bounding box
    //    boolean equal(in lref<Geometry> inC_gm) const; // Determines
    //    // whether or not the passed geometry is spatially equal to the geo-
metry
    //    // No tolerance considerations are taken here!
    //    Distance distance(in lref<Geometry> inC_gm) const; // Returns
    //    // spatial distance between two objects. Defined to be the shortest
distance
    //     // between any two points chosen from the two objects, within same
SpaceRefSys
    //    lref<Geometry> bufferZone(in Distance inDistance); // Returns a geo-
metry
    //    // that represents all points whose distance is less or equal to the
given
    //    // distance, within same SpaceRefSys
    //    lref<Geometry> boundary(); // Will return the closure of the
    //    // combinatorial boundary.
    //    boolean closed() const; // Returns TRUE if the current object is clo-
sed
    //    // i.e. curve w/identical endpoints, surface that bounds a solid...
    //     boolean simple() const; // TRUE if current objekt has no anomalous
geometric
    //    // points, such as self intersection or self tangency.
    //     boolean connected() const; // TRUE if current object is connected.
That is if
    //    // any two points in the set can be joined by a curve of points in
the set.
    //    boolean contains(in lref<Geometry> geometry) const; // TRUE if
    //    // passed geometry is spatially within the current object
    //    boolean isIn(in lref<Geometry> geometry) const; // TRUE if current
    //    // object is spatially within passed geometry.
  };
  // ----------------------------------------------------------------
  // Interface name: SH_Surface
  // Description:
  //(Abstract at the moment)
  // ----------------------------------------------------------------
  interface SH_Surface : public SH_Geometry {
  private:
    //    attributes
  public:
    //    void initialize(); // Constructor, dimension=2
    //    void finalize(); // Destructor
    //    void getCoordinateGeometry(inout CoordGeom inC_cg) const;
    //    // the actual geometry used.
     //     lref<Geometry> Export() const; // Returns geometry structure and
refsys
    //    lref<Envelope> minBoundBox() const; // Returns minimum bounding box
    //    boolean equal(in lref<Geometry> inC_gm) const; // Determines
    //    // whether or not the passed geometry is spatially equal to the geo-
metry
    //    // No tolerance considerations are taken here!
    //    Distance distance(in lref<Geometry> inC_gm) const; // Returns
    //    // spatial distance between two objects. Defined to be the shortest
distance
    //     // between any two points chosen from the two objects, within same
SpaceRefSys
    //    lref<Geometry> bufferZone(in Distance inDistance); // Returns a geo-
metry
    //    // that represents all points whose distance is less or equal to the
given
    //    // distance, within same SpaceRefSys
```

```
    //    lref<Geometry> boundary() const; // Will return the closure of the
    //    // combinatorial boundary.
    //    boolean closed() const; // Returns TRUE if the current object is clo-
sed
    //    // i.e. curve w/identical endpoints, surface that bounds a solid...
    //    boolean simple() const; // TRUE if current objekt has no anomalous
geometric
    //    // points, such as self intersection or self tangency.
    //    boolean connected() const; // TRUE if current object is connected.
That is if
    //    // any two points in the set can be joined by a curve of points in
the set.
    //    boolean contains(in lref<Geometry> inC_gm) const; // TRUE if
    //    // passed geometry is spatially within the current object
    //    boolean isIn(in lref<Geometry> inC_gm) const; // TRUE if current
    //    // object is spatially within passed geometry.
  };
  // ---------------------------------------------------------------
  // Interface name: SH_Line_String
  // Description:
  //
  // ---------------------------------------------------------------
  interface SH_Line_String : public SH_Curve {
  private:
  public:
    void initialize(in ref<SH_Accuracy> inAccuracyRef);
    void initWKS(in lref<LineString> inC_ls,
    in ref<SH_Accuracy> inAccuracyRef);
    void createRing(in lref<Ring> inC_ri,
    in ref<SH_Accuracy> inAccuracyRef);
    // Constructur  when a ring is created
    void finalize();
    void getCoordinateGeometry(inout CoordGeom inC_cg) const;
    relationship set<SH_Point> constituent inverse partOf;
    relationship set<SH_Polygon> partOf inverse constituent;
    //      lref<Geometry> Export() const; // Returns geometry structure and
refsys
    //    lref<Envelope> minBoundBox() const; // Returns minimum bounding box
    //    boolean equal(in lref<Geometry> inC_gm) const; // Determines
    //    // whether or not the passed geometry is spatially equal to the geo-
metry
    //    // No tolerance considerations are taken here!
    //    Distance distance(in lref<Geometry> inC_gm) const; // Returns
    //    // spatial distance between two objects. Defined to be the shortest
distance
    //    // between any two points chosen from the two objects, within same
SpaceRefSys
    //      lref<Geometry> bufferZone(in Distance inDistance); // Returns a
geometry
    //    // that represents all points whose distance is less or equal to the
given
    //    // distance, within same SpaceRefSys
    //    lref<Geometry> boundary(); // Will return the closure of the
    //    // combinatorial boundary.
    //    boolean closed() const; // Returns TRUE if the current object is
closed
    //    // i.e. curve w/identical endpoints, surface that bounds a solid...
    //    boolean simple() const; // TRUE if current objekt has no anomalous
geometric
    //    // points, such as self intersection or self tangency.
    //    boolean connected() const; // TRUE if current object is connected.
That is if
    //    // any two points in the set can be joined by a curve of points in
the set.
```

```
    //     boolean contains(in lref<Geometry> inC_gm) const; // TRUE if
    //     // passed geometry is spatially within the current object
    //     boolean isIn(in lref<Geometry> inC_gm) const; // TRUE if current
    //     // object is spatially within passed geometry.
  };
  // ------------------------------------------------------------
  // Interface name: SH_Polygon
  // Description:
  //
  // ------------------------------------------------------------
  interface SH_Polygon : public SH_Surface {
  private:
  public:
    void initialize(in ref<SH_Accuracy> inAccuracyRef);
    void initWKS(in lref<Polygon> inC_pg,
    in ref<SH_Accuracy> inAccuracyRef);
    void finalize();
    void getCoordinateGeometry(inout CoordGeom inC_cg) const;
    relationship set<SH_Line_String> constituent inverse partOf;
     //      lref<Geometry> Export() const; // Returns geometry structure and
refsys
    //     lref<Envelope> minBoundBox() const; // Returns minimum bounding box
    //     boolean equal(in lref<Geometry> inC_gm) const; // Determines
    //     // whether or not the passed geometry is spatially equal to the geo-
metry
    //     // No tolerance considerations are taken here!
    //     Distance distance(in lref<Geometry> inC_gm) const; // Returns
    //     // spatial distance between two objects. Defined to be the shortest
distance
    //     // between any two points chosen from the two objects, within same
SpaceRefSys
     //      lref<Geometry> bufferZone(in Distance inDistance); // Returns a
geometry
    //     // that represents all points whose distance is less or equal to the
given
    //     // distance, within same SpaceRefSys
    //     lref<Geometry> boundary(); // Will return the closure of the
    //     // combinatorial boundary.
     //      boolean closed() const; // Returns TRUE if the current object is
closed
    //     // i.e. curve w/identical endpoints, surface that bounds a solid...
    //     boolean simple() const; // TRUE if current objekt has no anomalous
geometric
    //     // points, such as self intersection or self tangency.
    //     boolean connected() const; // TRUE if current object is connected.
That is if
    //     // any two points in the set can be joined by a curve of points in
the set.
    //     boolean contains(in lref<Geometry> inC_gm) const; // TRUE if
    //     // passed geometry is spatially within the current object
    //     boolean isIn(in lref<Geometry> inC_gm) const; // TRUE if current
    //     // object is spatially within passed geometry.
  };
  // ------------------------------------------------------------
  // Interface name: SH_Indices
  // Description: All indices in system is in this object
  //
  // ------------------------------------------------------------
  interface SH_Indices {
  public:
    attribute index<A_Point,SH_Point> aPointInd;
    attribute index<string,SH_Spatial_Ref_System> aSpaceRefInd;
    attribute index<long,SH_Geometry> serialNoInd;
    attribute index<long,SH_Feature> themeInd;
```

```
    attribute index<string,SH_Feature> geomTypeInd;
    void initialize() const;

    // Methods on pointInd
    void newPoint(in A_Point inSH_pt, in ref<SH_Point> inaPoint) const;
    ref<SH_Point> findPoint(in A_Point inSH_pt) const;
    void killPoint(in A_Point inSH_pt) const;
    // Methods on refsysInd
    void newRefsys(in lref<char> inSpaceRef,
   in ref<SH_Spatial_Ref_System> inaSpaceRef) const;
    ref<SH_Spatial_Ref_System> findRefsys(in lref<char> inSpaceRef) const;
    void killRefsys(in lref<char> inSpaceRef) const;
    // Methods on serialNoInd
    void newSerialNo(in long inSerialNo,
     in ref<SH_Geometry> inaGeometry) const;
    ref<SH_Geometry> findSerialNo(in long inSerialNo) const;
    void killSerialNo(in long inSerialNo) const;
    void killAllSerialNoInx() const;
    // Methods on themeInd
    void newThemeInd(in long inTheme,
     in ref<SH_Feature> inaFeature) const;
    void killThemeInd(in long inTheme,
      in ref<SH_Feature> inaFeature) const;
    // Methods on geomTypeInd
    void newGeomTypeInd(in lref<char> inGeomType,
in ref<SH_Feature> inaFeature) const;
    void killGeomTypeInd(in lref<char> inGeomType,
 in ref<SH_Feature> inaFeature) const;
  }; // end interface SH_Indices
}
```

Vedlegg C SQL Datadefinisjoner; GO.sql

```
-- ************************************************************
-- File name: GO.sql
-- Project: GO
-- Copyright: IDT, NTNU
-- Language used: SQL
-- Short description: Table definitions for GO
-- Class-library of wxWindows is used with permission.
-- The Postgres95 databasesystem is used with permission
-- Modified:
-- who/when/what: Knut Aage Aksnes, 31.12.96 Created
-- ************************************************************

DROP TABLE Feature, Feature_parts, Spatial_Ref_System, Geometric_Property,
Accuracy, Geometry_parts, Point, Line_Points, Line_String,
Polygon_Lines, Polygon, Geometry;

DROP FUNCTION aGeomP_setGeometry(oid,oid);

CREATE TABLE Feature (
FT_name text,
description text,
aType int4,
someStuff text
);
CREATE TABLE Feature_parts (
FT_constitutent oid, -- Feature-master, should be unique
FT_partOf oid  -- Feature-parts
);

CREATE TABLE Spatial_Ref_System (
SP_name text,
linear_unit float,
angular_unit text,
prime_meridian text,
ellipsoide text,
GCS text,
CT text,
zone int4,
origoN int4,
origoOE int4,
maxN int4,
maxOE int4,
minN int4,
minOE int4,
PC text,
temporalSystem text,
spaceDimension int4,
timeDimension int4,
totalDimension int4
);

CREATE TABLE Geometric_Property (
feature oid, -- this, together with
```

```
geometry oid,  -- this should be unique in table
spaceRef oid
);

CREATE TABLE Accuracy (
accuracy int4,
measureMethod int4
);

CREATE TABLE Geometry (
coordDimension int4,
planar bool,
closed bool,
simple bool,
accuracyRef oid,
dimension int4,
interpretation int4
);

CREATE TABLE Geometry_parts (
GM_constituent oid, -- This should be unique
GM_partOf oid
);

CREATE TABLE Point (
east int4,
north int4,
height int4,
date int4
) INHERITS (Geometry);

CREATE TABLE Line_Points (
LS_constituent  oid,  -- This one should be unique
PT_partOf oid,
PT_seq_no int4
);

-- For simplyfying reasons, Curve and Surface aren’t included as tables

CREATE TABLE Line_String (
) INHERITS (Geometry);

CREATE TABLE Polygon_Lines (
PG_constituent oid, -- This one should be unique
LS_partOf oid,
LS_seq_no int4
);

CREATE TABLE Polygon (
) INHERITS (Geometry);

CREATE FUNCTION aGeomP_setGeometry(oid,oid) RETURNS int4
AS ‘UPDATE Geometric_Property
SET geometry = $2::oid
WHERE oid = $1::oid
SELECT 1’
LANGUAGE ‘sql’;

CREATE INDEX spaceRefinx
ON  Geometric_Property
USING btree(spaceRef oid);

CREATE INDEX PG_constituent_inx
ON  Polygon_Lines
```

```
USING btree(PG_constituent oid);

CREATE INDEX LS_constituent_inx
ON  Line_Points
USING btree(LS_constituent oid);
```

Vedlegg D Brukergrensesnitt (GUI); GO.h

```
// ************************************************************
// File name: GO.h
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Header file for GO.C
// Class-library of wxWindows is used with permission.
// The Shore databasesystem(beta-release) is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 26.11.96, Created
// ************************************************************
#ifndef GO_HDR
#define GO_HDR
// Forward decl's
class MyApp;
class MyTextWindow;
class MyCanvas;
class MyFrame;


// Define a new application
class MyApp : public wxApp {
  public:
  MyApp(void) ;
  wxFrame *OnInit(void);
};

// Define frames
class MyFrame: public wxFrame
{
  public:
  long dy, ymin, ymax, dx, xmin, xmax, origox, origoy;
  long xdomain, ydomain;
  float db_enhet, xzoom, yzoom, zoom_factor, xstart;
  Geometry C_gm_dbGeometry;
  wxPanel *panel;
  MyTextWindow *text_window;
  MyCanvas *canvas;
  wxText *canvasXmin;
  wxText *canvasYmin;
  wxText *canvasXmax;
  wxText *canvasYmax;
  MyFrame(wxFrame *frame, Const char *title, int x, int y, int w, int h);
  void OnSize(int w, int h);
  Bool OnClose(void);
  void OnMenuCommand(int id);
  void Draw(wxDC& dc,int id);
  void OnActivate(Bool) {};
  void NullArea();
  void NullFrame();
  void drawGeometry(wxDC& dc, CoordGeom& inC_cg);
  void drawText(wxDC& dc, CoordGeom& inC_cg, char* inString);
};
```

```
// Define a new text subwindow that can respond to drag-and-drop
class MyTextWindow: public wxTextWindow
{
  public:
  MyTextWindow(wxFrame *frame, int x=-1, int y=-1, int width=-1, int height=-
1,
               long style=0):
    wxTextWindow(frame, x, y, width, height, style)
  { DragAcceptFiles(TRUE);
  }

  void OnDropFiles(int n, char *files[], int x, int y)
  {
    int k = n + x + y;
    k += 5; // JUst kidding -- to kill some warnings...
    LoadFile(files[0]);
  }
  void OnChar(wxKeyEvent& event) {
    wxTextWindow::OnChar(event);
  }
};

// Define a new canvas which can receive some events
class MyCanvas: public wxCanvas
{
  public:
  float xpos;
  float ypos;
  MyCanvas(wxFrame *frame, int x, int y, int w, int h, long style = wxRETAI-
NED);
  ~MyCanvas(void) ;
  void OnPaint(void);
  void OnSize(int w, int h);
  void OnEvent(wxMouseEvent& event);
  void OnChar(wxKeyEvent& event);
  void OnScroll(wxCommandEvent& event);
};

// Callbacks
void text_proc(wxText& theText, wxCommandEvent& event);
void spaceRef_proc(wxListBox& theList, wxCommandEvent& event);
void theme_proc(wxListBox& theList, wxCommandEvent& event);
void db_proc(wxChoice& theChoice, wxCommandEvent& event);

#define CREATEDATA        1000
#define LOADSOSI          1100
#define GETDATA           1200
#define DELETEDATA        1300
#define RESETTHEMES       1350
#define CLOSEDB           1400
#define QUIT              1500
#defineSAVELOGFILE  1600
#define ZOOM              1700
#define GO_ABOUT          1800

#define MAXCANVAS         3000
#defineSPACEREF_TAG  2
#defineOK                1
#define NOTOK             0
#define SHORE             0
#define POSTGRES          1
#define SYBASE            2
```

```
// End of GO_HDR
#endif
```

Vedlegg E Brukergrensesnitt (GUI); GO.C

```
// ************************************************************
// File name: GO.C
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Implementation of user interface for
// the GO Project.
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 26.11.96 Created
// ************************************************************
#include <ctype.h>
#include “wx.h”
#include “aiai.xbm”
#include “wx_mf.h”
#include “wx_date.h”
#include “wx_timer.h”
#include “SH_Objects.h”
#include “PG_Objects.h”
#include “GO.h”
#include “PGDB.h”
#include “SHDB.h”
// Some gloabls
char* xmin_label = “Minimum X”;
char* ymin_label = “Minimum Y”;
char* xmax_label = “Maximum X”;
char* ymax_label = “Maximum Y”;
char* testSpaceRef = “Eksperimental”;
char* ALL = “Alle”;
char* SpaceRef = new char[80];
boolean choosedSpaceRef = FALSE;
// Declare two frames main, and map-canvas
MyFrame*      frame = NULL;
wxFrame*      subframe = NULL;
MyCanvas*     canvas = NULL;
MyTextWindow* myTextWindow = NULL;
MyDB*         myDB = NULL;
PGDB          pgDB;
SHDB          shDB;
wxMenuBar*    menu_bar = NULL;
wxIcon*       test_icon = NULL;
wxMenu*       the_file_menu = NULL;
wxMenu*       the_db_menu = NULL;
wxMenu*       the_help_menu = NULL;
wxListBox*    spaceRefList;
wxListBox*    themeList;
wxDate*       date;
// Must initialise these in OnInit, not statically
wxPen*        black_pen = NULL;
wxPen*        grid_pen = NULL;
wxFont*       labelFont = NULL;
wxFont*       itemFont = NULL;
```

```
wxFont*       textWindowFont = NULL;
wxCursor*     wxwait = NULL;
wxCursor*     arrow = NULL;



// This statement initialises the whole application
// ************************************************************
MyApp     myApp;
// ************************************************************
// A macro needed for some compilers (AIX) that need ‘main’ to be
// defined in the application itself.
IMPLEMENT_WXWIN_MAIN

/*-----------------------------------------------------------------
   Class name: MyApp
   Description:
  Application main class
  ----------------------------------------------------------------- */
// Testing of ressources
MyApp::MyApp() {
}

// The `main program’ equivalent, creating the windows and returning
// the main frame

wxFrame* MyApp::OnInit(void)
{
  // Create pens
  black_pen = new wxPen(“BLACK”, 2, wxSOLID);
  grid_pen = new wxPen(“LIGHT STEEL BLUE”, 1, wxSOLID);

  // Create fonts
  itemFont = new wxFont(11, wxROMAN, wxNORMAL, wxNORMAL);
  labelFont = new wxFont(12, wxROMAN, wxITALIC, wxBOLD);
  textWindowFont = new wxFont(12, wxSWISS, wxNORMAL, wxNORMAL);

  // Create main frame window
  frame = new MyFrame(NULL, “GO Hovedpanel”, 0, 0, 550, 500);
  // Give it a status line
  frame->CreateStatusLine(2);
  // Load icon and bitmap
  test_icon = new wxIcon(aiai_bits, aiai_width, aiai_height);
  frame->SetIcon(test_icon);

  // Menu database; options
  wxMenu* db_menu = new wxMenu;
  the_db_menu = db_menu;
  db_menu->Append(CREATEDATA, “Generer”,
  “Generer testdata”);
  db_menu->Append(LOADSOSI, “Laste SOSI-fil”,
  “Testdata fra SOSI-fil til database”);
  db_menu->Append(GETDATA, “Hente data”,
  “Hente data i database ut fra gjeldende valg”);
  db_menu->Append(RESETTHEMES, “Reset tema”,
  “Reset valgte temakoder til Alle”);
  db_menu->Append(DELETEDATA, “Slett testdata”,
  “Slette testdata ut fra gjeldende valg”);
  db_menu->Append(CLOSEDB, “Lukk database”,
  “Lukk database”);
  db_menu->AppendSeparator();
  db_menu->Append(QUIT, “Avslutt”,
  “Avslutt program”);
```

```
  // Menu file; options
  wxMenu* file_menu = new wxMenu;
  the_file_menu = file_menu ;
  file_menu->Append(SAVELOGFILE, “Lagre logg”,
    “Lagre data fra tekstvindu som loggfil”);
  file_menu->Append(ZOOM, “Zoom...”,
 “Zoom etter spesifikasjon”);
  file_menu->AppendSeparator();

  //Menu Help; options
  wxMenu* help_menu = new wxMenu;
  the_help_menu = help_menu;
  help_menu->Append(GO_ABOUT, “About”,
    “Copyright notice; wxWindows”);

  //Assiciate menus to menubar
  menu_bar = new wxMenuBar;
  menu_bar->Append(db_menu, “Database”);
  menu_bar->Append(file_menu, “File”);
  menu_bar->Append(help_menu, “Hjelp”);

  // Associate the menubar with the frame
  frame->SetMenuBar(menu_bar);

  // Some Menuitems are selectable on start(TRUE), some are not(FALSE)
  // User has to chose a database in frame before anything can be done...
  menu_bar->Enable(CREATEDATA,FALSE);  //
  menu_bar->Enable(LOADSOSI,FALSE);    //
  menu_bar->Enable(GETDATA,FALSE);     //
  menu_bar->Enable(DELETEDATA,FALSE);  //
  menu_bar->Enable(CLOSEDB,FALSE);     //
  menu_bar->Enable(QUIT,TRUE);         //
  menu_bar->Enable(SAVELOGFILE,FALSE); //
  menu_bar->Enable(ZOOM,FALSE);        //
  menu_bar->Enable(GO_ABOUT,TRUE);     //

  // Make a panel
  frame->panel = new wxPanel(frame,0,0,1000,500,0,”MyMainFrame”);
  frame->panel->SetLabelPosition(wxVERTICAL);
  frame->panel->SetLabelFont(labelFont);
  frame->panel->SetButtonFont(itemFont);

  // Create the panel items
  // A button to chose which database to run
  char *choice_strings[4];
  choice_strings[0] = “Shore”;
  choice_strings[1] = “Postgres95”;
  choice_strings[2] = “Sybase”;
  choice_strings[3] = “Ingen”;
  wxChoice *choice = new wxChoice(frame->panel,(wxFunction)&db_proc,
  “Valgt database”, -1, -1, -1, -1, 4,
  choice_strings);
  choice->SetSelection(3);
  choosedSpaceRef = NULL; // User hasn’t chosed SpaceRef yet
  strcpy(SpaceRef,testSpaceRef); // Just so nothin crazy happens.
  // A list of all SpaceRef-objs in database, initially empty
  spaceRefList = new wxListBox(frame->panel, (wxFunction)&spaceRef_proc,
       “Spatial Ref Systems”, wxSINGLE|wxNEEDED_SB,
       -1, -1, 150, 150);

  // A list of all chosen “theme” values currently included in canvas
  // theme-codes might be added & deleted later
  themeList = new wxListBox(frame->panel, (wxFunction)&theme_proc,
       “Gjeldende tema-koder”, wxSINGLE|wxNEEDED_SB,
```

```
       -1, -1, 150, 150);
  themeList->Append(ALL); // Initially sett to “Alle”
  frame->panel->NewLine();

  // The area represented by the canvas, by max&min values
  // that might be changed
  // Initially nothing of course....
  frame->NullArea();
  frame->canvasXmin = new wxText(frame->panel, (wxFunction)&text_proc,
 xmin_label, FloatToString((float)frame->xmin) ,
    -1, -1, 50, -1, wxPROCESS_ENTER) ;
  frame->canvasYmin = new wxText(frame->panel, (wxFunction)&text_proc,
 ymin_label, FloatToString((float)frame->ymin),
    -1, -1, 50, -1, wxPROCESS_ENTER) ;
  frame->canvasXmax = new wxText(frame->panel, (wxFunction)&text_proc,
 xmax_label, FloatToString((float)frame->xmax),
    -1, -1, 50, -1, wxPROCESS_ENTER) ;
  frame->canvasYmax = new wxText(frame->panel, (wxFunction)&text_proc,
 ymax_label, FloatToString((float)frame->ymax),
    -1, -1, 50, -1, wxPROCESS_ENTER) ;

  frame->panel->NewLine();
  // Make a text window
  myTextWindow = new MyTextWindow(frame, 0, 250, 400, 250, wxNATIVE_IMPL);
  frame->text_window = myTextWindow;
  frame->text_window->SetFont(textWindowFont);
  frame->text_window->DragAcceptFiles(TRUE);

  date = new wxDate;
  date->SetFormat(wxEUROPEAN);
       frame->text_window->SetInsertionPoint(frame->text_window->GetLastPosi-
tion());
  *frame->text_window << “Applikasjonen startet: “;
  *frame->text_window << date->Set();
  *frame->text_window << “\n”;


  // Make another frame, containing a canvas
  subframe = new wxFrame(NULL, “GO Kartpanel”, 300, 300, 500, 500);
  int width, height;
  subframe->GetClientSize(&width, &height);
  canvas = new MyCanvas(subframe, 0, 0,
width, height, wxRETAINED);
  wxCursor* cursor = new wxCursor(wxCURSOR_PENCIL);
  wxwait   = new wxCursor(wxCURSOR_WAIT);
  arrow  = new wxCursor(wxCURSOR_ARROW);
  canvas->SetBackground(wxWHITE_BRUSH);
  canvas->SetCursor(cursor);
  // Give it scrollbars: the virtual canvas is
  // 20 * 50 = 1000 pixels in each direction
  canvas->SetScrollbars(20, 20, 50, 50, 4, 4);
  canvas->SetPen(black_pen);
  wxDC* dc = canvas->GetDC();
  dc->SetMapMode(MM_LOMETRIC);
  frame->canvas = canvas;
  frame->OnSize(550, 500);
  subframe->Show(TRUE);
  frame->Show(TRUE);
  frame->SetStatusText(“Velg hvilken database som skal benyttes”);
  // Return the main frame window
  return frame;
}

/*-----------------------------------------------------------------
```

```
   Class name: MyFrame
   Description:
  Application main class
  ------------------------------------------------------------ */

// Define my frame constructor
MyFrame::MyFrame(wxFrame* frame, Const char* title,
 int x, int y, int w, int h):
  wxFrame(frame, title, x, y, w, h)
{
  panel = NULL;
}

// Intercept menu commands
void MyFrame::OnMenuCommand(int id)
{
  wxDC* dc = canvas->GetDC();
  switch (id)
  {
    case DELETEDATA:
      {
wxStartTimer();
frame->SetCursor(wxwait);
long rc = myDB->DeleteData();
if (rc == NOTOK)
  frame->SetStatusText(“Dette gikk ikke helt bra!”);
else
  frame->SetStatusText(“Testdata er slettet i Shore Databasen”);
// Update list og SpatialReference systems shown i frame
spaceRefList->Clear();
long okey = myDB->FillListBox(spaceRefList);
if (!okey)
   frame->SetStatusText(“Feil ved oppdatering av Spatial Ref Systems over-
sikt”);
else
  frame->SetStatusText(“Testfil er lagret i Shore Databasen”);
strcpy(SpaceRef,testSpaceRef); // Just so nothin crazy happens.
choosedSpaceRef = NULL; // User hasn’t chosed SpaceRef yet
themeList->Clear();
themeList->Append(ALL);
frame->NullFrame();
frame->SetCursor(arrow);
frame->Show(TRUE);
text_window->SetInsertionPoint(text_window->GetLastPosition());
*text_window << “Elapsed time, DELETEDATA “;
*text_window << wxGetElapsedTime(TRUE);
*text_window << “\n”;
// We are ready to run..
break;
      }

    case CREATEDATA:
      {
wxStartTimer();
frame->SetCursor(wxwait);
float inNo = 100;
char *inNo_s =FloatToString(inNo);
char  *text  =  wxGetTextFromUser(“Oppgi  antall  objekter”,  “Antall  input”,
inNo_s);
if (text) {
  StringToFloat(text, &inNo);
}
long rc = myDB->CreateData((long)inNo);
if (rc == NOTOK)
```

```
  frame->SetStatusText(“Dette gikk ikke helt bra!”);
else
  frame->SetStatusText(“Testdata er lagret i valgt database”);
// Update list og SpatialReference systems shown i frame
spaceRefList->Clear();
long okey = myDB->FillListBox(spaceRefList);
if (!okey)
  frame->SetStatusText(“Feil ved henting av Spatial Ref Systems oversikt”);

strcpy(SpaceRef,testSpaceRef); // Just so nothing crazy happens.
choosedSpaceRef = NULL; // User hasn’t chosed SpaceRef yet
themeList->Clear();
themeList->Append(ALL);
frame->SetCursor(arrow);
frame->Show(TRUE);
text_window->SetInsertionPoint(text_window->GetLastPosition());
*text_window << “Elapsed time, CREATEDATA “;
*text_window << wxGetElapsedTime(TRUE);
*text_window << “ no of elements: “;
*text_window << inNo;
*text_window << “\n”;
delete inNo_s;
delete text;
break;
      }
    case LOADSOSI:
      {
wxStartTimer();
char* aFile = wxFileSelector(“Laste SOSI-fil”, NULL, NULL, NULL, “*.sos”);
if (aFile) {
  frame->SetCursor(wxwait);
  frame->Show(TRUE);
  long rc = myDB->LoadSOSI(aFile); // Load the file
  if (rc == OK) {
    // Update list og SpatialReference systems shown i frame
    spaceRefList->Clear();
    long okey = myDB->FillListBox(spaceRefList);
    if (!okey)
      frame->SetStatusText(“Feil ved oppdatering av Spatial Ref Systems over-
sikt”);
    else
      frame->SetStatusText(“Testfil er lagret i Shore Databasen”);
  }
  else
    frame->SetStatusText(“Feil ved behandling av SOSI-fil”);
  strcpy(SpaceRef,testSpaceRef); // Just so nothin crazy happens.
  choosedSpaceRef = NULL; // User hasn’t chosed SpaceRef yet
  themeList->Clear();
  themeList->Append(ALL);
  frame->NullFrame();
  frame->SetCursor(arrow);
  frame->Show(TRUE);
}
text_window->SetInsertionPoint(text_window->GetLastPosition());
*text_window << “Elapsed time, LOADSOSI “;
*text_window << wxGetElapsedTime(TRUE);
*text_window << “\n”;
delete aFile;
break;
      }
    case GETDATA:
      {
wxStartTimer();
frame->SetCursor(wxwait);
```

```
// Start a transaction for initialization
C_gm_dbGeometry.reference_system_name = SpaceRef;
dc->Clear();
if (xzoom < yzoom)
  zoom_factor = xzoom;
else
  zoom_factor = yzoom;
if (zoom_factor > 2.5) {
  dc->SetMapMode(MM_METRIC);
  zoom_factor = (float)zoom_factor / 10;
}
else
  dc->SetMapMode(MM_LOMETRIC);
dc->SetUserScale(zoom_factor, zoom_factor);
Draw(*dc,id);
frame->SetCursor(arrow);
frame->SetStatusText(“Data er hentet i databasen”);
frame->Show(TRUE);
text_window->SetInsertionPoint(text_window->GetLastPosition());
*text_window << “Elapsed time, GETDATA “;
*text_window << wxGetElapsedTime(TRUE);
*text_window << “\n”;
break;
      }

    case CLOSEDB:
      {
wxStartTimer();
frame->SetCursor(wxwait);
myDB->CloseDB();
subframe->Show(FALSE);
frame->SetStatusText(“Lukket database!”);
frame->Show(TRUE);
menu_bar->Enable(CREATEDATA,FALSE);
menu_bar->Enable(LOADSOSI,FALSE);
menu_bar->Enable(GETDATA,FALSE);
menu_bar->Enable(DELETEDATA,FALSE);
menu_bar->Enable(CLOSEDB,FALSE);
menu_bar->Enable(QUIT,TRUE);
menu_bar->Enable(SAVELOGFILE,FALSE);
menu_bar->Enable(ZOOM,FALSE);
menu_bar->Enable(GO_ABOUT,TRUE);
frame->SetCursor(arrow);
text_window->SetInsertionPoint(text_window->GetLastPosition());
*text_window << “Elapsed time, CLOSEDB “;
*text_window << wxGetElapsedTime(TRUE);
*text_window << “\n”;
break;
      }
    case ZOOM:
      {
wxStartTimer();
char* zoom_factor_s = FloatToString(zoom_factor);
char* new_zoom_factor_s = wxGetTextFromUser(“Oppgi ny zoom-faktor”,
    “Input”, zoom_factor_s);
// no more malloced.
//delete [] zoom_factor_s ;
zoom_factor_s = new_zoom_factor_s ;
if (zoom_factor_s)
  {
    StringToFloat(zoom_factor_s, &zoom_factor);
    dc->SetUserScale(zoom_factor, zoom_factor);
    dc->Clear();
    // last param required for cl386
```

```
    Draw(*dc,id);
    // no more malloced.
    //delete [] zoom_factor_s ;
  }
text_window->SetInsertionPoint(text_window->GetLastPosition());
*text_window << “Elapsed time, ZOOM “;
*text_window << wxGetElapsedTime(TRUE);
*text_window << “\n”;
delete zoom_factor_s;
delete new_zoom_factor_s;
break;
      }
    case SAVELOGFILE:
      {
wxStartTimer();
         char* s = wxFileSelector(“Lagre logg-fil”, NULL, “GO.log”, “.log”,
“*.log”);
if (s) {
  text_window->SaveFile(s);
}
text_window->SetInsertionPoint(text_window->GetLastPosition());
*text_window << “Elapsed time, SAVELOGFILE “;
*text_window << wxGetElapsedTime(TRUE);
*text_window << “\n”;
delete s;
break;
      }
    case RESETTHEMES:
      {
themeList->Clear();
themeList->Append(ALL);
frame->Show(TRUE);
break;
      }
    case QUIT:
      {
if (myDB)
  myDB->CloseDB();
OnClose();
delete this;
break;
      }
    case GO_ABOUT:
      {
(void)wxMessageBox(“Demo versjon GO v.0.8 desember 1996\nForfatter Knut Aage
Aksnes, tm2akaa@idt.unit.no\nIDT, NTNU  (c) 1996”,
   “Om GO”, wxOK|wxCENTRE);
break;
      }
    default:
      {
wxCursor* cursor = new wxCursor(id);
frame->canvas->SetCursor(cursor);
break;
      }
    }
}

// Size the subwindows when the frame is resized
void MyFrame::OnSize(int w, int h) {
  int z;      // just to kill a warning
  z =  w + h; // nonsence
  if (panel && text_window)
  {
```

```
    int width, height;
    GetClientSize(&width, &height);
    panel->SetSize(0, 0, width, (int)(height/2));
    text_window->SetSize(0, (int)(height/2), width, (int)(height/2));
  }
}

void MyFrame::Draw(wxDC& dc, int id) {
  // Canvas is 1000 x 1000 pixels
  // When using MM_LOMETRIC, canvas is approcximately
  // 3000x3000 logical units
  // When MM_METRIC, 300x300
  // for defs of wxGREY_BRUSH etc, see ..wxWindows/include/base/wb_gdi.h
  if (!choosedSpaceRef)
    return;
  long rc;
  dc.SetPen(black_pen);
  dc.SetFont(itemFont);
  wxGREY_BRUSH->SetStyle(wxTRANSPARENT);
  dc.SetBrush(wxGREY_BRUSH);
  switch (dc.GetMapMode()) {
    case MM_LOMETRIC:
      xstart = (float)3000/zoom_factor;
      break;
    case MM_METRIC:
      xstart = (float)300/zoom_factor;
      break;
    default:
      xstart = (float)300/zoom_factor;
  }
  switch (id) {
    case ZOOM:
      if (!strcmp(themeList->GetString(0),ALL)) {
drawGeometry(dc,C_gm_dbGeometry.coordinate_geometry);
      }
      else {
rc = myDB->getThemes(*themeList,dc);
if (rc == NOTOK) {
  frame->SetStatusText(“Feil ved henting av temakoder i database”);
  return;
}
      }
      break;
    default:
      if (!strcmp(themeList->GetString(0),ALL)) {
rc = myDB->GetData(C_gm_dbGeometry);
if (rc == NOTOK) {
  frame->SetStatusText(“Feil ved henting av data i database”);
  return;
}
drawGeometry(dc,C_gm_dbGeometry.coordinate_geometry);
      }
      else {
rc = myDB->getThemes(*themeList,dc);
if (rc == NOTOK) {
  frame->SetStatusText(“Feil ved henting av temakoder i database”);
  return;
}
      }
  }
  return;
}
```

```
void MyFrame::drawGeometry(wxDC& dc, CoordGeom& inC_cg) {
  unsigned int i;
  float x,y;
  wxNode* node = NULL;
  wxList* someList = NULL;
  wxPoint* somePoint = NULL;
  Coordinate C_pt;
  LineString C_ls;
  CoordGeom C_cg;
  switch (inC_cg.get_CoordGeomType()) {
    case CoordCoordinateType:
      x = xstart -
(db_enhet*(float)inC_cg.get_point().get_elt(0) - dx);
      y = db_enhet*(float)inC_cg.get_point().get_elt(1) - dy;
      //y = xstart - (db_enhet*(float)inC_cg.get_point().get_elt(1) - dy);
      dc.DrawPoint(y,x);
      break;
    case CoordLineStringType:
      // iterate throug sequence of Coordinates and create wxList.
      someList = new wxList;
      for (i = 0;
   i < inC_cg.get_line_string().get_size();
   i++) {
C_pt = inC_cg.get_line_string().get_elt(i);
x = xstart - (db_enhet*(float)C_pt.get_elt(0) - dx);
y = db_enhet*(float)C_pt.get_elt(1) - dy;
//y = xstart - (db_enhet*(float)C_pt.get_elt(1) - dy);
somePoint = new wxPoint(y,x);
someList->Append(somePoint);
      }
      dc.DrawLines(someList);
      node = someList->First();
      while(node) {
somePoint = (wxPoint* )node->Data();
delete somePoint;
delete node;
node = someList->First();
      }
      delete someList;
      break;
    case CoordRingType:
      // iterate throug sequence of Coordinates and create wxList.
      someList = new wxList;
      for (i = 0;
   i < inC_cg.get_ring().get_size();
   i++) {
C_pt = inC_cg.get_ring().get_elt(i);
x = xstart - (db_enhet*(float)C_pt.get_elt(0) - dx);
y = db_enhet*(float)C_pt.get_elt(1) - dy;
//y = xstart - (db_enhet*(float)C_pt.get_elt(1) - dy);
somePoint = new wxPoint(y,x);
someList->Append(somePoint);
      }
      dc.DrawPolygon(someList,0,0,wxODDEVEN_RULE);
      node = someList->First();
      while(node) {
somePoint = (wxPoint* )node->Data();
delete somePoint;
delete node;
node = someList->First();
      }
      break;
      delete someList;
    case CoordPolygonType:
```

```
      // iterate through sequence and draw each item
      for (i = 0;
   i < inC_cg.get_polygon().get_size();
   i++ ) {
C_cg.set_CoordGeomType(CoordRingType);
C_cg.set_ring() = inC_cg.get_polygon().get_elt(i);
drawGeometry(dc, C_cg);
      }
      break;
    case CoordGeomCollectionType:
      // iterate through sequence and draw each item
      for (i = 0;
   i < inC_cg.get_cgc().get_size();
   i++ ) {
C_cg = inC_cg.get_cgc().get_elt(i);
drawGeometry(dc,C_cg);
      }
      break;
    case CoordEnvelopeType:
    case CoordPolyhedralSolidType:
    case CoordPolyhedronType:
    case CoordPolyhedralSurfaceType:
    case CoordTriangleType:
    default:
      break;
  }
  return;
}

void MyFrame::drawText(wxDC& dc, CoordGeom& inC_cg, char* inString) {
  float x,y;
  Coordinate C_pt;
  switch (inC_cg.get_CoordGeomType()) {
    case CoordCoordinateType:
      x = xstart - (db_enhet*(float)inC_cg.get_point().get_elt(0) - dx);
      y = db_enhet*(float)inC_cg.get_point().get_elt(1) - dy;

      dc.DrawText(inString,y,x);
      break;
    default:
      break;
  }
  return;
}

  // Define a constructor for my canvas
MyCanvas::MyCanvas(wxFrame* frame, int x, int y, int w, int h, long style):
 wxCanvas(frame, x, y, w, h, style)
{
  xpos = ypos = -1.0;
}

MyCanvas::~MyCanvas(void)
{
}

// Define the repainting behaviour
void MyCanvas::OnPaint(void)
{
  // need the last param for cl386.
  frame->Draw(*(GetDC()),100);
}
void MyCanvas::OnSize(int w, int h)
{
```

```
  // Just to kill som compiler-warnings
  if (w)
    return;
  else if (h)
    return;
}
// This implements a tiny doodling program! Drag the mouse using
// the left button
void MyCanvas::OnEvent(wxMouseEvent& event)
{
  float x, y;
  event.Position(&x, &y);
  if (xpos > -1 && ypos > -1 && event.Dragging() && event.LeftIsDown())
  {
    SetPen(wxBLACK_PEN);
    SetBrush(wxTRANSPARENT_BRUSH);
    DrawLine(xpos, ypos, x, y);
  }
  xpos = x;
  ypos = y;
}

void MyCanvas::OnScroll(wxCommandEvent& event)
{
  wxCanvas::OnScroll(event);
}

// Intercept character input
void MyCanvas::OnChar(wxKeyEvent& event)
{
  char buf[2];
  buf[0] = (char)event.KeyCode();
  buf[1] = 0;
  frame->SetStatusText(buf,1);

  // Process the default behaviour
  wxCanvas::OnChar(event);
}

// Define the behaviour for the frame closing
// - must delete all frames except for the main one.
Bool MyFrame::OnClose(void)
{
  if (subframe)
    delete subframe;
  Show(FALSE);

  return TRUE;
}

void MyFrame::NullArea(){
    ymin=xmin=ymax=xmax=origoy
      =origox=xdomain=ydomain=0;
    db_enhet=zoom_factor=1.0;
    return;
}
void MyFrame::NullFrame(){
    ymin=xmin=ymax=xmax=origoy
      =origox=xdomain=ydomain=0;
    db_enhet=zoom_factor=1.0;
    frame->canvasXmin->SetValue(FloatToString((float)frame->xmin));
    frame->canvasYmin->SetValue(FloatToString((float)frame->ymin));
    frame->canvasXmax->SetValue(FloatToString((float)frame->xmax));
    frame->canvasYmax->SetValue(FloatToString((float)frame->ymax));
```

```
    return;
}

void db_proc(wxChoice& theChoice, wxCommandEvent& event)
{
       frame->text_window->SetInsertionPoint(frame->text_window->GetLastPosi-
tion());
  *(frame->text_window) << “Valgt database er: “;
  *(frame->text_window) << event.commandString;
  *(frame->text_window) << “\n”;
  // When changing database-type:
  // Close previous opened database
  // Open connection to the new one
  // Reset frame wrt SpaceRefs, themecodes and canvas-area params.

  switch (event.commandInt) {
    case SHORE:
if (myDB)
  myDB->CloseDB();
myDB = &shDB;
break;
    case POSTGRES:
if (myDB)
  myDB->CloseDB();
myDB = &pgDB;
break;
    case SYBASE:
// wont change previous choice in this case
// Not implemented!
if (myDB)
  myDB->CloseDB();
myDB = NULL;
*(frame->text_window) << “Sybase database er ikke implementert!\n”;
return;
    default:
if (myDB)
  myDB->CloseDB();
myDB = NULL;
return;
  }
  long okey = myDB->InitDB();
  if (!okey) {
    *(frame->text_window) << “Åpning av database gikk Shit!”;
    return;
  }
  okey = myDB->FillListBox(spaceRefList);
  if (!okey) {
    *(frame->text_window) << “Klarte ikke å lese SpaceRef’s!”;
    return;
  }
  themeList->Clear();
  themeList->Append(ALL);
  frame->NullFrame();
  strcpy(SpaceRef,testSpaceRef);
  choosedSpaceRef = NULL;
  menu_bar->Enable(CREATEDATA,TRUE);  //
  menu_bar->Enable(LOADSOSI,TRUE);    //
  menu_bar->Enable(GETDATA,TRUE);     //
  menu_bar->Enable(DELETEDATA,TRUE);  //
  menu_bar->Enable(CLOSEDB,TRUE);     //
  menu_bar->Enable(QUIT,TRUE);         //
  menu_bar->Enable(SAVELOGFILE,TRUE); //
  menu_bar->Enable(ZOOM,TRUE);        //
  menu_bar->Enable(GO_ABOUT,TRUE);     //
```

```
  *(frame->text_window) << “Databasen aapnet!\n”;
  frame->SetStatusText(“Velg Spatial Ref System”);
  frame->Show(TRUE);
  return;
}

// Gets some user input, and sets the status line
// Load result to appropriate field in MyFrame::attributes
void text_proc(wxText& theText, wxCommandEvent& event)
{
  float shit;
       frame->text_window->SetInsertionPoint(frame->text_window->GetLastPosi-
tion());
  char* label = theText.GetLabel();
  if (!strcmp(label,xmin_label)) {
    StringToFloat(theText.GetValue(),&shit);
    frame->xmin = (long)shit;
    *(frame->text_window) << “Xmin-Verdi endret til: “ ;
    *(frame->text_window) << event.commandString;
    *(frame->text_window) << “\n”;
  }
  if (!strcmp(label,ymin_label)){
    StringToFloat(theText.GetValue(),&shit);
    frame->ymin = (long)shit;
    *(frame->text_window) << “Ymin-Verdi endret til: “ ;
    *(frame->text_window) << event.commandString;
    *(frame->text_window) << “\n”;
  }
  if (!strcmp(label,xmax_label)){
    StringToFloat(theText.GetValue(),&shit);
    frame->xmax = (long)shit;
    *(frame->text_window) << “Xmax-Verdi endret til: “ ;
    *(frame->text_window) << event.commandString;
    *(frame->text_window) << “\n”;
  }
  if (!strcmp(label,ymax_label)){
    StringToFloat(theText.GetValue(),&shit);
    frame->ymax = (long)shit;
    *(frame->text_window) << “Ymax-Verdi endret til: “ ;
    *(frame->text_window) << event.commandString;
    *(frame->text_window) << “\n”;
  }
    delete label;
}

// Update global parameter SpaceRef:
void spaceRef_proc(wxListBox& theList, wxCommandEvent& event)
{
       frame->text_window->SetInsertionPoint(frame->text_window->GetLastPosi-
tion());
  *(frame->text_window) << “Ny SpaceRef valgt: “;
  *(frame->text_window) << event.commandString;
  *(frame->text_window) << “\n”;
  strcpy(SpaceRef,theList.GetStringSelection());
  long okey = myDB->AreaDefs();
  if (!okey) {
  *(frame->text_window) << “Feil ved AreaDef\n”;
  }
  choosedSpaceRef = TRUE;
  frame->canvasXmin->SetValue(FloatToString((float)frame->xmin));
  frame->canvasYmin->SetValue(FloatToString((float)frame->ymin));
  frame->canvasXmax->SetValue(FloatToString((float)frame->xmax));
  frame->canvasYmax->SetValue(FloatToString((float)frame->ymax));
```

```
  if (strcmp(SpaceRef,testSpaceRef)) {
    themeList->Clear();
    themeList->Append(“1”);
    themeList->Append(“3100”);
    themeList->Append(“3101”);
    themeList->Append(“3205”);
    themeList->Append(“3206”);
    themeList->Append(“3207”);
    themeList->Append(“355”);
    themeList->Append(“4000”);
    themeList->Append(“4001”);
    themeList->Append(“4002”);
    themeList->Append(“4003”);
    themeList->Append(“7100”);
    themeList->Append(“801”);
    themeList->Append(“9205”);
    themeList->Append(“9206”);
    themeList->Append(“9207”);
    //    themeList->Append(“9400”); // Doubly defined in testfile...
    themeList->Append(“9401”);
    themeList->Append(“9999”);
  }
  return;
}
// Update which themes should be included
void theme_proc(wxListBox& theList, wxCommandEvent& event)
{
  char* newTheme_temp = new char[30];
  char* newTheme = wxGetTextFromUser(“Oppgi ny temakode”,”Temakode”,NULL);
  if (newTheme) {
    strcpy(newTheme_temp,newTheme);
    if (!strcmp(theList.GetString(0),ALL))
theList.Delete(0);
    theList.Append(newTheme_temp);
       frame->text_window->SetInsertionPoint(frame->text_window->GetLastPosi-
tion());
    *(frame->text_window) << “Ny temakode: “;
    *(frame->text_window) << newTheme;
    *(frame->text_window) << “\n”;
  }
  delete newTheme_temp;
  return;
}
```

Vedlegg F Databaselag; MyDB.h

```
// ****************************************************************
// File name: MyDB.h
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Headerfile for databaseclasses
// the GO Project.
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 10.12.96 Created
// ****************************************************************
#ifndef MYDB_HDR
#define MYDB_HDR
// Forward's:
// class MyDB;
// class SHDB;
// class PGDB;

class MyDB {
  public:
  MyDB();
  boolean opened; // Is connection to DB ok?
  // Initialize connection to server:
  virtual long InitDB();
  // Create a number of polygons:
  virtual long CreateData(long noOfObjects);
  // Collect according to current selections in database:
  virtual long GetData(Geometry& inC_gm);
  // Delete data based on selections in frame:
  virtual long DeleteData();
  // Read SOSI-file and store in database
  virtual long LoadSOSI(char* aFile);
  // Close connection to server:
  virtual long CloseDB();
  // Fill a listbox with all SpaceRef's in database
  virtual long FillListBox(wxListBox* theList);
  // Select a SpaceRef, and get some initial settings
  virtual long AreaDefs();
  // Find and draw according to given themes
  virtual long getThemes(wxListBox& theList, wxDC& dc);
};
// End MYDB_HDR
#endif
```

Vedlegg G Databaselag; MyDB.C

```
// ****************************************************************
// File name: MyDB.C
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Headerfile for databaseclasses
// the GO Project.
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 10.12.96 Created
// ****************************************************************
#include “wx.h”
#include “SH_Objects.h”
#include “MyDB.h”

MyDB::MyDB()
{
  cout << “Not implemented, MyDB::MyDB()”;
}
long MyDB::InitDB()
{
  cout << “Not implemented, MyDB::InitDB()”;
  return 0;
}
// Create a number of polygons:
long MyDB::CreateData(long noOfObjects)
{
  cout << “Not implemented, MyDB::CreateData”;
  return 0;
}
// Collect according to current selections in database:
long MyDB::GetData(Geometry& inC_gm)
{
  cout << “Not implemented, MyDB::GetData”;
  return 0;
}
// Delete data based on selections in frame:
long MyDB::DeleteData()
{
  cout << “Not implemented, MyDB::DeleteData()”;
  return 0;
}
// Read SOSI-file and store in database
long MyDB::LoadSOSI(char* aFile)
{
  cout << “Not implemented, MyDB::LoadSOSI”;
  return 0;
}
// Close connection to server:
long MyDB::CloseDB()
{
  cout << “Not implemented, MyDB::CloseDB()”;
```

```
  return 0;
}
// Fill a listbox with all SpaceRef's in database
long MyDB::FillListBox(wxListBox* theList)
{
  cout << "Not implemented, MyDB::FillListBox";
  return 0;
}
// Select a SpaceRef, and get some initial settings
long MyDB::AreaDefs()
{
  cout << "Not implemented, MyDB::AreaDefs()";
  return 0;
}
// Find and draw according to given themes
long MyDB::getThemes(wxListBox& theList, wxDC& dc)
{
  cout << "Not implemented, MyDB::getThemes";
  return 0;
}
```

Vedlegg H Databaselag; SHDB.h

```
// ****************************************************************
// File name: SHDB.h
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Headerfile for databaseclasses
// the GO Project.
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 10.12.96 Created
// ****************************************************************
#ifndef SHDB_HDR
#define SHDB_HDR
#include "MyDB.h"
class SHDB : public MyDB {
  private:
  long elTotal,elPrevious,elCurrent;
  int serialNo;
  int theSerialNo;
  int ocSpaceRef, ocPunkt,ocKurve,ocLinje,
    ocTekst, ocFlate, ocInxNew, ocInxFind,
    ocAccuracy, ocGeomP, ocPoint, ocLineString,
    ocPolygon;
  float xx,yy,zz;
  bool reverse;
  bool KP;
  public:
  SHDB() {;}
  // Initialize connection to server:
  long InitDB();
  // Create a number of polygons:
  long CreateData(long noOfObjects);
  // Collect according to current selections in database:
  long GetData(Geometry& inC_gm);
  // Delete data based on selections in frame:
  long DeleteData();
  // Read SOSI-file and store in database
  long LoadSOSI(char* aFile);
  // Close connection to server:
  long CloseDB();
  // Fill a listbox with all SpaceRef's in database
  long FillListBox(wxListBox* theList);
  // Select a SpaceRef, and get some initial settings
  long AreaDefs();
  // Find and draw according to given themes
  long getThemes(wxListBox& theList, wxDC& dc);
  long aSerialNo(char* & param1) {
    int theSerialNo = 0;
    char *someString = new char[80]; strcpy(someString,"\0");
    sscanf(param1,"%i :%[A-Za-z0-9: ()-]",&theSerialNo,someString);
    strcpy(param1,someString); delete someString;
    return (long)theSerialNo;
```

```
  }
};

// Special function needed by Shore-function.
long fillListBox(wxListBox* theList);
long theme(long& themeCode, wxDC& dc);

//end SHDB_HDR
#endif
```

Vedlegg I Databaselag; SHDB.C

```
// ************************************************************
// File name: MyDB.C
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Implementation of classes MyDB, SHDB and
// PGDB, that implements methods on databases
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission
// The Postgres95 databasesystem is used with permission
// Modified:
// who/when/what: Knut Aage Aksnes,22.12.96 Created
// ************************************************************
#include “wx.h”
#include “wx_timer.h”
#include “SH_Objects.h”
#include “GO.h”
#include “MyDB.h”
#include “SHDB.h”
// Some externals
extern char* SpaceRef;
extern MyFrame* frame;
// Some global object references
Ref<Pool>                  SH_Features;
Ref<Pool>                  SH_Geometries;
Ref<Pool>                  SH_Index;
Ref<SH_Query_Manager>      SH_qm;
Ref<SH_Spatial_Ref_System> aSpaceRef;
Ref<SH_Feature>            aFeature;
Ref<SH_Feature>            anotherFeature;
Ref<SH_Accuracy>           anAccuracy;
Ref<SH_Geometric_Property> aGeomP;
Ref<SH_Point>              aPoint;
Ref<SH_Line_String>        aLineString;
Ref<SH_Polygon>            aPolygon;
Ref<SH_Polygon>            anotherPolygon;
Ref<any> ref;

// Following char’s belongs to Shore...
const char *ROOT = “/”;   // System root
const char *SH_ROOT = “GO”;   // Under which our system is stored
// Container of Feature, SpatialRefSys, GeomProp, Accuracy objects:
const char *SH_Feature_Pool = “Feature_pool”;
// Container of Geometry, Point, Curve, Surface, LineString,
// Polygon objects:
const char *SH_Geometry_Pool = “Geometry_pool”;
// Container of QueryManager and Indices objects.
const char *SH_Indices_Pool = “Indices_pool”;

// Initializing connection to Shore-server, and creating/position on
// correct catalog in database.
long SHDB::InitDB() {
  // initialize connection to Shore-server
```

```
  if (opened)
    return OK; // Already opened
  char* canYouImagine[] = {“GO”};
  int bullshit = 1;
  // Some automagic
  SH_DO(Shore::init(bullshit, canYouImagine, 0, “.shoreconfig”));
  ref = NULL;
  shrc rc;
  SH_BEGIN_TRANSACTION(rc);
  if (rc)
    rc.fatal(); // Terminates
  rc = Shore::chdir(SH_ROOT);  // Check that our main directory exists
  if (rc != RCOK) {
    if (rc != RC(SH_NotFound))
      SH_ABORT_TRANSACTION(rc);
    // Not found.  Must be the first time through.
    // Create the root directory
    SH_DO(Shore::mkdir(SH_ROOT, 0777));
    SH_DO(Shore::chdir(SH_ROOT));
    // Creates the pools for allocating objects
    SH_DO(SH_Features.create_pool(SH_Feature_Pool, 0666, SH_Features));
    SH_DO(SH_Geometries.create_pool(SH_Geometry_Pool, 0666, SH_Geometries));
    SH_DO(SH_Index.create_pool(SH_Indices_Pool, 0666, SH_Index));
    SH_qm = new(SH_Index) SH_Query_Manager;
    SH_qm.update()->initialize();
  }
  else {
    SH_DO(SH_Features.lookup(SH_Feature_Pool,SH_Features ));
    SH_DO(SH_Geometries.lookup(SH_Geometry_Pool, SH_Geometries));
    SH_DO(SH_Index.lookup(SH_Indices_Pool, SH_Index));
    //Scans Indices_pool to find index-object
    PoolScan scan(SH_Index);
    if (scan.rc() != RCOK)
      SH_ABORT_TRANSACTION(scan.rc());
    while (TYPE_OBJECT(SH_Query_Manager).isa(ref) == NULL &&
   scan.rc() == RCOK) {
      scan.next(ref);
    };
    if (scan.rc() != RCOK)
      SH_ABORT_TRANSACTION(scan.rc());
    SH_qm = TYPE_OBJECT(SH_Query_Manager).isa(ref);
  }
  SH_DO(SH_COMMIT_TRANSACTION);
  opened = TRUE;
  // We are ready to run..
  return OK;
}
long SHDB::CreateData(long noOfObjects) {
  long okey;
  if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  // Start a transaction for initialization
  shrc rc;
  SH_BEGIN_TRANSACTION(rc);
  if (rc) {
    return NOTOK;
  }
   okey = SH_qm.update()->loadTestFig(noOfObjects); // Parameter = #Feature
objects
  // That should be all
  SH_DO(SH_COMMIT_TRANSACTION);
  if (!okey)
```

```
    return NOTOK;
  return OK;
}

long SHDB::GetData(Geometry& inC_gm) {
  //  long okey;
  if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  shrc rc;
  SH_BEGIN_TRANSACTION(rc);
  if (rc) {
    return NOTOK;
  }
  //  okey = SH_qm->getGeometry(inC_gm);
  SH_qm->getGeometry(inC_gm);
  SH_DO(SH_COMMIT_TRANSACTION);
  //   if (!okey)
  //     return NOTOK;
  return OK;
}


long SHDB::DeleteData() {
   if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  // remove some data from Shore:
  shrc rc;
  SH_BEGIN_TRANSACTION(rc);
  if (rc) {
    return NOTOK;
  }
  // Correct way to delete content of a pool
  W_IGNORE(SH_Features.destroy_contents());
  W_IGNORE(Shore::unlink(SH_Feature_Pool));
  SH_DO(SH_COMMIT_TRANSACTION);
  SH_BEGIN_TRANSACTION(rc);
  if (rc) {
    return NOTOK;
  }
  W_IGNORE(SH_Geometries.destroy_contents());
  W_IGNORE(Shore::unlink(SH_Geometry_Pool));
  SH_DO(SH_COMMIT_TRANSACTION);
  SH_BEGIN_TRANSACTION(rc);
  if (rc) {
    return NOTOK;
  }
  W_IGNORE(SH_Index.destroy_contents());
  W_IGNORE(Shore::unlink(SH_Indices_Pool));
  SH_DO(SH_COMMIT_TRANSACTION);
  SH_BEGIN_TRANSACTION(rc);
  if (rc) {
    return NOTOK;
  }
  W_IGNORE(Shore::chdir(ROOT));
  W_IGNORE(Shore::rmdir(SH_ROOT));
  SH_DO(SH_COMMIT_TRANSACTION);
  CloseDB();
  return InitDB();
}
```

```
long SHDB::CloseDB() {
  if (!opened) {
      return OK;
  }
  SH_DO(Shore::exit());
  opened = FALSE;
  return OK;
}

long SHDB::AreaDefs() {
  if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  shrc rc;
  SH_BEGIN_TRANSACTION(rc);
  if (rc) {
    return NOTOK;
  }
  aSpaceRef = SH_qm->getIndex()->findRefsys(SpaceRef);
  if (aSpaceRef) {
    aSpaceRef->getMin(frame->ymin,frame->xmin);
    aSpaceRef->getMax(frame->ymax,frame->xmax);
    aSpaceRef->getOrigo(frame->origoy, frame->origox);
    frame->db_enhet = aSpaceRef->getLinear_unit();
    frame->xdomain = abs(frame->xmax - frame->xmin);
    frame->ydomain = abs(frame->ymax - frame->ymin);
    frame->xzoom = (float)MAXCANVAS / frame->xdomain;
    frame->yzoom = (float)MAXCANVAS / frame->ydomain;
    frame->dx = abs(frame->origox - frame->xmin);
    frame->dy = abs(frame->origoy - frame->ymin);
  }
  SH_DO(SH_COMMIT_TRANSACTION);
  return OK;
}

long SHDB::FillListBox(wxListBox* theList) {
  if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  return fillListBox(theList);
}

long SHDB::LoadSOSI(char *aFile) {
  FILE *m_FP;
  char Line[80];
  char *param1 = new char[80];
  char *param2 = new char[80];
  char *param3 = new char[80];
  char *param4 = new char[80];
  char *dots = new char[80];
  char *featureName = “aFeature”;
  char *streng = new char[80];
  char *name = new char[80];
  int theme = 0;
  float date = 0;
  reverse = FALSE;
  KP = FALSE;
  A_Point SH_pt;
  // Coordinate-based variables
  Coordinate C_pt;
  LineString C_ls;
  Ring       C_ri;
```

```
  Polygon    C_pg;
  elTotal=elPrevious=elCurrent=0;
  ocSpaceRef=ocPunkt=ocKurve=ocLinje=
    ocTekst= ocFlate= ocInxNew= ocInxFind=
    ocAccuracy= ocGeomP= ocPoint= ocLineString=
    ocPolygon=0;
  KP = FALSE;
  //  CoordGeom  C_cg;
  //  Geometry   C_gm;
  shrc rc;
       frame->text_window->SetInsertionPoint(frame->text_window->GetLastPosi-
tion());
  wxStartTimer();
  float origoN=0, origoOE=0, linear_unit=0, maxN=0, maxOE=0,
    minN=0, minOE=0, measureMethod=0, accuracy=0, zone=0;;
  // Åpner og sjekker om filen er åpnet
  m_FP = fopen(aFile,"r");
  if (m_FP != NULL)
      fgets(Line,80,m_FP);
  while(!feof(m_FP)) {
    strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
    strcpy(param3,"\0"); strcpy(param4,"\0");
    sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);

    if (!strcmp(param1,"HODE")) {
      elPrevious = wxGetElapsedTime(FALSE);
      fgets(Line,80,m_FP);
      strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
      strcpy(param3,"\0"); strcpy(param4,"\0");
      sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      while (strstr(dots,"..")) {
if (strstr(param1,"ORIGO-NØ")) {
  StringToFloat(param2,&origoN);
  StringToFloat(param3,&origoOE);
}
else if (strstr(param1,"ENHET")) {
  StringToFloat(param2,&linear_unit);
}
else if (strstr(param1,"KOORDSYS")) {
  StringToFloat(param2,&zone);
}
else if (strstr(param1,"MAX-NØ")) {
  StringToFloat(param2,&maxN);
  StringToFloat(param3,&maxOE);
}
else if (strstr(param1,"MIN-NØ")) {
  StringToFloat(param2,&minN);
  StringToFloat(param3,&minOE);
}
else if (strstr(param1,"KVALITET")) {
  StringToFloat(param2,&measureMethod);
  StringToFloat(param3,&accuracy);
}
else if (strstr(param1,"DATO")) {
  StringToFloat(param2,&date);
}
fgets(Line,80,m_FP);
strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
strcpy(param3,"\0"); strcpy(param4,"\0");
sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      }
      strcpy(name,"\0");
      strcat(name,FloatToString(zone));
      strcat(name,FloatToString(origoN));
```

```
      strcat(name,FloatToString(origoOE));
      strcat(name,FloatToString(maxN));
      strcat(name,FloatToString(maxOE));
      strcat(name,FloatToString(minN));
      strcat(name,FloatToString(minOE));
      strcat(name,FloatToString(date));
      // After this, name is reasonable unique for my files...
      SH_BEGIN_TRANSACTION(rc);
      if (rc)
rc.fatal(); // terminates program
      // check if this refSys already exists
      aSpaceRef = SH_qm->getIndex()->findRefsys(name); ocInxFind++;
      if (!aSpaceRef) {
// Create Spatial Reference System; some oversimplifying here
aSpaceRef = new(SH_Features) SH_Spatial_Ref_System; ocSpaceRef++;
aSpaceRef.update()->initialize(name);
aSpaceRef.update()->setLinear_unit(linear_unit);
aSpaceRef.update()->setZone((long)zone);
aSpaceRef.update()->setOrigo((long)origoN, (long)origoOE);
aSpaceRef.update()->setMax((long)maxN, (long)maxOE);
aSpaceRef.update()->setMin((long)minN, (long)minOE);
strcpy(SpaceRef,name);
      }
      anAccuracy = new(SH_Features) SH_Accuracy; ocAccuracy++;
      anAccuracy.update()->initialize((long)accuracy, (long)measureMethod);
      SH_DO(SH_COMMIT_TRANSACTION);
      elCurrent = wxGetElapsedTime(FALSE);
      *frame->text_window << "Elapsed time, Hode(spaceref+accuracy): ";
      *frame->text_window << elCurrent - elPrevious;
      *frame->text_window << "\n";
      continue;
    }
    if (!strcmp(param1,"PUNKT")) {
      elPrevious = wxGetElapsedTime(FALSE);
      SH_BEGIN_TRANSACTION(rc);
      if (rc)
rc.fatal(); // terminates program
      strcpy(streng,"\0");
      sscanf(param2,"%i:",&serialNo);
      // read line-params
      strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
      fgets(Line,80,m_FP);
      sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      while (strcmp(dots,"\0")) {
if (!strcmp(param1,"PTEMA")) {
  sscanf(param2,"%i",&theme);
}
strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
fgets(Line,80,m_FP);
sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      }
      aFeature = new(SH_Features) SH_Feature; ocPunkt++;
        aFeature.update()->initialize(featureName,"PUNKT",(long)theme,"someS-
tuff");
      aGeomP = new(SH_Features) SH_Geometric_Property; ocGeomP++;
      aGeomP.update()->spaceRef = aSpaceRef;
      aGeomP.update()->feature = aFeature;
      while (!strcmp(dots,"\0")) {
xx = yy = zz = 0; KP = FALSE;
if (strstr(Line,"...KP")) // If this point is "KNUTEPUNKT"
  KP = TRUE;
sscanf(Line,"%f %f %f",&xx,&yy,&zz);
SH_pt.east = (long)xx;
SH_pt.north = (long)yy;
```

```
SH_pt.height = (long)zz;
SH_pt.date = (long)date;
strcpy(dots,”\0”);
fgets(Line,80,m_FP);
sscanf(Line,”%[.]%s %s %s %s”,dots,param1, param2, param3, param4);
      }
      aPoint = SH_qm->getIndex()->findPoint(SH_pt); ocInxFind++;
      if (!aPoint) {
aPoint = new(SH_Geometries) SH_Point; ocPoint++;
aPoint.update()->initialize(&SH_pt,anAccuracy);
      }
      if (KP)
aPoint.update()->KP = TRUE;
      aPoint.update()->geomProperty = aGeomP;
      SH_qm->getIndex()->newSerialNo((long)serialNo, aPoint); ocInxNew++;
      SH_DO(SH_COMMIT_TRANSACTION);
      elCurrent = wxGetElapsedTime(FALSE);
      *frame->text_window << “Elapsed time, Punkt: “;
      *frame->text_window  << serialNo;
      *frame->text_window << “ time: “;
      *frame->text_window  << elCurrent - elPrevious;
      *frame->text_window  << “\n”;
      theme = serialNo = 0;
      continue;
    }
    if (!strcmp(param1,”KURVE”)) {
      elPrevious = wxGetElapsedTime(FALSE);
      SH_BEGIN_TRANSACTION(rc);
      if (rc)
rc.fatal(); // terminates program
      strcpy(streng,”\0”);
      sscanf(param2,”%i:”,&serialNo);
      // read line-params
      strcpy(dots,”\0”); strcpy(param1,”\0”); strcpy(param2,”\0”);
      fgets(Line,80,m_FP);
      sscanf(Line,”%[.]%s %s %s %s”,dots,param1, param2, param3, param4);
      while (strcmp(dots,”\0”)) {
if (!strcmp(param1,”LTEMA”)) {
  sscanf(param2,”%i”,&theme);
}
strcpy(dots,”\0”); strcpy(param1,”\0”); strcpy(param2,”\0”);
fgets(Line,80,m_FP);
sscanf(Line,”%[.]%s %s %s %s”,dots,param1, param2, param3, param4);
      }
      aFeature = new(SH_Features) SH_Feature; ocKurve++;
        aFeature.update()->initialize(featureName,”KURVE”,(long)theme,”someS-
tuff”);
      aGeomP = new(SH_Features) SH_Geometric_Property; ocGeomP++;
      aGeomP.update()->feature = aFeature;
      aGeomP.update()->spaceRef = aSpaceRef;
      aLineString = new(SH_Geometries) SH_Line_String; ocLineString++;
      aLineString.update()->initialize(anAccuracy);
      aLineString.update()->geomProperty = aGeomP;
         SH_qm->getIndex()->newSerialNo((long)serialNo,  aLineString);  ocInx-
New++;

                  while   (!strcmp(dots,”\0”)   ||   !strcmp(param1,”NØ”)   ||
!strcmp(param1,”NØH”)) {
if (!strcmp(param1,”NØ”) || !strcmp(param1,”NØH”)) {
  strcpy(dots,”\0”); strcpy(param1,”\0”);
  fgets(Line,80,m_FP);
  sscanf(Line,”%[.]%s %s %s %s”,dots,param1, param2, param3, param4);
  continue;
}
```

```
xx = yy = zz = 0; KP = FALSE;
if (strstr(Line,"...KP")) // If this point is "KNUTEPUNKT"
  KP = TRUE;
sscanf(Line,"%f %f %f",&xx,&yy,&zz);
SH_pt.east = (long)xx;
SH_pt.north = (long)yy;
SH_pt.height = (long)zz;
SH_pt.date = (long)date;
aPoint = SH_qm->getIndex()->findPoint(SH_pt); ocInxFind++;
if (!aPoint) {
  aPoint = new(SH_Geometries) SH_Point; ocPoint++;
  aPoint.update()->initialize(&SH_pt,anAccuracy);
}
if (KP)
    aPoint.update()->KP = TRUE;
aPoint.update()->partOf.add(aLineString);
strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
fgets(Line,80,m_FP);
sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      }
      SH_DO(SH_COMMIT_TRANSACTION);
      elCurrent = wxGetElapsedTime(FALSE);
      *frame->text_window << "Elapsed time, Kurve: ";
      *frame->text_window  << serialNo;
      *frame->text_window << " time: ";
      *frame->text_window  << elCurrent - elPrevious;
      *frame->text_window  << "\n";
      theme = serialNo = 0;
      date = 991122;
      continue;
    }
    if (!strcmp(param1,"LINJE")) {
      elPrevious = wxGetElapsedTime(FALSE);
      SH_BEGIN_TRANSACTION(rc);
      if (rc)
rc.fatal(); // terminates program
      sscanf(param2,"%i:",&serialNo);
      // read line-params
      strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
      fgets(Line,80,m_FP);
      sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      while (strcmp(dots,"\0")) {
if (!strcmp(param1,"LTEMA")) {
  sscanf(param2,"%i",&theme);
}
strcpy(dots,"\0"); strcpy(param1,"\0"); fgets(Line,80,m_FP);
sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      }
      aFeature = new(SH_Features) SH_Feature; ocLinje++;
        aFeature.update()->initialize(featureName,"LINJE",(long)theme,"someS-
tuff");
      aGeomP = new(SH_Features) SH_Geometric_Property; ocGeomP++;
      aGeomP.update()->feature = aFeature;
      aGeomP.update()->spaceRef = aSpaceRef;
      aLineString = new(SH_Geometries) SH_Line_String; ocLineString++;
      aLineString.update()->initialize(anAccuracy);
      aLineString.update()->geomProperty = aGeomP;
         SH_qm->getIndex()->newSerialNo((long)serialNo,  aLineString);  ocInx-
New++;

                  while   (!strcmp(dots,"\0")   ||   !strcmp(param1,"NØ")   ||
!strcmp(param1,"NØH")) {
if (!strcmp(param1,"NØ") || !strcmp(param1,"NØH")) {
  strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
```

```
  fgets(Line,80,m_FP);
  sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
  continue;
}
xx = yy = zz = 0; KP = FALSE;
if (strstr(Line,"...KP")) // If this point is "KNUTEPUNKT"
  KP = TRUE;
sscanf(Line,"%f %f %f",&xx,&yy,&zz);
SH_pt.east = (long)xx;
SH_pt.north = (long)yy;
SH_pt.height = (long)zz;
SH_pt.date = (long)date;
aPoint = SH_qm->getIndex()->findPoint(SH_pt); ocInxFind++;
if (!aPoint) {
  aPoint = new(SH_Geometries) SH_Point; ocPoint++;
  aPoint.update()->initialize(&SH_pt,anAccuracy);
}
if (KP)
  aPoint.update()->KP = TRUE;
aPoint.update()->partOf.add(aLineString);
strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
fgets(Line,80,m_FP);
sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      }
      SH_DO(SH_COMMIT_TRANSACTION);
      elCurrent = wxGetElapsedTime(FALSE);
      *frame->text_window << "Elapsed time, Linje: ";
      *frame->text_window  << serialNo;
      *frame->text_window << " time: ";
      *frame->text_window  << elCurrent - elPrevious;
      *frame->text_window  << "\n";
      theme = serialNo = 0;
      date = 991122;
      continue;
    }
    if (!strcmp(param1,"BUE")) {
      //      cout <<"Bue"; // This won't be in my database.....
      fgets(Line,80,m_FP);
      continue;
    }
    if (!strcmp(param1,"TEKST")) {
      elPrevious = wxGetElapsedTime(FALSE);
      SH_BEGIN_TRANSACTION(rc);
      if (rc)
rc.fatal(); // terminates program
      sscanf(param2,"%i:",&serialNo);
      // read line-params
      strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
      fgets(Line,80,m_FP);
      sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      while (strcmp(dots,"\0")) {
if (!strcmp(param1,"TTEMA")) {
  sscanf(param2,"%i",&theme);
}
else
  if (!strcmp(param1,"STRENG")) {
    strcpy(param2,"\0");
    sscanf(Line,"%[.]%s %[\"]%[A-Za-zæøåÆØÅ ]",dots,param1,param2,param3);
    if (!strcmp(param2,"\0"))
      sscanf(Line,"%[.]%s %s",dots,param1,param2);
    strcpy(streng,param2);
}
strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
fgets(Line,80,m_FP);
```

```
sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      }
      aFeature = new(SH_Features) SH_Feature; ocTekst++;
      aFeature.update()->initialize(featureName,"TEKST",(long)theme,streng);
      aGeomP = new(SH_Features) SH_Geometric_Property; ocGeomP++;
      aGeomP.update()->feature = aFeature;
      aGeomP.update()->spaceRef = aSpaceRef;
      int i = 0;
      while (!strcmp(dots,"\0")) {
i++;
if (i<2) {
  xx = yy = zz = 0; KP = FALSE;
  if (strstr(Line,"...KP")) // If this point is "KNUTEPUNKT"
    KP = TRUE;
  sscanf(Line,"%f %f %f",&xx,&yy,&zz);
  SH_pt.east = (long)xx;
  SH_pt.north = (long)yy;
  SH_pt.height = (long)zz;
  SH_pt.date = (long)date;
  aPoint = SH_qm->getIndex()->findPoint(SH_pt); ocInxFind++;
  if (!aPoint) {
    aPoint = new(SH_Geometries) SH_Point; ocPoint++;
    aPoint.update()->initialize(&SH_pt,anAccuracy);
  }
  if (KP)
    aPoint.update()->KP = TRUE;
  aPoint.update()->geomProperty = aGeomP;
  SH_qm->getIndex()->newSerialNo((long)serialNo, aPoint); ocInxNew++;
}
strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
fgets(Line,80,m_FP);
sscanf(Line,"%[.]%s %s %s %s",dots,param1, param2, param3, param4);
      }
      SH_DO(SH_COMMIT_TRANSACTION);
      elCurrent = wxGetElapsedTime(FALSE);
      *frame->text_window << "Elapsed time, Tekst: ";
      *frame->text_window  << serialNo;
      *frame->text_window << " time: ";
      *frame->text_window  << elCurrent - elPrevious;
      *frame->text_window  << "\n";
      theme = serialNo = 0;
      continue;
    }
    if (!strcmp(param1,"FLATE")) {
      elPrevious = wxGetElapsedTime(FALSE);
      SH_BEGIN_TRANSACTION(rc);
      if (rc)
rc.fatal(); // terminates program
      sscanf(param2,"%i:",&serialNo);
      // get the theme-number
      strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
      fgets(Line,80,m_FP);
      sscanf(Line,"%[.]%s %s",dots,param1, param2);
      if (!strcmp(param1,"FTEMA")) {
sscanf(param2,"%i",&theme);
      }
      aFeature = new(SH_Features) SH_Feature; ocFlate++;
        aFeature.update()->initialize(featureName,"FLATE",(long)theme,"somes-
tuff");
      aGeomP = new(SH_Features) SH_Geometric_Property; ocGeomP++;
      aGeomP.update()->spaceRef = aSpaceRef;
      aGeomP.update()->feature = aFeature;
      aPolygon = new(SH_Geometries) SH_Polygon; ocPolygon++;
      aPolygon.update()->initialize(anAccuracy);
```

```
      aPolygon.update()->geomProperty = aGeomP;
      SH_qm->getIndex()->newSerialNo((long)serialNo, aPolygon); ocInxNew++;
      // Find and make refs to line-strings.
      strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
      fgets(Line,80,m_FP);
      sscanf(Line,"%[. :]%[A-Za-z0-9: ()-]",dots,param1);
      while (!strcmp(dots,".. :")) {
reverse = FALSE;
theSerialNo = (int)aSerialNo(param1);
while (theSerialNo != 0) {
  // read serialNo's referring to lineStrings
  // Create a Ring, containing all points in the referred lineStrings
  if (theSerialNo < 0) {
    reverse = TRUE;
    theSerialNo = -1*theSerialNo;
  }
  ref = SH_qm->getIndex()->findSerialNo((long)theSerialNo); ocInxFind++;
  aLineString = TYPE_OBJECT(SH_Line_String).isa(ref);
  if (aLineString) {
    aLineString.update()->partOf.add(aPolygon);
  }
  // some resettings and then we go again
  reverse = FALSE;
  theSerialNo = (int)aSerialNo(param1);
}
strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
fgets(Line,80,m_FP);
sscanf(Line,"%[. :]%[A-Za-z0-9: ()-]",dots,param1);
      }

      strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
      // if NØ or (:
      // catch a param like this: .. (:1234)
      sscanf(Line,"%[. (:]%s)",dots,param1);
      if (!strcmp(dots,".. (:")) {
theSerialNo = (int)aSerialNo(param1);
while (theSerialNo != 0) {
  ref = SH_qm->getIndex()->findSerialNo(theSerialNo); ocInxFind++;
  anotherPolygon = TYPE_OBJECT(SH_Polygon).isa(ref);
  if (anotherPolygon) {
    anotherPolygon.update()->partOf = aPolygon;
  }
  theSerialNo = (int)aSerialNo(param1);
}
strcpy(dots,"\0"); strcpy(param1,"\0"); strcpy(param2,"\0");
fgets(Line,80,m_FP);
      }
      // If the last param is a point associated with FLATE
      strcpy(dots,"\0");
      strcpy(param1,"\0");
      sscanf(Line,"%[.]%s",dots,param1);
      if (!strcmp(param1,"NØ") || !strcmp(param1,"NØH")) {
xx = yy = zz = 0; KP = FALSE;
fgets(Line,80,m_FP);
if (strstr(Line,"...KP")) // If this point is "KNUTEPUNKT"
  KP = TRUE;
sscanf(Line,"%f %f %f",&xx,&yy,&zz);
SH_pt.east = (long)xx;
SH_pt.north = (long)yy;
SH_pt.height = (long)zz;
SH_pt.date = (long)date;
aPoint = SH_qm->getIndex()->findPoint(SH_pt); ocInxFind++;
if (!aPoint) {
  aPoint = new(SH_Geometries) SH_Point; ocPoint++;
```

```
  aPoint.update()->initialize(&SH_pt,anAccuracy);
}
if (KP)
  aPoint.update()->KP = TRUE;
aPoint.update()->SH_Geometry::partOf = aPolygon;
C_pt.set_size(0);
fgets(Line,80,m_FP);
      }
      SH_DO(SH_COMMIT_TRANSACTION);
      elCurrent = wxGetElapsedTime(FALSE);
      *frame->text_window << “Elapsed time, Flate: “;
      *frame->text_window  << serialNo;
      *frame->text_window << “ time: “;
      *frame->text_window  << elCurrent - elPrevious;
      *frame->text_window  << “\n”;
      theme = serialNo = 0;
      date = 991122;
      continue;
    }
    if (!strcmp(param1,”SLUTT”)) {
      break;
    }
    fgets(Line,80,m_FP);
  }

  fclose(m_FP);
  delete param1;
  delete param2;
  delete param3;
  delete param4;
  delete dots;
  delete streng;
  delete featureName;
  delete name;
  SH_BEGIN_TRANSACTION(rc);
  if (rc)
    rc.fatal(); // terminates program
  SH_qm->getIndex()->killAllSerialNoInx();
  SH_DO(SH_COMMIT_TRANSACTION);
  elTotal = wxGetElapsedTime(TRUE);
  *frame->text_window << “Total elapsed time: “;
  *frame->text_window  << elTotal;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of SpaceRef-objs: “;
  *frame->text_window  << ocSpaceRef;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of Accuracy-objs: “;
  *frame->text_window  << ocAccuracy;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of GeomProp-objs: “;
  *frame->text_window  << ocGeomP;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of Point-objs: “;
  *frame->text_window  << ocPoint;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of LineString-objs: “;
  *frame->text_window  << ocLineString;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of Polygon-objs: “;
  *frame->text_window  << ocPolygon;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of Feature-PUNKT: “;
  *frame->text_window  << ocPunkt;
  *frame->text_window  << “\n”;
```

```
  *frame->text_window << “Number of Feature-KURVE: “;
  *frame->text_window  << ocKurve;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of Feature-LINJE: “;
  *frame->text_window  << ocLinje;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of Feature-TEKST: “;
  *frame->text_window  << ocTekst;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of Feature-FLATE: “;
  *frame->text_window  << ocFlate;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of new Inx-elements: “;
  *frame->text_window  << ocInxNew;
  *frame->text_window  << “\n”;
  *frame->text_window << “Number of Inx-lookups: “;
  *frame->text_window  << ocInxFind;
  *frame->text_window  << “\n”;
  *frame->text_window << “Goodbye\n”;
  return OK;
}

long SHDB::getThemes(wxListBox& theList,wxDC& dc) {
  // Find #elements in listbox
  long rc;
  float themecode;
  long code;
  for (int i=0; i < theList.Number(); i++) {
  // Lookup the themecodes in index and draw them
    StringToFloat(theList.GetString(i), &themecode);
    code = (long)themecode;
    rc = theme(code,dc);
    if (rc == NOTOK)
      return NOTOK;
  }
  return OK;
}

// Fills a given listbox with Space Ref’s found in the index
long fillListBox(wxListBox* theList) {
  shrc rc;
  SH_BEGIN_TRANSACTION(rc);
  if (rc)
    return NOTOK;
  // Scan all spaceRef’s, put them into the listbox
  IndexScanIter<sdl_string,Ref<SH_Spatial_Ref_System> >
    iter(SH_qm->getIndex()->aSpaceRefInd);
  SH_DO(iter.next());
  for ( ; !iter.eof; SH_DO(iter.next())) {
    theList->Append(iter.cur_val->metaData(“Name”));
  }
  SH_DO(SH_COMMIT_TRANSACTION);
  return OK;
}

long theme(long& themeCode, wxDC& dc) {
  Ref<any> ref;
  Ref<SH_Point> aPoint;
  Ref<SH_Line_String> aLineString;
  Ref<SH_Polygon> aPolygon;
  CoordGeom C_cg;
  shrc rc;
  SH_BEGIN_TRANSACTION(rc);
  if (rc)
```

```
    return NOTOK;
  // Scan all spaceRef's, put them into the listbox
  IndexScanIter<long,Ref<SH_Feature> >
    iter(SH_qm->getIndex()->themeInd);
  iter.SetLB(themeCode);
  iter.SetLowerCond(eqOp);
  iter.SetUB(themeCode);
  iter.SetUpperCond(eqOp);
  SH_DO(iter.next());
  for ( ; !iter.eof; SH_DO(iter.next())) {
    if (iter.cur_key != themeCode)
      continue;
    if (strcmp(iter.cur_val->geomProperty->spaceRef->metaData("Name"),
 SpaceRef))
      continue;
    ref = iter.cur_val->geomProperty->geometry;

    if (!strcmp(iter.cur_val->Description,"PUNKT")) {
      aPoint = TYPE_OBJECT(SH_Point).isa(ref);
      if (aPoint) {
  aPoint->getCoordinateGeometry(C_cg);
  frame->drawGeometry(dc,C_cg);
  continue;
      }
    }
    if (!strcmp(iter.cur_val->Description,"KURVE") ||
!strcmp(iter.cur_val->Description,"LINJE")) {
      aLineString = TYPE_OBJECT(SH_Line_String).isa(ref);
      if (aLineString) {
aLineString->getCoordinateGeometry(C_cg);
frame->drawGeometry(dc,C_cg);
continue;
      }
    }
    if (!strcmp(iter.cur_val->Description,"TEKST")) {
      aPoint = TYPE_OBJECT(SH_Point).isa(ref);
      if (aPoint) {
  aPoint->getCoordinateGeometry(C_cg);
  frame->drawText(dc,C_cg,iter.cur_val->someStuff);
  continue;
      }
    }
    if (!strcmp(iter.cur_val->Description,"FLATE")) {
aPolygon = TYPE_OBJECT(SH_Polygon).isa(ref);
if (aPolygon) {
    aPolygon->getCoordinateGeometry(C_cg);
    frame->drawGeometry(dc,C_cg);
    continue;
}
    }
  }
  SH_DO(SH_COMMIT_TRANSACTION);
  return OK;
}
```

Vedlegg J Databaselag; PGDB.h

```
// ****************************************************************
// File name: PGDB.h
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Headerfile for Postgres handling
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 04.01.97 Created
// ****************************************************************
#ifndef PGDB_HDR
#define PGDB_HDR
#include “libpq++.H”
#include “MyDB.h”
class PGDB : public MyDB {
  public:
  char* dbName;
  int i,j;
  char pgquery[500];
  PGenv env;
  PGdatabase* data;
  PGDB() {;}
  // Initialize connection to server:
  long InitDB();
  // Create a number of polygons:
  long CreateData(long noOfObjects);
  // Collect according to current selections in database:
  long GetData(Geometry& inC_gm);
  // Delete data based on selections in frame:
  long DeleteData(){
    cout << “Not implemened yet\n;”;
    return OK;
  }
  // Read SOSI-file and store in database
  long LoadSOSI(char* aFile){
    // Just to kill some compile-warnings!
    if (aFile) {
      cout << “Not implemened yet\n;”;
      return OK;
    }
    return OK;
  }
  // Close connection to server:
  long CloseDB();
  // Fill a listbox with all SpaceRef’s in database
  long FillListBox(wxListBox* theList);
  // Select a SpaceRef, and get some initial settings
  long AreaDefs();
  // Find and draw according to given themes
  long getThemes(wxListBox& theList, wxDC& dc) {
    cout << “Not implemened yet\n;”;
    return OK;
```

```
  }
};
// end PGDB_HDR
#endif
```

Vedlegg K Databaselag; PGDB.C

```
// ****************************************************************
// File name: PGDB.C
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Methods on Postgres database
// Class-library of wxWindows is used with permission.
// The Postgres databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 26.11.96 Created
// ****************************************************************
#include <ctype.h>
#include “wx.h”
#include “SH_Objects.h”
#include “PG_Objects.h”
#include “GO.h”
#include “PGDB.h”
// Some externals
extern MyFrame* frame;
extern char* SpaceRef;
PG_Query_Manager PG_qm;

long PGDB::InitDB() {
  dbName = “GO”;
  /* make a connection to the database */
  data = new PGdatabase(&env, dbName);
  // check to see that the backend connection was successfully made
  if (data->status() == CONNECTION_BAD) {
    *frame->text_window << “Connection to “;
    *frame->text_window << dbName;
    *frame->text_window << “database failed.\n”;
    *frame->text_window << data->errormessage();
    *frame->text_window << “\n”;
    delete data;
    return NOTOK;
  }
  return OK;
}

// Create a number of polygons:
long PGDB::CreateData(long noOfObjects){
  long okey;
  if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  // start a transaction block
//   if (data->exec(“BEGIN”) != PGRES_COMMAND_OK) {
//     *frame->text_window << “BEGIN command failed\n”;
//     CloseDB();
//     return NOTOK;
//   }
```

```
  // Okey here starts the trouble..............
  okey = PG_qm.loadTestFig(noOfObjects); // Parameter = #Feature objects
  // end the transaction
//   data->exec("END");
  return okey;
}

long PGDB::AreaDefs() {
  if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  // start a transaction block
  if (data->exec("BEGIN") != PGRES_COMMAND_OK) {
    *frame->text_window << "BEGIN command failed\n";
    CloseDB();
    return NOTOK;
  }
  float lu;
  int on, oe, xn, xe, nn, ne;
  // fetch instances from Spatial_Ref_System
  sprintf(pgquery,"SELECT linear_unit, origoN, origoOE,
    maxN, maxOE, minN, minOE FROM Spatial_Ref_System
    WHERE SP_name = '%s'",SpaceRef);
  switch (data->exec(pgquery)) {
    case PGRES_TUPLES_OK:
      StringToFloat(data->getvalue(0,0),&lu);
      StringToInt(data->getvalue(0,1),&on);
      StringToInt(data->getvalue(0,2),&oe);
      StringToInt(data->getvalue(0,3),&xn);
      StringToInt(data->getvalue(0,4),&xe);
      StringToInt(data->getvalue(0,5),&nn);
      StringToInt(data->getvalue(0,6),&ne);
      frame->db_enhet = lu;
      frame->origoy = on;
      frame->origox = oe;
      frame->ymax = xn;
      frame->xmax = xe;
      frame->ymin = nn;
      frame->xmin = ne;
      break;
    default:
      *frame->text_window << "Select Spatial_Ref_System(1).SP_name failed\n";
      CloseDB();
      return NOTOK;
  }
  frame->xdomain = abs(frame->xmax - frame->xmin);
  frame->ydomain = abs(frame->ymax - frame->ymin);
  frame->xzoom = (float)MAXCANVAS / frame->xdomain;
  frame->yzoom = (float)MAXCANVAS / frame->ydomain;
  frame->dx = abs(frame->origox - frame->xmin);
  frame->dy = abs(frame->origoy - frame->ymin);
  // end the transaction
  data->exec("END");
  return OK;
}

long PGDB::GetData(Geometry& inC_gm) {
  //  long okey;
  if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  // start a transaction block
```

```
  if(data->exec("BEGIN") != PGRES_COMMAND_OK) {
    *frame->text_window << "BEGIN command failed\n";
    CloseDB();
    return NOTOK;
  }
  PG_qm.getGeometry(inC_gm);
  // end the transaction
  data->exec("END");
  return OK;
}


// Close connection to server:
long PGDB::CloseDB(){
  // close the connection to the database and cleanup
  if (!opened) {
      return OK;
  }
  delete data;
  opened = FALSE;
  return OK;
}
// Find all SpaceRef's and display in theList...
long PGDB::FillListBox(wxListBox* theList) {
  if (!opened) {
    if (!InitDB())
      return NOTOK;
  }
  // start a transaction block
  if(data->exec("BEGIN") != PGRES_COMMAND_OK) {
    *frame->text_window << "BEGIN command failed\n";
    CloseDB();
    return NOTOK;
  }
  // fetch instances from Spatial_Ref_System
  if (data->exec("SELECT SP_name FROM Spatial_Ref_System")
      != PGRES_TUPLES_OK) {
    *frame->text_window << "Select Spatial_Ref_System(1).SP_name failed\n";
    CloseDB();
    return NOTOK;
  }
  for (int i=0; i < data->ntuples(); i++)
    theList->Append(data->getvalue(i,0));
  // end the transaction
  data->exec("END");
  return OK;
}
```

Vedlegg L Metodeimplementasjon; SH_defs.C

Denne filen genererer metoder for datadefinisjonene i SH_Objects.h

```
// ************************************************************
// File name: SH_defs.C
// Project: GO
// Copyright: IDT, NTNU
// Language used: SDL (Shore Data Language)
// Short description: Datatype definitions for all objects
// to be stored in the database
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 22.11.96 Created
// ************************************************************
#define MODULE_CODE
#include “SH_Objects.h”
```

Vedlegg M Metodeimplementasjon; SH_Objects.h

Denne headerfilen er generert av Shore's prekompilator fra SH_Objects.sdl:

```
#ifndef SH_Objects_mod
#define SH_Objects_mod 1
#ifndef WKS_mod
#define WKS_mod 1
#include "ShoreApp.h"
class sdlModule_SH_Objects { public: virtual int oid_20297(); };
static sdlModule_SH_Objects SH_Objects_header_version;
class sdlModule_WKS { public: virtual int oid_20051(); };
static sdlModule_WKS WKS_header_version;
INTERFACE_PREDEFS( SH_Query_Manager);
INTERFACE_PREDEFS( SH_Feature);
INTERFACE_PREDEFS( SH_Accuracy);
INTERFACE_PREDEFS( SH_Geometric_Property);
INTERFACE_PREDEFS( SH_Spatial_Ref_System);
INTERFACE_PREDEFS( SH_Geometry);
INTERFACE_PREDEFS1(SH_Point,SH_Geometry);
INTERFACE_PREDEFS1(SH_Curve,SH_Geometry);
INTERFACE_PREDEFS1(SH_Surface,SH_Geometry);
INTERFACE_PREDEFS1(SH_Line_String,SH_Curve);
INTERFACE_PREDEFS1(SH_Polygon,SH_Surface);
INTERFACE_PREDEFS( SH_Indices);
struct Distance {
 long meters ;
 long time ;
void __apply(HeapOps) {}
static void sdl_apply(void *p,HeapOps op);
};
;
struct A_Point {
 long east ;
 long north ;
 long height ;
 long date ;
void __apply(HeapOps) {}
static void sdl_apply(void *p,HeapOps op);
};
;
enum InterpretationType {
GO_Arc ,
GO_Ellipse ,
GO_Icon ,
GO_Line ,
GO_Lines ,
GO_Polygon ,
GO_Point ,
GO_Rectangle ,
GO_RoundedRectangle ,
GO_Spline ,
GO_Text };
```

```
OVERRIDE_INDVAL(InterpretationType)
NO_APPLY(InterpretationType)
;
typedef Sequence< Ref< void  >    > Refs_any;
 class SH_Query_Manager ;
 class SH_Feature ;
 class SH_Accuracy ;
 class SH_Geometric_Property ;
 class SH_Spatial_Ref_System ;
 class SH_Geometry ;
 class SH_Point ;
 class SH_Curve ;
 class SH_Surface ;
 class SH_Line_String ;
 class SH_Polygon ;
class SH_Query_Manager  : public sdlObj {
COMMON_FCT_DECLS(SH_Query_Manager)
private:
 Ref< SH_Indices  >  inx ;
virtual  void  createRing ( LREF(Sequence< long    > )  inC_pt_zero ,Sequ-
ence<Sequence< long    >  > &inC_ri_theRing, long inEfz , long inNfz , long
inHigh , long inBroad );
public:
virtual  void  initialize ();
virtual  void  finalize ( LREF( char  )  inSpaceRef );
virtual  long loadTestFig ( long inNo );
virtual  Ref< SH_Indices  >  getIndex () const ;
virtual  void  getGeometry ( struct Geometry &inC_gm ) const ;
};
class SH_Feature  : public sdlObj {
COMMON_FCT_DECLS(SH_Feature)
public:
 sdl_string Name ;
 sdl_string Description ;
 long Type ;
 sdl_string someStuff ;
virtual  void  initialize ( LREF( char  )  inName , LREF( char  )  inDescrip-
tion , long inType , LREF( char  )  inSomeStuff );
virtual  void  initWKS ( LREF( char  )  inName , LREF( char  )  inDescription
, long inType , LREF( char  )  inSomeStuff , LREF( struct Geometry  )  inC_gm
, Ref< SH_Accuracy  >  inAccuracyRef );
virtual  void  finalize () const ;
virtual  long getType () const ;
RefInv<SH_Geometric_Property,SH_Feature,32> geomProperty
;
RefInv<SH_Feature,SH_Feature,36> partOf
;
SetInv<SH_Feature,SH_Feature,40> constituent
;
};
class SH_Accuracy  : public sdlObj {
COMMON_FCT_DECLS(SH_Accuracy)
private:
 long Accuracy ;
 long MeasureMethod ;
public:
virtual  void  initialize ( long inAccuracy , long inMeasureMethod );
virtual  long getAccuracy () const ;
};
class SH_Geometric_Property  : public sdlObj {
COMMON_FCT_DECLS(SH_Geometric_Property)
public:
virtual  void  finalize () const ;
```

```
virtual  void  createGeometry ( LREF( struct CoordGeom  )  inC_cg , Ref<
SH_Accuracy  >  inAccuracyRef );
RefInv<SH_Feature,SH_Geometric_Property,4> feature
;
RefInv<SH_Spatial_Ref_System,SH_Geometric_Property,8> spaceRef
;
RefInv<SH_Geometry,SH_Geometric_Property,12> geometry
;
};
class SH_Spatial_Ref_System  : public sdlObj {
COMMON_FCT_DECLS(SH_Spatial_Ref_System)
private:
 sdl_string Name ;
 float Linear_unit ;
 sdl_string Angular_unit ;
 sdl_string Prime_meridian ;
 sdl_string Ellipsoide ;
 sdl_string GCS ;
 sdl_string CT ;
 long Zone ;
 long OrigoN ;
 long OrigoOE ;
 long MaxN ;
 long MaxOE ;
 long MinN ;
 long MinOE ;
 sdl_string PC ;
 sdl_string Temporal_system ;
 long SpaceDimension ;
 long TimeDimension ;
 long TotalDimension ;
public:
virtual  void  initialize ( LREF( char  )  inName );
virtual  void  finalize () const ;
virtual  long spaceDimension () const ;
virtual  long timeDimension () const ;
virtual  long totalDimension () const ;
virtual  void  setLinear_unit ( float inLinear_unit );
virtual  void  setZone ( long inZone );
virtual  void  setOrigo ( long inOrigoN , long inOrigoOE );
virtual  void  setMax ( long inMaxN , long inMaxOE );
virtual  void  setMin ( long inMinN , long inMinOE );
virtual  float getLinear_unit () const ;
virtual  long getZone () const ;
virtual  void  getOrigo ( long &inOrigoN , long &inOrigoOE ) const ;
virtual  void  getMax ( long &inMaxN , long &inMaxOE ) const ;
virtual  void  getMin ( long &inMinN , long &inMinOE ) const ;
virtual  LREF( char  )  metaData ( LREF( char  )  name ) const ;
SetInv<SH_Geometric_Property,SH_Spatial_Ref_System,112> properties
;
};
class SH_Geometry  : public sdlObj {
COMMON_FCT_DECLS(SH_Geometry)
protected:
 long CoordDimension ;
 boolean Planar ;
 boolean Closed ;
 boolean Simple ;
 Ref< SH_Accuracy  >  accuracyRef ;
 long Dimension ;
 enum InterpretationType Interpretation ;
public:
virtual  Ref< SH_Spatial_Ref_System  >  referenceSystem () const ;
virtual  void  getCoordinateGeometry ( struct CoordGeom &inC_cg ) const ;
```

```
virtual  Ref< SH_Accuracy  >  getAccuracyRef () const ;
virtual  void  setAccuracyRef ( Ref< SH_Accuracy  >  inAccuracyRef );
RefInv<SH_Geometry,SH_Geometry,32> partOf
;
SetInv<SH_Geometry,SH_Geometry,36> constituent
;
RefInv<SH_Geometric_Property,SH_Geometry,48> geomProperty
;
};
class SH_Point :public SH_Geometry  {
COMMON_FCT_DECLS(SH_Point)
private:
 struct A_Point Point ;
public:
 boolean KP ;
virtual  void  initialize ( LREF( struct A_Point  )  inPt , Ref< SH_Accuracy
>  inAccuracyRef );
virtual  void  initWKS ( LREF(Sequence< long   > )  inPt , Ref< SH_Accuracy  >
inAccuracyRef );
virtual  void  finalize ();
virtual  void  getCoordinateGeometry ( struct CoordGeom &inC_cg ) const ;
SetInv<SH_Line_String,SH_Point,72> partOf
;
};
class SH_Curve :public SH_Geometry  {
COMMON_FCT_DECLS(SH_Curve)
;
};
class SH_Surface :public SH_Geometry  {
COMMON_FCT_DECLS(SH_Surface)
;
};
class SH_Line_String :public SH_Curve  {
COMMON_FCT_DECLS(SH_Line_String)
public:
virtual  void  initialize ( Ref< SH_Accuracy  >  inAccuracyRef );
virtual  void  initWKS ( LREF(Sequence<Sequence< long   >  > )  inC_ls , Ref<
SH_Accuracy  >  inAccuracyRef );
virtual  void  createRing ( LREF(Sequence<Sequence< long   >  > )  inC_ri ,
Ref< SH_Accuracy  >  inAccuracyRef );
virtual  void  finalize ();
virtual  void  getCoordinateGeometry ( struct CoordGeom &inC_cg ) const ;
SetInv<SH_Point,SH_Line_String,52> constituent
;
SetInv<SH_Polygon,SH_Line_String,64> partOf
;
};
class SH_Polygon :public SH_Surface  {
COMMON_FCT_DECLS(SH_Polygon)
public:
virtual  void  initialize ( Ref< SH_Accuracy  >  inAccuracyRef );
virtual  void  initWKS ( LREF(Sequence<Sequence<Sequence< long   >  >  > )
inC_pg , Ref< SH_Accuracy  >  inAccuracyRef );
virtual  void  finalize ();
virtual  void  getCoordinateGeometry ( struct CoordGeom &inC_cg ) const ;
SetInv<SH_Line_String,SH_Polygon,52> constituent
;
};
class SH_Indices  : public sdlObj {
COMMON_FCT_DECLS(SH_Indices)
public:
Index< struct A_Point  , Ref< SH_Point  >    >  aPointInd ;
Index< sdl_string  , Ref< SH_Spatial_Ref_System  >    >  aSpaceRefInd ;
Index< long  , Ref< SH_Geometry  >    >  serialNoInd ;
```

```
Index< long  , Ref< SH_Feature  >    >  themeInd ;
Index< sdl_string  , Ref< SH_Feature  >    >  geomTypeInd ;
virtual  void  initialize () const ;
virtual  void  newPoint ( struct A_Point inSH_pt , Ref< SH_Point  >  inaPoint
) const ;
virtual  Ref< SH_Point  >  findPoint ( struct A_Point inSH_pt ) const ;
virtual  void  killPoint ( struct A_Point inSH_pt ) const ;
virtual    void    newRefsys  (  LREF(  char    )    inSpaceRef  ,  Ref<
SH_Spatial_Ref_System  >  inaSpaceRef ) const ;
virtual  Ref< SH_Spatial_Ref_System  >  findRefsys ( LREF( char  )  inSpaceRef
) const ;
virtual  void  killRefsys ( LREF( char  )  inSpaceRef ) const ;
virtual  void  newSerialNo ( long inSerialNo , Ref< SH_Geometry  >  inaGeome-
try ) const ;
virtual  Ref< SH_Geometry  >  findSerialNo ( long inSerialNo ) const ;
virtual  void  killSerialNo ( long inSerialNo ) const ;
virtual  void  killAllSerialNoInx () const ;
virtual  void  newThemeInd ( long inTheme , Ref< SH_Feature  >  inaFeature )
const ;
virtual  void  killThemeInd ( long inTheme , Ref< SH_Feature  >  inaFeature )
const ;
virtual  void  newGeomTypeInd ( LREF( char  )  inGeomType , Ref< SH_Feature  >
inaFeature ) const ;
virtual  void  killGeomTypeInd ( LREF( char  )  inGeomType , Ref< SH_Feature
>  inaFeature ) const ;
};
const  long MaxDim = 4 ;
typedef Sequence< long   > LongBSeq;
typedef Sequence< double   > DoubleBSeq;
enum CoordinateTypeTag {
TypeDouble ,
TypeLong };
OVERRIDE_INDVAL(CoordinateTypeTag)
NO_APPLY(CoordinateTypeTag)
;
typedef Sequence< long   > Coordinate;
typedef Sequence<Sequence< long   >  > LineString;
typedef Sequence<Sequence< long   >  > Ring;
typedef Sequence<Sequence<Sequence< long   >  >  > Polygon;
const  long TriangleDim = 3 ;
typedef Sequence<Sequence< long   >  > Triangle;
typedef Sequence<Sequence<Sequence<Sequence< long   >  >  >  > PolyhedralSur-
face;
typedef Sequence<Sequence<Sequence<Sequence< long   >  >  >  > Polyhedron;
typedef  Sequence<Sequence<Sequence<Sequence<Sequence< long    >  >  >  >  >
PolyhedralSolid;
struct Envelope {
Sequence< long   > min;
Sequence< long   > max;
void __apply(HeapOps op) {
min.__apply(op);
max.__apply(op);
};
static void sdl_apply(void *p,HeapOps op);
};
;
enum CoordGeomTypeTag {
CoordCoordinateType ,
CoordLineStringType ,
CoordRingType ,
CoordPolygonType ,
CoordTriangleType ,
CoordPolyhedralSurfaceType ,
CoordPolyhedronType ,
```

```
CoordPolyhedralSolidType ,
CoordEnvelopeType ,
CoordGeomCollectionType };
OVERRIDE_INDVAL(CoordGeomTypeTag)
NO_APPLY(CoordGeomTypeTag)
;
struct CoordGeom : sdl_heap_base {
CoordGeom(){}
CoordGeom(const CoordGeom & arg) { *this = arg;}
const CoordGeom & operator=(const CoordGeom &);
const  enum CoordGeomTypeTag CoordGeomType ;
int _armi( ) const;
void set_utag( enum CoordGeomTypeTag  _tg_val ){int old_arm =_armi();
( enum CoordGeomTypeTag & )CoordGeomType = _tg_val;
if (old_arm!= _armi()) __free_space();}
void set_CoordGeomType( enum CoordGeomTypeTag  _tg_val ){set_utag(_tg_val);}
 enum CoordGeomTypeTag  get_utag() const { return CoordGeomType;}
 enum CoordGeomTypeTag  get_CoordGeomType() const { return CoordGeomType;}
void __apply(HeapOps op);
Sequence< long   >  & set_point() {
assert(_armi()==0);
if (!space) __set_space(12);

return *(Sequence< long   >  *  )space ;}
const Sequence< long   >  & get_point() const {
assert(_armi()==0);
return  *(const Sequence< long   >  * ) space;}
Sequence<Sequence< long   >  >  & set_line_string() {
assert(_armi()==1);
if (!space) __set_space(12);

return *(Sequence<Sequence< long   >  >  *  )space ;}
const Sequence<Sequence< long   >  >  & get_line_string() const {
assert(_armi()==1);
return  *(const Sequence<Sequence< long   >  >  * ) space;}
Sequence<Sequence< long   >  >  & set_ring() {
assert(_armi()==2);
if (!space) __set_space(12);

return *(Sequence<Sequence< long   >  >  *  )space ;}
const Sequence<Sequence< long   >  >  & get_ring() const {
assert(_armi()==2);
return  *(const Sequence<Sequence< long   >  >  * ) space;}
Sequence<Sequence<Sequence< long   >  >  >  & set_polygon() {
assert(_armi()==3);
if (!space) __set_space(12);

return *(Sequence<Sequence<Sequence< long   >  >  >  *  )space ;}
const Sequence<Sequence<Sequence< long   >  >  >  & get_polygon() const {
assert(_armi()==3);
return  *(const Sequence<Sequence<Sequence< long   >  >  >  * ) space;}
Sequence<Sequence< long   >  >  & set_triangle() {
assert(_armi()==4);
if (!space) __set_space(12);

return *(Sequence<Sequence< long   >  >  *  )space ;}
const Sequence<Sequence< long   >  >  & get_triangle() const {
assert(_armi()==4);
return  *(const Sequence<Sequence< long   >  >  * ) space;}
Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  & set_surface() {
assert(_armi()==5);
if (!space) __set_space(12);

return *(Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  *  )space ;}
```

```
const Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  & get_surface()
const {
assert(_armi()==5);
return  *(const Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  * )
space;}
Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  & set_polyhedron() {
assert(_armi()==6);
if (!space) __set_space(12);

return *(Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  *  )space ;}
const  Sequence<Sequence<Sequence<Sequence<  long      >    >    >    >    &
get_polyhedron() const {
assert(_armi()==6);
return  *(const Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  * )
space;}
Sequence<Sequence<Sequence<Sequence<Sequence<  long    >   >   >   >   >   &
set_solid() {
assert(_armi()==7);
if (!space) __set_space(12);

return *(Sequence<Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  >  *
)space ;}
const Sequence<Sequence<Sequence<Sequence<Sequence< long   >  >  >  >  >  &
get_solid() const {
assert(_armi()==7);
return  *(const Sequence<Sequence<Sequence<Sequence<Sequence< long   >  >  >
>  >  * ) space;}
 struct Envelope   & set_envelope() {
assert(_armi()==8);
if (!space) __set_space(24);

return *( struct Envelope  *   )space ;}
const  struct Envelope   & get_envelope() const {
assert(_armi()==8);
return  *(const  struct Envelope  *  ) space;}
Sequence< struct CoordGeom   >  & set_cgc() {
assert(_armi()==9);
if (!space) __set_space(12);

return *(Sequence< struct CoordGeom   >  *  )space ;}
const Sequence< struct CoordGeom   >  & get_cgc() const {
assert(_armi()==9);
return  *(const Sequence< struct CoordGeom   >  * ) space;}
};
;
typedef Sequence< struct CoordGeom   > CoordGeomCollection;
typedef  sdl_string SpatialReferenceName ;
struct Geometry {
 struct CoordGeom coordinate_geometry ;
 sdl_string reference_system_name ;
void __apply(HeapOps op) {
coordinate_geometry.__apply(op);
reference_system_name.__apply(op);
};
static void sdl_apply(void *p,HeapOps op);
};
;
const int SH_Query_Manager_OID = 20373;
INTERFACE_POSTDEFS(SH_Query_Manager)
const int SH_Feature_OID = 20375;
INTERFACE_POSTDEFS(SH_Feature)
const int SH_Accuracy_OID = 20377;
INTERFACE_POSTDEFS(SH_Accuracy)
const int SH_Geometric_Property_OID = 20379;
```

```
INTERFACE_POSTDEFS(SH_Geometric_Property)
const int SH_Spatial_Ref_System_OID = 20381;
INTERFACE_POSTDEFS(SH_Spatial_Ref_System)
const int SH_Geometry_OID = 20383;
INTERFACE_POSTDEFS(SH_Geometry)
const int SH_Point_OID = 20385;
INTERFACE_POSTDEFS(SH_Point)
const int SH_Curve_OID = 20387;
INTERFACE_POSTDEFS(SH_Curve)
const int SH_Surface_OID = 20389;
INTERFACE_POSTDEFS(SH_Surface)
const int SH_Line_String_OID = 20391;
INTERFACE_POSTDEFS(SH_Line_String)
const int SH_Polygon_OID = 20393;
INTERFACE_POSTDEFS(SH_Polygon)
const int SH_Indices_OID = 20395;
INTERFACE_POSTDEFS(SH_Indices)

#ifdef MODULE_CODE
struct rModule SH_Objects(“SH_Objects”,0,10,20297 );
#define CUR_MOD SH_Objects
extern char * metatype_version_1_29;
int sdlModule_SH_Objects::oid_20297(){ return (int)metatype_version_1_29; };

void Distance::sdl_apply(void *p,HeapOps op)
{ ((Distance *)p)->__apply(op);};
template class Apply<Distance>;

void A_Point::sdl_apply(void *p,HeapOps op)
{ ((A_Point *)p)->__apply(op);};
template class Apply<A_Point>;
template class noappIndVal<InterpretationType>;
SETUP_VTAB(SH_Query_Manager,0)
NEW_PERSISTENT(SH_Query_Manager)
CLASS_VIRTUALS(SH_Query_Manager)
template class BRef<SH_Query_Manager,Ref<any> >;
template class WRef<SH_Query_Manager>;
template class Apply<Ref<SH_Query_Manager> >;
template class RefPin<SH_Query_Manager>;
template class WRefPin<SH_Query_Manager>;
template class srt_type<SH_Query_Manager>;
TYPE(SH_Query_Manager) TYPE_OBJECT(SH_Query_Manager);
TYPE_CAST_DEF(SH_Query_Manager)
TYPE_CAST_END(SH_Query_Manager)
void SH_Query_Manager::__apply(HeapOps op) {
inx.__apply(op);
} // end SH_Query_Manager::__apply
SETUP_VTAB(SH_Feature,0)
NEW_PERSISTENT(SH_Feature)
CLASS_VIRTUALS(SH_Feature)
template class BRef<SH_Feature,Ref<any> >;
template class WRef<SH_Feature>;
template class Apply<Ref<SH_Feature> >;
template class RefPin<SH_Feature>;
template class WRefPin<SH_Feature>;
template class srt_type<SH_Feature>;
TYPE(SH_Feature) TYPE_OBJECT(SH_Feature);
TYPE_CAST_DEF(SH_Feature)
TYPE_CAST_END(SH_Feature)
void SH_Feature::__apply(HeapOps op) {
Name.__apply(op);
Description.__apply(op);
someStuff.__apply(op);
geomProperty.__apply(op);
```

```
partOf.__apply(op);
constituent.__apply(op);
} // end SH_Feature::__apply
REF_INV_IMPL(SH_Geometric_Property,feature,SH_Feature,32)
REF_INV_IMPL(SH_Feature,constituent,SH_Feature,36)
SET_INV_IMPL(SH_Feature,partOf,SH_Feature,40)
SETUP_VTAB(SH_Accuracy,0)
NEW_PERSISTENT(SH_Accuracy)
CLASS_VIRTUALS(SH_Accuracy)
template class BRef<SH_Accuracy,Ref<any> >;
template class WRef<SH_Accuracy>;
template class Apply<Ref<SH_Accuracy> >;
template class RefPin<SH_Accuracy>;
template class WRefPin<SH_Accuracy>;
template class srt_type<SH_Accuracy>;
TYPE(SH_Accuracy) TYPE_OBJECT(SH_Accuracy);
TYPE_CAST_DEF(SH_Accuracy)
TYPE_CAST_END(SH_Accuracy)
void SH_Accuracy::__apply(HeapOps ) {}
SETUP_VTAB(SH_Geometric_Property,0)
NEW_PERSISTENT(SH_Geometric_Property)
CLASS_VIRTUALS(SH_Geometric_Property)
template class BRef<SH_Geometric_Property,Ref<any> >;
template class WRef<SH_Geometric_Property>;
template class Apply<Ref<SH_Geometric_Property> >;
template class RefPin<SH_Geometric_Property>;
template class WRefPin<SH_Geometric_Property>;
template class srt_type<SH_Geometric_Property>;
TYPE(SH_Geometric_Property) TYPE_OBJECT(SH_Geometric_Property);
TYPE_CAST_DEF(SH_Geometric_Property)
TYPE_CAST_END(SH_Geometric_Property)
void SH_Geometric_Property::__apply(HeapOps op) {
feature.__apply(op);
spaceRef.__apply(op);
geometry.__apply(op);
} // end SH_Geometric_Property::__apply
REF_INV_IMPL(SH_Feature,geomProperty,SH_Geometric_Property,4)
REF_INV_IMPL(SH_Spatial_Ref_System,properties,SH_Geometric_Property,8)
REF_INV_IMPL(SH_Geometry,geomProperty,SH_Geometric_Property,12)
SETUP_VTAB(SH_Spatial_Ref_System,0)
NEW_PERSISTENT(SH_Spatial_Ref_System)
CLASS_VIRTUALS(SH_Spatial_Ref_System)
template class BRef<SH_Spatial_Ref_System,Ref<any> >;
template class WRef<SH_Spatial_Ref_System>;
template class Apply<Ref<SH_Spatial_Ref_System> >;
template class RefPin<SH_Spatial_Ref_System>;
template class WRefPin<SH_Spatial_Ref_System>;
template class srt_type<SH_Spatial_Ref_System>;
TYPE(SH_Spatial_Ref_System) TYPE_OBJECT(SH_Spatial_Ref_System);
TYPE_CAST_DEF(SH_Spatial_Ref_System)
TYPE_CAST_END(SH_Spatial_Ref_System)
void SH_Spatial_Ref_System::__apply(HeapOps op) {
Name.__apply(op);
Angular_unit.__apply(op);
Prime_meridian.__apply(op);
Ellipsoide.__apply(op);
GCS.__apply(op);
CT.__apply(op);
PC.__apply(op);
Temporal_system.__apply(op);
properties.__apply(op);
} // end SH_Spatial_Ref_System::__apply
SET_INV_IMPL(SH_Geometric_Property,spaceRef,SH_Spatial_Ref_System,112)
SETUP_VTAB(SH_Geometry,0)
```

```
NEW_PERSISTENT(SH_Geometry)
CLASS_VIRTUALS(SH_Geometry)
template class BRef<SH_Geometry,Ref<any> >;
template class WRef<SH_Geometry>;
template class Apply<Ref<SH_Geometry> >;
template class RefPin<SH_Geometry>;
template class WRefPin<SH_Geometry>;
template class srt_type<SH_Geometry>;
TYPE(SH_Geometry) TYPE_OBJECT(SH_Geometry);
TYPE_CAST_DEF(SH_Geometry)
TYPE_CAST_END(SH_Geometry)
void SH_Geometry::__apply(HeapOps op) {
accuracyRef.__apply(op);
partOf.__apply(op);
constituent.__apply(op);
geomProperty.__apply(op);
} // end SH_Geometry::__apply
REF_INV_IMPL(SH_Geometry,constituent,SH_Geometry,32)
SET_INV_IMPL(SH_Geometry,partOf,SH_Geometry,36)
REF_INV_IMPL(SH_Geometric_Property,geometry,SH_Geometry,48)
SETUP_VTAB(SH_Point,0)
NEW_PERSISTENT(SH_Point)
CLASS_VIRTUALS(SH_Point)
template class BRef<SH_Point,Ref<SH_Geometry> >;
template class WRef<SH_Point>;
template class Apply<Ref<SH_Point> >;
template class RefPin<SH_Point>;
template class WRefPin<SH_Point>;
template class srt_type<SH_Point>;
TYPE(SH_Point) TYPE_OBJECT(SH_Point);
TYPE_CAST_DEF(SH_Point)
TYPE_CAST_CASE(SH_Point,SH_Geometry)
TYPE_CAST_END(SH_Point)
void SH_Point::__apply(HeapOps op) {
SH_Geometry::__apply(op);
partOf.__apply(op);
} // end SH_Point::__apply
SET_INV_IMPL(SH_Line_String,constituent,SH_Point,72)
SETUP_VTAB(SH_Curve,0)
NEW_PERSISTENT(SH_Curve)
CLASS_VIRTUALS(SH_Curve)
template class BRef<SH_Curve,Ref<SH_Geometry> >;
template class WRef<SH_Curve>;
template class Apply<Ref<SH_Curve> >;
template class RefPin<SH_Curve>;
template class WRefPin<SH_Curve>;
template class srt_type<SH_Curve>;
TYPE(SH_Curve) TYPE_OBJECT(SH_Curve);
TYPE_CAST_DEF(SH_Curve)
TYPE_CAST_CASE(SH_Curve,SH_Geometry)
TYPE_CAST_END(SH_Curve)
void SH_Curve::__apply(HeapOps op) {
SH_Geometry::__apply(op);
} // end SH_Curve::__apply
SETUP_VTAB(SH_Surface,0)
NEW_PERSISTENT(SH_Surface)
CLASS_VIRTUALS(SH_Surface)
template class BRef<SH_Surface,Ref<SH_Geometry> >;
template class WRef<SH_Surface>;
template class Apply<Ref<SH_Surface> >;
template class RefPin<SH_Surface>;
template class WRefPin<SH_Surface>;
template class srt_type<SH_Surface>;
TYPE(SH_Surface) TYPE_OBJECT(SH_Surface);
```

```
TYPE_CAST_DEF(SH_Surface)
TYPE_CAST_CASE(SH_Surface,SH_Geometry)
TYPE_CAST_END(SH_Surface)
void SH_Surface::__apply(HeapOps op) {
SH_Geometry::__apply(op);
} // end SH_Surface::__apply
SETUP_VTAB(SH_Line_String,0)
NEW_PERSISTENT(SH_Line_String)
CLASS_VIRTUALS(SH_Line_String)
template class BRef<SH_Line_String,Ref<SH_Curve> >;
template class WRef<SH_Line_String>;
template class Apply<Ref<SH_Line_String> >;
template class RefPin<SH_Line_String>;
template class WRefPin<SH_Line_String>;
template class srt_type<SH_Line_String>;
TYPE(SH_Line_String) TYPE_OBJECT(SH_Line_String);
TYPE_CAST_DEF(SH_Line_String)
TYPE_CAST_CASE(SH_Line_String,SH_Curve)
TYPE_CAST_END(SH_Line_String)
void SH_Line_String::__apply(HeapOps op) {
SH_Curve::__apply(op);
constituent.__apply(op);
partOf.__apply(op);
} // end SH_Line_String::__apply
SET_INV_IMPL(SH_Point,partOf,SH_Line_String,52)
SET_INV_IMPL(SH_Polygon,constituent,SH_Line_String,64)
SETUP_VTAB(SH_Polygon,0)
NEW_PERSISTENT(SH_Polygon)
CLASS_VIRTUALS(SH_Polygon)
template class BRef<SH_Polygon,Ref<SH_Surface> >;
template class WRef<SH_Polygon>;
template class Apply<Ref<SH_Polygon> >;
template class RefPin<SH_Polygon>;
template class WRefPin<SH_Polygon>;
template class srt_type<SH_Polygon>;
TYPE(SH_Polygon) TYPE_OBJECT(SH_Polygon);
TYPE_CAST_DEF(SH_Polygon)
TYPE_CAST_CASE(SH_Polygon,SH_Surface)
TYPE_CAST_END(SH_Polygon)
void SH_Polygon::__apply(HeapOps op) {
SH_Surface::__apply(op);
constituent.__apply(op);
} // end SH_Polygon::__apply
SET_INV_IMPL(SH_Line_String,partOf,SH_Polygon,52)
SETUP_VTAB(SH_Indices,5)
NEW_PERSISTENT(SH_Indices)
CLASS_VIRTUALS(SH_Indices)
template class BRef<SH_Indices,Ref<any> >;
template class WRef<SH_Indices>;
template class Apply<Ref<SH_Indices> >;
template class RefPin<SH_Indices>;
template class WRefPin<SH_Indices>;
template class srt_type<SH_Indices>;
TYPE(SH_Indices) TYPE_OBJECT(SH_Indices);
TYPE_CAST_DEF(SH_Indices)
TYPE_CAST_END(SH_Indices)
void SH_Indices::__apply(HeapOps op) {
aPointInd.__apply(op);
aSpaceRefInd.__apply(op);
serialNoInd.__apply(op);
themeInd.__apply(op);
geomTypeInd.__apply(op);
} // end SH_Indices::__apply
#undef CUR_MOD SH_Objects
```

```
struct rModule WKS(“WKS”,0,10,20051 );
#define CUR_MOD WKS
extern char * metatype_version_1_29;
int sdlModule_WKS::oid_20051(){ return (int)metatype_version_1_29; };
template class noappIndVal<CoordinateTypeTag>;

void Envelope::sdl_apply(void *p,HeapOps op)
{ ((Envelope *)p)->__apply(op);};
template class Apply<Envelope>;
template class noappIndVal<CoordGeomTypeTag>;
void CoordGeom::__apply(HeapOps op)
{
sdl_heap_base::__apply(op);
if (space != 0) switch(CoordGeomType){
case  CoordCoordinateType :
((Sequence< long   >  *  )space)->__apply(op); return ;
case  CoordLineStringType :
((Sequence<Sequence< long   >  >  *  )space)->__apply(op); return ;
case  CoordRingType :
((Sequence<Sequence< long   >  >  *  )space)->__apply(op); return ;
case  CoordPolygonType :
((Sequence<Sequence<Sequence< long   >  >  >  *  )space)->__apply(op); return
;
case  CoordTriangleType :
((Sequence<Sequence< long   >  >  *  )space)->__apply(op); return ;
case  CoordPolyhedralSurfaceType :
((Sequence<Sequence<Sequence<Sequence<  long    >   >   >   >   *   )space)-
>__apply(op); return ;
case  CoordPolyhedronType :
((Sequence<Sequence<Sequence<Sequence<  long    >   >   >   >   *   )space)-
>__apply(op); return ;
case  CoordPolyhedralSolidType :
((Sequence<Sequence<Sequence<Sequence<Sequence<  long    >   >   >   >   >   *
)space)->__apply(op); return ;
case  CoordEnvelopeType :
(( struct Envelope  *   )space)->__apply(op); return ;
case  CoordGeomCollectionType :
((Sequence< struct CoordGeom   >  *  )space)->__apply(op); return ;
default: return;
}}
int CoordGeom::_armi( ) const
{ switch(CoordGeomType) {
case  CoordCoordinateType :return(0);
case  CoordLineStringType :return(1);
case  CoordRingType :return(2);
case  CoordPolygonType :return(3);
case  CoordTriangleType :return(4);
case  CoordPolyhedralSurfaceType :return(5);
case  CoordPolyhedronType :return(6);
case  CoordPolyhedralSolidType :return(7);
case  CoordEnvelopeType :return(8);
case  CoordGeomCollectionType :return(9);
default: return -1;}}
const CoordGeom & CoordGeom::operator=(const CoordGeom  &arg ) {
set_utag(arg.get_utag());
 switch(CoordGeomType) {
case  CoordCoordinateType :set_point() = arg.get_point(); break;
case  CoordLineStringType :set_line_string() = arg.get_line_string(); break;
case  CoordRingType :set_ring() = arg.get_ring(); break;
case  CoordPolygonType :set_polygon() = arg.get_polygon(); break;
case  CoordTriangleType :set_triangle() = arg.get_triangle(); break;
case  CoordPolyhedralSurfaceType :set_surface() = arg.get_surface(); break;
case  CoordPolyhedronType :set_polyhedron() = arg.get_polyhedron(); break;
case  CoordPolyhedralSolidType :set_solid() = arg.get_solid(); break;
```

```
case  CoordEnvelopeType :set_envelope() = arg.get_envelope(); break;
case  CoordGeomCollectionType :set_cgc() = arg.get_cgc(); break;
default: sdl_heap_base::__clear();}
return *this;}
template class Apply<CoordGeom>;

void Geometry::sdl_apply(void *p,HeapOps op)
{ ((Geometry *)p)->__apply(op);};
template class Apply<Geometry>;
#undef CUR_MOD WKS
template class Sequence< Ref< void  >    > ;
template class Sequence< long   > ;
template class Sequence<Sequence< long   >  > ;
template class  Set< Ref< SH_Feature  >   >  ;
 template class Sequence< Ref< SH_Feature  >    > ;
template class  Set< Ref< SH_Geometric_Property  >   >  ;
 template class Sequence< Ref< SH_Geometric_Property  >    > ;
template class  Set< Ref< SH_Geometry  >   >  ;
 template class Sequence< Ref< SH_Geometry  >    > ;
template class  Set< Ref< SH_Line_String  >   >  ;
 template class Sequence< Ref< SH_Line_String  >    > ;
template class  Set< Ref< SH_Point  >   >  ;
 template class Sequence< Ref< SH_Point  >    > ;
template class  Set< Ref< SH_Polygon  >   >  ;
 template class Sequence< Ref< SH_Polygon  >    > ;
template class Sequence<Sequence<Sequence< long   >  >  > ;
template class Sequence< double   > ;
template class Sequence<Sequence<Sequence<Sequence< long   >  >  >  > ;
template class Sequence<Sequence<Sequence<Sequence<Sequence< long   >  >  >  >
> ;
template class Sequence< struct CoordGeom   > ;

#endif MODULE_CODE
#endif SH_Objects_mod
#endif WKS_mod
```

Vedlegg N Metodeimplementasjon; SH_Objects.C

```
// ***************************************************************
// File name: SH_Objects.C
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Implementation of methods for persistent classes
// stored in Shore-database
// Class-library of wxWindows is used with permission.
// The Shore databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 25.11.96 Created
// ***************************************************************
#include <stdlib.h>
#include “SH_Objects.h”

extern Ref<Pool> SH_Features, SH_Geometries, SH_Index;
extern Ref<SH_Query_Manager> SH_qm;
extern char* testSpaceRef;
//extern MyTextWindow* myTextWindow;
// ---------------------------------------------------------------
// Class name: SH_Query_Manager
// Description: Act as the only link between GO-application and
//Shore-database
// ---------------------------------------------------------------
void SH_Query_Manager::initialize() {
  inx = new(SH_Index) SH_Indices;
  inx.update()->initialize();
}

Ref<SH_Indices> SH_Query_Manager::getIndex() const {
  return inx;
}
// Creates two squares that connect, and act as a compund polygon.
// Using WKS in all work.
long SH_Query_Manager::loadTestFig(long inNo) {
  // Input parameter for number of objects to create
  int  generated = 0, inner_limit, outer_limit;
  char *name = “aHouse”;
  long east,north,height,date; // Part of a coordinate, in this sequence
  long east_from_zero, north_from_zero;
  long polygon_height, polygon_width;
  // Coordinate-based variables
  Coordinate C_pt_zero;
  Ring       C_ri;
  Polygon    C_pg;
  Geometry   C_gm;
  // Object references
  Ref<SH_Feature>            aFeature;
  Ref<SH_Spatial_Ref_System> aSpaceRef;
  Ref<SH_Accuracy>           anAccuracyRef;

   // Create Spatial Reference System
  aSpaceRef = new(SH_Features) SH_Spatial_Ref_System;
```

```
  aSpaceRef.update()->initialize(testSpaceRef);// Name of this
    aSpaceRef.update()->setLinear_unit((float)1.0);//  Coordinates  given  in
meters
  aSpaceRef.update()->setZone(99);
  aSpaceRef.update()->setOrigo(72000,62000);
  aSpaceRef.update()->setMin(72000,62000);
  aSpaceRef.update()->setMax(73000,63000);

  // Create Accuracy object
  anAccuracyRef = new(SH_Features) SH_Accuracy;
  anAccuracyRef.update()->initialize(82,36);

   // Establich zero-point; Never to be changed I think!
  C_pt_zero.append_elt((east = 0));
  C_pt_zero.append_elt((north = 0));
  C_pt_zero.append_elt((height = 0));
  C_pt_zero.append_elt((date = 991122));

  // Fills a square with 15x15 double polygons (max 225)
  // or up to 30 x 100 (max 3000)
  if (inNo < 226)
    inner_limit = outer_limit = 15;
  else {
    inner_limit = 30;
    outer_limit = 100;
  }
  // Supposed to create a number of polygons, given by parameter inNo
  for (int outer = 0; outer <= outer_limit; outer++) {
    north_from_zero = 10 + 60*outer + outer%2;
    // Left-bottom corner northern displacement from zero
    for (int inner = 0; inner <= inner_limit; inner++) {
      east_from_zero = 10 + 60*inner + inner%3;
      // Left-bottom corner eastern displacement from zero
      polygon_height = 20 + inner%5;
      polygon_width = 20 + inner%7;
      createRing(&C_pt_zero, C_ri, east_from_zero, north_from_zero,
 polygon_height, polygon_width);
      C_pg.append_elt(C_ri);
      // Empty the coordinate-points from ring
      C_ri.set_size(0);

      // Another polygon adjacent to previous
      east_from_zero += polygon_width; // Push the corner to the right
      polygon_height = 30 + inner%3;
      polygon_width = 30 + inner%6;
      createRing(&C_pt_zero, C_ri, east_from_zero, north_from_zero,
 polygon_height, polygon_width);
      C_pg.append_elt(C_ri);
      // Empty the coordinate-points from ring (again)
      C_ri.set_size(0);

      // Create a feature
      C_gm.reference_system_name.strcpy(aSpaceRef->metaData(“Name”));
      C_gm.coordinate_geometry.set_CoordGeomType(CoordPolygonType);
      C_gm.coordinate_geometry.set_polygon() = C_pg;
      aFeature = new(SH_Features) SH_Feature;
      aFeature.update()->initWKS(name,name,5000,”someStuff”,&C_gm,anAccuracy-
Ref);
      // Empty rings from the polygon
      C_pg.set_size(0);
      generated +=1;
      if (generated > inNo) return(0);
    }
  }
```

```
  // That should be all
  return(0);
}

void SH_Query_Manager::createRing(Coordinate *inC_pt_zero,
     Ring &inC_ri_theRing,
      long inEfz, long inNfz, long inHigh,
      long inBroad) {
  Coordinate C_pt;
  long aPoint_efz, aPoint_nfz, aPoint_height, time_value;
  //  First point
  aPoint_efz    = inC_pt_zero->get_elt(0) + inEfz;
  aPoint_nfz    = inC_pt_zero->get_elt(1) + inNfz;
  aPoint_height = inC_pt_zero->get_elt(2);
  time_value    = inC_pt_zero->get_elt(3);

  C_pt.append_elt(aPoint_efz); // East-value
  C_pt.append_elt(aPoint_nfz); // North-value
  C_pt.append_elt(aPoint_height); // Height above sealevel (not used)
  C_pt.append_elt(time_value); // Time-value
  inC_ri_theRing.append_elt(C_pt);

  // Second Point
  aPoint_nfz += inHigh;
  C_pt.write_elt(1)= aPoint_nfz; // Norht-value
  inC_ri_theRing.append_elt(C_pt);

  // Third Point
  aPoint_efz += inBroad;
  C_pt.write_elt(0)= aPoint_efz; // Norht-value
  inC_ri_theRing.append_elt(C_pt);

  // Forth and last  Point
  aPoint_nfz -= inHigh;
  C_pt.write_elt(1)= aPoint_nfz; // Norht-value
  inC_ri_theRing.append_elt(C_pt);

  return;
}

void SH_Query_Manager::getGeometry(Geometry &inC_gm) const {
  // Here's the idea:
  // Find SpaceRef-object via index, and traverse all ref's to
  // geometries.
  // Pick those Geometries satisfying given criteria
  // geometry is added to the geometry-collection
  // All polygons are added, so are line-strings without ref's to polygons,
  // and point without refs to linestrings
  Ref<SH_Spatial_Ref_System> aSpaceRef;
  Ref<SH_Geometric_Property> aGeomP;
  Ref<any> ref;
  Ref<SH_Point> aPoint;
  Ref<SH_Line_String> aLineString;
  Ref<SH_Polygon> aPolygon;
  CoordGeomCollection C_cgc;
  CoordGeom C_cg;
  bool pointCriteria=TRUE, lineStringCriteria=TRUE, polygonCriteria=TRUE;
  aSpaceRef = inx->findRefsys(inC_gm.reference_system_name);
  if (!aSpaceRef)
    return; // Not found!
  int i;
  // Check all ref's found from Refsys
  for (aGeomP = aSpaceRef->properties.get_elt(i=0);
       aGeomP != NULL;
```

```
       aGeomP = aSpaceRef->properties.get_elt(++i)) {
    ref = aGeomP->geometry;
    aPoint = TYPE_OBJECT(SH_Point).isa(ref);
    if (aPoint) {
      // points with no partOf-relation are
      // put into coordinate-collection
      if (aPoint->partOf.get_size() == 0) {
if (pointCriteria) {
  aPoint->getCoordinateGeometry(C_cg);
  C_cgc.append_elt(C_cg);

}
      }
    }
    else {
      aLineString = TYPE_OBJECT(SH_Line_String).isa(ref);
      if (aLineString) {
if (lineStringCriteria) {
  if (aLineString->partOf.get_size() == 0) {
    aLineString->getCoordinateGeometry(C_cg);
    C_cgc.append_elt(C_cg);
  }
}
      }
      else {
aPolygon = TYPE_OBJECT(SH_Polygon).isa(ref);
if (aPolygon) {
  if (polygonCriteria) {
    aPolygon->getCoordinateGeometry(C_cg);
    C_cgc.append_elt(C_cg);
  }
}
      }
    }
  }
  inC_gm.coordinate_geometry.set_CoordGeomType(CoordGeomCollectionType);
  inC_gm.coordinate_geometry.set_cgc()  = C_cgc;
  return;
}
void SH_Query_Manager::finalize(char *inSpaceRef) {
  // Delete all objects in database, or those connected to
  // given spaceRef
  Ref<SH_Spatial_Ref_System> aSpaceRef;
  if (strcmp(inSpaceRef,"ALL")) {
    aSpaceRef = inx->findRefsys(inSpaceRef);
    aSpaceRef->finalize();
    SH_DO(aSpaceRef.destroy());
    return;
  }
  // Scan all spaceRef's, delete them one by one.
   IndexScanIter<sdl_string,Ref<SH_Spatial_Ref_System> > iter(inx->aSpaceRe-
fInd);
  SH_DO(iter.next());
  for ( ; !iter.eof; SH_DO(iter.next())) {
    iter.cur_val->finalize();
    SH_DO(iter.cur_val.destroy());
  }
}


// ----------------------------------------------------------------
// Class name: SH_Feature
// Description :
//
```

```
// ------------------------------------------------------------
void SH_Feature::initialize(char *inName, char *inDescription,
long inType, char *inSomeStuff) {
  Name = inName;
  Description = inDescription;
  someStuff = inSomeStuff;
  Type = inType;
  partOf = NULL; // Feature-Feature-Relation: Won't use this
  SH_qm->getIndex()->newThemeInd(Type,this);
  SH_qm->getIndex()->newGeomTypeInd(Description,this);
  return;
}
void SH_Feature::finalize() const {
  SH_qm->getIndex()->killThemeInd(Type,this);
  SH_qm->getIndex()->killGeomTypeInd(Description,this);
}

void SH_Feature::initWKS(char *inName, char *inDescription,
long inType, char *inSomeStuff,
Geometry * inC_gm,
Ref<SH_Accuracy> inAccuracyRef) {
  // Object references
  Ref<SH_Geometric_Property> aGeomP;
  Ref<SH_Spatial_Ref_System> aSpaceRef;
  // Update some attributes
  Name = inName;
  Description = inDescription;
  someStuff = inSomeStuff;
  Type = inType;
  // Create Geometric-Property, and establish relationship Feature - Property
  aGeomP = new(SH_Features) SH_Geometric_Property;
  geomProperty = aGeomP; // Relationship
  // Find reference to given ReferenceSystem through the index
  aSpaceRef = (SH_qm->getIndex())->findRefsys(inC_gm->reference_system_name);
  if (aSpaceRef == NULL)
    return; //Didn't find the given refsys!
  // - establish relationship to RefSys
  aGeomP.update()->spaceRef = aSpaceRef; // relationship
  // Create Geometry
   aGeomP.update()->createGeometry(&(inC_gm->coordinate_geometry),inAccuracy-
Ref);
  // Done!
  partOf = NULL; // Feature-Feature-Relation: Won't use this
  return;
}

long SH_Feature::getType() const {
  return Type;
}

// ------------------------------------------------------------
// Class name: SH_Geometric_Property
// Description:
//
// ------------------------------------------------------------
void SH_Geometric_Property::finalize() const{
  Ref<SH_Point> aPoint;
  Ref<SH_Line_String> aLineString;
  Ref<SH_Polygon> aPolygon;
  feature->finalize();
  SH_DO(feature.destroy());
  SH_DO(geometry->getAccuracyRef().destroy());
  aPoint = TYPE_OBJECT(SH_Point).isa(geometry);
  if (aPoint) {
```

```
    aPoint.update()->finalize();
    SH_DO(aPoint.destroy());
  }
  else {
    aLineString = TYPE_OBJECT(SH_Line_String).isa(geometry);
    if (aLineString) {
    aLineString.update()->finalize();
    SH_DO(aLineString.destroy());
    }
    else {
      aPolygon = TYPE_OBJECT(SH_Polygon).isa(geometry);
      if (aPolygon) {
aPolygon.update()->finalize();
SH_DO(aPolygon.destroy());
      }
    }
  }
  return;
} // Destructor

void         SH_Geometric_Property::createGeometry(CoordGeom*         inC_cg,
Ref<SH_Accuracy> inAccuracy) {
  // Creates the correct Geometry
  Ref<SH_Point> aPoint;
  Ref<SH_Line_String> aLine;
  Ref<SH_Polygon> aPolygon;
  A_Point SH_pt;
  Coordinate C_pt;
  LineString C_ls;
  Polygon C_pg;
  switch (inC_cg->get_CoordGeomType()) {
    case CoordCoordinateType:
      SH_pt.east   = inC_cg->get_point().get_elt(0);
      SH_pt.north  = inC_cg->get_point().get_elt(1);
      SH_pt.height = inC_cg->get_point().get_elt(2);
      SH_pt.date   = inC_cg->get_point().get_elt(3);
      //look for the point in inx-obect
      aPoint = (SH_qm->getIndex())->findPoint(SH_pt);
      //if found; thats it
      if (!aPoint) {
aPoint = new(SH_Geometries) SH_Point;
C_pt = inC_cg->get_point();
aPoint.update()->initWKS(&C_pt,inAccuracy);
      }
      geometry = aPoint;
      break;
    case CoordLineStringType:
      // Choosed not to test for existing lines or part of lines
      // Means they might be duplicated as Line_String-objects, but
      // on the other hand the direction of a line will be defined
      // with respect to each Line_String.
      aLine = new (SH_Geometries) SH_Line_String;
      C_ls = inC_cg->get_line_string();
      aLine.update()->initWKS(&C_ls,inAccuracy);
      geometry = aLine;
      break;
    case CoordRingType:
      //    Ring ring;
      break;
    case CoordPolygonType:
      aPolygon = new (SH_Geometries) SH_Polygon;
      C_pg = inC_cg->get_polygon();
      aPolygon.update()->initWKS(&C_pg,inAccuracy);
      geometry = aPolygon;
```

```
      break;
    case CoordTriangleType:
      //    Triangle triangle;
      break;
    case CoordPolyhedralSurfaceType:
      //    PolyhedralSurface surface;
      break;
    case CoordPolyhedronType:
      //    Polyhedron polyhedron;
      break;
    case CoordPolyhedralSolidType:
      //    PolyhedralSolid solid;
      break;
    case CoordEnvelopeType:
      //      Envelope envelope;
      break;
    case CoordGeomCollectionType:
      //      CoordGeomCollection cgc;
      break;
    default:
      break;
  }
  return;
}

// ----------------------------------------------------------------
// Class name: SH_Spatial_Ref_System
// Description: Finds locations in real world, given geometry
//                  (place/time)
//                        A very limited version is implemented
// ----------------------------------------------------------------
void SH_Spatial_Ref_System::initialize(char* inName) {
  Name = inName;
  Angular_unit.strcpy(“Degree”);
  Prime_meridian.strcpy(“Greenwich”);
  Ellipsoide.strcpy(“GauzzKrugers”);
  GCS.strcpy(“WGS84”);
  CT.strcpy(“Transverse Mercator”);
  PC.strcpy(“WGS/Europeen datum ED50”);
  Temporal_system.strcpy(“YYMMDD”);
  SpaceDimension = 3; // Third dimension always set to zero at the moment
  TimeDimension = 1; // This is 1
  TotalDimension = 4; // This is the sum of space- and timeDimension
  SH_qm->getIndex()->newRefsys(Name, this);
}

void SH_Spatial_Ref_System::finalize() const{
  int i;
  Ref<SH_Geometric_Property> aGeomP;
  for (aGeomP = properties.get_elt(i=0);
       aGeomP != NULL;
       aGeomP = properties.get_elt(++i)) {
    aGeomP->finalize();
    SH_DO(aGeomP.destroy());
  }
  SH_qm->getIndex()->killRefsys(Name);
}

void SH_Spatial_Ref_System::setLinear_unit(float inLinear_unit) {
  Linear_unit = inLinear_unit;
}
void SH_Spatial_Ref_System::setZone(long inZone) {
  Zone = inZone;
}
```

```
void SH_Spatial_Ref_System::setOrigo(long inOrigoN, long inOrigoOE) {
  OrigoN = inOrigoN;
  OrigoOE = inOrigoOE;
}
void SH_Spatial_Ref_System::setMax(long inMaxN, long inMaxOE) {
  MaxN = inMaxN;
  MaxOE = inMaxOE;
}
void SH_Spatial_Ref_System::setMin(long inMinN, long inMinOE) {
  MinN = inMinN;
  MinOE = inMinOE;
}
float SH_Spatial_Ref_System::getLinear_unit() const {
  return Linear_unit;
}
long SH_Spatial_Ref_System::getZone() const{
  return Zone;
}
void SH_Spatial_Ref_System::getOrigo(long &inOrigoN, long &inOrigoOE) const{
  inOrigoN = OrigoN;
  inOrigoOE = OrigoOE;
}
void SH_Spatial_Ref_System::getMax(long& inMaxN, long& inMaxOE) const{
  inMaxN = MaxN;
  inMaxOE = MaxOE;
}
void SH_Spatial_Ref_System::getMin(long& inMinN, long& inMinOE) const{
  inMinN = MinN;
  inMinOE = MinOE;
}
long SH_Spatial_Ref_System::spaceDimension() const {
  return SpaceDimension;
}
long SH_Spatial_Ref_System::timeDimension() const {
return TimeDimension;
}
long SH_Spatial_Ref_System::totalDimension() const {
return TotalDimension;
}
char* SH_Spatial_Ref_System::metaData(char* inName) const {
  if (!strcmp(“Name”,inName))
    return Name;
  if (!strcmp(“Angular_unit”,inName))
    return Angular_unit;
  if (!strcmp(“Prime_meridian”,inName))
    return Prime_meridian;
  if (!strcmp(“Ellipsoide”,inName))
    return Ellipsoide;
  if (!strcmp(“GCS”,inName))
    return GCS;
  if (!strcmp(“CT”,inName))
    return CT;
  if (!strcmp(“PC”,inName))
    return PC;
  if (!strcmp(“Temporal_system”,inName))
    return Temporal_system;
  return (“Unkown metadata”);
}

// ---------------------------------------------------------------
// Class name: SH_Accuracy
// Description:
//
// ---------------------------------------------------------------
```

```
void SH_Accuracy::initialize(long inMeasureMethod, long inAccuracy) {
  MeasureMethod = inMeasureMethod;
  Accuracy = inAccuracy;
}
long SH_Accuracy::getAccuracy() const {
  return Accuracy;
}
// ---------------------------------------------------------------
// Class name: SH_Geometry
// Description:
//
// ---------------------------------------------------------------
Ref<SH_Spatial_Ref_System> SH_Geometry::referenceSystem() const {
  return geomProperty->spaceRef;
}
void SH_Geometry::getCoordinateGeometry(CoordGeom &inC_cg) const {
  return;  // This is virtual (as all others are..)
}
Ref<SH_Accuracy> SH_Geometry::getAccuracyRef() const {
  return accuracyRef;
}
void SH_Geometry::setAccuracyRef(Ref<SH_Accuracy> inAccuracyRef) {
accuracyRef = inAccuracyRef;
}
// ---------------------------------------------------------------
// Class name: SH_Point
// Description:
//
// ---------------------------------------------------------------
//    attribute SH_point Point; // The actual point; sequence<long,MaxDim>
// Constructor, sets dimension = 0
void SH_Point::initialize(A_Point *inC_pt, Ref<SH_Accuracy> inAccuracy) {
  //update attributes of Geometry and in Point
  CoordDimension = 4;
  Dimension = 0;
  Interpretation = GO_Point;
  Planar = TRUE;
  Closed = TRUE;
  Simple = TRUE;
  accuracyRef = inAccuracy;
  Point = *inC_pt;
  // update index
  SH_qm->getIndex()->newPoint(Point,this);
}

void SH_Point::initWKS(Coordinate *inC_pt, Ref<SH_Accuracy> inAccuracy) {
  //update attributes of Geometry and in Point
  CoordDimension = 4;
  Dimension = 0;
  Interpretation = GO_Point;
  Planar = TRUE;
  Closed = TRUE;
  Simple = TRUE;
  accuracyRef = inAccuracy;
  Point.east = inC_pt->get_elt(0);
  Point.north = inC_pt->get_elt(1);
  Point.height = inC_pt->get_elt(2);
  Point.date = inC_pt->get_elt(3);
  // update index
  SH_qm->getIndex()->newPoint(Point,this);
}

void SH_Point::finalize() {
  SH_qm->getIndex()->killPoint(Point);
```

```
} // Destructor

void SH_Point::getCoordinateGeometry(CoordGeom &inC_cg) const {
  Coordinate C_pt;
  C_pt.append_elt(Point.east);
  C_pt.append_elt(Point.north);
  C_pt.append_elt(Point.height);
  C_pt.append_elt(Point.date);
  inC_cg.set_CoordGeomType(CoordCoordinateType);
  inC_cg.set_point() = C_pt;
  return;
}

// ----------------------------------------------------------------
// Interface name: SH_Line_String
// Description:
//
// ----------------------------------------------------------------
void SH_Line_String::initialize(Ref<SH_Accuracy> inAccuracy){
  // Constructor: dimension=1, set interpretation(line or ring)
  // For each point in LineString-sequence, establish
  // relation to existing or new Point or
  CoordDimension = 4;
  Dimension = 1;
  Interpretation = GO_Lines;
  Planar = TRUE;
  Closed = FALSE;
  Simple = TRUE;
  accuracyRef = inAccuracy;
}

void  SH_Line_String::initWKS(LineString*  inC_ls,  Ref<SH_Accuracy>  inAccu-
racy){
  // Constructor: dimension=1, set interpretation(line or ring)
  // For each point in LineString-sequence, establish
  // relation to existing or new Point or
  CoordDimension = 4;
  Dimension = 1;
  Interpretation = GO_Lines;
  Planar = TRUE;
  Closed = FALSE;
  Simple = TRUE;
  accuracyRef = inAccuracy;
  Coordinate C_pt;
  A_Point SH_pt;
  Ref<SH_Point> aPoint ;
  for (unsigned int i = 0; i < inC_ls->get_size(); i++) {
    C_pt = inC_ls->get_elt(i);
    SH_pt.east = C_pt.get_elt(0);
    SH_pt.north = C_pt.get_elt(1);
    SH_pt.height = C_pt.get_elt(2);
    SH_pt.date = C_pt.get_elt(3);
    //look for the point in inx-obect
    aPoint = (SH_qm->getIndex())->findPoint(SH_pt);
    //if found; thats it
    if (!aPoint) {
      aPoint = new(SH_Geometries) SH_Point;
      aPoint.update()->initWKS(&C_pt,inAccuracy);
    }
    constituent.add(aPoint);
  }
}
void SH_Line_String::createRing(Ring* inC_ri, Ref<SH_Accuracy> inAccuracy){
  // Constructor of rings (Shore doesn’t support method overloading
```

```
  // so I have to use another method name than initialize......
  // dimension=1, set interpretation(line or ring)
  // For each point in LineString-sequence, establish
  // relation to existing or new Point
  CoordDimension = 4;
  Dimension = 1;
  Interpretation = GO_Polygon;
  Planar = TRUE;
  Closed = FALSE;
  Simple = TRUE;
  accuracyRef = inAccuracy;
  Coordinate C_pt;
  A_Point SH_pt;
  Ref<SH_Point> aPoint;
  for (unsigned int i = 0; i < inC_ri->get_size(); i++) {
    C_pt = inC_ri->get_elt(i);
    SH_pt.east = C_pt.get_elt(0);
    SH_pt.north = C_pt.get_elt(1);
    SH_pt.height = C_pt.get_elt(2);
    SH_pt.date = C_pt.get_elt(3);
    //look for the point in inx-obect
    aPoint = (SH_qm->getIndex())->findPoint(SH_pt);
    //if found; thats it
    if (aPoint == NULL) {
      aPoint = new(SH_Geometries) SH_Point;
      aPoint.update()->initWKS(&C_pt, inAccuracy);
    }
    constituent.add(aPoint);
  }
}

void SH_Line_String::finalize() {
  Ref<SH_Point> aPoint;
  for (aPoint = constituent.delete_one();
       aPoint != NULL;
       aPoint = constituent.delete_one()) {
    if (aPoint->partOf.get_size() == 0) {
      aPoint.update()->finalize();
      SH_DO(aPoint.destroy());
    }
  }
  return;
} // Destructor

void SH_Line_String::getCoordinateGeometry(CoordGeom &inC_cg) const {
  Ref<SH_Point> aPoint;
  CoordGeom C_cg;
  LineString C_ls;
  Ring C_ri;
  Coordinate C_pt;
  int i;
  switch (Interpretation) {
    case GO_Lines:
      for (aPoint = constituent.get_elt(i=0);
   aPoint != NULL;
   aPoint = constituent.get_elt(++i)) {
aPoint->getCoordinateGeometry(C_cg);
C_ls.append_elt(C_cg.get_point());
      }
      inC_cg.set_CoordGeomType(CoordLineStringType);
      inC_cg.set_line_string() = C_ls;
      return;
    case GO_Polygon:
      for (aPoint = constituent.get_elt(i=0);
```

```
   aPoint != NULL;
   aPoint = constituent.get_elt(++i)) {
aPoint->getCoordinateGeometry(C_cg);
C_ri.append_elt(C_cg.get_point());
      }
      inC_cg.set_CoordGeomType(CoordRingType);
      inC_cg.set_ring() = C_ri;
      return;
    default:
      break;
  }
  return;
}// Returns the actual geometry used.

// ----------------------------------------------------------------
// Interface name: SH_Polygon
// Description:
//
// ----------------------------------------------------------------
void SH_Polygon::initialize(Ref<SH_Accuracy> inAccuracy){
  // Constructor: dimension=2, set interpretation
  // For each ring in Polygon-sequence, create LineString and
  // keep track of the relations
  CoordDimension = 4;
  Dimension = 2;
  Interpretation = GO_Polygon;
  Planar = TRUE;
  Closed = TRUE;
  Simple = TRUE;
  accuracyRef = inAccuracy;
}
void SH_Polygon::initWKS(Polygon* inC_pg, Ref<SH_Accuracy> inAccuracy){
  // Constructor: dimension=2, set interpretation
  // For each ring in Polygon-sequence, create LineString and
  // keep track of the relations
  CoordDimension = 4;
  Dimension = 2;
  Interpretation = GO_Polygon;
  Planar = TRUE;
  Closed = TRUE;
  Simple = TRUE;
  accuracyRef = inAccuracy;
  Coordinate C_pt;
  Ring C_ri;
  Ref<SH_Line_String> aLineString;
  for (unsigned int i = 0; i < inC_pg->get_size(); i++) {
    // Get a ring
    // For each ring create Line_String
    C_ri = inC_pg->get_elt(i);
    aLineString = new(SH_Geometries) SH_Line_String;
    aLineString.update()->createRing(&C_ri, inAccuracy);
    constituent.add(aLineString);
  }
}

void SH_Polygon::finalize() {
  Ref<SH_Line_String> aLineString;
  for (aLineString = constituent.delete_one();
       aLineString != NULL;
       aLineString = constituent.delete_one()) {
    if (aLineString->partOf.get_size() == 0) {
      aLineString.update()->finalize();
      SH_DO(aLineString.destroy());
    }
```

```
  }
} // Destructor


void SH_Polygon::getCoordinateGeometry(CoordGeom &inC_cg) const {
  int i;
  CoordGeom C_cg;
  Polygon C_pg;
  CoordGeomCollection C_cgc;
  boolean lineString = FALSE, ring = FALSE;
  Ref<SH_Line_String> aLine;
  for (aLine = constituent.get_elt(i=0);
       aLine != NULL;
       aLine = constituent.get_elt(++i)) {
    // TODO if linestring or ring....
    aLine->getCoordinateGeometry(C_cg);
    switch (C_cg.get_CoordGeomType()) {
      case CoordLineStringType:
if (ring)
  return; // shouldn't happen
C_cgc.append_elt(C_cg);
lineString = TRUE;
break;
      case CoordRingType:
if (lineString)
  return; // shouldn't happen
C_pg.append_elt(C_cg.get_ring());
ring = TRUE;
break;
      default:
return;  // shouldn't happen either...
    }
  }
  if (lineString) {
    inC_cg.set_CoordGeomType(CoordGeomCollectionType);
    inC_cg.set_cgc() = C_cgc;
    return;
  }
  if (ring) {
    inC_cg.set_CoordGeomType(CoordPolygonType);
    inC_cg.set_polygon() = C_pg;
  }
  return;
} // Returns the actual geometry used.

// ----------------------------------------------------------------
// Class name: SH_Indices
// Description:
//
// ----------------------------------------------------------------
void SH_Indices::initialize() const {
  SH_DO(aSpaceRefInd.init(UniqueBTree));
  SH_DO(aPointInd.init(UniqueBTree));
  SH_DO(serialNoInd.init(UniqueBTree));
  SH_DO(themeInd.init(BTree));
  SH_DO(geomTypeInd.init(BTree));
} // Constructor

void SH_Indices::newPoint(A_Point inSH_pt, Ref<SH_Point> inaPoint) const
{
  SH_DO(aPointInd.insert(inSH_pt, inaPoint));
}

Ref<SH_Point> SH_Indices::findPoint(A_Point inSH_pt) const {
```

```
  Ref<SH_Point> aPoint;
  bool found;
  SH_DO(aPointInd.find(inSH_pt, aPoint, found));
  if (found) return aPoint;
  else return NULL;
}

void SH_Indices::killPoint(A_Point inSH_pt) const {
  int count;
  SH_DO(aPointInd.remove(inSH_pt, count));
}

void SH_Indices::newRefsys(char* inSpaceRef,
      Ref<SH_Spatial_Ref_System> inaSpaceRef) const
{
  SH_DO(aSpaceRefInd.insert(inSpaceRef, inaSpaceRef));
}

Ref<SH_Spatial_Ref_System> SH_Indices::findRefsys(char* inSpaceRef) const
{
  Ref<SH_Spatial_Ref_System> aSpaceRef;
  bool found = FALSE;
  SH_DO(aSpaceRefInd.find(inSpaceRef, aSpaceRef, found));
  if (found) return aSpaceRef;
  else return NULL;
}

void SH_Indices::killRefsys(char* inSpaceRef) const {
  int count;
  SH_DO(aSpaceRefInd.remove(inSpaceRef,count));
}

void SH_Indices::newSerialNo(long inSerialNo,
Ref<SH_Geometry> inaGeometry) const
{
  SH_DO(serialNoInd.insert(inSerialNo, inaGeometry));
}

Ref<SH_Geometry> SH_Indices::findSerialNo(long inSerialNo) const
{
  Ref<SH_Geometry> aGeometry;
  bool found = FALSE;
  SH_DO(serialNoInd.find(inSerialNo, aGeometry, found));
  if (found) return aGeometry;
  else return NULL;
}
void SH_Indices::killSerialNo(long inSerialNo) const
{
  int count;
  W_IGNORE(serialNoInd.remove(inSerialNo,count));
}

void SH_Indices::killAllSerialNoInx() const {
  long low_end=99999,high_end=1;
  IndexScanIter<long,Ref<SH_Geometry> > iter(serialNoInd);
  SH_DO(iter.next());
  for ( ; !iter.eof; SH_DO(iter.next())) {
    if (low_end > iter.cur_key)
      low_end = iter.cur_key;
    if (high_end < iter.cur_key)
      high_end = iter.cur_key;
  }
  for (long i = low_end; i <= high_end; i++) {
        killSerialNo(i);
```

```
  }
}

void SH_Indices::newThemeInd(long inThemeInd,
Ref<SH_Feature> inaFeature) const
{
  SH_DO(themeInd.insert(inThemeInd, inaFeature));
}

void SH_Indices::killThemeInd(long inThemeInd,
      Ref<SH_Feature> inaFeature) const
{
  SH_DO(themeInd.remove(inThemeInd,inaFeature));
}

void SH_Indices::newGeomTypeInd(char* inGeomTypeInd,
Ref<SH_Feature> inaFeature) const
{
  SH_DO(geomTypeInd.insert(inGeomTypeInd, inaFeature));
}

void SH_Indices::killGeomTypeInd(char* inGeomTypeInd,
      Ref<SH_Feature> inaFeature) const
{
  SH_DO(geomTypeInd.remove(inGeomTypeInd,inaFeature));
}
```

Vedlegg O Metodeimplementasjon; PG_Objects.h

```
// ****************************************************************
// File name: PG_Objects.h
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Classes used in cooperation with Postgres
// database system
// Class-library of wxWindows is used with permission.
// The Postgres databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 01.01.97 Created
// ****************************************************************
#ifndef PG_OBJECTS
#define PG_OBJECTS
class PG_Query_Manager {
  private:
  void createRing(Coordinate* inC_pt_zero,
  Ring& inC_ri_theRing,
  long inEfz,  long inNfz,
  long inHigh,  long inBroad);
  public:
  PG_Query_Manager() {;}
  long loadTestFig(long inNo); // Creates inNo double-polygons
  void getGeometry(Geometry& inC_gm); // First version;
};

void aPGFeature_initWKS(char* inName, char* inDescription,
long inType, char* inSomeStuff,
Geometry* inC_gm, char* inAccuracyRef,
char* aSpaceRef);
char* aPGPoint_initWKS(Coordinate* inPt, char* inAccuracyRef);
char* aPGPoint_findOrAdd(Coordinate* inC_pt, char* inAccuracy);
char* aPGLineString_initWKS(LineString* inC_ls, char* inAccuracyRef);
char* aPGLineString_createRing(Ring* inC_ri, char* inAccuracyRef);
char* aPGPolygon_initWKS(Polygon* inC_pg,char* inAccuracyRef);

#endif
```

Vedlegg P Metodeimplementasjon; PG_Objects.C

```
// ****************************************************************
// File name: PG_Objects.C
// Project: GO
// Copyright: IDT, NTNU
// Language used: C++
// Short description: Implementation of methods used in cooperation
// with Postgres-database
// Class-library of wxWindows is used with permission.
// The Postgres databasesystem is used with permission,
// Modified:
// who/when/what: Knut Aage Aksnes, 01.01.97 Created
// ****************************************************************
#include <stdlib.h>
#include “wx.h”
#include “SH_Objects.h”
#include “PG_Objects.h”
#include “GO.h”
#include “PGDB.h”
extern PGDB pgDB;
extern char* testSpaceRef;
extern char* SpaceRef;
extern MyFrame* frame;

// ----------------------------------------------------------------
// Class name: PG_Query_Manager
// Description: Act as link between GO-application and
//Postgres-database
// ----------------------------------------------------------------
// Creates two squares that connect, and act as a compound polygon.
// Using WKS in all work.
long PG_Query_Manager::loadTestFig(long inNo) {
  // Input parameter for number of objects to create
  int  generated = 1, inner_limit, outer_limit;
  char *name = “aHouse”;
  long east,north,height,date; // Part of a coordinate, in this sequence
  long east_from_zero, north_from_zero;
  long polygon_height, polygon_width;
  // Coordinate-based variables
  Coordinate C_pt_zero;
  Ring       C_ri;
  Polygon    C_pg;
  Geometry   C_gm;
  // Object references
  char* aSpaceRef = new char[20];
  char* anAccuracyRef = new char[20];
  // start a transaction block
  if (pgDB.data->exec(“BEGIN”) != PGRES_COMMAND_OK) {
     *frame->text_window << “BEGIN command failed\n”;
     pgDB.CloseDB();
     return NOTOK;
   }
  // Create Spatial Reference System
  sprintf(pgDB.pgquery,”INSERT INTO Spatial_Ref_System
```

```
 (SP_name, linear_unit, angular_unit, prime_meridian, ellipsoide, GCS,
  CT, zone, origoN, origoOE, maxN, maxOE, minN, minOE, PC, temporalSystem,
  spaceDimension, timeDimension, totalDimension)
 VALUES ('%s',1.0,'Degree','Greenwich','GauzzKrugers','WGS84',
        'Transverse Mercator',99,72000,62000,73000,63000,72000,62000,
        'WGS/Europeen datum ED50','YYMMDD',3,1,4)" ,testSpaceRef);
 if (pgDB.data->exec(pgDB.pgquery)  != PGRES_COMMAND_OK) {
   *frame->text_window << "Insert Spatial_Ref_System command(1) failed\n";
   *frame->text_window << pgDB.data->errormessage();
   *frame->text_window << "\n";
   pgDB.CloseDB();
   return NOTOK;
 }
 strcpy(aSpaceRef,pgDB.data->OidStatus());
 //aSpaceRef now holds a unique identifier of this tuple...

 // Create Accuracy object
 sprintf(pgDB.pgquery,"INSERT INTO Accuracy(accuracy, measureMethod)
                       VALUES (82,36)");
 if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK)
 {
   *frame->text_window << "Insert Accuracy(1) failed\n";
   *frame->text_window << pgDB.data->errormessage();
   *frame->text_window << "\n";
   pgDB.CloseDB();
   return NOTOK;
 }
 strcpy(anAccuracyRef,pgDB.data->OidStatus());
 // anAccuracyRef now holds an id of the current Accuracy-tuple
 // end the transaction
 pgDB.data->exec("END");

  // Establich zero-point; Never to be changed I think!
 C_pt_zero.append_elt((east = 0));
 C_pt_zero.append_elt((north = 0));
 C_pt_zero.append_elt((height = 0));
 C_pt_zero.append_elt((date = 991122));

 // Fills a square with 15x15 double polygons (max 225)
 // or up to 30 x 100 (max 3000)
 if (inNo < 226)
   inner_limit = outer_limit = 15;
 else {
   inner_limit = 30;
   outer_limit = 100;
 }
 // Supposed to create a number of polygons, given by parameter inNo
 for (int outer = 0; outer <= outer_limit; outer++) {
   north_from_zero = 10 + 60*outer + outer%2;
   // Left-bottom corner northern displacement from zero
   for (int inner = 0; inner <= inner_limit; inner++) {
     east_from_zero = 10 + 60*inner + inner%3;
     // Left-bottom corner eastern displacement from zero
     polygon_height = 20 + inner%5;
     polygon_width = 20 + inner%7;
     createRing(&C_pt_zero, C_ri, east_from_zero, north_from_zero,
polygon_height, polygon_width);
     C_pg.append_elt(C_ri);
     // Empty the coordinate-points from ring
     C_ri.set_size(0);

     // Another polygon adjacent to previous
     east_from_zero += polygon_width; // Push the corner to the right
     polygon_height = 30 + inner%3;
```

```
      polygon_width = 30 + inner%6;
      createRing(&C_pt_zero, C_ri, east_from_zero, north_from_zero,
 polygon_height, polygon_width);
      C_pg.append_elt(C_ri);
      // Empty the coordinate-points from ring (again)
      C_ri.set_size(0);

      // Create a feature
      C_gm.reference_system_name.strcpy(testSpaceRef);
      C_gm.coordinate_geometry.set_CoordGeomType(CoordPolygonType);
      C_gm.coordinate_geometry.set_polygon() = C_pg;

      // start a transaction block
      if (pgDB.data->exec("BEGIN") != PGRES_COMMAND_OK) {
*frame->text_window << "BEGIN command failed\n";
pgDB.CloseDB();
return NOTOK;
      }
      // Should test for success, though......
      aPGFeature_initWKS(name,name,generated,"someStuff"
,&C_gm,anAccuracyRef,aSpaceRef);
      // end the transaction
      pgDB.data->exec("END");

      // Empty rings from the polygon
      C_pg.set_size(0);
      generated +=1;
      if (generated > inNo) return(0);
    }
  }
  // That should be all
  delete aSpaceRef;
  delete anAccuracyRef;
  delete name;
  return(0);
}

void PG_Query_Manager::createRing(Coordinate *inC_pt_zero,
     Ring &inC_ri_theRing,
      long inEfz, long inNfz, long inHigh,
      long inBroad) {
  Coordinate C_pt;
  long aPoint_efz, aPoint_nfz, aPoint_height, time_value;
  //  First point
  aPoint_efz    = inC_pt_zero->get_elt(0) + inEfz;
  aPoint_nfz    = inC_pt_zero->get_elt(1) + inNfz;
  aPoint_height = inC_pt_zero->get_elt(2);
  time_value    = inC_pt_zero->get_elt(3);

  C_pt.append_elt(aPoint_efz); // East-value
  C_pt.append_elt(aPoint_nfz); // North-value
  C_pt.append_elt(aPoint_height); // Height above sealevel (not used)
  C_pt.append_elt(time_value); // Time-value
  inC_ri_theRing.append_elt(C_pt);

  // Second Point
  aPoint_nfz += inHigh;
  C_pt.write_elt(1)= aPoint_nfz; // Norht-value
  inC_ri_theRing.append_elt(C_pt);

  // Third Point
  aPoint_efz += inBroad;
  C_pt.write_elt(0)= aPoint_efz; // Norht-value
  inC_ri_theRing.append_elt(C_pt);
```

```
  // Forth and last  Point
  aPoint_nfz -= inHigh;
  C_pt.write_elt(1)= aPoint_nfz; // Norht-value
  inC_ri_theRing.append_elt(C_pt);

  return;
}

void PG_Query_Manager::getGeometry(Geometry &inC_gm) {
  char* prevPolygon = new char[20];
  char* prevLineOid = new char[20];
  int east=0, north=0, height=0, date=0, nTuples=0, i=0;
  Coordinate C_pt;
  Ring C_ri;
  Polygon C_pg;
  CoordGeomCollection C_cgc;
  CoordGeom C_cg;
  // fields:
  // 0: thePolygon 1: LS_seq_no 2: theLine 3: PT_seq_no
  // 4: east 5: north 6: height 7: date
  // iterator i
  // Declare Cursorp
  sprintf(pgDB.pgquery,"DECLARE aGeometry_cursor CURSOR FOR
 SELECT C.oid AS thePolygon, D.LS_seq_no, E.LS_constituent,
   E.PT_seq_no, F.east, F.north, F.height, F.date
 FROM Spatial_Ref_System A, Geometric_Property B, Geometry* C,
   Polygon_Lines D, Line_Points E, Point F
 WHERE A.SP_name = '%s' AND A.oid = B.spaceRef
   AND B.geometry = C.oidAND C.dimension = %i
   AND C.oid = D.PG_constituent AND D.LS_partOf = E.LS_constituent
   AND E.PT_partOf = F.oid
 ORDER BY thePolygon, LS_seq_no, PT_seq_no",SpaceRef,2);
  // 2 is the dimension
  if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK) {
    *frame->text_window << "DECLARE CURSOR command failed (GetGM:1)\n";
    *frame->text_window << pgDB.data->errormessage();
    *frame->text_window << "\n";
    pgDB.CloseDB();
    delete prevPolygon;
    delete prevLineOid;
    return;
  }
  // Fetch first element
  // Iterate through the tuples returned, i.e. all Geometries
  if (pgDB.data->exec("FETCH ALL IN aGeometry_cursor")
      != PGRES_TUPLES_OK) {
    *frame->text_window << "First FETCH command failed (GetGM:1)\n";
    *frame->text_window << pgDB.data->errormessage();
    *frame->text_window << "\n";
    pgDB.CloseDB();
    delete prevPolygon;
    delete prevLineOid;
    return;
  }
  nTuples = 0;
  i = 0;
  strcpy(prevLineOid,pgDB.data->getvalue(i,2));
  strcpy(prevPolygon,pgDB.data->getvalue(i,0));
  nTuples = pgDB.data->ntuples();
  C_cg.set_CoordGeomType(CoordPolygonType);
  // Treating all tuples in CURSOR
  while (i < nTuples) {
    // Treating  each Polygon by itself
```

```
    while (!strcmp(pgDB.data->getvalue(i,0),prevPolygon)) {
      // Treating each LineString by itself, get the points...
      while (!strcmp(pgDB.data->getvalue(i,2),prevLineOid)) {
// At the innermost level, got the point
StringToInt(pgDB.data->getvalue(i,4),&east);
StringToInt(pgDB.data->getvalue(i,5),&north);
StringToInt(pgDB.data->getvalue(i,6),&height);
StringToInt(pgDB.data->getvalue(i,7),&date);
C_pt.append_elt((long)east);
C_pt.append_elt((long)north);
C_pt.append_elt((long)height);
C_pt.append_elt((long)date);
C_ri.append_elt(C_pt);
C_pt.set_size(0);
// get next element
i++;
if (i == nTuples)
  break;
      }
      if (i == nTuples)
break;
      strcpy(prevLineOid,pgDB.data->getvalue(i,2));
      C_pg.append_elt(C_ri);
      C_ri.set_size(0);
      // Ready for next Ring
    }
    if (i == nTuples)
      break;
    strcpy(prevPolygon,pgDB.data->getvalue(i,0));
    C_cg.set_polygon() = C_pg;
    C_pg.set_size(0);
    C_cgc.append_elt(C_cg);
    // Ready for next Polygon
  }
  // Read all tuples; finished
  inC_gm.coordinate_geometry.set_CoordGeomType(CoordGeomCollectionType);
  inC_gm.coordinate_geometry.set_cgc()  = C_cgc;
  // some deletes maybe...
  delete prevPolygon;
  delete prevLineOid;
  return;
}

// ----------------------------------------------------------------
// Class name: PG_Feature
// ----------------------------------------------------------------
void aPGFeature_initWKS(char *inName, char *inDescription,
long inType, char *inSomeStuff,
Geometry * inC_gm,char* inAccuracyRef,
char* inSpaceRef) {
  // Object references
  char* thisGeomP = new char[20];
  char* thisFeature = new char[20];
  // Update some attributes,
  // this on also sets partOf and constituent
  sprintf(pgDB.pgquery,"INSERT INTO Feature
          (FT_name, description, aType, someStuff)
          VALUES ('%s','%s',%i,'%s')",
  inName,inDescription,(int)inType,inSomeStuff);
  if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK)
  {
      *frame->text_window << "Insert Feature(1) failed\n";
      *frame->text_window << pgDB.data->errormessage();
      *frame->text_window << "\n";
```

```
      pgDB.CloseDB();
      delete thisGeomP;
      delete thisFeature;
      return;
  }
  strcpy(thisFeature,pgDB.data->OidStatus());
  //  thisFeature now holds an id of the current Feature

  // Create Geometric-Property, and establish relationship
  // Feature - Spatial_Ref_System
  sprintf(pgDB.pgquery,"INSERT INTO Geometric_Property(feature,spaceRef)
     VALUES ('%s','%s')",thisFeature, inSpaceRef);
  if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK)
  {
    *frame->text_window << "Insert Geometric_Property(1) failed\n";
    *frame->text_window << pgDB.data->errormessage();
    *frame->text_window << "\n";
    pgDB.CloseDB();
    delete thisGeomP;
    delete thisFeature;
    return;
  }
  strcpy(thisGeomP,pgDB.data->OidStatus());
  //  thisGeomP now holds an id of the current Geometric_Property
  // Creates the correct Geometry
  char* thisGeometry = new char[20];
  char* aGeometry;
  Coordinate C_pt;
  LineString C_ls;
  Polygon C_pg;
  switch (inC_gm->coordinate_geometry.get_CoordGeomType()) {
    case CoordCoordinateType:
      C_pt = inC_gm->coordinate_geometry.get_point();
      // Look for the point....
      aGeometry = aPGPoint_findOrAdd(&C_pt,inAccuracyRef);
      if (aGeometry)  {   // NULL returned when error
strcpy(thisGeometry,aGeometry);
delete aGeometry;
      }
      else {
delete thisGeometry;
delete thisGeomP;
delete thisFeature;
return;
      }
      // add reference to Point to Geometric_Property.geometry
      sprintf(pgDB.pgquery,"SELECT aGeomP_setGeometry('%s','%s')",
  thisGeomP,thisGeometry);
      if (pgDB.data->exec(pgDB.pgquery) != PGRES_TUPLES_OK) {
*frame->text_window << "aGeomP_setGeometry(1) failed\n";
*frame->text_window << pgDB.data->errormessage();
*frame->text_window << "\n";
pgDB.CloseDB();
delete thisGeometry;
delete thisGeomP;
delete thisFeature;
return;
      }
      break;
    case CoordLineStringType:
      // Choosed not to test for existing lines or part of lines
      // Means they might be duplicated as Line_String-objects, but
      // on the other hand the direction of a line will be defined
      // with respect to each Line_String.
```

```
      C_ls = inC_gm->coordinate_geometry.get_line_string();
      aGeometry = aPGLineString_initWKS(&C_ls,inAccuracyRef);
      if (aGeometry) {// NULL returned when error
strcpy(thisGeometry,aGeometry);
delete aGeometry;
      }
      else {
delete thisGeometry;
delete thisGeomP;
delete thisFeature;
return;
      }
      // add reference to LineString to Geometric_Property.geometry
      sprintf(pgDB.pgquery,"SELECT aGeomP_setGeometry('%s','%s')",
  thisGeomP,thisGeometry);
      if (pgDB.data->exec(pgDB.pgquery) != PGRES_TUPLES_OK) {
*frame->text_window << "aGeomP_setGeometry(2) failed\n";
*frame->text_window << pgDB.data->errormessage();
*frame->text_window << "\n";
pgDB.CloseDB();
delete thisGeometry;
delete thisGeomP;
delete thisFeature;
return;
      }
      break;
    case CoordRingType:
      //    Ring ring;
      break;
    case CoordPolygonType:
      C_pg = inC_gm->coordinate_geometry.get_polygon();
      aGeometry = aPGPolygon_initWKS(&C_pg,inAccuracyRef);
      if (aGeometry) { // NULL returned when error
strcpy(thisGeometry,aGeometry);
delete aGeometry;
      }
      else {
delete thisGeometry;
delete thisGeomP;
delete thisFeature;
return;
      }
      // add reference to Polygon to Geometric_Property.geometry
      sprintf(pgDB.pgquery,"SELECT aGeomP_setGeometry('%s','%s')",
  thisGeomP,thisGeometry);
      if (pgDB.data->exec(pgDB.pgquery) != PGRES_TUPLES_OK) {
*frame->text_window << "aGeomP_setGeometry(3) failed\n";
*frame->text_window << pgDB.data->errormessage();
*frame->text_window << "\n";
pgDB.CloseDB();
delete thisGeometry;
delete thisGeomP;
delete thisFeature;
return;
      }
      break;
    default:
      break;
  }
  delete thisGeometry;
  delete thisGeomP;
  delete thisFeature;
  return;
}
```

```
// ---------------------------------------------------------------
// Class name: PG_Point
// ---------------------------------------------------------------
// Return oid of the created point

char* aPGPoint_initWKS(Coordinate *inC_pt, char* inAccuracy) {
  //update attributes of Point
  char* thisPoint = new char[20];
  sprintf(pgDB.pgquery,"INSERT INTO Point
          (coordDimension, planar, closed, simple, accuracyRef,
          dimension, interpretation, east,north,height,date)
          VALUES (4,\'TRUE\',\'TRUE\',\'TRUE\',\'%s',0,%i,
          %i,%i,%i,%i)",
  inAccuracy, GO_Point, (int)inC_pt->get_elt(0), (int)inC_pt->get_elt(1),
  (int)inC_pt->get_elt(2), (int)inC_pt->get_elt(3) );
  if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK) {
    *frame->text_window << "Insert Point(1) failed\n";
    *frame->text_window << pgDB.data->errormessage();
    *frame->text_window << "\n";
    pgDB.CloseDB();
    delete thisPoint;
    return NULL;
  }
  strcpy(thisPoint,pgDB.data->OidStatus());
  return thisPoint;
}
// Finds or inserts a new point
char* aPGPoint_findOrAdd(Coordinate* inC_pt, char* inAccuracy) {
  char* thisPoint = new char[20];
  int east,north,height,date;
  east = (int)inC_pt->get_elt(0);
  north = (int)inC_pt->get_elt(1);
  height = (int)inC_pt->get_elt(2);
  date = (int)inC_pt->get_elt(3);
  sprintf(pgDB.pgquery,"SELECT oid FROM Point
     WHERE east = %i and north = %i and height = %i and date = %i",
  east, north, height, date);
  switch (pgDB.data->exec(pgDB.pgquery)) {
    case PGRES_TUPLES_OK:
      switch (pgDB.data->ntuples()) {
case 0: // Point is not found...
  char* thisPoint_temp;
  thisPoint_temp = aPGPoint_initWKS(inC_pt,inAccuracy);
  if (thisPoint_temp) {
    strcpy(thisPoint,thisPoint_temp);
    delete thisPoint_temp;
    break;
  }
  else {
    // Error when inserting a new point
    delete thisPoint;
    return NULL;
  }
case 1: // One point is found, good!
  strcpy(thisPoint,pgDB.data->getvalue(0,0));
  break;
default:
  // Some mysterious error!
  *frame->text_window << "FindOrAdd strange (1), PQntuples= \n";
  *frame->text_window << pgDB.data->ntuples();
  *frame->text_window << "\n";
  delete thisPoint;
  return NULL;
```

```
      }
      break;
    default:
      *frame->text_window << "FindOrAdd failed (1)\n";
      *frame->text_window << pgDB.data->errormessage();
      *frame->text_window << "\n";
      pgDB.CloseDB();
      delete thisPoint;
      return NULL;
  }
  return thisPoint;
}


// ----------------------------------------------------------------
// Interface name: PG_Line_String
// ----------------------------------------------------------------
// Return oid of the created LineString object
char* aPGLineString_initWKS(LineString* inC_ls, char* inAccuracy) {
  // Constructor: dimension=1, set interpretation(line or ring)
  // For each point in LineString-sequence, establish
  // relation to existing or new Point or
  // update attributes of LineString
  char* thisLineString = new char[20];
  char* thisPoint = new char[20];
  char* thisPoint_temp;

  sprintf(pgDB.pgquery,"INSERT INTO Line_String
    (coordDimension, planar, closed, simple, accuracyRef,
    dimension, interpretation)
    VALUES (4,\'TRUE\',\'TRUE\',\'TRUE\','%s',1,%i)",
          inAccuracy, GO_Lines);
  if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK) {
      *frame->text_window << "LineString insert (initWKS:1) failed\n";
      *frame->text_window << pgDB.data->errormessage();
      *frame->text_window << "\n";
      pgDB.CloseDB();
      delete thisLineString;
      delete thisPoint;
      return NULL;
  }
  strcpy(thisLineString,pgDB.data->OidStatus());
  // Got the oid...

  Coordinate C_pt;
  for (unsigned int i = 0; i < inC_ls->get_size(); i++) {
    C_pt = inC_ls->get_elt(i);
    // Look for the point....
    thisPoint_temp = aPGPoint_findOrAdd(&C_pt,inAccuracy);
    if (thisPoint_temp)  {   // returned NULL when error
      strcpy(thisPoint,thisPoint_temp);
      delete thisPoint_temp;
    }
    else {
      delete thisLineString;
      delete thisPoint;
      return NULL;
    }
    // Update relation Line_Points
    sprintf(pgDB.pgquery,"INSERT INTO Line_Points (LS_constituent,
       PT_partOf,PT_seq_no) VALUES ('%s', '%s', %i)",
    thisLineString, thisPoint, i);
    if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK) {
      *frame->text_window << "Insert Line_Points(initWKS:1) failed\n";
```

```
      *frame->text_window << pgDB.data->errormessage();
      *frame->text_window << "\n";
      pgDB.CloseDB();
      delete thisLineString;
      delete thisPoint;
      return NULL;
    }
  }
  delete thisPoint;
  return thisLineString;
}

char* aPGLineString_createRing(Ring* inC_ri, char* inAccuracy){
  // Constructor of rings
  // dimension=1, set interpretation(line or ring)
  // For each point in LineString-sequence, establish
  // relation to existing or new Point
  char* thisLineString = new char[20];
  char* thisPoint = new char[20];
  char* thisPoint_temp;

  sprintf(pgDB.pgquery,"INSERT INTO Line_String
         (coordDimension, planar, closed, simple, accuracyRef,
  dimension, interpretation)
  VALUES (4,\'TRUE\',\'TRUE\',\'TRUE\','%s',1,%i)",
          inAccuracy, GO_Polygon);
  if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK) {
    *frame->text_window << "LineString insert (createRing:1) failed\n";
    *frame->text_window << pgDB.data->errormessage();
    *frame->text_window << "\n";
    pgDB.CloseDB();
    delete thisLineString;
    delete thisPoint;
    return NULL;
  }
  strcpy(thisLineString,pgDB.data->OidStatus());
  // Got the oid...

  Coordinate C_pt;
  for (unsigned int i = 0; i < inC_ri->get_size(); i++) {
    C_pt = inC_ri->get_elt(i);
    // Look for the point....
    thisPoint_temp = aPGPoint_findOrAdd(&C_pt,inAccuracy);
    if (thisPoint_temp)  {   // returned NULL when error
      strcpy(thisPoint,thisPoint_temp);
      delete thisPoint_temp;
    }
    else {
      delete thisLineString;
      delete thisPoint;
      return NULL;
    }
    // Update relation Line_Points
    sprintf(pgDB.pgquery,"INSERT INTO Line_Points
       (LS_constituent,PT_partOf,PT_seq_no)
       VALUES ('%s', '%s', %i)",
    thisLineString, thisPoint, i);
    if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK) {
      *frame->text_window << "Insert Line_Points(createRing:1) failed\n";
      *frame->text_window << pgDB.data->errormessage();
      *frame->text_window << "\n";

      pgDB.CloseDB();
      delete thisLineString;
```

```
      delete thisPoint;
      return NULL;
    }

  }
  delete thisPoint;
  return thisLineString;
}


// ----------------------------------------------------------------
// Interface name: PG_Polygon
// ----------------------------------------------------------------
// Return oid of the created Polygon
char* aPGPolygon_initWKS(Polygon* inC_pg, char* inAccuracy){
  // Constructor: dimension=2, set interpretation
  // For each ring in Polygon-sequence, create LineString and
  // keep track of the relations
  char* thisPolygon = new char[20];
  char* thisLineString = new char[20];
  char* thisLineString_temp;;
  sprintf(pgDB.pgquery,"INSERT INTO Polygon
         (coordDimension, planar, closed, simple, accuracyRef,
  dimension, interpretation)
  VALUES (4,\'TRUE\',\'TRUE\',\'TRUE\','%s',2,%i)",
          inAccuracy, GO_Polygon);

  if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK) {
    *frame->text_window << "Polygon insert (initWKS:1) failed\n";
    *frame->text_window << pgDB.data->errormessage();
    *frame->text_window << "\n";
    pgDB.CloseDB();
    delete thisPolygon;
    delete thisLineString;
    return NULL;
  }
  strcpy(thisPolygon,pgDB.data->OidStatus());
  Coordinate C_pt;
  Ring C_ri;
  for (unsigned int i = 0; i < inC_pg->get_size(); i++) {
    // Get a ring
    // For each ring create Line_String
    C_ri = inC_pg->get_elt(i);
    thisLineString_temp = aPGLineString_createRing(&C_ri,inAccuracy);
    if (thisLineString_temp) { // NULL returned when error
      strcpy(thisLineString, thisLineString_temp);
      delete thisLineString_temp;
    }
    else { // Error when creating Line_String, better get out of here..
      delete thisPolygon;
      delete thisLineString;
      return NULL;
    }
    // Update relations in LineString and Polygon
    sprintf(pgDB.pgquery,"INSERT INTO Polygon_Lines
           (PG_constituent, LS_partOf, LS_seq_no)
           VALUES ('%s','%s',%i)",
    thisPolygon,thisLineString,i);
    if (pgDB.data->exec(pgDB.pgquery) != PGRES_COMMAND_OK) {
      *frame->text_window << "Insert Polygon_Lines(initWKS:1) failed\n";
      *frame->text_window << pgDB.data->errormessage();
      *frame->text_window << "\n";
      pgDB.CloseDB();
      delete thisPolygon;
```

```
      delete thisLineString;
      return NULL;
    }
  }
  delete thisLineString;
  return thisPolygon;
}
```

Vedlegg Q Makefile for GO

```
#
# File:Makefile
# Author:Knut Aage Aksnes
# Created:1996
# Updated:
# Description:Makefile for GO

#Purify
PURIFY = /progs/pure/purify-3.0a-sunos4/purify -linker=/bin/ld -collector=/sesz_arkiv/progs/lib/gcc-lib/sparc-sun-
sunos4.1.2/2.7.2/ld -chain-length=”15”

#Directories for Motif
MOTIFDIR = /home/inga/ttbwt/Motif1.2
MOTIFINCLUDE = -I$(MOTIFDIR)/include
MOTIFLIB= -L$(MOTIFDIR)/lib


#Directories for wxWindows
WXDIR = /sesz_arkiv/progs/wxWindows
WXINCLUDE = -I$(WXDIR)/include/base -I$(WXDIR)/include/x -I$(WXDIR)/utils/image/src
WXLIB= -L$(WXDIR)/lib
WXLIBNAME = -lwx_motif

#Directories for Shore
#SHDIR = /home/storlind/d/oodb/shore
SHDIR = /sesz_arkiv/shore
SHINCLUDE= -I$(SHDIR)/include
SHLIB= $(SHDIR)/lib
SDL= $(SHDIR)/bin/sdl
SHLIB= -L$(SHDIR)/lib
SHLIBNAME= -lshore

#Directories for Postgres95
PGDIR= /home/postgres95
PGSRCB= $(PGDIR)/src/backend
PGINCLUDE= -I$(PGDIR)/include
# -I$(PGSRCB) -I$(PGSRCB)/include
PGLIB= -L$(PGDIR)/lib
PGDEPLIBS= $(PGDIR)/lib/libpq.a $(PGDIR)/lib/libpq++.a
PGLIBNAME= -lpq -lpq++

#The lib’s and flag’s:
INCLUDE = $(MOTIFINCLUDE) $(WXINCLUDE) $(SHINCLUDE) $(PGINCLUDE)
CPPFLAGS = -Dwx_motif -ggdb -DDEBUG=’0’ -Wall
LDLIBS= $(MOTIFLIB) $(WXLIB) $(SHLIB) $(PGLIB)
LDNAMES= $(WXLIBNAME) $(SHLIBNAME) $(PGLIBNAME) -lXm -lXt -lX11 -lm

#And this application: GO
SH_SRCS= SH_defs.C SH_Objects.C GO.C MyDB.C SHDB.C PGDB.C PG_Objects.C
SH_HDRS= WKS.h SH_Objects.h GO.h MyDB.h PGDB.h SHDB.h PG_Objects.h
SH_OBJS= SH_defs.o GO.o SH_Objects.o MyDB.o SHDB.o PGDB.o PG_Objects.o
SDLFILES= WKS.sdl SH_Objects.sdl
SQLFILES= GO.sql
```

```
#Various dependencies
GO_DEPS= SH_Objects.h GO.h MyDB.h

GCC = /sesz_arkiv/progs/bin/g++
#I seriously hope the files with conn. to postgres may be compiled w g++..

OBJECTS=  objects/SH_defs.o objects/GO.o objects/SH_Objects.o objects/SHDB.o objects/PGDB.o objects/
MyDB.o objects/PG_Objects.o

.SUFFIXES:

all:objects GO

objects:
mkdir objects

GO: $(OBJECTS) $(PGDEPLIBS)
$(GCC) $(LDLIBS) -o GO $(OBJECTS) $(LDNAMES)
#Evt med Purify:
#$(PURIFY) $(GCC) $(LDLIBS) -o GO $(OBJECTS) $(LDNAMES)

objects/GO.o:       GO.C $(SH_HDRS)
$(GCC) -c $(CPPFLAGS) $(INCLUDE) -o $@ GO.C

objects/SHDB.o:       SHDB.C $(SH_HDRS)
$(GCC) -c $(CPPFLAGS) $(INCLUDE) -o $@ SHDB.C

objects/MyDB.o:       MyDB.C $(SH_HDRS)
$(GCC) -c $(CPPFLAGS) $(INCLUDE) -o $@ MyDB.C

objects/PGDB.o:       PGDB.C $(SH_HDRS)
$(GCC) -c $(CPPFLAGS) $(INCLUDE) -o $@ PGDB.C

objects/SH_defs.o:SH_Objects.h
$(GCC) -c $(CPPFLAGS) $(INCLUDE) -o $@ SH_defs.C

objects/SH_Objects.o: SH_Objects.C $(SH_HDRS)
$(GCC) -c $(CPPFLAGS) $(INCLUDE) -o $@ SH_Objects.C

objects/PG_Objects.o: PG_Objects.C $(SH_HDRS)
$(GCC) -c $(CPPFLAGS) $(INCLUDE) -o $@ PG_Objects.C

SH_Objects.h: SH_Objects.sdl WKS.h
$(SDL) -f -s SH_Objects.sdl -b SH_Objects -o SH_Objects.h -L -B

WKS.h: WKS.sdl
$(SDL) -f -s WKS.sdl -b  WKS -o WKS.h -L -B

sdl: WKS.h SH_Objects.h




#clean:
#rm -f $(OBJECTS) core

#cleanany:
#rm -f $(OBJECTS) GO core
```