Vestlandsforsking
Boks 163, 6851 Sogndal
Tlf. 57 67 61 50
Internett: www.vestforsk.no

VF-notat 8/2004

Konvertering av kommunale organisasjonsdata

Yngve Håkonsen

og

Thomas Sløk Tvedt

VF Notat

Tittel Konvertering av kommunale organisasjonsdata	**Rapportnummer** 8/2004
	Dato August 2004
	Gradering Open
Prosjekttittel Kommunal organisasjonsdatabase	**Antall sider** 50
	Prosjektnr 5251
Forskarar Yngve Håkonsen og Thomas Sløk Tvedt (forskingsass.), Svein Ølnes, Terje Aaberge og Nils Arne Hove (veiledere)	**Prosjektansvarlig** Ivar Petter Grøtte
Oppdragsgiver Kommunal- og regionaldepartementet	**Emneord** Konvertering Kommunedata XML SPSS

Sammendrag

Notatet er basert på en studentoppgave utarbeidet av Yngve Håkonsen og Thomas Sløk Tvedt som en del av prosjektet Kommunal organisasjonsdatabase. Som del av studentarbeidet tok de på seg oppgaven med å lage en overføringsrutine fra data lagret i statistikk-verktøyet SPSS og over til et XML-format. De har løst dette på en generell måte som har interesse utover dette prosjektet. Løsningen er en del av Vestlandsforskings oppdrag for KRD med å tilrettelegge kommunale organisasjonsdata på nettet.

Høgskolelektor Knut Erling Øien ved Høgskulen i Sogn og Fjordane har vært hoved-veileder for prosjektoppgaven mens forsker Svein Ølnes, forsker Terje Aaberge og systemutvikler Nils Arne Hove har vært veiledere for det videre arbeidet i prosjektet.

Andre publikasjoner fra prosjektet

ISSN: 0804-8835	**Pris:**

KONVERTERING AV KOMMUNALE ORGANISASJONSDATA

Prosjektoppgave skrevet av

Yngve Håkonsen
Thomas Sløk Tvedt

Conversion of organisation data from local government

Dette prosjektet er utført som et prosjektarbeid i kurset DA690 (15 stp) som avslutning på det toårige studiet i informasjonsbehandling ved Høgskulen i Sogn og Fjordane våren 2004. Hovedveileder har vært høgskulelektor Knut Erling Øien, Høgskulen i Sogn og Fjordane.

Sammendrag

"CONVERSION OF GOVERNMENT ORGANISATION DATA"

The Norwegian institute for city- and regional research (NIBR) carried out a survey in 2004 where they collected data about the organisation of local government on assignment for the Ministry of Local Government and Regional Development (KRD). This information is stored in SPSS – Statistical Product and Service Solutions. SPSS is an advanced statistical data-management and analysis tool, which means that the stored information is mainly available for researchers, scientists and others who are familiar with SPSS.

Our assignment from Vestlandsforskning AS was to convert data from the local government organisation survey 2004 to a more applicable format as part of the KRD-assignment.
This report shows how we converted data from SPSS to a Oracle 8i database (via XML) to obtain easy accessibility to the data. To solve this task we developed a Java application that takes any given set of data from SPSS as input and converts this data to a database (via XML). Not only did we solve our spesific problem, we also came out with a general solution for converting data from a survey (stored in SPSS) to a database. We believe that this part may be of general interest.

Sammendrag

”KONVERTERING AV KOMMUNALE ORGANISASJONSDATA”

Norsk institutt for by- og regionalforskning (NIBR) har på oppdrag fra Kommunal- og regionaldepartementet (KRD) gjennomført en spørreundersøkelse i 2004 der de har samlet inn data om organisering av kommuner. Disse dataene er lagret i SPSS – Statistical Product and Service Solutions. SPSS er et omfattende statistisk datahåndterings- og dataanalyseverktøy, noe som betyr at disse dataene først og fremst er tilgjengelig for forskere og andre med god kjennskap til SPSS.

Vi fikk i oppdrag fra Vestlandsforskning AS å konvertere de kommunale organisasjonsdataene fra spørreundersøkelsen 2004 til et mer anvendelig format som en del av KRD-oppdraget.
Denne rapporten tar for seg hvordan vi har konvertert disse dataene fra SPSS til en Oracle 8i database (via XML) med den hensikt å gjøre de lettere tilgjengelig. For å løse denne oppgaven har vi utviklet en Java applikasjon som tar et hvilket som helst datasett fra SPSS og konverterer disse dataene til en database (via XML). Vi har altså ikke bare kommet frem til en løsning på den spesifikke problemstillingen vi ble stilt ovenfor, vi har også kommet frem til en generell løsning som kan benyttes av andre som møter en lignende problemstilling. Vi mener derfor at dette prosjektet kan være av allmenn interesse.

Forord

Denne prosjektoppgaven er en avslutning på det toårige studiet i informasjonsbehandling ved Høgskulen i Sogn og Fjordane. Prosjektoppgaven er vektet med 15 studiepoeng, og har blitt gjennomført i Sogndal i perioden januar til juni 2004.

Oppgaven er utført for Vestlandsforskning AS på oppdrag fra Kommunal- og regionaldepartementet (KRD). Vi har lært mye etter å ha gjennomført dette prosjektet, og vi håper at oppdragsgiver er tilfreds med sluttproduktet.

Vi håper også at arbeidet vi har utført kan være til nytte for andre som møter en lignende problemstilling.

I tillegg til å takke vår hovedveileder, Knut Erling Øien, vil vi rette en spesiell takk til Jarle Håvik og Morten Simonsen for god veiledning gjennom prosjektet. Jarle Håvik har hjulpet oss med databaserelaterte spørsmål, og Morten Simonsen har hjulpet oss med spørsmål knyttet til programmering. Vi vil og takke Svein Ølnes, Terje Aaberge og spesielt Nils Arne Hove ved Vestlandsforskning AS.

Sogndal 04.juni 2004

_____ _____

Thomas Sløk Tvedt Yngve Håkonsen

1 INNLEDNING

1.1 Bakgrunn

Kommunal- og regionaldepartementet (KRD) har siden 1996 samlet inn data om organisering av norske kommuner. I tillegg ble det gjort en forundersøkelse i 1995. Statistisk sentralbyrå sto for datainnsamling og bearbeiding i 1996 og norsk institutt for by- og regionsforsking (NIBR) stod for tilsvarende arbeid i 2000. I år (2004) gjennomfører NIBR en ny datainnsamling for KRD.

Til nå har målgruppen for disse organisasjonsdataene stort sett vært forskere. Data har vært relativt vanskelig tilgjengelig, og bare personer med god kjennskap innen fagfeltet har kunnet nytte seg av informasjonen. KRD ønsker å utvide målgruppen og gjøre informasjonen lettere tilgjengelig. Særlig kan informasjonen være nyttig for kommunene i deres arbeid med omorganisering. Planen er å konvertere den statiske informasjonen som i dag er lagret i SPSS til et mer anvendelig format for så å utvikle et brukergrensesnitt på Internett.

Det finnes i dag en Access database tilgjengelig på Kommunal- og Regionaldepartementet sine hjemmesider[1]. Denne databasen inneholder data fra undersøkelsene gjennomført i 1996 og 2000. HISF/Vestlandsforskning sin analyse av denne Access databasen viser at den har en struktur som gjør den lite egnet til brukersøk. Basen er bygd opp slik at en først må velge kommune, deretter kan en ved hjelp av basen finne fram til trekk ved organiseringen av denne kommunen. HISF/Vestlandsforskning sin hypotese er derimot at kommunale brukere som begynner et søk først vil være interessert i å finne ut hvilke kommuner som deler et bestemt organisasjonstrekk eller tiltak. Så kan det hende at de i neste omgang vil ha kjennskap til flere trekk ved den aktuelle populasjonen av kommuner. På denne måten kan brukerne finne frem til relevante sammenlikningskommuner for benchmarking eller kommuner som er foregangskommuner i utprøvingen av en bestemt organisasjonsform eller utviklingstiltak.

På bakgrunn av dette har Vestlandsforskning fått i oppdrag av KRD å gjøre kommunale organisasjonsdata fra den aktuelle spørreundersøkelsen lettere tilgjengelig.
I denne sammenheng fikk vi forespørsel fra faglærere ved Høyskolen i Sogn og Fjordane (HISF) om vi kunne tenke oss å ha nevnte problemstilling som prosjektoppgave. Etter et informasjonsmøte med Vestlandsforskning, og faglærere ved HISF, satt vi igjen med et inntrykk av at dette var en krevende, men meget interessant oppgave. Selv om dette innebar at vi måtte sette oss inn i ny teknologi, og områder vi tidligere ikke hadde vært presentert for i det 2-årige kandidatstudiet, takket vi ja til forespørselen – vi hadde funnet oppgaven til prosjektet.

[1] http://odin.dep.no/krd/norsk/databaser/kommorg/index-b-n-a.html (3. juni 2004)

1.2 Mandat

Mandatet fikk vi overlevert av prosjektleder fra Vestlandsforskning, 28.01.04. Det består av 6 deloppgaver, og går ut på å løse følgende som en del av KRD-oppdraget:

- Lage en analyse av dagens SPSS[2] baserte datagrunnlag
- Utvikle en datamodell
- Importere statistikk data fra SPSS til database
- Eksportere databasegrunnlag til XML-struktur
- Utarbeide enkelte spørringer etter nærmere definisjon
- Vurdere generalisering av datamaterialet

Som overliggende krav til løsning skal det være mulig med benchmarking mellom kommuner eller grupper av kommuner. Det skal videre være mulig å generere krysstabeller, og å gjøre synonymsøk mulig.
Generelt kan oppgaven beskrives som å utvikle en robust og fleksibel datamodell for bedre utnytting av et spesifikt spørreskjemagenerert datasett i tillegg til å foreslå løsninger for best mulig gjenbruk av data.

Etter hvert skulle det vise seg at vi ikke fikk gjennomført alle punktene i mandatet innen leveringsfristen på prosjektet. Forsinkelser og andre problemer har vi diskutert i kapittel 5.1.

1.3 Problemstilling

Oppdraget Vestlandsforskning fikk fra KRD var å gjøre data fra spørreundersøkelsen utført i 2004 tilgjengelig på Internett. Det var altså ikke et krav at løsningen skulle ta hensyn til historikk.

Oppgaven vår gikk ut på å eksportere dataene fra spørreundersøkelsen til et format, eller datagrunnlag, som gjør det mulig å utvikle et brukergrensesnitt mot disse. For å få til dette måtte vi først finne ut hvilket datagrunnlag som var best egnet, og deretter flytte dataene fra spørreundersøkelsen over til dette datagrunnlaget. Med det nest siste punktet i mandatet, utarbeiding av enkelte spørringer etter nærmere definisjon, forstå vi at vi skal begynne på utviklingen av brukergrensesnittet.

Mandatet legger opp til en løsning basert på en database, men sier også at vi skal vurdere en generalisering av datamaterialet. Her har vi valgt å se på XML som et alternativt datagrunnlag for å oppnå en slik generalisering av datamaterialet.

Ettersom vi har fått et mandat som sier noe om oppgaven vi skal løse vil problemstillingen være å **løse alle punktene i mandatet**.

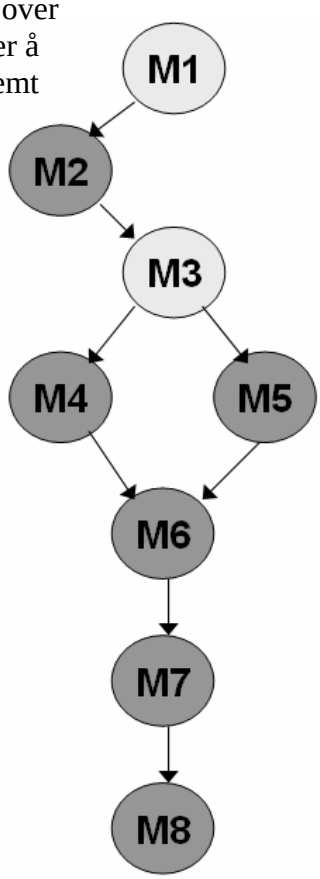
En overordnet problemstilling for hele prosjektet vil være som følger:

Hvordan kan vi bidra til å gjøre dataene fra spørreundersøkelsen 2004 bedre tilgjengelig.

[2] SPSS er omtalt i kapittel 3.4.2

1.4 Fremdriftsplan

Lengden på prosjektet vårt er litt mer enn 4 måneder. Da det knyttet seg en del usikkerhet rundt enkelte detaljer i prosjektet, hadde vi vanskeligheter med å si noe nøyaktig om varigheten på prosjektet. Men for å ha en oversikt over hvor langt vi til en hver tid var kommet laget vi en del milepæler å styre etter. En intern milepæl markerer avslutningen på en bestemt aktivitet, mens en ekstern milepæl markerer at vi er avhengig av en ekstern aktør. Vi definerte følgende milepæler i begynnelsen av prosjektet vårt:



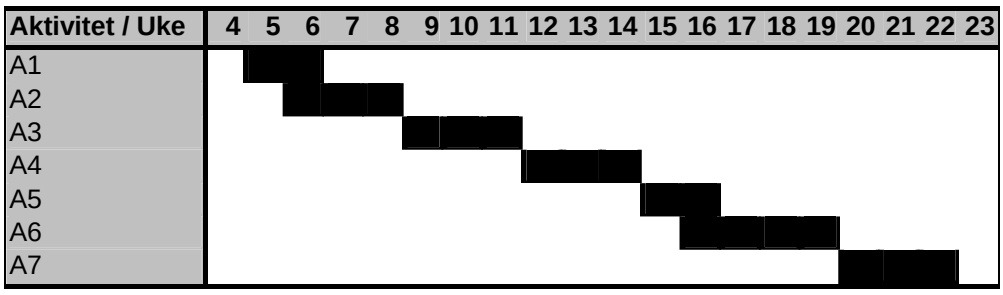
Figur 1 - Milepældiagram

- M1 Fått mandat av Vestlandsforskning (ekstern)
- M2 Prosjekt godkjent av veileder
- M3 Fått datafil fra NIBR (ekstern)
- M4 Analysert dagens SPSS-baserte datagrunnlag
- M5 Bestemt organisering av datagrunnlag
- M6 Overført data fra SPSS til nytt datagrunnlag
- M7 Utviklet brukergrensesnitt
- M8 Rapporten er ferdigstilt og klar til levering

Ut ifra disse milepælene har vi definert følgende aktiviteter:

- A1 Formulere problemstilling og utarbeide tidsplan
- A2 Analysere dagens SPSS-baserte datagrunnlag
- A3 Bestemme organisering av datagrunnlag
- A4 Utvikle database?
- A5 Overføre data fra SPSS til database?
- A6 Utvikle brukergrensesnitt
- A7 Ferdigstille rapport

Disse aktivitetene har vi satt inn i et gantt-diagram (se figuren under). Vi har satt spørsmålstegn ved aktivitetene A4 og A5 da disse representerer en mulig løsning. Aktiviteter som å sette seg inn i nye teknologier og rapportskriving er aktiviteter som går igjen gjennom hele prosjektet. I gantt-diagrammet på figur 1 har vi kun vist planlagt tidsbruk. Etter hvert viste det seg at vi fikk store problemer med å forholde oss til denne fremdriftsplanen. Forsinkelser og andre problemer har vi diskutert i kapittel 5.1.



Figur 2 – Gantt-diagram

2 METODE

2.1 Metodedefinisjon

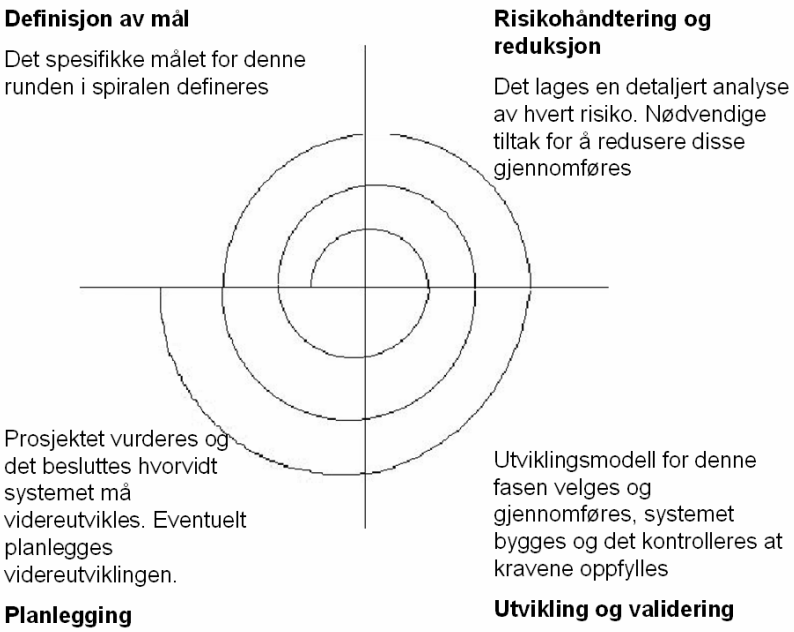
En metode er samlingen av teorier, hypoteser og fremgangsmåter som viser seg å være brukbare til å løse et problem, og gjør det mulig å bruke løsningen til å lære noe om problemet. Siden en metode er en slags oppskrift på løsningen av et problem er det viktig at metoden man bruker blir dokumentert. Det er for at andre som blir presentert for en lignende problemstilling senere skal kunne benytte den samme metoden og komme frem til samme resultat.

2.2 Fossefallsmodellen

En velkjent systemutviklingsmetode er fossefallsmodellen. Denne metoden er bygd på prinsippet om en forhåndsbestemt rekkefølge av bestemte aktiviteter. Den eneste fleksibiliteten er at vi underveis i prosessen, avhengig av situasjonen, kan velge mellom et antall aktiviteter, og at enkelte aktiviteter kan vise seg å være "tomme" og derfor kan hoppes over (Gerhard Skagestein, 2002:56). Fossefallsmodellen består av følgende aktiviteter: Initiering, utvikling, implementering, bruk og vedlikehold.

2.3 Spiralmodellen

Ettersom det var en god del usikkerhet forbundet med dette prosjektet, har vi valgt å benytte oss av spiralmodellen som en overordnet utviklingsmodell for hele prosjektet. Denne modellen passer bedre enn tradisjonelle utviklingsmodeller siden den opprinnelige usikkerheten er stor[3]. Spiralmodellen sørger for å kombinere prototypens fleksibilitet og iterative natur med fossefallsmodellens kontroll og systematikk. I spiralmodellen utvikles det ønskede systemet iterativt gjennom et antall versjoner. Spiralmodellen illustreres ved en spiral (se figuren til høyre). Man starter i midten og hver omgang i spiralen beskriver en ny versjon av systemet. Spiralens plan er oppdelt i et antall faser, tilsvarende de prosjektfaser som en versjon skal gjennomgå før neste versjon kan påbegynnes. For hver versjon kan således spiralmodellen være like nøyaktig og systematisk som fossefallsmetoden. Metodens fleksibilitet oppstår ved at det først ved det ytterste nivået i spiralen representerer fremstillingen av det ferdige



Figur 3 - Spiralmodellen

[3] http://www.ddre.dk/rapporter/rap-04/F%2005-2004.pdf (3. juni 2004)

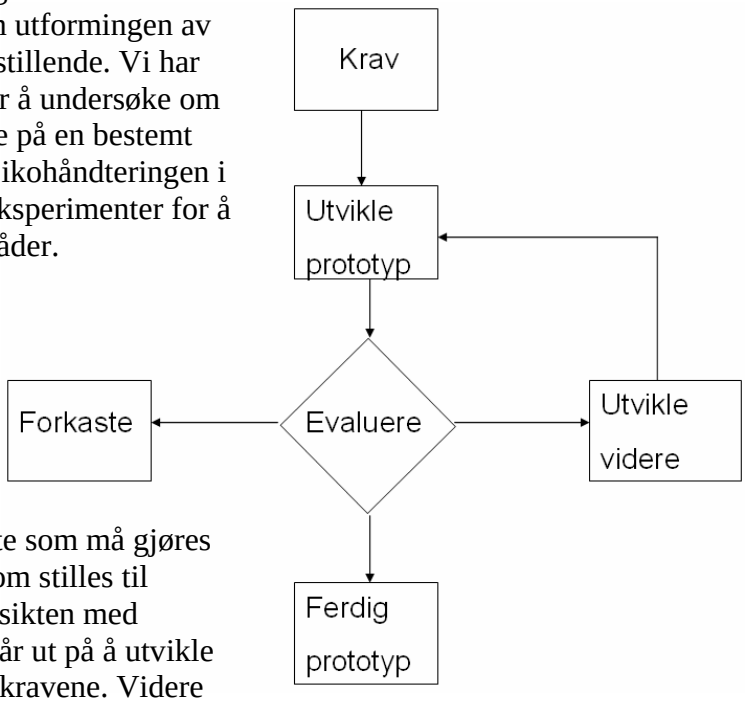
produktet. Hvert enkelt av nivåene innenfor har derimot som formål å avklare utgangsbetingelsene for det neste nivået.

Vi vil nå se litt på de fire fasene i spiralmodellen. Antall faser kan variere, men en må minimum ha med de fire fasene som vist i figur 3. Det første steget, definisjon av mål, går ut på å definere målene for denne runden i spiralen. Deretter følger en vurdering av alternativene, samt en vurdering av risikoen knyttet til disse. En slik risikovurdering vil ofte innebære konstruksjon av en eller flere prototyper for å identifisere risikoområder. I den tredje fasen bygges systemet, og det kontrolleres om kravene oppfylles. Til slutt vurderes prosjektet, og det avgjøres om en skal fortsette utviklingen.

Spiralmodellen er ikke helt uten ulemper. Metodens suksess er svært avhengig av en god risikohåndtering. På grunn av at selve metodens fleksibilitet kan det være vanskelig å kontrollere hvor mange runder i spiralen som kreves. Det kan derfor være vanskelig å si noe om prosjektets varighet.

2.4 Prototyping

Vi har også benyttet oss av prototyping som systemutviklingsmetode i dette prosjektet. Prototyper blir ofte brukt når de presise kravene er vanskelige å identifisere (Steven Alter, 2002:489), noe som også var tilfellet i vårt prosjekt. Ved utvikling av brukergrensesnitt er det vanlig å utvikle prototyper for å undersøke om utformingen av brukergrensesnittet er tilfredsstillende. Vi har derimot utviklet prototyper for å undersøke om det er mulig å løse en oppgave på en bestemt måte. Vi har som en del av risikohåndteringen i spiralmodellen gjennomført eksperimenter for å avdekke eventuelle risikoområder.



Figur 4 - Prototyping

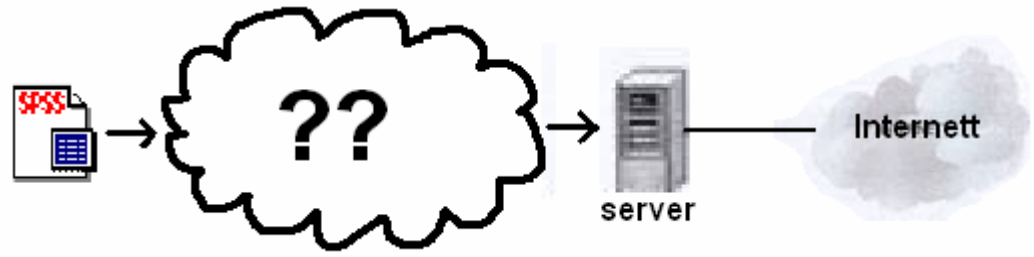
En prototyp er en sterkt forenklet utgave av et ferdig system (Gerhard Skagestein, 2002:249). På figuren til høyre ser vi de forskjellige stegene som inngår i prototyping. Det første som må gjøres er å identifisere hvilke krav som stilles til prototypen og hva som er hensikten med prototypen. Det neste steget går ut på å utvikle prototypen i henhold til disse kravene. Videre vil det bli foretatt en vurdering av prototypen. Her kan en velge mellom å forkaste prototypen, videreutvikle den, eller ende opp med et ferdig system.

I vår bruk av prototyping har vi endt opp med å forkaste prototypen en del ganger, uten at dette nødvendigvis var negativt, for på denne måten kan vi som nevnt tidligere identifisere eventuelle risikoområder. Med disse prototypene har formålet ofte vært å undersøke om noe i det hele tatt var mulig.

3 ANALYSE

3.1 Innledning

Som nevnt i problemstillingen (kapittel 1.3) går oppgaven ut på å gjøre dataene fra spørreundersøkelsen tilgjengelig på Internett. Dataene må altså ut fra SPSS[4], og presenteres på Internett. Dette kan gjøres på mange forskjellige måter, og vi vil i dette kapittelet se på noen alternative løsninger. Boblen med spørsmålstegn på figur 5 illustrerer de forskjellige løsningsalternativene.



Figur 5 – Oversikt over problemet

I kapittel 3.3 har vi vurdert om det er mulig å la XML være selve datagrunnlaget, enten som en vanlig XML-fil eller som en såkalt "Web Service[5]". Vi har også sett på fordeler og ulemper ved å ha XML som datagrunnlag i forhold til en database.

Etter at vi hadde bestemt oss for å benytte en Oracle 8i database (se kapittel 4.1.1) og datamodellen var på plass måtte vi importere dataene fra SPSS til databasen. Etter hvert viste det seg at denne eksport/import jobben var meget ressurs- og tidkrevende. Som nevnt i kapittel 2 har vi fulgt spiralmodellen i dette arbeidet for å avdekke eventuelle risikoområder. Vi har altså utført en god del prøving og feiling før vi fikk til å importere dataene til Oracle databasen. I kapittel 3.4 har vi sett på en del alternative måter å løse denne importen på. Hvordan vi faktisk gjorde det har vi diskutert i kapitlene 4.3 til 4.6. Her beskriver vi blant annet utviklingen av en Java applikasjon som importerer data fra et hvilket som helst datasett i SPSS til en Oracle database (via XML). Vi mener at denne Java applikasjonen også kan være av allmenn interesse ettersom den er generell for alle SPSS-filer. Dette har vi diskutert i kapittel 4.6.

Til slutt i denne analysedelen har vi tatt med en vurdering på om det er mulig å presentere dataene som en selvstendig applikasjon på disk. Denne vurderingen var ikke en del av mandatet, men vi følte at en slik vurdering kan være nyttig ettersom dataene er statiske, og det ikke er noe i veien for å distribuere disse på for eksempel en cd.

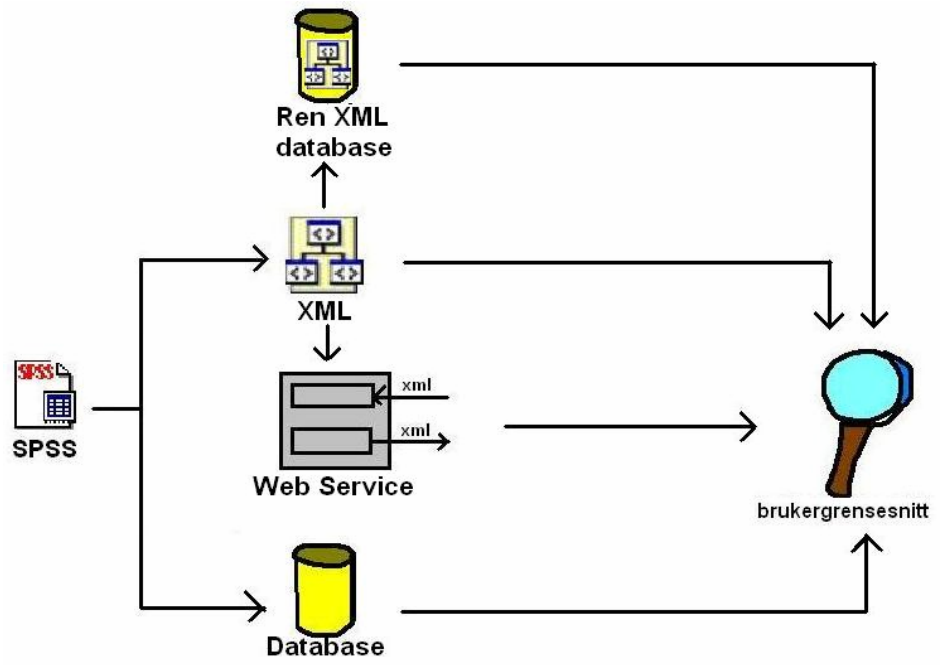
3.2 Tidligere arbeid

Når det gjelder tidligere arbeid med å eksportere data fra SPSS til en Oracle-database, har vi ikke funnet eksempler på at dette har vært gjort tidligere. Vi har vært i kontakt med flere personer i miljøer som kunne ha hørt om dette, blant annet daglig leder ved SPSS Norge, men ingen av de vi spurte kunne svare oss. Vi har også søkt på Internett, uten resultater. Dette betyr ikke at det aldri har vært gjort tidligere, men konklusjonen får bli at det mest sannsynlig ikke er mange som har gjort dette før oss, kanskje ingen.

[4] SPSS er beskrevet i kapittel 3.4.1.1
[5] "Web Service" har en bestemt betydning i denne sammenhengen. Se kapittel 3.3.1.4

3.3 XML som datagrunnlag

I mandatet står det indirekte at en løsning bør være basert på en database, men som vi nevnte innledningsvis vil vi også diskutere andre alternativer. Mandatet (kapittel 1.2) sier at vi skal vurdere en generalisering av datamaterialet, og for å oppfylle dette punktet har vi sett på XML som et alternativt datagrunnlag til en database. I dette kapittelet vil vi se litt på fordeler og ulemper ved å la datagrunnlaget være XML, samt hvordan dette kan løses i praksis. Figur 6 gir en oversikt over de alternativene vi vil diskutere i dette kapittelet.



Figur 6 – Alternative datagrunnlag

Som vi ser av figur 6 kan XML representeres eller lagres på forskjellige måter. Det er mulig å utvikle et brukergrensesnitt opp mot en vanlig XML-fil, en såkalt ren XML database, eller mot en såkalt Web Service. Men før vi ser på de forskjellige alternativene vil vi gå gjennom litt relevant teori.

3.3.1 Teori

3.3.1.1 XML

XML - Extensible Markup Language**Feil! Bokmerke er ikke definert.** er kode som ligner litt på HTML. Den største forskjellen er at XML ikke er fastlåst til ett sett med ”tags”. I tillegg har XML strengere krav til struktur. XML e utviklet av en arbeidsgruppe under W3C – World Wide Web Consortium[6]. Hensikten med XML er å skille form og innhold. XML er et forholdsvis nytt språk, og de første offisielle spesifikasjonene ble lansert av W3C i februar 1998. XML versjon 1.1 ble en anbefalt standard den 4.februar 2004, altså midt i prosjektperioden vår.

XML har blitt svært populært i mange sammenhenger. En av grunnene til dette er nettopp det at det er en åpen og plattformuavhengig standard. Det er altså ingen som har ”monopol” på XML, og fritt frem for alle som ønsker å bruke det. XML er en godt

[6] http://www.w3c.org/ (3. juni 2004)

egnet måte å lagre data på, ikke bare for Internett, men også for applikasjoner. XML er ikke binære filer, men er lagret som ren tekst. Her er noen bruksområder for XML:

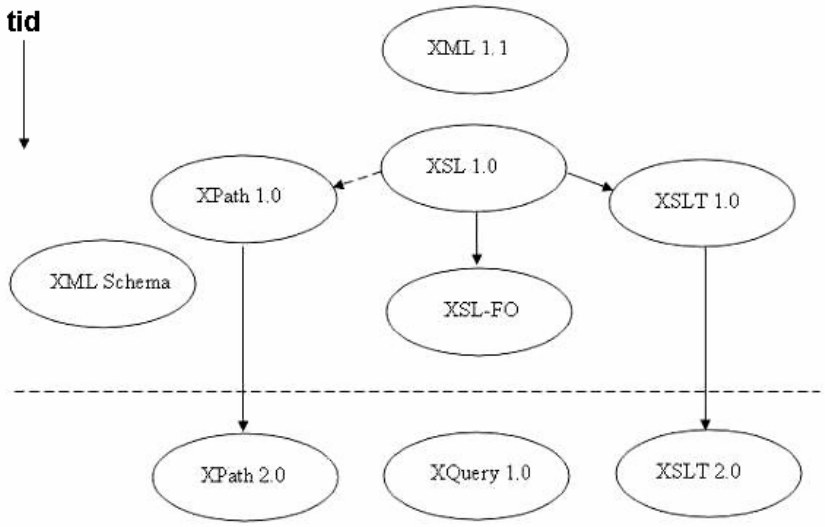
- Bruke XML for å lagre/behandle informasjon som benyttes av applikasjoner.
- Bruke XML som et "utvekslingsformat". Siden XML-formatet er basert på en standard er dette et anvendelig format for deling av data mellom programmer.
- Bruke XML for å lagre/behandle Internettdata.
- Bruke XML til å lage et felles datalagringsformat som kan brukes i mange sammenhenger og av mange forskjellige applikasjoner.

3.3.1.2 XML Schema

XSD – XML Schema Definition[7], er skjema som definerer struktur, innhold, og semantikken til et XML-dokument. XML skjemaer gjør det altså mulig for maskiner å følge regler som er definerte av mennesker. I motsetning til en DTD (Document Type Definition kan et XML-skjema angi type for et element (North og Hermans, 2000:94-105).

3.3.1.3 XPath og XQuery

Det finnes en rekke forskjellige "spørre-språk" for XML. Vi kan nevne XQL, XPath, XML-QL, Quilt, og XQuery. Av disse er XPath – the XML Path Language8 og XQuery9 omtalt som de mest lovende, ettersom dette er disse to standardene som blir utviklet av W3C. XPath og XQuery er to relativt nye standarder. På figuren til høre ser vi hvilke standarder som er anbefalte standarder og hvilke som fortsatt er under utvikling hos W3C.
Den prikkete linjen viser hvor vi er i dag. XPath er et språk som brukes av blant annet XSLT for å hente ut eller referere til deler av et XML-dokument. XQuery er en standard for å kjøre spørringer opp mot XML-filer, og er på mange måter en videreutvikling av XPath. Det siste utkastet til XQuery fra W3C er datert 20.februar 2004.



Figur 7 – Anbefalte standarder

3.3.1.4 Web Services

På Internett blir det større og større behov for kommunikasjon mellom applikasjoner. Det grensesnittet som gjør dette mulig kalles Web Services. Faggruppen Web Services Architecture Working Group definerer WS slik:
Web Services er programvare som identifiseres av en URL, hvis offentlige grensesnitt og utseende er definert med bruk av XML. Definisjonene kan oppdages av andre

[7] http://www.w3schools.com/schema/schema_intro.asp (3. juni 2004)
[8] http://www.w3.org/TR/xpath (3. juni 2004)
[9] http://www.w3.org/XML/Query.html (3. juni 2004)

systemer. Disse systemene kan igjen innvirke på Web Service slik som det er fastsatt i definisjonen, med bruk av XML baserte meldinger tilkjennegitt ved Internett protokoller.[10]

3.3.1.5 Tabellbasert tilordning

Et viktig begrep ved import / eksport av data mellom XML og en relasjonsdatabase er en såkalt tabellbasert tilordning. En slik tilordning blir også benyttet av såkalte wrappers som vi har omtalt i kapittel 3.3.4 Det modulerer XML-dokumenter som en enkel tabell eller et sett med tabeller. På figur 8 ser vi et eksempel på en XML-fil med en tabellbasert tilordning.

```
<database>
      <table>
         <row>
            <column1>...</column1>
            <column2>...</column2>
            ...
         </row>
         <row>
            ...
         </row>
         ...
      </table>
      <table>
         ...
      </table>
      ...
</database>
```

Figur 8 – XML-fil med tabellbasert tilordning

3.3.2 Hvilke data passer for lagring i XML

En XML-fil lagrer data i en trestruktur. Et godt eksempel på data som passer for lagring i XML er en fil som inneholder informasjon om innstillinger til en applikasjon, ofte kalt .ini filer. Det er mye enklere å lage en liten XML-fil og behandle denne med ett programmerings grensesnitt, enn å gjøre det på den ”vanlige” måten med kommaseparerte filer. Andre eksempler på hvor XML kan egne seg som lagringsformat kan være personlige kontaktlister, nettleserfavoritter, og beskrivelser på MP3-filer.

3.3.3 XML vs DBMS

Ved å la datagrunnlaget være XML oppnår vi en generalisering av datamaterialet. Med det mener vi at det er mulig for hvem som helst, uavhengig av plattform og programmeringsspråk, å bruke filen.
Ettersom mandatet indirekte legger opp til en løsning som benytter seg av en database, vil vi se litt på forskjellene mellom det å lagre data i en XML-fil i motsetning til en database. Er det ene bedre enn det andre? Lagring av data i XML sørger for mange av de funksjonene som finnes i en tradisjonell database:

- Lagring (XML dokumenter)
- Skjemaer (DTD[11], XML skjemaer)

[10] http://www.w3.org/TR/2003/WD-ws-arch-20030514/ (3. juni 2004)
[11] http://www.w3.org/TR/REC-xml/ (3. juni 2004)

- Spørrespråk (XQuery, XPath, XQL, etc.)
- Programmerings grensesnitt (SAX, DOM)[12]

Men der stopper også likheten. En database tilbyr blant annet følgende funksjoner som ikke finnes ved lagring av data i XML:

- Effektiv lagring
- Indeksering
- Sikkerhet
- Transaksjoner og dataintegritet
- Flerbruker tilgang
- Spørringer gjort på tvers av flere dokumenter

Det finnes også andre fordeler med å lagre data som XML. En stor fordel er at data lagret i XML er transportable. Det er enkelt å flytte data mellom ulike plattformer og systemer ettersom XML er lagret i Unicode, eller som ren tekst.

3.3.4 Kjøre spørringer mot XML

Det var først da vi fant ut at det var mulig å kjøre spørringer mot en XML fil at vi så at det var en mulighet å la datagrunnlaget være XML. Dersom det skal være aktuelt å la datagrunnlaget være XML, må det være mulig å utvikle et brukergrensesnitt opp mot datagrunnlaget. Det at det er mulig å kjøre spørringer mot en XML-fil er viktig ved utvikling av et brukergrensesnitt der det vil være aktuelt å hente ut bestemte deler data.

XPath[8] er i dag det mest vanlige spørrespråket opp mot XML. En slik XPath spørring kan for eksempel inngå i en XSLT-fil, eller i en applikasjon skrevet i .NET (Microsoft rammeverk). På figuren under ser vi et eksempel på en XSLT-fil med en XPath spørring. I dette tilfellet hentes verdien i et bestemt element, men det er også mulig å hente en samling med elementer.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" media-type="text/html"/>
        <xsl:template match="/">
                Dato: <xsl:value-of select="/sav_file/info/printed"/>
        </xsl:template>
</xsl:stylesheet>
```

Figur 9 – XSLT med XPath spørring

Et problem med XPath er at det mangler mulighet til å gruppere, sortere, og kombinere flere dokument.

Det er også mulig å kjøre SQL-spørringer mot XML med såkalte wrappers[13]. Generelt kan man si at wrappers benyttes når man vil kjøre SQL-spørringer mot XML. Wrappers er et system som behandler XML dokumenter som et opphav for relasjonsdata. (Terminologien kommer fra sammensluttede database systemer, hvor en wrapper er en komponent som ”pakker inn” et kildesystem slik at dataene bruker modellen fra et målsystem). En wrapper behandler XML data som relasjonsdata. En wrapper kan for eksempel benyttes ved kobling av data fra forskjellige systemer. Karakteristisk nok

[12] SAX og DOM har vi diskutert på side 21.
[13] http://www.rpbourret.com/xml/ProdsWrappers.htm (3. juni 2004)

implementerer wrappers en SQL spørrings motor, der den bruker en objektbasert, eller tabellbasert tilordning.

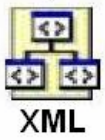
Som et eksempel på en slik wrapper har vi sett litt på Ashpool[14]. Dette er en enkel XML database utviklet i Java. Den bruker standard SQL92 syntaks for å kjøre spørringer, innsetting, oppdatering, og sletting av XML dokumenter med en tabellbasert tilordning. Grunnen til at XML dokumentene må ha en tabellbasert tilordning er at SQL er designet for å kjøre spørringer mot tabelldata og ikke hierarkiske data. Begrensningen med såkalte wrappers blir altså at XML filen må ha en tabellbasert tilordning.
Dersom XML-filene har et skjema, vil disse skjemareglene benyttes ved oppdateringer og innsettinger. Ashpool har også en innebygd JDBC driver og kan lett benyttes sammen med andre applikasjoner som har støtte for JDBC En ulempe med Ashpool er, i følge produsenten selv, at det kan gå tregt å kjøre spørringer når store mengder informasjon blir involvert. Ashpool benytter seg av SAX og kan derfor håndtere litt større dokumenter enn applikasjoner som benytter seg av DOM, men anbefales fortsatt ikke for større mengder data.

Ashpool fungerer slik at SQL setninger blir konvertert til XSLT stylesheets, som deretter blir koblet til XML dokumentet. Det er støtte for SQL syntaks som f.eks. SELECT, UPDATE, INSERT, DELETE, CREATE TABLE. Men det er ikke støtte for f.eks. joins i SELECT setninger.
Den største begrensningen er imidlertid at XML-dokumentene må ha en tabellbasert tilordning (se kapittel 3.3.1.5).

3.3.5 Hvordan lagre XML

Som nevnt innledningsvis i kapittel 3.3 kan XML representeres eller lagres på forskjellige måter:

- Som en vanlig XML-fil
- I en ren XML database
- Som en Web Service

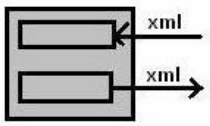


Et alternativ er å ha en vanlig XML-fil liggende på enten disk eller Internett-server for så å la brukergrensesnittet lese denne filen. Vi mener at dette kunne vært et godt alternativ dersom en hadde satset på å utvikle systemet som en selvstendig applikasjon. Et argument for dette er at XML er en fri standard, og det er ingen kostnader forbundet med å benytte XML som datagrunnlag på for eksempel en cd. Dersom XML-filen skulle ligget på en Internett-server mener vi at det kan oppstå problemer når samme filen benyttes av flere brukere samtidig, men dette er vi litt usikre på. En annen ulempe med å benytte en XML-fil som datagrunnlag er at utvikleren av brukergrensesnittet må kunne en del om hvordan en behandler XML-dataene i det aktuelle programmeringsspråket.
Det er også mulig å satse på en løsning med en XML-fil og tilhørende stilark(XSL) som henter de aktuelle data vha. XPath, men dette mener vi er en dårlig løsning da det ikke er god støtte for XML og XSL i alle nettlesere.

En såkalt ”native XML database”, eller en ren XML database er en database spesielt designet for lagring av XML eller et administrativsystem (en applikasjon designet for å styre dokumenter og som bygges på toppen av rene

[14] http://ashpool.sourceforge.net/ (3. juni 2004)

XML databaser). I motsetning til en tradisjonell database som lagrer data i et binært format lagres data som tekst i en ren XML database.
Ronald Bourret vedlikeholder en liste på Internett over slike rene XML databaser[15]. Som eksempler på slike databaser kan vi nevne NeoCore XMS, dbXML og Apache Xindice. NeoCore XMS er et XML database håndteringssystem som tar imot og sender hele, eller deler av XML-dokumenter via HTTP. NeoCore er et kommersielt produkt, og er derfor ikke gratis. En stor ulempe, eller begrensning er at NeoCore bare støtter en spørring om gangen.



Et siste alternativ for å representere XML-dataene er vha. en såkalt Web Service. Dersom vi skulle ha representert de kommunale organisasjonsdataene som en Web Service ville et brukergrensesnitt kommunisert med denne tjenesten via Internett. Et poeng med Web Services er at dataene skal bli tilgjengelig på alle plattformer og systemer som forstår HTTP og XML. Vi får altså en generalisering av datagrunnlaget, og dataene kan brukes av mange applikasjoner via HTTP. Vi mener at det ikke er nødvendig å utvikle en Web Service i vårt tilfelle, da dataene kun skal benyttes av en applikasjon, og denne skal utvikles av Vestlandsforskning. Dersom det hadde vært et krav at dataene skulle kunne brukes av mange applikasjoner ville det vært mer naturlig å utvikle en Web Service. Dersom det skulle vise seg i ettertid at andre ønsker å utvikle en applikasjon som benytter de kommunale organisasjonsdataene mener vi at det vil være et like godt alternativ å tilby disse som en vanlig XML-fil.

3.3.6 Konklusjon

Det er fullt mulig å utvikle et system med XML som datagrunnlag, men vi mener at dette ikke er den beste løsningen. En ulempe med XML som datagrunnlag er at den som utvikler brukergrensesnittet må ha god kjennskap til XML teknologier som for eksempel XPath. En annen ulempe er at XPath har begrensninger som ikke finnes i SQL.
Under arbeidet med å sette oss inn i XML og alle de tilhørende teknologier, fikk vi et inntrykk av at det å bruke XML som datagrunnlag og deretter utvikle et brukergrensesnitt på Internett opp mot disse XML-dataene, ikke har vært gjort så mange ganger tidligere. Det er i dag fullt mulig å kjøre spørringer mot XML, men språket for å gjennomføre disse spørringene er fortsatt under utvikling. Dette mener vi taler imot å la dataene fra spørreundersøkelsen være lagret som XML. Det første språket for å kjøre spørringer mot XML, XPath, finnes i dag i versjon 1.0. Versjon 2.0 er under utvikling, men er ikke en anbefalt standard av w3c[16]. XQuery er enda et spørrespråk under utvikling, men dette er kun på utviklingsstadiet og er ikke en anbefalt standard av w3c[17].

En fordel med at datagrunnlaget er en XML-fil er at denne kan brukes i et hvilket som helst system og på en hvilken som helst plattform. Vi får med andre ord en generalisering av datamaterialet.

Dersom vi hadde valgt å benytte oss av XML som datagrunnlag ville mye av arbeidet ligget i utviklingen av brukergrensesnittet. Et brukergrensesnitt kunne for eksempel bygges opp med XML-stilark med dynamiske XPath-spørringer. En bakdel med dette er

[15] http://www.rpbourret.com/xml/XMLDatabaseProds.htm#native (3. juni 2004)
[16] http://www.w3.org/XML/XPath (3. juni 2004)
[17] http://www.w3.org/XML/Query (3. juni 2004)

at noen nettlesere, som for eksempel Opera, ikke har støtte for å vise XML-filer med tilhørende stilark. En annen måte å bygge opp et brukergrensesnitt mot dataene er å lese hele filen inn i en datastruktur som for eksempel et DOM-tre, omtalt i kapittel 3.4.2, og deretter utføre søk i dette treet. Dette vil medføre en god del programmering, noe vi mener er unødvendig når det kan gjøres på en enklere måte.

Vi så også litt på muligheten for å la datagrunnlaget være en såkalt Web Service. Å utvikle en slik tjeneste som leverer ut data som XML er en god løsning i den forstand at det bidrar til en generalisering av datamaterialet. XML er ikke plattformavhengig, og et brukergrensesnitt kan således utvikles på en hvilken som helst plattform og programmeringsspråk. Vi mener likevel at det ikke er en god løsning å la datagrunnlaget være en såkalt Web Service. Dette fordi at en fortsatt må utvikle et brukergrensesnitt opp mot disse XML-dokumentene som returneres. (I hvert fall dersom dataene skal komme til nytte for Ola Nordmann, noe som er hovedtanken bak dette prosjektet). Utviklingen av et brukergrensesnitt vil altså medføre en god del arbeid.

Mange Internett-sider er i dag basert på en database og tilhørende script. En fordel med å lagre dataene i en database er at mange databaser har et kraftig spørrespråk, og det er enkelt å utvikle et brukergrensesnitt på Internett opp mot en database. En av hovedgrunnene til at det er enklere å utvikle et brukergrensesnitt opp mot en database enn mot en XML-fil er at det har blitt gjort mange ganger tidligere. Det er for eksempel mye lettere å finne litteratur om dette emnet. Det er også mye vanligere med kompetanse på dette området ettersom XML er en forholdsvis ny teknologi.

Å ha et DBMS som datagrunnlag i en Internettapplikasjon mener vi er den beste løsningen. Hvorfor skal en gjøre det vanskelig når det kan gjøres enkelt?

3.4 Import / eksport

Det første punktet i mandatet(kapittel 1.2) var å lage en analyse av dagens SPSS-baserte datagrunnlag. Denne analysen ble gjennomført ca. en måned senere enn beskrevet i fremdriftsplanen i kapittel 1.5 på grunn av diverse forsinkelser (se kapittel 5.1). I denne analysen vil vi først og fremst se på hvordan dataene fra spørreundersøkelsen er lagret i SPSS[18]. I tillegg vil vi se på forskjellige eksport muligheter i SPSS.

Etter at det ble bestemt at vi skulle benytte en Oracle 8i database(se kapittel 4.1.1) var neste oppgave å importere dataene fra spørreundersøkelsen til en database. I kapittel 3.4.3 vil vi sett på alternativene som finnes for å gjennomføre denne importen.

3.4.1 Teori

Vi vil her se på forskjellig teori knyttet til import og eksport av data.

3.4.1.1 SPSS

SPSS - Statistical Product and Service Solutions[18] er et omfattende statistisk datahåndterings- og dataanalyseverktøy. Programmet inneholder over 50 statistiske rutiner, inkludert beskrivende statistikk, krysstabuleringer, frekvensanalyse, undersøkende dataanalyse, variansanalyse og klyngeanalyse. Programmet kan generere mange forskjellige plot – som histogrammer, søylediagrammer, scatterplott, etc. Datahåndteringsverktøyene lar brukeren lese inn data i mange forskjellige formater som for eksempel faste og kommaseparerte filer, hierarkiske, grupperte og blandede filer.

SPSS[18] betyr noe annet i dag enn det gjorde sa produktet ble lansert. På slutten av 60 – tallet samarbeidet SPSS styrformann Norman H. Nie med C. Hadlai (Tex) og Dale Bent, to av hans studenter ved Standforduniversitetet, med å utvikle programmet. De gav det navnet ”**Statistical Package for the Social Sciences”** eller “SPSS”. Ettersom pakken har vokst til et multinasjonalt produkt, med høyst ulike brukegrupper, har navnet blitt endret til å reflektere det selskapet og produktene nå star for.

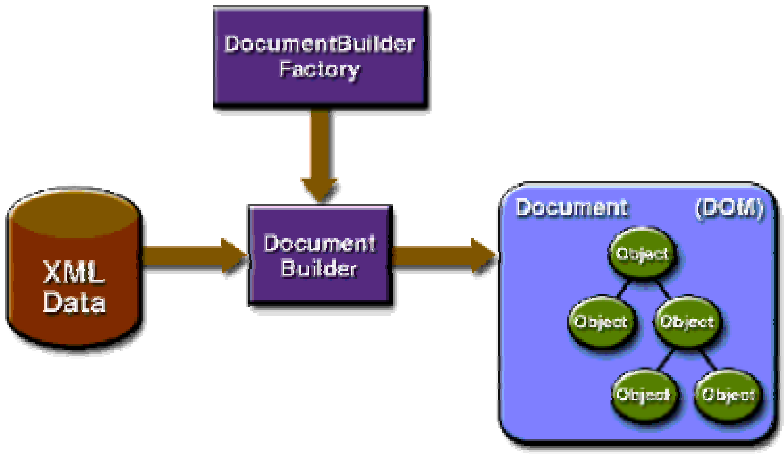
SPSS er som nevnt et svært nyttig verktøy innenfor statistikk, og brukergruppen har vokst mye de siste årene. Verktøyet finnes i nær sagt alle universiteter og høyskoler i landet i dag, og dette forgrener seg videre i bedriftene. Bedriftenes kompetanse på dataanalyseverktøy og dataminingverktøy har også økt.

3.4.1.2 SAX og DOM

SAX – Simple Application Programming Interface (API) for XML[19] så dagens lys etter at Peter Murray-Rust, en av de tidligere utviklerene av XML, kom med en klage til dem som stod på XML-utviklernes adresseliste. Han stilte spørsmålet om de kunne bli enige om en enkel aktivitetsorientert API – Application Interface mens de ventet på at API’en som ble definert av W3C(DOM) skulle bli ferdig. SAX ble opprinnelig utviklet for Java, men er i dag tilgjengelig på et par andre programmeringsspråk. For å skrive en SAX-applikasjon i Java må man ha en XML-parser som støtter SAX og SAX-distribusjonen, som er en samling av klasser og grensesnitt.

[18] http://www.spss.com/no/ (3. juni 2004)
[19] http://www.saxproject.org/ (3. juni 2004)

DOM – Document Object Model[20], som er et standard API, har et plattformuavhengig og språknøytralt grensesnitt som tillater programmer og skript dynamisk tilgang og oppdatering av innhold, struktur og stil på dokumentene. Dokumentet kan videre bli behandlet, og resultatet av denne behandlingen kan legges tilbake til den presenterte siden.
DOM er en modell av et dokument som inneholder objekter(elementer) med egenskaper(attributter) og metoder, slik at de kan manipuleres(North og Hermans,2000:344) (se figur 10). Disse objektene er organiserte som noder i en trestruktur (heretter omtalt som et DOM-tre)



Figur 10 – Document Object Model

3.4.1.3 XSL og XSLT

XSL – Extensible Stylesheet Language[21] er et språk for å uttrykke stilark som beskriver hvordan XML-dokument av en bestemt type skal vises. I motsetning til CSS, bruker XSL en XML notasjon. Denne notasjonen er uttrykt med en XML DTD som definerer et sett med elementer kalt ”formaterende objekter”, og attributter. Ved å knytte en XSL-fil til en XML-fil vil nettleseren Vise XML-filen formatert pent og pyntelig. XSL deler funksjonaliteten, og er kompatibel med CSS2. Men de har som sagt forskjellig syntaks. I tillegg introduserer XSL et omformingsspråk for XML-dokumenter, nemlig XSLT – Extesible Stylesheet Language Transformation. Tanken bak XSLT var å utføre komplekse ”styling-operasjoner” som for eksempel å lage tabeller. Språket blir nå brukt til å behandle XML-data generelt. XSLT brukes for eksempel til å generere HTML-sider fra XML-data.

3.4.2 Analyse av dagens datagrunnlag i SPSS

Det første punktet i mandatet var å lage en analyse av dagens SPSS-baserte datagrunnlag. I denne analysen vil vi se på hvordan dataene lagres i SPSS, samt hvilke eksport muligheter som finnes.

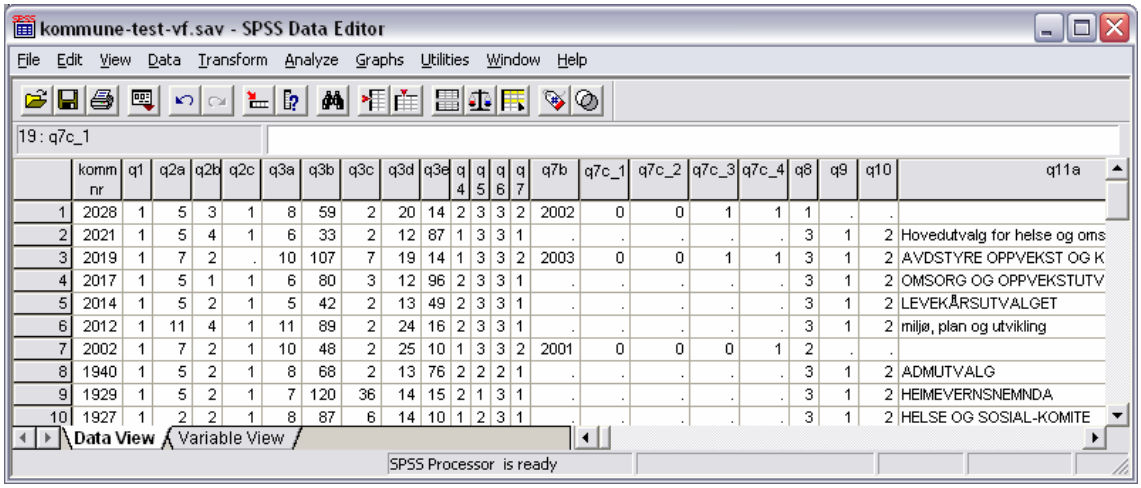
3.4.2.1 Hvordan lagres data i SPSS

SPSS er et som nevnt tidligere et program som brukes til å registrere og analysere statistikk. Dette betyr igjen at datamengden kan bli ganske stor. Data i SPSS lagres i en

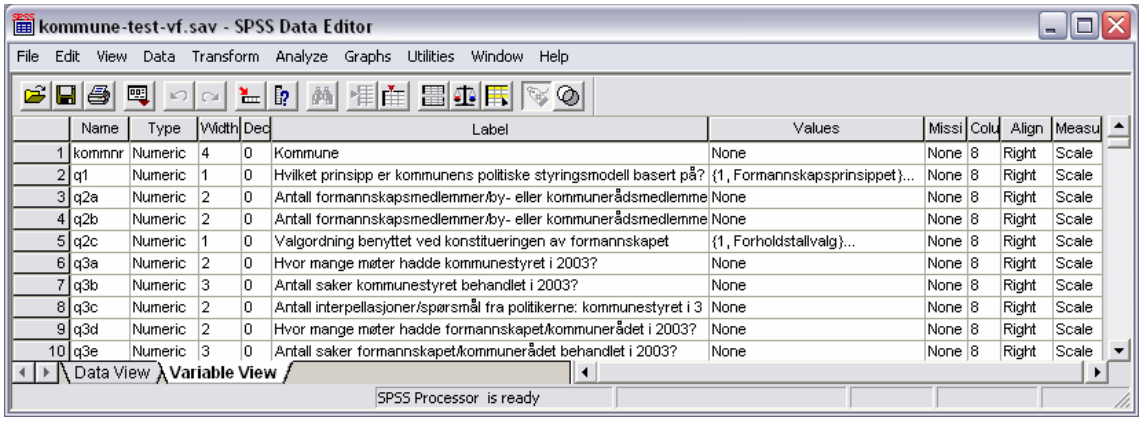
[20] http://www.w3.org/DOM/ (3. juni 2004)
[21] http://www.w3.org/Style/XSL/WhatIsXSL.html (3. juni 2004)

matrise der hver rad representerer en ”post”. I spørreundersøkelsen som NIBR har gjennomført vil det være en rad per kommune som har svart på undersøkelsen.

Hva kommunene svarte på de forskjellige spørsmålene er lagret i kolonner (se figur 11). Den aktuelle spørreundersøkelsen har 806 kolonner, og det er altså en rimelig stor matrise vi har med å gjøre. For hver kolonne (variabel) lagres det en tallverdi som representerer den virkelige verdiene. På figur 11 ser vi at de fleste kommuner er registrert med et 1-tall i kolonnen q1. På figur 12 ser vi at variabelen q1 har to lovlige verdier (values); formannskapsprinsippet og det parlamentariske prinsipp. Her er også spørsmålsteksten lagret. Denne informasjonen som er lagret om variablene kan vi kalle metadata, eller beskrivende data.



figur 11 – data view



figur 12 – variable view

En datafil i SPSS blir lagret som *.sav, denne filen er i et binært format. Dette betyr igjen at filen bare er leselig av SPSS, og en er derfor avhengig av å kunne eksportere data dersom en ønsker å bruke dataene i en annen applikasjon. Som nevnt i kapittel 1.4 er det ønskelig å eksportere dataene fra SPSS til et mer anvendelig format. Vi vil nå se litt på eksportmulighetene som tilbys i SPSS.

3.4.2.2 Eksport muligheter i SPSS

SPSS versjon 11.5 har følgende muligheter for eksport av data

- Tab-delimited
- Fixed ASCII
- Excel
- SAS
- dBase
- XML m.m. (vha. script)

Vi vil nå se på noen av disse eksportmulighetene og si litt om fordeler og ulemper.

En tab-delimited fil er en tekstfil hvor dataene er lagret kolonnevis delt med tabulator. Fixed ASCII er en tekstfil uten tabulator eller mellomrom mellom variabel feltene. Ved eksport til disse formatene vil de representerende tallverdiene bli lagret, men alle variabler som finnes i SPSS-filen vil gå tapt.

I tillegg til de innebygde eksportmulighetene har SPSS 11.5 et scriptverktøy der en kan lage sine egne skript for eksport av data. Dette verktøyet er basert på Visual Basic 6.0 kode. På Internett finnes det ferdige script som eksporterer en datafil i SPSS til XML etter en bestemt oppskrift[22]. I SPSS versjon 12.0 introduseres OMS – Output Management System, som er et internt syntaks språk i SPSS. Det er mulig å eksportere SPSS data til XML ved hjelp av OMS, men ettersom vi fikk versjon 12.0 litt sent i prosjektperioden har vi valgt å ikke se så nøye på denne funksjonen. Problemer med forsinkelser har vi diskutert i kapittel 5.1

Som nevnt tidligere i rapporten skal det blant annet være mulig med benchmarking og synonymsøk. Det må altså være mulig å utvikle et dynamisk brukergrensesnitt opp mot datakilden vi velger å bruke. Tabluator-separert, ASCII-fixed, og Excel – format ser vi på som lite anvendelige format til vårt formål. I disse tre formatene vil ikke informasjon om variablene lagres, og det vil for eksempel ikke være mulig med synonymsøk. Videre har vi vurdert det slik at å ha dataene lagret i SAS eller dBase heller ikke er noen god løsning da vi hadde liten kjennskap til disse produktene.

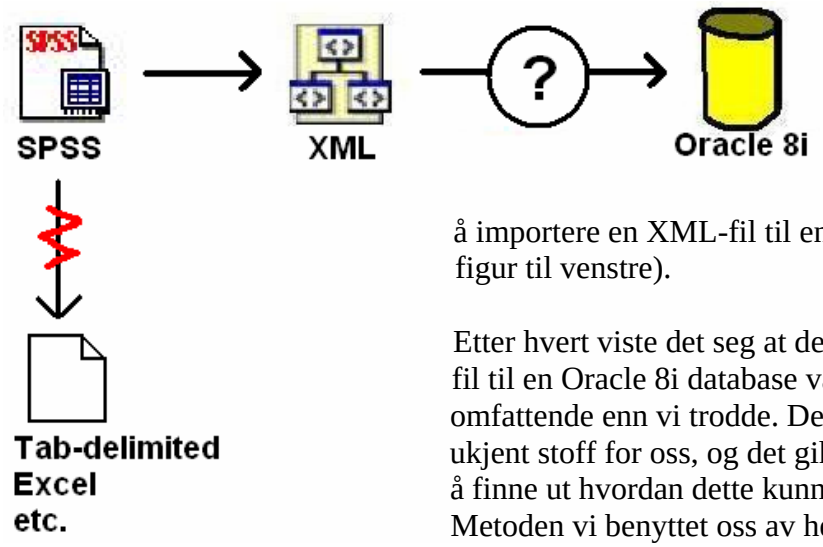
XML er ikke bare et populært lagringsformat, men også et populært ”utvekslingsformat”. En av grunnene til dette er at XML er en felles standard som ikke eies av noen, og det er derfor ingen kostnader forbundet med å lagre data i XML format.

Vi mener at å eksportere SPSS dataene til XML er veien å gå. Hovedgrunnen til dette er at vi får med alle beskrivende data, eller metadata. Eksempel på metadata er spørsmålsteksten og etiketter på eventuelle svaralternativer. Denne informasjonen mener vi er nødvendig ved utvikling av et brukergrensesnitt.

[22] http://pages.infinit.net/Scripts/ImportExport/td_ExportAsXMLv2.txt (3. juni 2004)

3.4.3 Importere data til Oracle 8i database

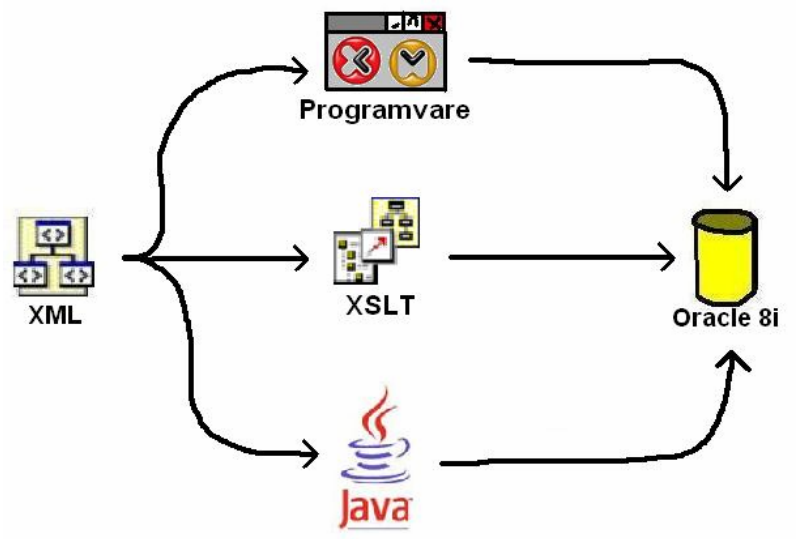
Vi vil i dette kapittelet si litt om hvilke muligheter som finnes for å importere data fra SPSS til en Oracle 8i database. Vi har valgt å konsentrere oss om Oracle 8i da det ble bestemt at vi skulle benytte denne databasetjeneren (se kapittel 4.1.1).



Figur 13 – Veien fra SPSS til Oracle 8i

I kapittel 3.4.2.2 så vi at å eksportere dataene fra SPSS til XML var det beste alternativet. Vi har derfor valgt å se på mulighetene for å importere en XML-fil til en Oracle 8i database (se figur til venstre).

Etter hvert viste det seg at det å importere en XML-fil til en Oracle 8i database var vanskeligere og mer omfattende enn vi trodde. Dette var relativt nytt og ukjent stoff for oss, og det gikk med en god del tid til å finne ut hvordan dette kunne løses.
Metoden vi benyttet oss av her var prøving og feiling helt til vi fikk det til. Vi kan si at vi utviklet prototyper for å sjekke om en fremgangsmåte var mulig (se kapittel 2) Figuren under illustrerer de forskjellige fremgangsmåtene vi forsøkte for å få dataene fra XML-filen og inn i Oracle 8i databasen. I realiseringsdelen av denne rapporten (kapittel 4.4, 4.5, og 4.6) har vi beskrevet disse forsøkene på å importere data.



Figur 14 – Veien fra XML til Oracle 8i

3.5 Internettapplikasjon eller selvstendig applikasjon

Den gamle Access databasen som ble utviklet på grunnlag av undersøkelsene gjort i 1996 og 2000 er tilgjengelig som en enkeltstående applikasjon på KRD sine hjemmesider[23]. Da vi fikk i oppgave å gjøre de kommunale organisasjonsdataene fra spørreundersøkelsen 2004 var det et krav fra oppdragsgiver at systemet skulle utvikles som en Internettapplikasjon. Vi vil imidlertid se på om det er mulig å utvikle en selvstendig applikasjon som kan leveres på for eksempel en cd. Vi vil også si litt om fordeler og ulemper ved disse to løsningene.

Dataene fra spørreundersøkelsen er statiske data, og disse vil aldri bli oppdatert etter at de er lagt inn første gangen. Dette åpner for at dataene kan lagres på en disk som en selvstendig applikasjon. Et brukergrensesnitt skal altså ikke sette inn eller oppdatere data.

Å utvikle systemet som en selvstendig applikasjon har en del fordeler fremfor en Internettapplikasjon. De viktigste fordelene er som følger:

- Bedre responstid
- Tilgang til Internett er ikke nødvendig
- En slipper løpende kostnader som en hadde hatt ved en Internettapplikasjon

En selvstendig applikasjon vil sannsynligvis ha bedre responstid enn en Internettapplikasjon. Grunnen til dette er at responstiden til en Internettapplikasjon er avhengig av båndbredden, eller hastigheten på Internett tilkoblingen. Responstiden i en selvstendig applikasjon er avhengig av maskinvaren. En selvstendig applikasjon er ikke bare uavhengig av båndbredde, den er ikke avhengig av en Internett tilkobling i det hele tatt. Dette er vel kanskje ikke det beste argumentet i denne sammenheng, da tilgangen til Internett er gangske bra i Norge(og spesielt i kommunesektoren). En annen fordel med en selvstendig applikasjon er at en slipper de løpende kostnadene en hadde hatt ved en Internettapplikasjon (for eksempel leie av serverplass). Hvorvidt dette er en kjempefordel kan diskuteres, for disse løpende kostnadene vil bli erstattet av en engangskostnad ved publisering av en selvstendig applikasjon på for eksempel cd.

En Internettapplikasjon har følgende fordeler fremfor en selvstendig applikasjon.

- God tilgjengelighet
- Enkelt å oppdatere system
- En slipper engangskostnader forbundet med distribusjon av applikasjon

En Internettapplikasjon vil være mye lettere tilgjengelig enn en selvstendig applikasjon. En kunne selvsagt gjort den selvstendige applikasjonen tilgjengelig på Internet, man da er også noe av vitsen borte. En annen fordel med en Internett variant er at bruker allerede er kjent med deler av brukergrensesnittet. Som et eksempel så vet brukeren at når musepekeren forandrer seg fra en pil til en hånd så er det mulig å trykke på dette området på skjermen.
I mandatet (kapittel 1.2) så vi at det var et overliggende krav at det skulle være mulig med synonymsøk. For å få til dette må det implementeres en synonymordliste. Her vil det kanskje bli aktuelt å legge til nye synonym etter hvert, og systemet må dermed oppdateres. En stor fordel med en Internettapplikasjon er at oppdateringen av systemet

[23] http://odin.dep.no/filarkiv/121343/komsetup.zip (3. juni 2004)

blir mye enklere enn dersom en hadde valgt en selvstendig applikasjon. I en Internettapplikasjon er det bare nødvendig å oppdatere systemet på en plass (serveren), mens en oppdatering av en selvstendig applikasjon ville vært mer omstendelig.

Vi ser ikke at det er noe i veien for å utvikle systemet som en selvstendig applikasjon, men vi mener at en Internettapplikasjon er den beste løsningen for å gjøre de kommunale organisasjonsdataene lettere tilgjengelig. Dette mener vi først og fremst fordi en Internettapplikasjon vil være lettere tilgjengelig enn en selvstendig applikasjon, men også fordi det vil være lettere å oppdatere systemet.

4. REALISERING

4.1 Våre valg

Etter en grundig analyse av hvilke løsningsalternativ som er tilgjengelige var det på tide å ta noen valg. Vi vil i dette kapittelet si litt om valgene vi har tatt og begrunne disse.

4.1.1 Valg av datagrunnlag

Dataene fra spørreundersøkelsen skal som nevnt i problemstillingen (kapittel 1.3) gjøres tilgjengelig på Internett. For å få til dette kan ikke dataene være lagret i SPSS, og må derfor overføres til et annet datagrunnlag som er mer anvendelig ved utvikling av et brukergrensesnitt. Vi har i kapittel 3.3 sett på om det er mulig å la datagrunnlaget være XML. I kapittel 3.6 konkluderte vi med at en databaseløsning er et bedre alternativ en XML basert løsning.

Den 13.april hadde vi et møte med Vestlandsforskning der vi ble enige om å benytte oss av en Oracle 8i database som datagrunnlag. Hovedgrunnen til dette er at Vestlandsforskning allerede har satt opp en Oracle 8i databasetjener, og det er dermed ingen ekstra kostnader forbundet med å benytte denne. Under dette møtet kom vi også frem til at en løsning basert på XML ville være vanskeligere å utvikle enn en databaseløsning, og dette var også noe som talte for å la datagrunnlaget være en database. Ettersom det ble bestemt at vi skulle benytte en Oracle 8i database har vi ikke vurdert andre databasetjenere.

4.1.2 Eksport / import

Da vi hadde bestemt oss for å benytte Oracle 8i som datagrunnlag var neste steg å eksportere dataene fra SPSS. Som nevnt i kapittel 3.4.3 så finnes det mange muligheter for eksport av data fra SPSS. Vi har valgt å benytte oss av muligheten til å kjøre Visual Basic script. Vi fant et script[24]**Feil! Bokmerke er ikke definert.** på Internett skrevet av Tom Derrick som eksporterer dataene fra SPSS til en XML-fil. I SPSS versjon 12.0 er det mulig å eksportere til XML ved hjelp av OMS – Output Management System, som er en egen syntaks eller språk. På grunn av at vi fikk versjon 12.0 alt for sent i prosjektperioden hadde vi ikke tid til å sette oss inn i OMS. Forsinkelser og andre problemer underveis er diskutert i kapittel 5.1. Dersom vi hadde hatt mer tid til rådighet kunne vi muligens benyttet oss av OMS til å eksportere dataene til XML.

En av grunnene til at vi valgte å gå veien om XML er at da kan vi også gjøre denne XML filen tilgjengelig. Denne delen har altså nytte i seg selv. En annen fordel med å benytte XML som utvekslingsformat er at vi får med metadata som for eksempel

[24] http://pages.infinit.net/rlevesqu/Scripts/ImportExport/td_ExportAsXMLv2.txt (3. juni 2004)

spørsmålsteksten fra spørreundersøkelsen. Dette ville for eksempel gått tapt dersom vi gikk veien om kommaseparerte filer. Vi har sett litt nærmere på scriptet til Tom Derrick og strukturen på XML-filen i kapittel 4.3

Etter at vi hadde fått dataene fra SPSS ut i en XML-fil var neste steg å importere disse dataene til Oracle. Denne import biten viste seg etter hvert å være mer komplisert enn vi hadde forventet i utgangspunktet. Fremgangsmåten vi benyttet oss av for å få til denne importen var rett og slett prøving og feiling helt til vi fikk det til. Som nevnt i kapittel 2 har vi fulgt spiralmodellen ved dette arbeidet for å identifisere eventuelle problemområder. Vi kan si at vi har utviklet prototyper med den hensikt å undersøke om ting fungerer. Vi har forsøkt å importere dataene til Oracle databasen på følgende måter:

- Ved å benytte programvare
- Ved å skrive XSLT
- Ved å utvikle en egen Java applikasjon

De to første forsøkene var mer eller mindre mislykkede, mens Java applikasjonen løser problemet. Vi har gått gjennom de forskjellige prosessene i kapitelene 4.4, 4.5 og 4.6

4.1.3 Brukergrensesnitt

Et punkt i mandatet (kapittel 1.2) er å utvikle enkelte spørringer etter nærmere definisjon. Med dette forstår vi at vi skal begynne på utviklingen av et brukergrensesnitt.
I den originale fremdriftsplanen i kapittel 1.5 ser vi at utvikling av brukergrensesnitt er med som en egen aktivitet. Men på grunn av diverse forsinkelser (se kapittel 5.1) har vi ikke fått tid til å gjennomføre dette punktet.

4.2 Utvikling av datamodell

4.2.1 Innledning

Ved utviklingen av datamodellen har vi benyttet oss av prototyping som metode (se kapittel 2). Vi har altså utviklet en datamodell og deretter evaluert denne. Denne evalueringen ble gjort i samarbeid med Vestlandsforskning. Utviklingen av datamodellen ble gjort parallelt med utviklingen av Java applikasjonen. Vi kom litt sent i gang med datamodelleringen, og føler vel at vi ikke er komt helt i mål med datamodellen. Denne bør altså utvikles videre etter at prosjektperioden vår er over.

4.2.2 Oppsett av Oracle 8i

Da vi hadde bestemt oss for å importere dataene fra SPSS til en Oracle database begynte vi å se på oppsettet av databasen. Vi fikk opprettet en bruker på Vestlandsforskning sin Oracle tjener og tildelt et passord. Deretter installerte vi en Oracle klient fra en cd vi fikk låne av Kåre Asbjørn Byrkenes ved HISF. Etter at det ble ordnet opp i noen rettighetsproblemer hadde vi nå tilgang til Oracle databasen hos Vestlandsforskning.

4.2.3 Krav til datamodellen

4.2.3.1 Historikk

Undersøkelsen fra Kommunal- og regionaldepartementet gjennomføres med en periode på hvert fjerde år (1996, 2000, 2004). Datamodellen vi har utviklet tar ikke hensyn til historikk. Det er altså ikke lagret informasjon om hvilket årstall denne

spørreundersøkelsen gjelder for. Dette er gjort slik fordi det i oppgaven fra oppdragsgiver er spesifisert at det kun er informasjon fra datainnsamlingen i år (2004) som skal være med.

4.2.3.2 Språk

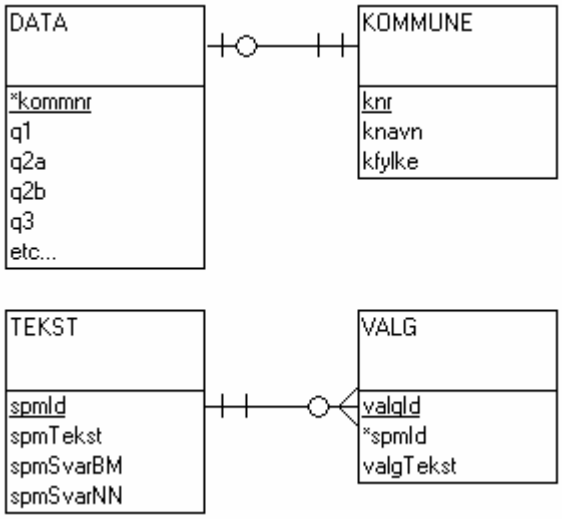
I kommunesektoren finnes det strenge regler om at publikasjoner skal være tilgjengelig på både nynorsk og bokmål. Spørsmålene som er definert i SPSS er på bokmål. For å gi bruker mulighet til å velge nynorsk eller bokmål i det nye systemet, må spørsmålstekst og svaralternativer på nynorsk legges inn manuelt. Dette kan enten gjøres i XML-filen eller i databasen.

4.2.3.3 Andre krav

Som overliggende krav til løsning skal det være mulig med benchmarking mellom kommuner eller grupper av kommuner. Det skal også være mulig å generere krysstabeller, og å gjøre synonymsøk mulig. Disse kravene fikk vi oppgitt i mandatet (kapittel 1.2).

4.2.4 Den første prototypen

4.2.4.1 Datamodell



4.2.4.2 Forklaring til datamodell

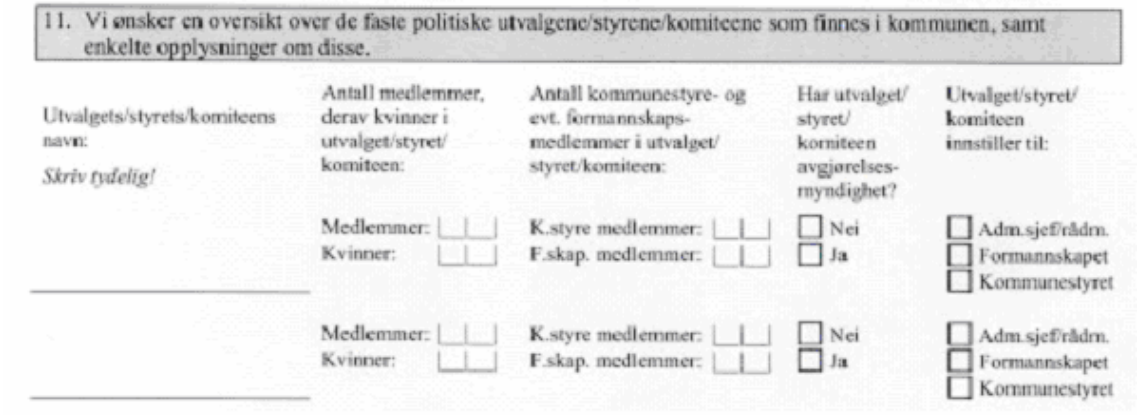
Den første prototypen består av fire entiteter. Entitetstypen DATA inneholder selve dataene fra spørreundersøkelsen, mens TEKST inneholder spørsmålsteksten fra alle spørsmålene, og VALG inneholder eventuelle tekstetiketter på svaralternativene. Attributtene kommnr, q1, q2a, q2b, etc. i DATA er kolonner fra SPSS (spørreundersøkelsen). Attributtene spmSvarBM og spmSvarNN skal inneholde en svartekst på henholdsvis bokmål og nynorsk. Dersom spørsmålsteksten er: ” Hvilket prinsipp er kommunens politiske styringsmodell basert på?” vil en svartekst på bokmål være: ”Kommunens politiske styringsmodell er basert på”. En slik svartekst må nødvendigvis legges inn manuelt. Entitetstypen KOMMUNE er importert fra den gamle Access-databasen som vi fant på Kommunal- og Regionaldepartementet sine Internettsider[25]. Dette er grunnen til at vi har attributtet knr i KOMMUNE og kommnr i

[25] http://odin.dep.no/krd/norsk/databaser/kommorg/index-b-n-a.html (3. juni 2004)

DATA(fra spørreundersøkelsen). Entitetene DATA, TEKST og VALG er generelle, og kan benyttes på data fra en hvilken som helst SPSS-fil. Vi ser at TEKST og VALG ikke er koblet sammen med DATA. Dette er fordi TEKST og VALG inneholder metadata, eller beskrivende data. Her må koblingen gjøres ved utvikling av brukergrensesnittet.

I entitestypen TEKST har vi lagt inn attributtene spmSvarBM og spmSvarNN. I disse attributtene lagres svarteksten på bokmål og nynorsk.

Vi har valgt å ikke gjennomføre noen normalisering på entitetstypen DATA. I denne entitetstypen har vi eksempel på repeterende grupper som skulle vært skilt ut i en egen tabell ved normalisering (se figur 15).


11. Vi ønsker en oversikt over de faste politiske utvalgene/styrene/komiteene som finnes i kommunen, samt enkelte opplysninger om disse.

Utvalgets/styrets/komiteens navn: *Skriv tydelig!*	Antall medlemmer, derav kvinner i utvalget/styret/komiteen:	Antall kommunestyre- og evt. formannskaps-medlemmer i utvalget/styret/komiteen:	Har utvalget/styret/komiteen avgjørelses-myndighet?	Utvalget/styret/komiteen innstiller til:
	Medlemmer: ☐☐ Kvinner: ☐☐	K.styre medlemmer: ☐☐ F.skap. medlemmer: ☐☐	☐ Nei ☐ Ja	☐ Adm.sjef/rådm. ☐ Formannskapet ☐ Kommunestyret
	Medlemmer: ☐☐ Kvinner: ☐☐	K.styre medlemmer: ☐☐ F.skap. medlemmer: ☐☐	☐ Nei ☐ Ja	☐ Adm.sjef/rådm. ☐ Formannskapet ☐ Kommunestyret

Figur 15 – Repeterende grupper i spørreskjemaet

Ettersom vi ikke gjennomfører noen normalisering på entiteten DATA blir det en del ekstra arbeid ved utvikling av brukergrensesnitt, men vi mener likevel at det ikke er nødvendig å normalisere denne entiteten. Dette begrunner vi med at dataene som er lagret er statiske og lagres kun en gang. Faren for feil er derfor sterkt redusert siden det her er snakk om statiske data. Det som derimot kan skje er inntastingsfeil når de statiske dataene legges inn.

4.2.4.3 Evaluering

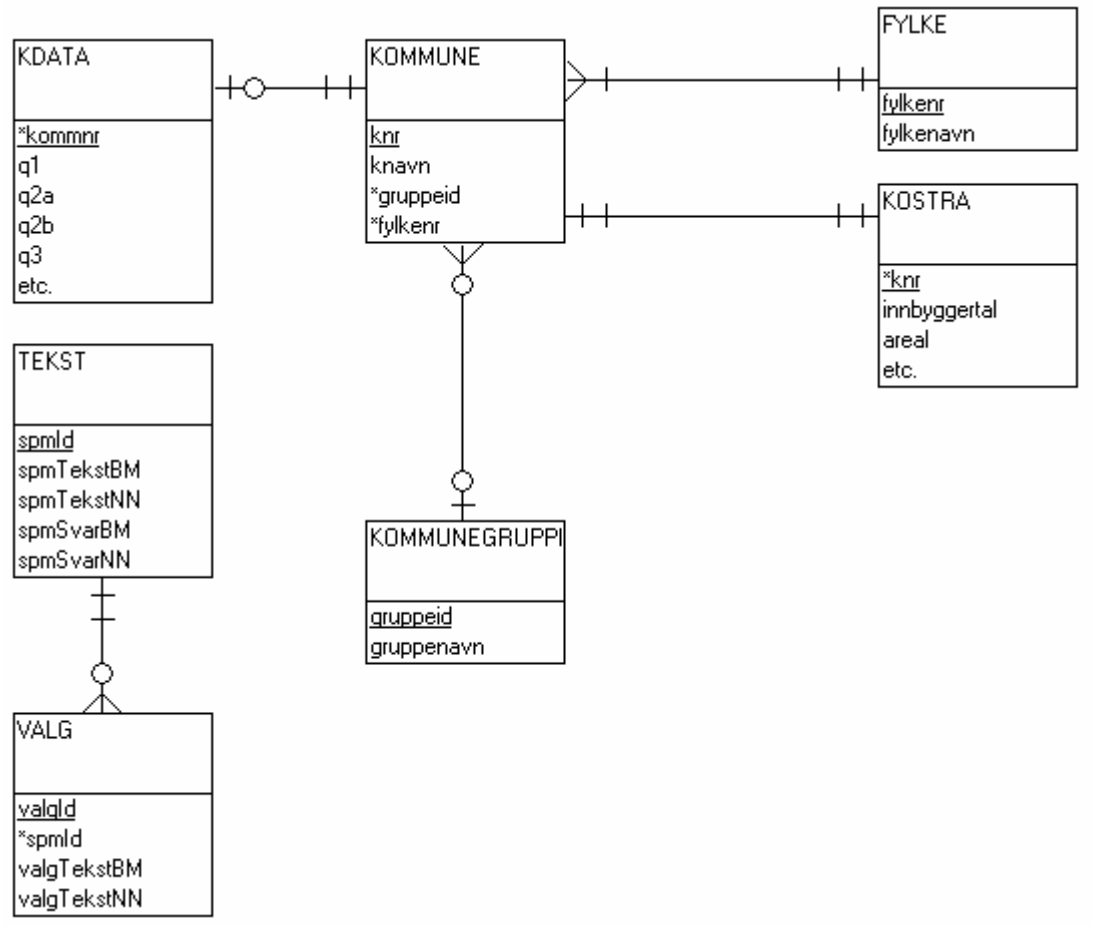
Strukturen på entiteten KOMMUNE er hentet direkte fra den gamle Access databasen. Denne inneholder en repeterende gruppe, fylke. KOMMUNE bør normaliseres, og fylke bør skilles ut i en egen entitet.

I entitetstypen TEKST bør spørsmålsteksten representeres med to attributt på henholdsvis bokmål og nynorsk. Det samme gjelder for attributtet valgTekst i VALG.

Videre ser vi at modellen ikke tar hensyn til kravene angående benchmarking og synonymsøk

4.2.5 Den andre prototypen

4.2.5.1 Datamodell



4.2.5.2 Forklaring til datamodell

I denne datamodellen har vi forandret entitetsnavnet fra DATA til KDATA for å vise at denne gjelder for det aktuelle datasettet. I TEKST har vi definert to attributt spmTekstBM og spmTekstNN som erstatter spmTekst fra den første prototypen. En nynorsk spørsmålstekst må legges inn manuelt. I entitetstypen VALG har vi lagt inn valgTekstBM og valgTekstNN slik at etikettene på eventuelle svaralternativ også kan lagres på bokmål og nynorsk.

Vi ser at KOMMUNE er normalisert, og vi har fått en ny entitet FYLKE. I et kommunenummer identifiserer de to første sifrene hvilket fylke kommunen hører til. For eksempel Sogndal kommune har kommunenummer 1420, der 14 identifiserer fylke, og 20 identifiserer kommunen.

Det har vært snakk om fra arbeidsgiver sin side å ta med data fra eksterne datasett. I denne sammenheng er det foreslått å hente data fra Kostra, som er en database fra Statistisk Sentralbyrå[26]. Entitetstypen KOSTRA inneholder antall innbyggere, antall kommuneansatte, areal, etc. i en kommune. Disse attributtene som for eksempel

[26] http://www.ssb.no/kostra/ (3. juni 2004)

innbyggertall og areal kunne vært plassert i KOMMUNE, men vi har valgt å skille disse ut i en egen entitetstype for å vise at dette er eksterne data.

Den siste entitetstypen, KOMMUNEGRUPPE, har vi valgt å ha med fordi det i følge oppdragsgiver vil være hensiktsmessig å kunne gruppere på antall innbyggere i en kommune.

4.2.5.3 Evaluering

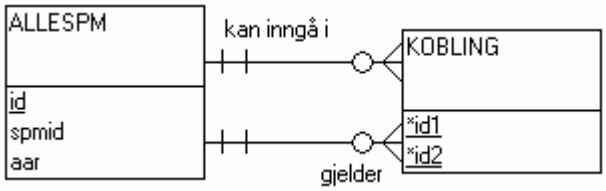
I denne datamodellen er det bare mulig å gruppere på en, og bare en, gruppe. Dersom en ønsker å gruppere på flere kriterier får vi en mange-til-mange relasjon mellom KOMMUNE og KOMMUNEGRUPPE.

Videre ser vi at heller ikke denne modellen tar hensyn til de kravet til synonymsøk. Dette skyldes diverse forsinkelser, noe vi har diskutert i kapittel 5.1. Dette bør implementeres i den endelige datamodellen.

I kapittel 4.2.3.1 så vi at det ikke var noe krav fra oppdragsgivers side at datamodellen skal ta hensyn til historikk. Et scenario derimot vil være at det i fremtiden skal kunne sammenliknes svar fra periode til periode. Siden dette kan være en aktuell problemstilling senere, vil vi her gi en redegjørelse for hvordan dette **kan** gjøres.

Ett alternativ ville vært å lagt til et attributt årstall i entitetstypene KDATA og TEKST. For at dette skal fungere må alle spørsmålene være like fra år til år. Dette er ikke tilfellet i spørreundersøkelsene fra 1996, 2000 og 2004. De kan inneholde noen av de samme spørsmålene, men ofte på forskjellig plass. Dersom spørreundersøkelsene hadde blitt utformet med tanke på at dataene skulle inn i en database ville det vært enklere, men dette er som sagt ikke tilfellet.

Et annet alternativ er å koble sammen spørsmål fra en periode til en annen med en koblingstabell. En vil da på en ny entitetstype tilsvarende KDATA for hver undersøkelse, og tilsvarende med TEKST og VALG. En løsning kan være å legge inn alle spørsmålene fra alle undersøkelsene i en egen entitet ALLESPM (se figur til høyre). En vil da få en mange-til-mange relasjon, som også er en egenrelasjon, ettersom et spørsmål kan tilsvare et eller flere andre spørsmål. Ved å entitetisere mange-til-mange relasjonen vil vi få en ny entitet KOBLING (se figur oppe til høyre).



Figur 16 - Koblingstabell

Figuren under viser noen eksempeldata fra ALLESPM og KOBLING.

id	spmid	aar
1	kommnr	2004
2	q1	2004
3	q2	2004
4	knr	1996
5	spm1	1996
6	spm2	1996
7	q645a	2008

id1	id2
1	4
2	5
2	7
3	6

Figur 17 – Data i koblingstabell

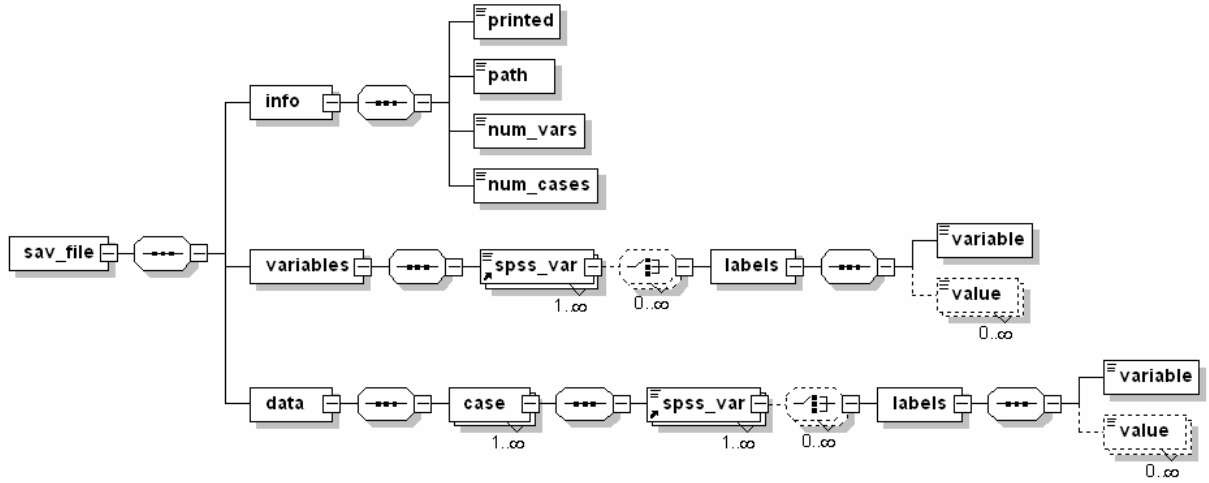
Vi ser at kommnr fra spørreundersøkelsen i 2004 er koblet sammen med knr fra 1996. Ulempen med å løse problemet med historikk på denne måten er at disse koblingene må gjøres manuelt. Og med 800 variabler i spørreundersøkelsen fra 2004 vil dette være en meget omfattende jobb.

4.3 Eksport av data fra SPSS til XML

Som nevnt i kapittel 4.1.2 bestemte vi oss for å eksportere dataene i SPSS til en XML-fil. For å få til dette kjørte vi et script av Tom Derrick[27] som genererte en XML-fil. En ulempe med å benytte seg av dette scriptet er at strukturen på XML-filen er ferdig definert. Dersom vi hadde hatt mer tid til rådighet ville vi satt oss bedre inn i Visual Basic (6.0) slik at kunne forandret litt på denne strukturen.

Vi vil ikke gå i detaljer på koden i scriptet ettersom vi ikke har gode nok kunnskaper i Visual Basic. Det som skjer ved kjøring av scriptet er at dataene i SPSS leses inn i en trestruktur som deretter skrives til en fil.

Etter at vi hadde kjørt scriptet åpnet vi XML-filen i XMLSpy[28] og genererte et XML-skjema basert på denne. Til å beskrive strukturen på XML-filen har vi lagt ved en grafisk fremstilling av XML-skjemaet (se figur 18). Denne figuren illustrerer ganske bra hvordan strukturen på XML-filen er.



Figur 18 – XML-skjema

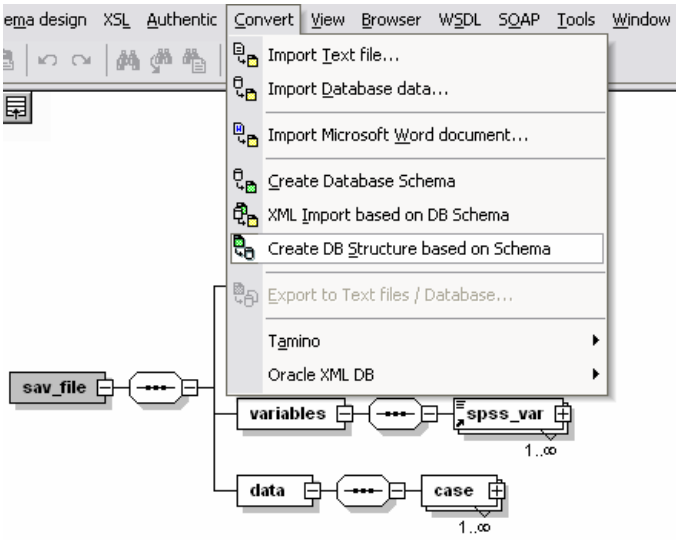
Info-delen inneholder informasjon om hvor mange variabler og hvor mange registreringer vi har. Videre ser vi at variablene er skilt fra dataene, akkurat som i SPSS-filen.
Det eneste som ikke fremgår av dette skjemaet er eventuelle attributter. Vi vil derfor gå gjennom de attributtene som finnes. Elementet <spss_var> har følgende attributter: Name, Type, Width, Decimals, Format, Columns, Align, og Measure. Elementet <case> har attributtet case_num, og elementet <value> har id som attributt.
En innvendning til denne strukturen er at attributtene kunne vært representert som elementer, men dette er ikke av stor betydning.

[27] http://pages.infinit.net/rlevesqu/Scripts/ImportExport/td_ExportAsXMLv2.txt (3. juni 2004)

4.4 Import av data vha. programvare

Det første vi prøvde da vi skulle importere data fra XML-filen til Oracle databasen var å gjøre dette vha. programvare. Etter å ha søkt på Internett fant vi eksempler på hvordan en kunne importere data fra en XML-fil til en Oracle 8i database vha. programvare. Vi gjorde et forsøk der vi prøvde å importere dataene fra XML-filen til Oracle databasen ved hjelp av programmene XMLSpy[28] og Mapforce[28].

XMLSpy har en innebygd funksjon for å opprette en database ut fra et XML-skjema (se figur til høyre). Forutsetningen for at dette skal fungere er at XML-filen er på en tabell basert tilordning (se kapittel 3.3.1.5). Vi kunne altså ikke ta utgangspunkt i den eksporterte XML-filen da denne ikke har en tabellbasert tilordning.



Figur 19 – Oppretting av database i XMLSpy

Vi måtte altså endre på strukturen til XML-filen. Altova tilbyr også et program som gjør dette, kalt Mapforce. Dette er et avansert tilordningsverktøy som kan brukes til å endre strukturen på en XML-fil. Mapforce genererer kode i XSLT, Java, C# og C++ som tilordner data fra ett XML-skjema til ett annet. Vi prøvde først med et par enkle eksempler fra en Mapforce tutorial, og dette gikk helt fint. Men da vi gikk løs på den eksporterte XML-filen fikk vi et problem. Mapforce er godt egnet til å få data fra en struktur til en annen der disse strukturene er gitt på forhånd i form av XML-skjema. Men i vårt tilfelle er vi også ute etter å generere strukturen på dette skjemaet. Dersom vi skulle ha benyttet oss av Mapforce til å få en tabellbasert tilordning på XML-filen måtte vi ha laget strukturen på forhånd. Vi måtte altså ha laget et skjema med 800 elementer (q1, q2,…, q800). Denne jobben var lite fristende, og vi konkluderte med at det ikke lot seg gjøre å importere dataene fra XML-filen til Oracle databasen på denne måten.

[28] http://www.altova.com (3. juni 2004)

4.5 Import av data vha. XSLT

Da vi ikke fikk til å importere dataene fra XML-filen til Oracle databasen vha. programvare tenkte vi å prøve på dette ved å benytte XSLT. Ideen til dette fikk vi da vi så hvordan Mapforce[28] genererte XSLT for å tilordne en XML-fil til en annen. Ettersom dette var nytt og ukjent stoff fant vi på et par enkle eksempler for å se hvordan ting fungerte. Egentlig kan vi vel kalle dette for små prototyper med det formål å undersøke om ting fungerer.

4.5.1 Den første prototypen

Teorien var at vi kunne skrive en XSLT-fil som konverterte XML-filen til SQL-setninger. For å se om det var noe hold i denne teorien prøvde vi først med en litt mindre XML-fil som inneholdt et par personer og telefonnummer. Slik så denne XML-filen ut:

```
<?xml version="1.0" encoding="UTF-8"?>
<PersonRegister>
    <Person>
        <navn>Thomas</navn>
        <telefon>90522123</telefon>
    </Person>
    <Person>
        <navn>Yngve</navn>
        <telefon>23456534</telefon>
    </Person>
</PersonRegister>
```

Figur 20 – Enkel XML-fil

Deretter skrev vi en XSLT-fil som skulle oversette denne XML-filen til SQL-kall for å sette inn de aktuelle personene i en tenkt tabell (se figur 21).

```
<?xml version="1.0" encoding="UTF-8"?>
<?xmlspysamplexml d:\Thomas\Skule\semester 4\DA690 Prosjekt\Prosjekter
(prototyper)\XMLProsjekt_XPath\PersonRegister.xml?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" media-type="text"/>
    <xsl:template match="/PersonRegister">
        <xsl:for-each select="Person">
            INSERT INTO person VALUES('<xsl:value-of select="navn"></xsl:value-of>',
            '<xsl:value-of select="telefon"></xsl:value-of>');
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Figur 21 – XSLT for generering av SQL (versjon 1)

Da vi kjørte denne XSLT-filen i XMLSpy fikk vi følgende resultat:

```
INSERT INTO person VALUES('Thomas', '90522123'); INSERT INTO person
VALUES('Yngve', '23456534');
```

Figur 22 – Resultat

Konklusjonen av dette forsøket var at det sannsynligvis var mulig å generere SQL-kall for oppretting av tabeller og innsetting av data ved hjelp av XSLT. Vi satte derfor i gang forsøk (eller prototyp) nummer to.

4.5.2 Den andre prototypen

Etter det oppløftende resultatet fra den første prototypen gikk vi løs på den eksporterte XML-filen med en gang. Strukturen på denne XML-filen ser vi på figur18 i kapittel 4.3.

Målet med denne prototypen var å generere SQL-kall for oppretting av tabellene i kapittel 4.2.5.1 (kun DATA, TEKST og VALG), samt for innsetting av data. I denne andre prototypen konsentrerte vi oss om Datatabellen (se datamodell). Den endelige XSLT-filen ser vi under på figur 23.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" media-type="text"/>
    <xsl:template match="/">
        CREATE TABLE data(
        <xsl:for-each select="/sav_file/variables/spss_var">
                <xsl:value-of select="@name"></xsl:value-of>
                <xsl:choose>
                    <xsl:when test="@type='Numeric'">
                        NUMBER(<xsl:value-of select="@width"></xsl:value-of>)
                    </xsl:when>
                    <xsl:when test="@type='String'">
                        VARCHAR2(<xsl:value-of select="@width"></xsl:value-of>)
                    </xsl:when>
                </xsl:choose>
                ,
        </xsl:for-each>
        );
        <xsl:for-each select="/sav_file/data/case">
            INSERT INTO data VALUES(
            <xsl:for-each select="spss_var">
                <xsl:apply-templates/>
                '<xsl:value-of select="spss_var" disable-output-escaping="no"/> ',
            </xsl:for-each>
            );
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Figur 23 - XSLT for generering av SQL (versjon 2)

Her er et utdrag fra resultatet som ble produsert:

```
CREATE TABLE data(kommnr NUMBER(4), q1 NUMBER(1),...);
INSERT INTO data VALUES('2028','1',..,'kommunalt nÃ¦ringsarbeid',..);
```

Figur 24 – Resultat

Som vi ser av dette utdraget genereres SQL slik det skal. Det er bare et problem, nemlig Æ, Ø, og Å. Disse bokstavene blir til uforståelige tegn i det produserte resultatet. Vi forsøkte vel og lenge å få til dette, men uten suksess.

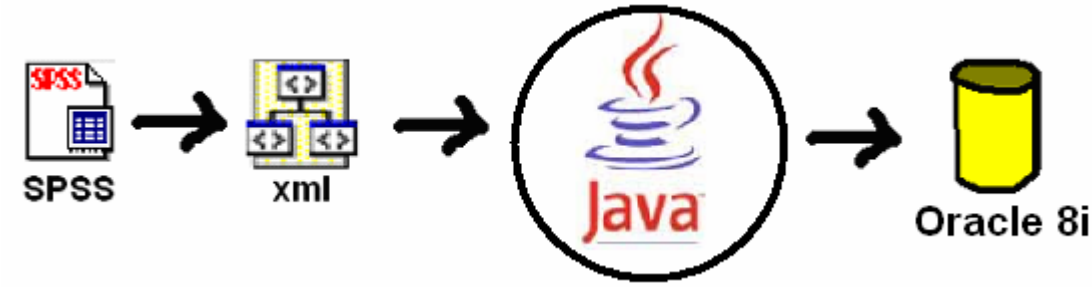
I tillegg til disse språkproblemene var det et ganske omfattende arbeid å sette seg inn i XSLT teknologien. Etter hvert så vi at det ville ta mye lenger tid enn planlagt, og vi bestemte til slutt å gi opp å importere dataene til Oracle på denne måten.

Da vi igjen måtte se oss om etter alternative løsninger tenkte vi på Java som et alternativ da det meste kan løses i Java.

4.6 Utvikling av Java applikasjon for import av data

4.6.1 Innledning

Vi har tidligere i rapporten diskutert forskjellige løsninger for å få dataene fra SPSS over i en database. Som nevnt i kapittel 4.1.2 valgte vi å benytte oss av et script av Tom Derrick for å eksportere dataene i SPSS til XML, og deretter utvikle en Java applikasjon som importerer XML filen til Oracle databasen. Denne veien fra SPSS til Oracle har vi illustrert på figur 25. Vi vil i dette kapittelet si litt om utviklingen av en Java applikasjon som importerer dataene fra XML-filen til en Oracle database. Denne Java applikasjonen ble i første omgang utviklet med tanke på att jobben skulle gjøres en gang og kun for et spesifikt datasett, men vi så tidlig at denne applikasjonen kunne benyttes på en hvilken som helst XML-fil som er generert av scriptet til Tom Derrick. Det er altså mulig å ta en hvilken som helst SPSS fil, kjøre scriptet til Tom Derrick, kjøre Java applikasjonen, og få opprettet en Oracle database. Dermed får denne Java applikasjonen en nytteverdi ettersom den kan brukes ved lignende problemstillinger. Den eneste begrensningen er at datastrukturen blir slik vi definerte i kapittel 4.2.5.1 (gjelder kun entitetstypene DATA, TEKST, og VALG).



Figur 25 – Veien fra SPSS til Oracle

4.6.2 Teori

4.6.2.1Java

Sun microsystems Java J2SE[29] (Java 2 platform, standard edition) versjon 1.4.2 er et objektorientert programmeringsspråk. Java er plattformuavhengig, og en Java applikasjon kan kjøres på for eksempel Windows, Linux eller Mac OS. Vi fant det naturlig å benytte Java til utviklingen av denne applikasjonen da vi kan en del Java fra DA675 – Objektorientert programmering, og DA677 – Algoritmer og datastrukturer.

4.6.2.2 ODBC, JDBC og ADO

ODBC[30] (Open Data Base Connectivity) er en del av Windows operativsystem som holder orden på koblinger mellom programmer og databaser, m.a.o. en standard mellomvare for databaseaksess. En slik kobling er en datakilde.

ADO (ActiveX Data Objects) er en samling objekter som innkapsler en database. ADO kan instansieres og brukes direkte fra et ASP-skript, men kan også aksesseres fra komponenter og applikasjoner skrevet i andre språk, for eksempel Visual Basic.

[29] http://java.sun.com/j2se/index.jsp (3. juni 2004)
[30] http://www.cantor.no/obs2.nsf/0/71e1616e4d33c749c1256b3c00492566 (3. juni 2004)

ADO bruker OLE DB for å knytte seg til databaser. OLE DB kan igjen bruke ODBC. OLE DB er en driver med et lavere nivå grensesnitt, som ADO bruker for å kommuniserer med databasen.

JDBC (Java Data Base Connectivity) definerer en standard API (Application Programming Interface) til hjelp ved utvikling av Java applikasjoner for å kunne aksesserere en database. API er et grensesnitt for eksterne programmer eller systemer for å få tilgang til data i databasen. JDBC ”pakken” definerer en databasetilgang med API som støtter enkel SQL funksjonalitet og som gjør det mulig å aksessere en rekke realsjonsdatabase produkter.

4.6.3 Krav til løsning

Som nevnt i kapittel 2 benyttet vi oss av prototyping ved utvikling av Java applikasjonen. I vårt tilfelle vil den siste prototypen bli det endelige systemet. Målet med den første prototypen var å undersøke om det var vanskelig eller i det hele tatt mulig å opprette databasen (se klassediagram kapittel 4.2) ved hjelp av en Java applikasjon. Til den første prototypen la vi til grunn følgende krav:

- Skal lese XML-fil inn i en datastruktur(DOM)
- Skal generere SQL for oppretting av Data-tabellen (se datamodell i kapittel 4.2.5.1).

Da vi hadde utviklet den første prototypen så vi at applikasjonen kunne brukes mer enn en gang, og vi la til følgende krav til løsning:

- Skal tilordne dataene i XML-filen til en klassestruktur
- Skal generere SQL for oppretting av tabellene Data, Tekst, og Valg (se datamodell i kapittel 4.2.5.1).
- Skal generere SQL for innsetting av data.
- Skal koble til en Oracle 8i database og utføre SQL kall direkte
- Skal tilby et enkelt brukergrensesnitt
- Skal være enkelt å forbedre brukergrensesnittet

Den andre prototypen har en noe begrenset funksjonalitet, men likevel tilstrekkelig til å løse det spesifikke problemet vårt. Vi vil derfor ikke utvikle denne prototypen videre innen tidsrammene til denne prosjektoppgaven, men vi har kommet frem til et par krav som eventuelt bør implementeres i neste versjon:

- Støtte for flere databasetjenere enn Oracle
- Mulighet til å velge mellom å lagre SQL eller å utføre SQL direkte mot database
- Ta hensyn til bokmål/nynorsk endring i XML-filen.

4.6.4 Den første prototypen

4.6.4.1 Beskrivelse

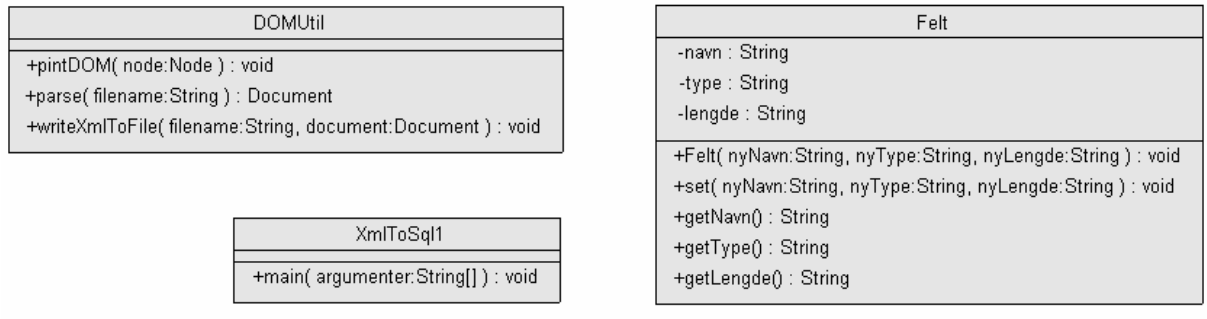
Den første prototypen var som nevnt tidligere utviklet med tanke på å løse en bestemt jobb en gang. På klassediagrammet ser vi at den første prototypen består av tre klasser. Felt-klassen representerer en Variabel fra SPSS-filen. DOMUtil-klassen er en klasse hentet fra Internett[31] med tre statiske metoder. Den ene metoden(parse) leser en XML-fil og returnerer et objekt av typen Document. Et Document objekt er et såkalt DOM-tre (se kapittel 3.4.1.2). I denne prototypen valgte vi å legge mesteparten av koden i main-

[31] http://developers.sun.com/sw/building/codesamples/dom/src/DOMSample.zip (3. juni 2004)

metoden til klassen XmlToSql1. Det er altså her at mesteparten av jobben gjøres. Hvilken fil som skal leses, hvor SQL-filen skal lagres, etc. er representert med vanlige strengvariabler i main-metoden. Vi vil ikke gi eksempler fra koden til den første prototypen, men vi kan beskrive det som skjer slik:

- XML-fil leses inn i en trestruktur (DOM-tre)
- Traverserer DOM-tre og legger Felt-objekt inn i et array.
- Går gjennom array og genererer SQL for opprettelse av Data tabellen
- Lagrer SQL til fil

4.6.4.2 Klassediagram



4.6.4.3 Evaluering

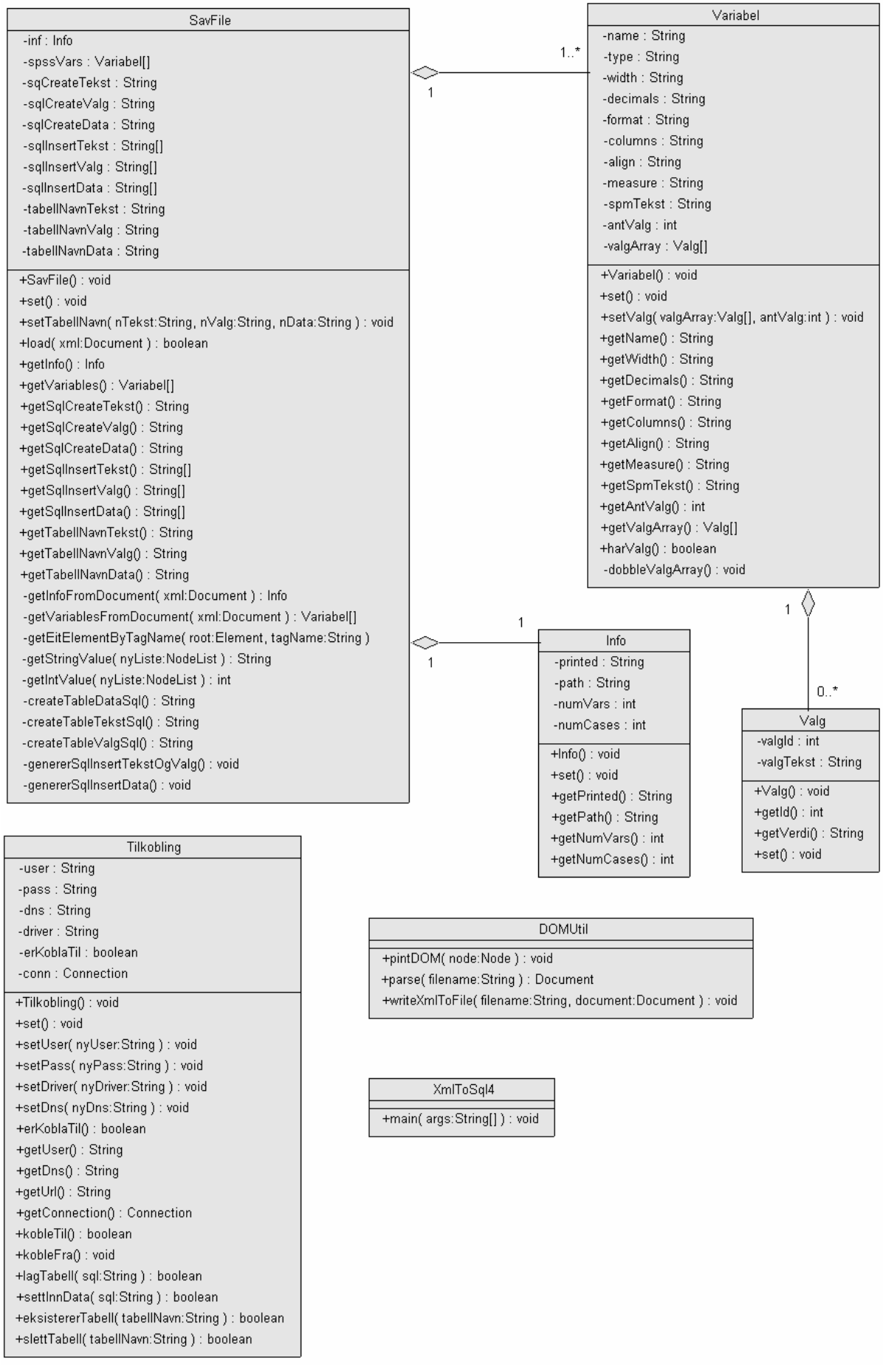
Det andre steget i prototyping er å evaluere prototypen (se kapittel 2.4). En slik evaluering kan også endre på kravspesifikasjonen, noe som var tilfellet hos oss. Denne første prototypen hjalp oss nemlig med å se potensialet og mulighetene som lå der. Noe som igjen førte til en drastisk endring på kravspesifikasjonen. Bortsett fra dette kom det ikke så mye godt ut av denne prototypen. Koden bærer preg av dårlig objektorientert programmering, og gir liten mulighet for gjenbruk. Det eneste som ikke blir skrevet om i prototyp nummer 2 er klassen DOMUtil.

4.6.5 Den andre prototypen

4.6.5.1 Beskrivelse

Denne prototypen har tilstrekkelig med funksjonalitet for å løse oppgaven med å importere XML-filen til en Oracle database. I denne prototypen la vi vekt på å få til en god tilordning av dataene i XML-filen til en klassestruktur. Denne tilordningen bidrar til å dele hoved-problemet opp i mindre del-problem. Vi benyttet oss altså av den såkalte splitt og hersk strategien(Weiss, 2002:473), noe som skal gi god objektorientert programmering. Vi la også vekt på at det skulle være enkel å utvikle et bedre brukergrensesnitt på et senere tidspunkt. Denne andre prototypen er på 1500 linjer med kode.

4.6.5.2Klassediagram



4.6.5.3 Tilordning til klassestruktur

Vi har valgt å tilordne XML-filen til en klassestruktur da dette hjalp oss med å dele opp hovedproblemet i små del-problemer. Vi vil her gå gjennom og forklare hvordan vi har tilordnet strukturen i XML-filen til en klassestruktur.

Vi ser at klassen SavFile i klassediagrammet på side 39 representerer en importjobb, samt roten i XML-filen. Denne klassen har kun en konstruktør, og denne er uten parameter. Load-metoden tar et objekt av typen Document, altså et DOM-tre, som argument. Når denne metoden kjøres traverserer applikasjonen treet, og instansvariablene får verdier. Ved å kjøre koden i figur 26 vil XML-filen leses inn i en trestruktur og videre inn i en klassestruktur. Deretter vil den generere SQL for oppretting av tabeller og innsetting av data. Disse SQL kallene er lagret som vanlige strengvariabler og array med strenger.

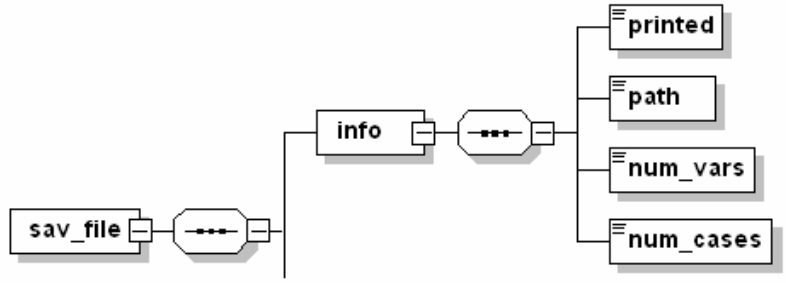
```
Document  xml = DOMUtil.parseSavFile(”test.xml”);
SavFile   sf = new SavFile();
sf.setTabellNavn("TEKST","VALG","KDATA");
sf.load(xml);
```

Figur 26 – Kodeeksempel for instansiering av SavFile-objekt

Load-metoden gjennomfører mange og ressurskrevende operasjoner, og vil derfor ta noe tid å kjøre. Vi har valgt å la denne metoden returnere en boolean-verdi som indikerer hvorvidt operasjonen var vellykket eller ikke. Dersom vi hadde hatt litt bedre tid ville vi håndtert eventuelle feil på en annen måte. Dette har vi diskutert i evalueringen av denne prototypen.

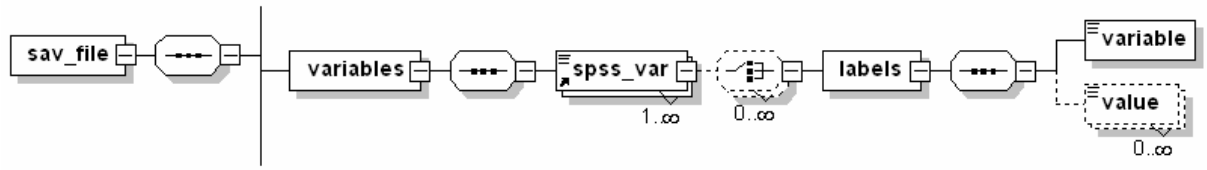
SavFile-klassen har aksessor-metoder som returnerer SQL kallene som ble generert. Det vil altså være enkelt å utvide applikasjonen slik at det blir mulig å lagre disse SQL kallene til disk.

Info-klassen representerer info-elementet i XML-filen som vist på figur 27. Her ser vi tydelig hvordan en kan tilordne elementer i en XML-fil til en klassestruktur. Elementene printed, path, num_vars, og num_cases har blitt instansvariabler i klassen Info. En slik tilordning er ganske enkel og grei når dataene i XML-filen er representert som elementer. Dersom en velger å benytte seg av attributter kan det være litt verre å se hvordan tilordningen blir, men prinsippet er det samme. På klassediagrammet ser vi at vi har en 1-til-1 komposisjon mellom SavFile og Info klassen. Dette betyr at det finnes ett Info-objekt per SavFile-objekt. Denne koblingen er en komposisjon ettersom et Info-objekt ikke kan eksistere uavhengig av SavFile-objektet(Oestereich, 1999:245).



Figur 27 – Struktur på Info-delen i XML-filen

Variabel-klassen representerer en variabel fra SPSS eller XML-filen. På figur 28 ser vi hvordan variablene er strukturert i XML-filen. Elementet spss_var representerer altså en variabel. Instansvariablene i Variabel-klassen som name, type, width, etc. er representert som attributter til spss_var elementet i XML-filen. Ved traversering av DOM-treet er det enklere å forholde seg til attributter med hensyn til koding, men det kan være litt vanskeligere å se hvordan en skal få en XML-struktur inn i en klassestruktur. Dersom vi ikke hadde benyttet oss av scriptet til Tom Derrick, men definert strukturen på XML-filen selv ville vi representert disse attributtene som elementer istedenfor.



Figur 28 – Struktur på Variabel-delen i XML-filen

En variabel i SPSS inneholder informasjon om type, målestokk, spørsmålstekst, og teksten på eventuelle svaralternativer. På figur 29 ser vi et eksempel på en variabel fra XML-filen. Vi ser at spørsmålet har to svaralternativ. Disse svar alternativene er i XML-filen lagret i value elementet. Som vi ser av skjemaet på figur 28 så er value elementet valgfritt ettersom ikke alle spørsmål har svaralternativer. I klassestrukturen har vi laget en klasse Valg, som er en null-til-mange komposisjon til klassen Variabel. Denne Valg-klassen representerer ett valgalternativ.

```
<spss_var name="q1" type="Numeric" width="1" decimals="0"
format="F1"
  columns="8" align="Right" measure="Scale">
      <labels>
            <variable>Hvilket prinsipp er kommunens politiske
            styringsmodell basert på?</variable>
            <value id="1">Formannskapsprinsippet</value>
            <value id="2">Det parlamentariske prinsipp</value>
      </labels>
</spss_var>
```

Figur 29 – Eksempel på en variabel i XML-filen

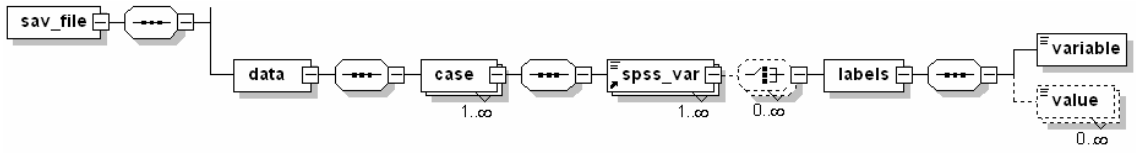
4.6.5.4 Generering av SQL

Etter at dataene i XML-filen er lastet inn i en klassestruktur genererer load-metoden SQL. Disse SQL kallene blir deretter lagret i SavFile klassen sine instansvariabler. På figur 30 ser vi metoden som genererer SQL for oppretting av datatabellen. SQL koden for oppretting av tekst- og valgtabellen er fast, ettersom strukturen på disse tabellene er fastsatt ut ifra datamodellen (se kapittel 4.2.5.1).

```
/*************************************************
* @return SQL-syntaks for oppretting av Data-tabellen
*************************************************/
private String createTableDataSql()
{
      String sql="create table "+tabellNavnData+" (";
      for(int i=0;i<inf.getNumVars();i++)
      {
            Variabel v = spssVars[i];
            sql+="\n"+v.getName() + " ";
            if(v.getType().equals("Numeric"))
                  sql+="NUMBER("+v.getWidth()+")";
            else //string
                  sql+="VARCHAR2("+v.getWidth()+")";
            if(i<(inf.getNumVars()-1))
                  sql+=",";
      }
      sql+=")";
      return sql;
}
```

Figur 30 – Kodeeksempel for generering av SQL

Som vi ser av klassediagrammet så er SQL kallene for innsetting av data lagret i et array med strenger. SQL for innsetting av data i Data tabellen blir generert ved å gå gjennom selve XML-filen, og ikke en samling med objekter. Dette er fordi vi ikke har tilordnet selve datadelen i XML-filen til en klassestruktur. Strukturen på datadelen i XML-filen ser vi på figur 31.



Figur 31 – Struktur på datadelen i XML-filen

Vi har også lagt ved et kodeeksempel fra metoden som genererer SQL for innsetting av data på figur 32. I denne metoden har vi en nøstet for-løkke, antall operasjoner som vil bli utført i denne metoden er antall variabler ganger antall cases. Denne metoden er også et godt eksempel på hvordan man kan traversere et DOM-tre bestående av elementer og attributter.

```
/*******************************************
* Metode for å generere SQL for innsetting av data
* i DATA-tabellen.
*******************************/
private void genererSqlInsertData(Document xml)
{
      Element     rota = xml.getDocumentElement();
      Element     dataElement = getEitElementByTagName(rota,"data");
      NodeList    caseListe =
dataElement.getElementsByTagName("case");

      for(int i=0; i<inf.getNumCases();i++)
      {
         String   sql="INSERT INTO "+tabellNavnData+" VALUES(";
         Node     tmp = (Node) caseListe.item(i);
         if(tmp.getNodeType() == Node.ELEMENT_NODE)
         {
            Element  caseElement = (Element) tmp;
                  String   caseId =
caseElement.getAttribute("casenum");
             NodeList sVarsListe =
                  caseElement.getElementsByTagName("spss_var");
             for(int j=0;j<sVarsListe.getLength();j++)
             {
                   Node  t = (Node) sVarsListe.item(j);
                   if(t.getNodeType() == Node.ELEMENT_NODE)
                   {
                         Element  sVariabel = (Element) t;
                         NodeList sListe= sVariabel.getChildNodes();
                               String   verdi =
getStringValue(sListe);
                         sql+="'"+verdi+="'";
                         if(j<inf.getNumCases()-1)
                               sql+=", ";
                   }
             }
             sql+=")";
         }
         sqlInsertData[i]=sql;
      }
}
```

Figur 32 – Kodeeksempel for generering av SQL

Da vi utviklet denne applikasjonen var kravet at den skulle kommunisere med en Oracle database. I figuren over har vi brukt SQL-typene NUMBER og VARCHAR2.

4.6.5.5 Tilkobling til databasen

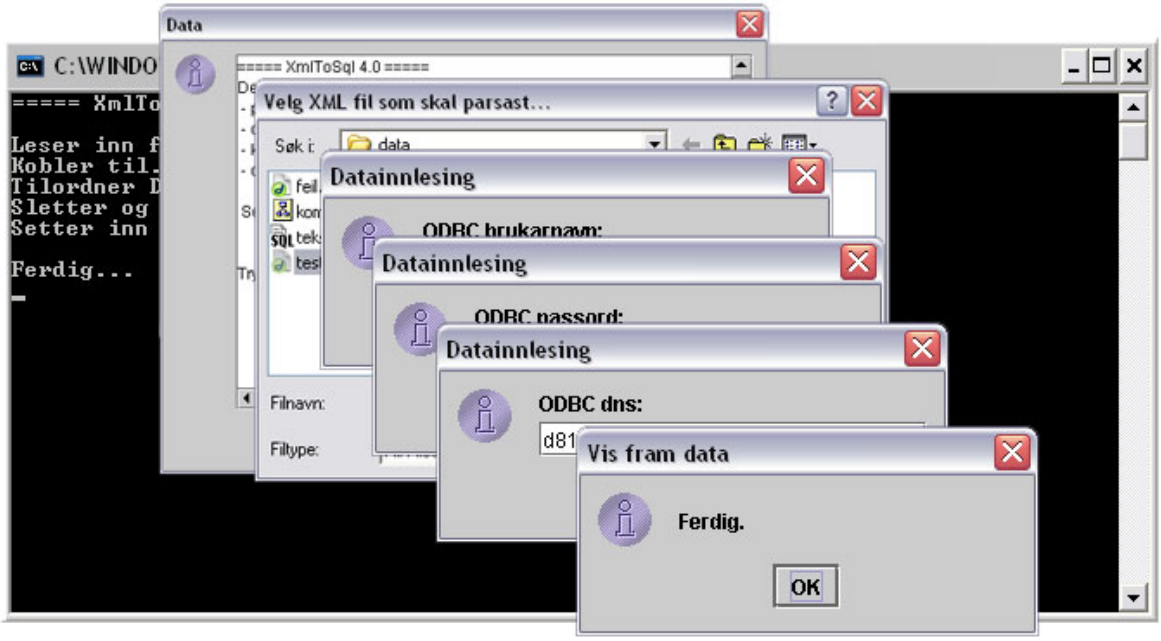
Vi har laget en egen klasse Tilkobling som tar seg av kontakten med databasen. Denne klassen har en instansvariabel av typen Connection. Dette objektet er den egentlige koblingen mot databasen. På figur 33 ser vi koden for å koble til en Oracle database med en ODBC-JDBC bridge driver og opprette en tabell. For at denne koden skal fungere må det finnes en ODBC-kilde med navnet ”kilde”. ODBC og JDBC har vi diskutert i kapittel 4.6.2.2.

```
import java.sql.*;
import sun.jdbc.odbc.*;

// utelatt kode //

try{
Class.forName(”sun.jdbc.odbc.JdbcOdbcDriver”);
Connection  conn=DriverManager.getConnection(”jdbc:odbc:kilde”,”usr”,”p”);
Statement   s = conn.createStatement();
s.executeUpdate(”CREATE TABLE test(id Varchar2(3), navn Varchar2(20))”);
}
catch(Exception e){
e.printStackTrace();
}
```

Figur 33 – Kodeeksempel for tilkobling til database

4.6.5.6 Brukergrensesnitt

Som nevnt tidligere ønsket vi at applikasjonen skulle tilby et enkelt brukergrensesnitt. Med det mener vi at det skal finnes et enkelt brukergrensesnitt som tilbyr et minimum av interaktivitet mellom bruker og applikasjon. På figur 34 ser vi et skjermbilde som viser hvordan brukergrensesnittet ser ut i den andre prototypen.



Figur 34 – Skjermbilde fra applikasjonen

Som nevnt tidligere har vi lagt vekt på at det skulle være enkelt å utvikle et bedre brukergrensesnitt på et senere tidspunkt. Ettersom vi ikke har prioritert utviklingen av brukergrensesnittet har vi løst denne delen på en enklest mulig måte. Vi har benyttet oss av en verktøys-klasse[32] utviklet av Morten Simonsen til innlesing og presentasjon av data. Ett unntak er dialogboksen for å velge XML-fil. På figur 35 ser vi koden for å lage denne dialogboksen.

[32] http://www.hisf.no/aos/ibh/da675/docs/simTools/Simonsen.html (3. juni 2004)

```
FileDialog  f = new FileDialog(new JFrame("Bla gjennom"),"Velg XML
fil");
f.setVisible(true);
String filnavn = f.getDirectory() + f.getFile();
```

Figur 35 – Kodeeksempel for å vise dialogboks for velging av fil

4.6.5.7 Evaluering

Det andre steget i prototyping er evaluering. Etter dette steget kan vi enten kaste prototypen, fortsette utviklingen, eller komme ut med en fungerende versjon. Som nevnt tidligere var hovedkravet til denne Java applikasjonen at den skulle løse vårt spesifikke problem, og dersom den kan brukes av andre blir dette bare en bonus. Ut ifra dette har vi valgt å ikke utvikle applikasjonen videre innen den gitte prosjektperioden. Applikasjonen fungerer i den forstand at den oppretter tabeller og setter inn data i en Oracle database basert på XML-filen som leses inn. Men selv om den fungerer, er det likevel mange momenter som kan forbedres.

Som vi ser av Tilkobling-klassen i klassediagrammet på side 39 så har vi løst eventuelle feil som måtte oppstå ved å la enkelte metoder returnere falsk dersom det oppstår en feil. Dette mener vi burde gjøres på en annen måte. Istedenfor at metodene som kommuniserer med databasen returnerer enten sann eller falsk, burde disse metodene delegere unntak som oppstår og la det bli opp til den som bruker disse metodene å ta seg av feilbehandlingen. Dette kan gjøres ved å angi bak metode deklarasjonen hvilke unntak som skal delegeres videre (se figur 36).

```
/**************************************
* @param s SQL for oppretting av tabell
**************************************/
Public void lagTabell(String s) throws SQLException
{
      Try
      {
            //utfør kritisk operasjon (utelatt kode)
      }
      catch(Exception e)
      {
            throw exception e;
      }
}
```

Figur 36 – Kodeeksempel for bedre feilbehandling

Applikasjonen er også litt ”grådig” på minnebruken. Dersom vi skulle ha videreutviklet applikasjonen ville vi sett på hvordan dette kan gjøres bedre. Vi ville også vurdert å utvikle et bedre brukergrensesnitt, og i minste fall gjøre det mulig å lagre SQL til fil istedenfor å utføre disse direkte mot Oracle.

4.6.5.8 Laste ned program og kildekode

Programmet er tilgjengelig sammen med kildekode for nedlasting på følgende Internettadresse:
http://kjelda.hisf.no/~thotved/XmlToSql/index.html (03.juni 2004)

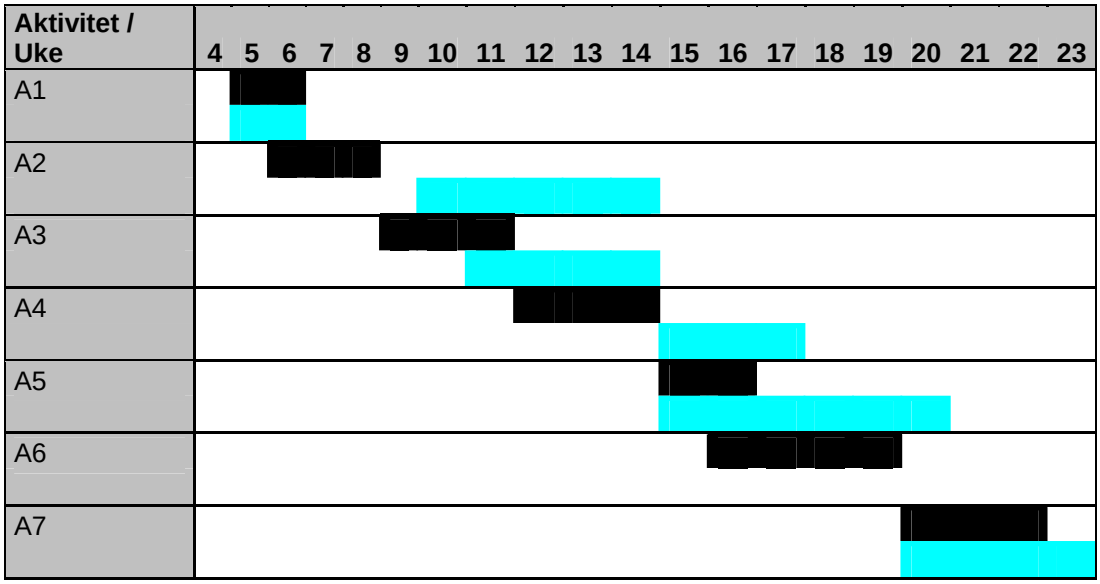
5 OPPSUMMERING OG VURDERING

5.1 Problemer underveis

Vi har i denne prosjektoppgaven vært plaget en del med forsinkelser og andre problemer, og vi vil i dette kapittelet se litt nærmere på hva som forårsaket dette. I kapittel 1.5 definerte vi følgende aktiviteter som skulle gjennomføres i løpet av prosjektperioden:

A1	Formulere problemstilling og utarbeide tidsplan
A2	Analysere dagens SPSS-baserte datagrunnlag
A3	Bestemme organisering av datagrunnlag
A4	Utvikle database?
A5	Overføre data fra SPSS til database?
A6	Utvikle brukergrensesnitt
A7	Ferdigstille rapport

Vi har også lagt ved gantt-diagrammet fra kapittel 1.5. Her har vi i tillegg til den planlagte tidsbruken(markert med svart) markert den faktiske tidsbruken med lyseblått.



Figur 37 – Gantt-diagram med faktisk tidsbruk

Som vi ser av diagrammet så stemmer ikke den planlagte tidsbruken så godt overens med den tiden vi faktisk brukte på aktivitetene. Den første aktiviteten, som var å formulere problemstilling og utarbeide en tidsplan, er faktisk den eneste som ble utført innen de planlagte tidsrammene.

NIBR er den institusjonen som behandler svarene fra undersøkelsen. Vi ble lovet en rådatafil i SPSS-format fra NIBR, kun med spørsmålsdefinisjonene, i slutten av uke 6 eller begynnelsen av uke 7. Denne fikk vi først 2. mars, altså 25 dager senere.
Før vi i det hele tatt kunne analysere dagens SPSS-baserte datagrunnlag (A2) måtte vi ha tilgang til den nevnte rådatafila for å kunne se strukturen på fila. Denne forsinkelsen på nesten en måned var meget alvorlig ettersom hele prosjektperioden var på litt mer enn fire måneder. I denne tiden var vi meget frustrerte, og vi fikk føle på kroppen hvor frustrerende det kan være å være avhengig av eksterne aktører for å kunne følge fremdriftsplanen. Disse eksterne aktørene var ikke direkte involvert i prosjektet vårt, og

et problem var at de ikke hadde kjennskap til de tidsbegrensningene vi hadde å forholde oss til. Vi vurderte i denne perioden å finne en ny prosjektoppgave, men ettersom vi fikk stadig nye løfter om at filen skulle komme snart gjorde vi ikke dette. I denne perioden hadde vi ikke mulighet til å jobbe med den problemstillingen vi har definert i kapittel 1.3. Tiden vi hadde til rådighet brukte vi til å sette oss bedre inn i SPSS og XML samt å skrive litt på teoridelen til rapporten. Grunnen til at vi valgte å se nærmere på XML var at vi hadde en forestilling om at vi kunne bruke dette som et utvekslingsformat.

Den versjonen av SPSS vi hadde tilgjengelig var SPSS v. 11.5. Den nyeste versjonen av SPSS på markedet, SPSS v. 12.0, hadde vi ikke tilgjenglighet til, da høgskolen ventet og ventet på lisensen. Vi forhørte oss jevnlig med it-ansvarlig om lisensen var kommet, og først 9. april ble SPSS v.12.0 installert. Denne siste versjonen har utvidede muligheter for eksport til for eksempel XML, men ettersom lisensen kom så sent hadde vi ikke tid til å sette oss inn i eksportmulighetene til denne versjonen.

Som vi ser av gantt-diagrammet hadde vi planlagt å bruke tre uker på å analysere dagens SPSS-baserte datagrunnlag. Den faktiske tiden vi brukte på dette var rund fem uker. En av grunnene til dette var at vi ønsket å ta med eksportmulighetene som finnes i SPSS i den nevnte analysen. For å gjøre dette måtte SPSS-filen inneholde noen data. Vi tastet inn fiktive svar, noe som tok tid, men svarene vi la inn i SPSS ga ingen sammenheng. For å lette arbeidet videre ble vi lovet en ny datafil i SPSS-format med et par testdata av svarene som NIBR hadde samlet inn. 30. mars fikk vi oversendt en fil med svar fra 10 kommuner. Selv om dataene ikke hadde gjennomgått noen form for kvalitetskontroll, var dette et bedre grunnlag å jobbe videre med enn de fiktive svarene vi selv hadde lagt inn.
En annen grunn til at denne analysen tok lenger tid enn planlagt var at vi jobbet parallelt med den tredje aktiviteten, som var å se på forskjellige organiseringsmuligheter av datagrunnlaget. Her så vi på om det var mulig å generalisere datamaterialet ved å la datagrunnlaget være XML.

Aktivitet nummer fire, utvikling av datamodell, ble forsinket i forhold til den opprinnelige planen som et resultat av at de foregående aktivitetene ble forsinket.

I den opprinnelige planen hadde vi satt av 3 uker til arbeidet med å overføre data fra SPSS til en database (aktivitet nr. 5). Dette viste seg å være langt fra sannheten, og vi endte opp med å bruke seks uker på dette. I utgangspunktet trodde vi det skulle være enkelt å importere dataene fra SPSS til en database, men dette var vanskeligere enn vi trodde.

Som et resultat av alle de foregående forsinkelsene ble aktivitet nummer 6, utvikling av brukergrensesnitt, ikke gjennomført. I mandatet (kapittel 1.2) var et punkt å utarbeide enkelte spørringer etter nærmere definisjon. Med dette forsto vi at vi skulle begynne på utviklingen av brukergrensesnittet. Vi har jobbet med prosjektet hver mandag, onsdag og fredag (med noen unntak) gjennom hele prosjektperioden, og vi vil påstå at det ikke har vært noe å utsette på arbeidsinnsatsen. Vi mener at dersom den første datafilen fra NIBR ikke hadde vært en måned forsinket ville vi klart å gjennomføre alle punktene i den opprinnelige fremdriftplanen. Således mener vi at hovedgrunnen til at vi ikke har komt helt i mål skyldes forsinkelser hos eksterne aktører.

5.2 Konklusjon

Mandatet vi fikk overlevert av prosjektleder fra Vestlandsforskning inneholder følgende 6 punkter:

- Lage en analyse av dagens SPSS-baserte datagrunnlag
- Utvikle en datamodell
- Importere statistikk data fra SPSS til database
- Eksportere databasegrunnlag til XML-struktur
- Utarbeide enkelte spørringer etter nærmere definisjon
- Vurdere generalisering av datamaterialet

Vi har som nevnt i forrige kapittel ikke oppfylt alle disse punktene.

Det første punktet har vi skrevet om i kapittel 3.4.2 og det siste punktet, som går på å vurdere en generalisering av datamaterialet, har vi dekket i kapittel 3.3. Utviklingen av datamodellen er ikke helt ferdig, men vi har sett litt på hva som bør forbedres i neste datamodell

Selve importen av statistikk data fra SPSS til databasen mener vi at vi har løst på en meget god måte. Her har vi utviklet en Java applikasjon som kan importere data fra en SPSS-fil til en database. Denne Java applikasjonen er generell og kan benyttes på et hvilket som helst datasett i SPSS. Dette betyr at andre som kommer bort i en lignende problemstilling som oss kan ha nytte av arbeidet vi har utført.

Vi har ikke eksportert databasegrunnlaget til XML-struktur (punkt nr.4). Dette mener vi ikke er nødvendig da vi allerede har en XML-fil fra eksporten fra SPSS. Dersom noen ønsker data i form av XML mener vi det er et like bra alternativ å tilby hele denne XML-filen.
Det femte punktet i mandatet var å utarbeide enkelte spørringer etter nærmere definisjon. Dette har vi som nevnt i forrige kapittel ikke gjort.

Selv om vi ikke har oppfylt alle punktene i mandatet håper vi at oppdragsgiver er tilfreds med sluttproduktet tatt i betraktning de problemene vi har støtt på underveis.

For vår egen del kan vi trygt si at vi har hatt bra utbytte av denne prosjektoppgaven. Vi har lært om nye teknologier, noe vi også så på som en utfordring og motivasjon ved start. Vi har fått en relativt god oversikt over XML og tilknyttede teknologier. Vi har lånt mange bøker, men ofte var Internett kilden for oppdatert informasjon. Vi føler også at vi har dratt nytte av kunnskaper vi har tilegnet oss i det toårige informasjonsbehandlingsstudiet i dette prosjektet.

Vi har ikke bare lært å bruke egnede teknikker og ny teknologi, men også hvor forandret arbeidssituasjonen kan bli når tidsplaner ikke kan følges. Sånn sett kan vi si at det å jobbe med dette prosjektet har vært som en reell arbeidssituasjon i en hvilken som helst bedrift, og vi ser klare fordeler av å ha kunnet erfare en slik situasjon.

6 LITTERATURLISTE

- Mark Allen Weiss, Data structures and problem solving using Java, Addison Wesley, 2002.
- Steven Alter, Information Systems, Prentice Hall, 2002.
- Simon North og Paul Hermans, Lær XML på 21 dager, Tano Aschehoug, 2000.
- Bernd Oestereich, Developing Software with UML, Addison Wessley, 1999.
- Elio Bonazzi og Glenn Stokol, Oracle 8i & Java, Prentice Hall, 2001.
- Lasse Berntzen, Utvikling av avanserte Internett-baserte applikasjoner, Nki forlaget, 2002.
- Thomas Connoly, Carolyn Begg og Anne Strachan, Database Systems, Addison Wesley, 1999.
- Elio Bonazzi, Software Engineering with Oracle, Prentice Hall, 2000.
- Charles F. GoldFarb's, XML handbook 4th edition, Prentice Hall, 2002.

7 VERKTØYLISTE

- J2SE, Java standard edition, versjon 1.4.2
- TextPad 4.7.2
- Eclipse platform 2.1.3
- Aonix Ameos UML Developer 9.0
- Altova XMLSpy 2004
- Altova Mapforce 2004
- Modelator 4.0
- Oracle 8i
- SPSS 11.5 og SPSS 12.0
- Microsoft Paint
- Adobe Photoshop CS
- Microsoft Office XP