Forord

Jeg vil gjerne takke min veileder Trond Aalberg for samarbeidet. Møtene har vært gode og engasjerende, og har alltid gitt meg godt med arbeidslyst. Jeg må også takke min samboer, Marit Brodshaug, for at hun har lest gjennom oppgaven og kommet med kommentarer til forbedringer som jeg ikke ville klart å se selv. Hun trenger også en takk for at hun har vært så positiv til arbeidet, også i innspurten.

Sammendrag

Høstprosjektet jeg skreiv før jul konkluderte med at det var viktig å se på hva som kunne gjøres videre med dokumentformater som åpne standarder. Oppgaven viste også at lukkede formater gjorde konvertering og dermed dokumentutveksling vanskelig. Ett åpent format som kunne brukes av alle tekstbehandlerne hadde vært det beste.

OASIS sin spesifikasjon er den eneste åpne dokumentformatstandarden, og denne vil bli utforsket for å se på nye måter å bruke den på. Bruken av tekstbehandlere blir også godt beskrevet, med tanke på de underliggende formatene.

Oppgaven ser på hva bruk av dokumentklasser kan tilføre dokumentformatene, og hvordan modularisering av dokumentformatene kan gjøres ved hjelp av prinsippet om dokumentklasser.

Innhold

Kapittel 1

Innledning

1.1 Introduksjon

Dokumentformatene for kontorstøtteprogrammer har hatt en gradvis utvikling fra filer av ren tekst, via store kompliserte binære formater, til like store, men ikke fullt så kompliserte men mer åpne formater definert i XML.

1.1.1 Dokumentformater

I dag brukes mest de programmene som er enklest å få tak i, og som det er enklest å lære. De som trenger avansert funksjonalitet som ikke er tilgjengelig i kontorprogrammene skjønner når de må bytte til andre verktøy.

Utviklingen av dokumentformatene har ført til at tekstbehandlerne har fått stadig mer funksjonalitet, og dermed blitt større og tregere, og ikke alltid egner seg like godt til den jobben de var ment å løse.

De første tekstbehandlere var tekstbaserte og skreiv ut tekst akkurat som den så ut på skjermen. Den største fordelen i forhold til en skrivemaskin var at man kunne lagre dokumentet og skrive det ut flere ganger, med endringer.

Etterhvert som skriverteknologien ble forbedret, ble tekstbehandlerne utvidet med funksjonalitet for å lage større overskrifter, understreking og å bruke andre skrifttyper. Nye måter å lagre dokumentene på gjorde det også mulig å sette inn sidetall, topptekster, bunntekster, fotnoter og annet automatisk innhold.

Seinere fikk vi også muligheter til å skrive ut bilder, og utvikling på lagringsformater gjorde det mulig å lagre bilder i dokumentene. Det ble også mulig å lime andre frittstående dokumenter, også av helt forskjellige typer, inn i dokumentene.

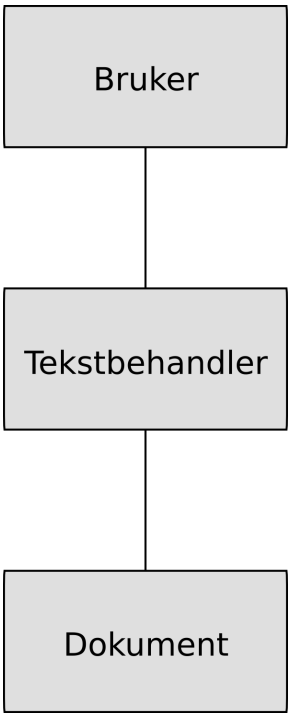
I de siste årene har det blitt stadig mer fokus på XML-formater, strukturen som XML tilbyr har vist seg å være nyttig også for dokumentformatene. Men dette trenger ikke å bety at tekstbehandlere i dag kan utveksle dokumenter på en enkel måte. Det er problemer med kommunikasjon siden det brukes forskjellige dokumentformater, og det er problemer med hvordan strukturen i formatene er bygd opp, som hindrer effektiv uthenting av informasjon i dokumentene.

1.1.2 Skriveprogrammer og tekstbehandlere

Skriveprogrammer er programmer som lar brukerne skrive tekst. Tekstbehandlere er også skriveprogrammer, men det som gjør dem til tekstbehandlere er at de fungerer etter «det

du ser er det du får» eller «what you see is what you get» (WYSIWYG). Brukeren skriver en tekst, og teksten på skjermen er tilnærmet lik det ferdige resultatet.

En tekstbehandler sitter mellom brukeren og dokumentet. Det er tekstbehandleren som mottar kommandoene og lager et dokument. Brukeren kommuniserer med programmet via brukergrensesnittet, og programmet lagrer dokumentet i filsystemet. Dette ser vi i figur 1.1.



Figur 1.1: Forholdet mellom brukere, tekstbehandlere og dokumenter.

1.2 Problemdefinisjon

De fleste tekstbehandlere lar forfatterne bruke all funksjonalitet som finnes i programmet til enhver tid. Mye funksjonalitet blir dermed brukt til ting det ikke var ment for, noe som gjør at dokumentene ikke nødvendigvis får en så strukturert semantikk som XML er laget for.

Oppgaven vil se på om det er mulig å lage en enkel tekstbehandler der forfatterne kun får gjøre dokumentoperasjoner som er beregnet på den type dokument de holder på å skrive, der dokumenttype bestemmes gjennom bruk av dokumentklasser.

Dokumentformatenes spesifikasjoner er også så store at tekstbehandlernes størrelse, både i lagringsplass og minnebruk, nesten er uhåndterlig uten å brukes spesielle triks. Oppgaven vil derfor se på muligheter for å modularisere dokumentformatene for å gjøre ressursbruken mer effektiv.

1.3 Arbeid

Arbeidet i denne oppgaven har vært mangeartet. Høstprosjektet jeg hadde i forrige semester avsluttet med å anbefale å studere OpenDocument videre, som den eneste åpne

standarden for dokumentformater til bruk i avanserte tekstbehandlere.

Jeg har blitt inspirert av LaTeXtil å se på bruken av dokumentklasser, og se hva disse kunne gjøre for en stor spesifikasjon som OpenDocument. Dette har også ført videre til å se om dokumentklassene kunne brukes som utgangspunkt for modularisering av dokumentformatene, etter en kikk på andre måter å modularisere formatene.

Meningen var også å se på bruken av OpenDocument og prøve å endre på en enkel tekstbehandler for å få den til å bruke OpenDocument. Dette viste seg å være veldig tungvint, og etterhvert nøyde jeg meg med å kun bruke dette programmets dokumentformat som utgangspunkt for eksportering til OpenDocument. Dette er presentert i C på side 67, og har havnet litt på siden av oppgaven, slik den har utviklet seg i en mer teoretisk retning.

Tomboy er programmet som jeg har brukt som utgangspunkt for eksporteringen. Dette er et program som jeg kjenner godt til, fordi jeg tidligere har implementert støtte for punktlister for det. Tomboy egner seg også fordi det bruker veldokumenterte komponenter, som er mye enklere å beskrive for dette formålet enn OpenOffice sine API-er. Dette programmet er også interessant fordi det aldri vil kunne implementere en så stor spesifikasjon som OpenDocument har, fordi målet er at det skal være så lite som mulig, og krever en fleksibel bruk av spesifikasjonen.

1.4 Innholdsbeskrivelse

Kapittel 2 på side 9 er et generelt teorikapittel der dagens situasjon blir beskrevet. Kaptitelet starter i 2.1 på side 9 med en generell introduksjon om dokumentformater som finnes og hva som skiller dem. Videre i kapittelet ser vi i 2.2 på side 12 på XML som dokumentformat, hvilke fordeler XML gir, og en introduksjon til forskjellige typer XML-teknologier. Delkapittel 2.3 på side 15 har en lengre utgreiing om standardisering av dokumentformater, og hvordan en god standard defineres. Kapittelet avslutter med en introduksjon til OpenDocument i 2.4 på side 18, og viser hvordan denne spesifikasjonen benytter andre velkjente standarder i sin oppbygning. Vi ser også på hvordan filstrukturen i OpenDocument er satt sammen.

Det neste kapittelet, kapittel 3 på side 23, ser på bruken av dokumentformater. Begynnelsen av kapittelet går med på å forklare hvor utviklingen av dokumenter og de omkringliggende skrivebordsmiljøene er på vei i delkapittel 3.1 på side 23. Etterpå ser vi på hvordan skriveprosessen kan være inndelt i forskjellige faser i 3.2 på side 27, og hvilke verktøy og formater som brukes eller er godt egnet i de forskjellige fasene, og hvordan dokumentenes innhold kan overføres mellom de forskjellige fasene når dokumentformatene er forskjellige. I slutten av kapittelet ser vi på hvordan ett dokumentformat kan brukes til det meste, og hvordan programmer kan bruke et slikt felles format uten å støtte all funksjonalitet fullt ut, i 3.3 på side 36.

Kapittel 4 på side 39 går gjennom forskjellige måter å dele opp dokumentformatene. Delkapittel 4.1 på side 39 ser på hvordan modularisering gjøres i dag, og hva som gjøres med tanke på ressursbruk. I delkaptitel 4.2 på side 40 ser vi på hvordan vi kan dele inn dokumentfunksjonaliteten i dokumentklasser, og hvordan vi kan se for oss bruken av dokumentklasser. Kapittelet avslutter med å se nøyere på modularisering i 4.3 på side 51, hvordan dette kan gjøres på ulike måter, og kommer til slutt med et forslag for modularisering av OpenDocument basert på dokumentklasser.

Det siste kapittelet, kapittel 5 på side 55 avslutter oppgaven. Det starter med en oppsummering, fortsetter med konklusjon og avslutter med forslag til videre arbeid.

Kapittel 2

Teori

Dette kapittelet vil gå gjennom teorier knyttet til dokumentformater. Her ser vi på hva et dokumentformat er, og . . .

Formatene som dokumenter bruker er tilpasset bruksområdet programmet som lagrer dokumentene.

Oppgaven fokuserer på tekstdokumenter, og vil i starten av kapittelet sammenlikne forskjellige tekstlige dokumentformater med filformater for å lagre grafikk og musikk.

2.1 Dokumentformater

Et *dokumentformat*, eller et *filformat for dokumenter*, er i praksis måten dokumenter av en type lagres på. Dokumenter av en bestemt type lagres i dokumenttypens format.

Tekstdokumenter er alle typer dokumenter der tekst utgjør hoveddelen av innholdet. Eksempler på tekstdokumenter:

- Regneark
- Presentasjoner
- Notater
- Brev
- Artikler
- Rapporter

2.1.1 Spredning i formater

Da det en gang på syttitallet ble aktuelt å lagre filer, ble de lagret som ren tekst, og funksjonaliteten var beregnet på å erstatte skrivemaskinene. De digitale dokumentenes fremste egenskap var at de kunne endres, noe som var vanskelig med skrivemaskiner.

Etterhvert ble det behov for å skille ut dokumenter med spesiell funksjonalitet, som regneark, presentasjoner, nettsider, osv. Denne inndelinga gjorde at hver dokumenttype fikk hvert sitt dokumentformat og ble redigert på i hvert sitt program. Hvis en dokumenttype skulle inkluderes i en annen, som et regneark inn i en rapport, ville regnearket bli limt inn i tekstdokumentet i sitt eget format. Det er dette vi ser i dagens tekstbehandlingssystemer.

Tekstdokumentene blir også inndelt enda finere. LaTeX, deler tekstdokumentene inn i dokumentklasser, der typiske eksempler er: bok, rapport og artikkel. Disse klassene kan igjen spesialiseres i nye klasser. De mer brukte tekstbehandlerne spesialiserer også dokumentene i mindre grupper, ved bruk av maler (templates). Vi ser nærmere på dokumentspesialisering ved hjelp av maler og klasser i 4.2 på side 41.

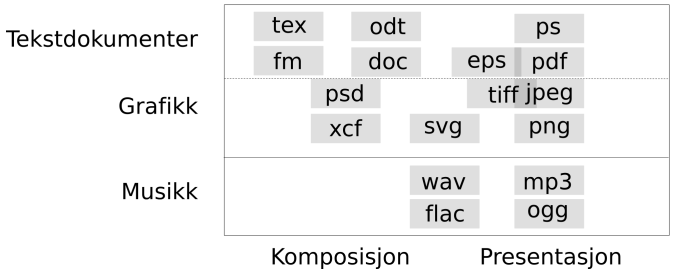
2.1.2 Komposisjon og presentasjon

Det finnes en mengde dokumentformater, laget til forskjellig bruk. Noen dokumentformater er laget for å vises pent, enten det er på skjerm eller papir. Andre dokumentformater er laget for å hjelpe forfatteren mest mulig i komposisjonsarbeidet, enten det er ved å være minst mulig i veien og så raskt som mulig i bruk, bidra med avansert angre- eller tilbakerullingsfunksjonalitet på tvers av lagringer, eller det er å sørge for en ryddig dokumentstruktur som gjør det mulig å gjenoppta arbeidet seinere eller å bruke det til andre formål.

I figur 2.1 sammenlikner jeg enkelte mye brukte filformater innen tekstdokumenter, grafikk og musikk, og deler disse inn i formater for komposisjon og formater for presentasjon.

Komposisjonsformatene, uansett om det er tekst, grafikk eller musikk, er laget for å skrive, tegne eller komponere. Ferdige tekstdokumenter er optimalisert for utskrift eller annen skriftlig presentasjon, i likhet med at ferdige bildeformater er laget for å vise klare, fine bilder, og ferdige lydfiler er laget for å presentere lyd godt.

Å lage og redigere et halvferdig tekstdokument kan sammenliknes med å komponere musikk eller grafikk. En halvferdig tegning trenger en del funksjonalitet som en ferdig tegning ikke trenger, som for eksempel angrefunksjonalitet, komponentbasert tegning på flere lag, og muligheter for å redigere tegningen ved å flytte på eksisterende elementer i den. Tilsvarende finnes det funksjonalitet som man trenger når man komponerer musikk, men som er helt poengløst når man kun er ute etter å høre på musikken. Tekstdokumenter er på mange måter enklere i det ferdige uttrykket enn både grafikk og musikk, men likevel har vi en rekke formater som kun er for presentasjon av tekstdokumenter.



Figur 2.1: Komposisjons- og presentasjonsformater for enkelte medietyper

Musikk som komponeres blir ofte lagret som noter eller som sekvenser, og komponert grafikk blir ofte lagret i et format som PSD (Adobe Photoshop) eller XCF (Gimp), som bidrar med komposisjonsfunksjonalitet som nevnt tidligere.

Ferdig musikk lagres gjerne i et hørbart format, enten som ukomprimert lyd (som WAV, Microsoft Windows audio file format), som tapsløst komprimert lyd (som FLAC, Free Lossless Audio Codec), eller sterkere komprimert lyd (som MP3, WMA OGG). Grafikk som er ferdig lagres gjerne i et lettere format som kan sees uten å eie grafikkprogrammene de er laget i. De lagres da enten ukomprimert (som TIFF, Tagged Image File Format) eller

komprimert (som JPEG, GIF eller PNG). Både musikk og grafikk har tapsløse formater, som FLAC og TIFF som beholder all informasjon i musikken eller bildet [1].

Når musikk og grafikk er ferdig og befinner seg i et presentasjonsformat, går det fremdeles an å gjøre endringer på dem, for eksempel ved å klippe ut utsnitt, forlenge musikk ved å kopiere deler av det som et refreng. Det er også mulig å skalere et ferdig bilde og endre fargedybde, eller endre tonehøyde og jevne ut volum for musikk. Det kan da diskuteres om dette er å endre originalen eller om det er å lage et nytt uttrykk av det gamle, hvor en eventuelt kaster originalen.

Tekstdokumenter skiller seg fra musikk og grafikk ved at tekstdokumentene kan være så mye forskjellig. Det finnes ikke mange typer lagringsformer for forskjellige typer bilder eller musikk, mens det finnes et hav av tekstdokumenttyper, gjerne knyttet til hvert sitt skriveverktøy. Noe som også gjør tekst annerledes, er at tekstdokumenter er enklere å kode, og enklere å vise. Dette gjør at det ikke er like vanlig å skille mellom tekst som komponeres og ferdig tekst som for de andre medietypene.

Enkelte mener at et slikt skille mellom komposisjon og presentasjon også for tekstdokumenter er nødvendig for å gjøre koding av tekst så enkel som mulig, og overlate mest mulig av presentasjonsproblematikken til presentasjonsprogrammene. I vitenskapelige miljøer er det tradisjon for å gjøre dette skillet, der man i stor grad har brukt LaTeXsom komposisjonsformat og PS eller PDF som presentasjonsformat, fordi TeXformaterer teksten tilnærmet typografisk korrekt [2].

PDF er et godt eksempel på et presentasjonsformat for tekst. Det er et veldig godt format til å få ting til å se pent ut, på tross av maskinplattform, og selv om det finnes verktøy som kan endre PDF-dokumenter, så er det ikke det det er laget for. Å bruke et program til å endre på en PDF-fil kan sammenliknes med å gjøre endringer på et ferdig bilde eller en ferdig lydfil.

2.1.3 Stiler brukes til å skille innhold og presentasjon

Komposisjonsformater har i stor grad innført stiler eller liknende virkemidler for å forenkle formatering av dokumentene, og ikke minst for å skille innhold og presentasjon. Før stiler ble innført var det vanskelig å formatere dokumenter på en enkel og enhetlig måte. Det enkleste er å forklare med et eksempel for overskrifter.

Det som tidligere skilte en overskrift fra dokumentets brødtekst, var hvordan den så ut. Man forstørret skrift og gjorde den fet, for å få det til å se ut som overskrifter. Dersom man fant ut at overskriftene skulle ha en annen skrifttype, måtte man gå inn og redigere skrifttypen for hver overskrift individuelt. Det at meningen ble vist ved bruk av formatering gjorde ikke bare formateringsjobben vanskelig, men dokumentene kunne heller ikke tolkes maskinelt på en enkel måte, slik at dokumentet kunne brukes til andre ting.

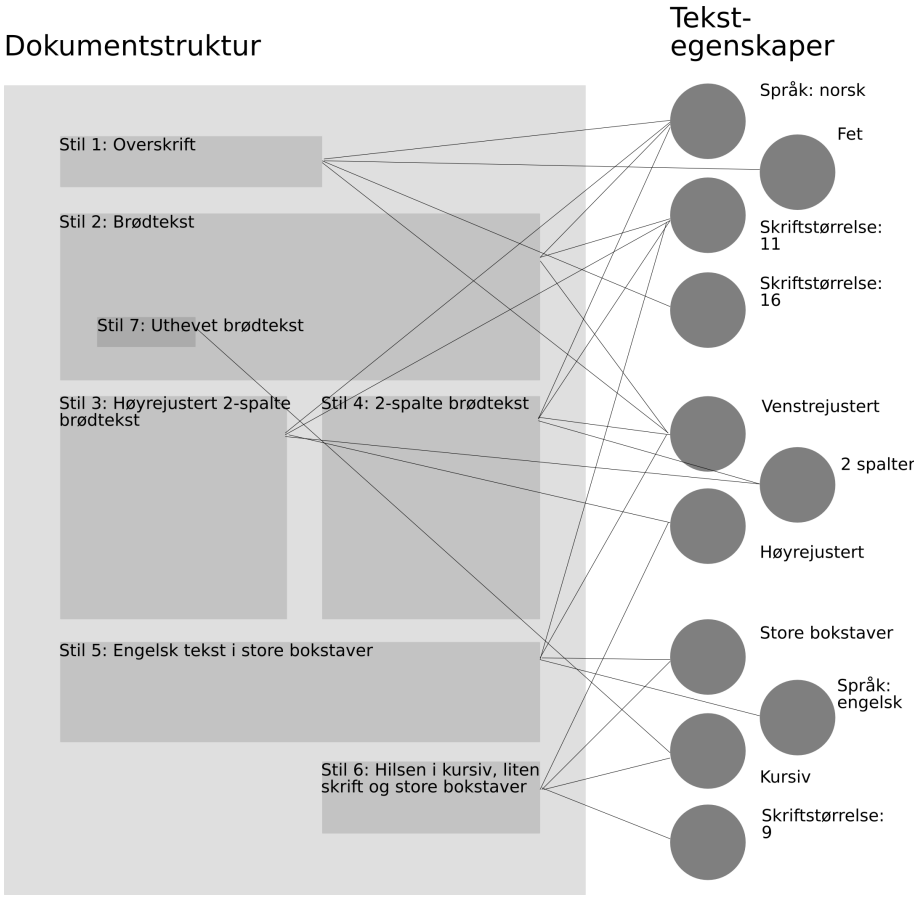
I dag bruker vi stilene til å definere overskrifter, og det følger som standard med en del stiler som kan brukes til det meste. Når man i dag lager en overskrift, gjøres det ved å tilordne en overskriftstil til merket tekst. Stilens formatering bestemmer sånn sett utseendet til overskriften. Når man i dag bestemmer seg for å endre skrifttypen

[1] Musikk som skrives som noter og spilles inn av et orkester kan lett miste mye av lydbildet under opptak, mest fordi det er umulig å fange opp et fullstendig analogt lydbilde. Selv om mye informasjon mistes på denne måten, er det gjerne et rikere lydbilde som skapes enn om det hadde blitt laget fullstendig digitalt uten bruk av analoge instrumenter.

[2] Det har tidligere også vært en grunn at kjente tekstbehandlere ikke har klart å håndtere store dokumenter, noe som har ført til tap av informasjon eller stort ekstra formateringsarbeid. Dette har ført til begreper som «400-sidersgrensa», som henspeiler på at dokumenter er trygge hvis de er godt under 400 sider.

for overskriftene, redigerer man dette kun én plass, i stildefinisjonen for overskrift. Alle overskriftene merket med denne stilen endrer da utseende.

Stilene gjør også at dokumentformatene blir mer semantisk oppbygd. Strukturen består nå av dokumenter med overskrifter og avsnitt, i stedet for masse tekst med forskjellig formatering. Slik strukturering gjør det også mulig å skille innhold og presentasjon, ved at presentasjonen blir kodet et annet sted enn innholdet i dokumentet.



Figur 2.2: Et dokument og dets egenskaper

Figur 2.2 viser et eksempeldokument med en rekke egenskaper. I denne figuren blir ingen av stilene gjenbrukt, selv om det ikke er noe problem å gjøre det.

Vi ser altså en stadig bedre strukturering som gjør at dokumentene blir mer semantisk kodet i lagringsformatene, der presentasjonen har blitt flyttet ut av innholdet. Vi ser mer på automatisk behandling av dokumenter i neste del om XML.

2.2 XML

Extensible Markup Language (XML) er et sett av regler for å bygge opp merkespråk (markup languages). Merker blir lagt til i teksten for å beskrive teksten, ved å identifisere deler tekstelementer, og relatere tekstelementer til hverandre [1]. XML bygger på GML (Generalized Markup Language) og den standardiserte formen SGML (Standard General Markup Language), og innsnevrer funksjonaliteten til SGML og tilbyr heller et utvidbart

subsett [3].

XML har vist seg å være en god måte å beskrive, strukturere og lagre dokumenter og annet tekstlig og ikketekstlig innhold. Det brukes i dag, ikke bare til å kode dokumenter og nettsider, men også til matematiske formler, figurtegninger og annen vektorgrafikk, og mye annet. Figur 2.3 viser hvordan XML-merkene brukes til å strukturere informasjon. Et fullstendig eksempel på en godt definert XML-struktur ligger i vedlegg A på side 63.

```
<message>
        <exclamation>Hello, World!</exclamation>
        <paragraph>XML is <emphasis>fun</emphasis> and
                <emphasis>easy</emphasis> to use.
                <graphic fileref="smiley_face.pict"/></paragraph>
</message>
```

Figur 2.3: Et eksempelutdrag med XML-kode hentet fra [1].

2.2.1 Nye former for tekstbehandling

Det at så mange forskjellige formater bygger på XML, gjør det mulig å ekstrahere og konvertere informasjon fra disse formatene med felles ekstraherings- og konverteringsverktøy. XML tilbyr en mye enklere måte å lese dokumenter på maskinelt enn forrige generasjons binærformater og dårlig strukturerte tekstformater.

Dokumenter, som andre XML-filer, kan leses og tolkes automatisk, slik at informasjonen i dem kan brukes av andre programmer. Dette kan for eksempel være ekstraheringsverktøy, som lagrer informasjonen i databaser, eller programmer som bruker informasjonen direkte til å lage nye XML-dokumenter. XML-filer i ett format kan også konverteres automatisk til et nytt format, dersom strukturen er god. Felles verktøy for XML-prosessering gjør det også enkelt å endre på eksisterende XML-dokumenter, både endre på dokumentelementer og opprette nye elementer i XML-strukturen.

2.2.2 Formater redefineres i XML

XML har blitt brukt til å redefinere andre språk. Hypertext Markup Language (HTML) ble laget av Tim Berners-Lee og Anders Berglund på begynnelsen av 1990-tallet, og er en kompakt og effektiv SGML-dokumenttype for hypertekst. HTML har blitt sett på som et tilbakeskritt av flere grunner. Blant annet fordi HTML kun hadde enn dokumenttype som måtte bli brukt til alle typer hypertekstdokumenter. HTML har også enkelte merker som kun koder presentasjon, og som dermed strider med prinsippet om å skille innhold og presentasjon. XHTML er en reimplementasjon av HTML i XML.

DocBook ble laget som et merkespråk for teknisk dokumentasjon, og ble opprinnelig spesifisert SGML. I de seinere årene har en XML-versjon av DocBook tatt over nesten hele bruksområdet [2], og nok et eksempel på hvordan XML-baserte formater tar over for de SGML-baserte.

2.2.3 Eksempler på XML-teknologier

Det finnes en stor mengde XML-baserte standarder, både for å definere verktøy og prosesser for å arbeide med XML-dokumenter, og en rekke dokumentformater og

[3] Derav navnet Extensible Markup Language

kommunikasjonsprotokoller. Vi vil nå se videre på hvordan XML brukes til å kode andre typer filformater, kommunikasjonsprotokoller, og ikke minst definisjonsspråk som igjen brukes for å behandle XML-dokumenter.

En lengre liste over hva XML brukes til av dokumentformater, kommunikasjonsprotokoller og strukturspråk finnes i [3].

Andre filformater

XML brukes til mer enn tekstdokumenter.

Scalable Vector Graphics (SVG) er et XML-format for å lagre vektorgrafikk. Skalerbar vektorgrafikk kjennetegnes ved at man spesifiserer en tegning som et sett av streker og andre elementer, der man lagrer endepunktene sammen med strekenes utseendeegenskaper, i stedet for å lagre hele bildet punkt for punkt [4].

Synchronized Multimedia Integration Language (SMIL) er et XML-basert språk for å skrive interaktive multimediapresentasjoner. SMIL-komponenter kan også brukes i XHTML og SVG [5].

Mathematical Markup Language (MathML) brukes til å kode matematiske formler ved hjelp av XML [6].

Resource Description Framework (RDF) er et rammeverk for å kode metadata i dokumenter, og koding av relasjoner mellom data gjennom en subjekt-predikat-objekt-modell [7] [8].

Really Simple Syndication (RSS) [4] er en standard i flere versjoner for å tilby nyhetsstrømmer, korte beskrivelser av web-innhold sammen med en lenke til nettstedet med hele innholdet. Dette er en standard som brukes av nettaviser, diskusjonsfora, og andre nettsteder med innhold som endres ofte. RSS startet som et RDF format, men versjon 2.0 bruker ikke lenger RDF [9] [10]. RSS 2.0 inkluderer også en publiseringsprotokoll.

Atom som er definert av IETF, er et annet XML-format og publiseringsprotokoll for nyhetsstrømmer som heller ikke bruker RDF.

Protokoller

XML brukes også i kommunikasjonsprotokoller. XML-RPC (Remote Procedure Call) [11] og SOAP (Simple Object Access Protocol) [12] er de mest kjente protokollene i dag, og brukes blant annet i RSS, Atom og i Web Services. Web Service Description Language (WSDL) [13] er et utvekslingsformat for Web Services, som bruker SOAP som underliggende protokoll. XML-RPC startet som en enkel protokoll til å sende kommandoer over nettet, og har blitt bygd på og fått en større avart, kjent som SOAP. Dette er forholdsvis nye standarder, som tok i bruk XML fordi det var så mye i bruk fra før, og fordi det fantes så mange ferdige verktøy for lesing, tolking og skriving av XML-dokumenter.

Det finnes også andre protokoller med helt mer spesifikke bruksområder. XMPP, med RFC-nummer 3920, er en av dem, og utgjør kjernen av Jabber-protokollen for hurtigmeldinger (Instant Messages) [14]. Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) er en protokoll for å hente ut metadata fra store digitale samlinger [15].

[4] Også kalt Rich Site Summary og RDF Site Summmary i tidligere versjoner

Definisjonsspråk

XSL (Extensible Stylesheet Language) er en rekke standarder for å definere XML-dokumenttransformering og -formatering [16]. XSL består av XSLT (XSL Transformations) [17] som definerer dokumenttransformasjoner, XPath (XML Path Language) [18] som brukes av XSLT til å få tilgang eller referere til deler av et XML-dokument, og XSL-FO (XSL Formatting Objects) [19] som er et sett med regler for å spesifisere formateringssemantikk. XSL-FO sies å være mer gjennomtenkt, og beregnet på mer komplekse strukturer og for mer kompleks visning enn Cascading Stylesheet (CSS) [20]. XPath brukes også av XLink (XML Linking Language) [21], et språk definert i XML for å opprette og beskrive lenker mellom ressurser.

XML Schema [22] er et XML-format for å definere struktur, innhold og semantikk i XML-dokumenter, enten det er dokumentformater, protokoller eller andre XML-strukturer. Dokumenter kan da valideres opp mot et XML Schema for å se om de er korrekt kodet.

XML kan også brukes til å definere spørringer mot XML-databaser. XQuery [23] er et slikt XML-basert databasespørrespråk (query language).

Det finnes også et eget XML-språk for skjemaer og brukerinteraksjon. XForms definerer neste generasjons webskjemaer, og skiller skjemaer i tre deler: modell, instansdata og brukergrensesnitt.

Dokumentformater i XML

I tillegg til HTML og DocBook, ser vi også at XML-formater er tatt i bruk i dokumentformatene til Microsoft Word, Adobe Framemaker og OpenOffice. Microsofts WordprocessingML [24] er laget med utgangspunkt i det tradisjonelle dokumentformatet i Microsoft Word, og vi ser i 3.2.2 på side 31 at WordprocessingML bærer preg av dette [5]. Framemaker har de siste årene kommet i to versjoner, vanlig Framemaker og strukturert Framemaker. Vi vil se i 3.2.2 på side 32 at også dette formatet har visse problemer. OASIS Open Document Format for Office Applications», også kjent som OpenDocument, er et tredje XML-format for dokumenter. OpenDocument er en åpen standarden som prøver å bli universelt dokumentformat, og behandles mer i 2.4 på side 18.

2.3 Standardisering av dokumentformater

For å sikre at dokumenter kan utveksles mellom forskjellige parter og at de skal kunne brukes av dokumenthåndteringsverktøy i lang tid framover, har bedrifter og myndigheter behov for å følge gjeldende standarder, enten man mener at standarden er det mest brukte dokumentformatet, eller et dokumentformat som er standardisert av en uavhengig standardiseringsorganisasjon. Det har alltid vært kniving om hvilket dokumentformater som skal være standarden, og Microsoft har stort sett hatt brukermassen på sin side.

2.3.1 Tidligere standardiserte dokumentformater

På åttitallet og begynnelsen av nittitallet definerte Den internasjonale teleunionens telekommunikasjons standardiseringssektor (ITU-T) et standard dokumentformat som skulle erstatte alle proprietære dokumentformater. «Open Document Architecture»

[5] Petter Merok, Direktør Teknologiutvikling i Microsoft Norge, har sagt at grunnen til at WordprocessingML ikke ble lansert tidligere har vært at formatet har vært for nært knyttet til skrivere, og derfor ikke vært ferdig til bruk som et utvekslingsformat [25].

(ODA) definerer et sammensatt dokumentformat som inneholder ren tekst, bilder og vektorgrafikk. En pilotimplementasjon ble implementert allerede i 1985, og dette dokumentformatet ble en ISO-standard i 1994. Dokumentformatet ble veldig fort utdatert, og ble aldri tatt i utstrakt bruk, men idéene om sammensatte dokumentformater lever videre den dag i dag [26].

DocBook er et annet dokumentformat som er standardisert gjennom en standardiserings-organisasjon. Dette formatet er standardisert gjennom Organization for the Advancement of Structured Information Standards (OASIS), og er opprinnelig et merkespråk for teknisk dokumentasjon, men har vide bruksområder. DocBook er definert i SGML og XML, og dokumenter skrevet i formatet kan konverteres til andre XML-formater med XSL-transformasjoner. Formatet er i utstrakt bruk i dag, til bøker og teknisk dokumentasjon, spesielt innen Open Source-prosjekter [2].

2.3.2 Definisjoner av standarder

Det finnes en del definisjoner av hva som er en standard, og det finnes en del forskjellige typer standarder. Uniform Code Council, som blant annet står bak strekkodene, sier at en standard er [27]:

> A specification for hardware, software, or data that is either widely used and accepted (de facto) or is sanctioned by a standards organization (de jure).

En standard bør være tilgjengelig for de parter som skal bruke den, for at den skal bli brukt riktig. Og myndigheter og andre aktører setter pris på å få vite hvordan informasjonen kodet i dokumenter blir lagret. European Interoperability Framework (EIF) [6] sier at en *åpen* standard må oppfylle fire kriterier [28]:

- Open standards are adopted and managed by a non-profit organisation open to all interested parties,
- Open standards are fully published, and specifications are available without or for a nominal charge,
- Intellectual property is irrevocably made available on a royalty free basis,
- There are no constraints on the re-use of the standard.

Det norske teknologirådet følger opp med mye av det samme [29]:

- Standarden skal maksimere sluttbrukerens valgfrihet og ikke låse denne til en spesifikk leverandør eller gruppe av leverandører.
- Det skal ikke kreves royalties eller gebyrer av de som ønsker å implementere standarden. Sertifisering i forhold til standarden kan derimot koste penger.
- Det skal være lov, innenfor gitte begrensninger, å tilby standarden i redusert eller utvidet form.
- En åpen standard kan ha lisensbetingelser som krever at eventuelle utvidelser offentliggjøres og at alle andre tillates å distribuere programvare som er kompatibel med utvidelsene. Dette for å forhindre en taktikk hvor en leverandør tar i bruk en standard, men utvider den med tillegg som ikke er åpne og tilgjengelige for andre i håp om å løse brukerne til sin programvare.

[6] EIF er et av prosjektene til IDABC (Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens).

Åpne dokumentformater

Vi avslutter definisjonene av åpne standarder med å ta med en definisjon av et åpent dokumentformat. Bob Sutor i IBM sier i [30] at «ekte åpne dokumentformater» i 2005 har visse karakteristikker:

- it is supported by multiple applications with demonstrated interoperability
- it is preferably produced but at least maintained by a standards group with representation from many companies, organizations, and individuals,
- is therefore not under the control of a single vendor who can change the format and the licensing at its whim,
- is available on a royalty-free basis and has no restrictions that might limit its use for any reason in any software, be it customer-unique code, a vendor product or open source.

2.3.3 OASIS Open Document Format for Office Applications (OpenDocument) og Microsoft Open XML Formats

Formatene i OpenOffice, som OpenOffice.org og Sun har utviklet, har blitt sendt til standardiseringsorganisasjonen OASIS, der en arbeidsgruppe ble satt sammen med representanter fra Sun, OpenOffice.org og andre tekstbehandlingsprodusenter som ville bruke et felles dokumentformat. Arbeidsgruppa tok utgangspunkt i formatene brukt i OpenOffice, og endret på disse med stor hjelp fra de produsentene som brukte andre formater, og utviklet en spesifikasjon for office-formater kjent som «OASIS Open Document Format for Office Applications (OpenDocument)». Dette er første gang i den 25 år lange office-historien at et format har blitt utviklet av en uavhengig standardiseringsorganisasjon. Denne spesifikasjonen er på vei til å bli en ISO-standard [31].

Microsoft har lenge vært en stor produsent av programvare, og har fått flesteparten til å bruke verktøyene og formatene deres. Det har med andre ord oppnådd å bli en de-facto-standard. Dokumentformatene til Microsoft har ikke blitt sendt til noen ekstern standardiseringsautoritet, og det finnes også restriksjoner på bruken av formatene. Microsoft har tidligere sendt inn andre formater til standardiseringsorganisasjoner, og har deltatt aktivt i World Wide Web Consortium på en flere av de mest kjente standardene, som for eksempel Web Service Description Language (WSDL) [13].

Microsoft har annonsert [32] at de kommer med nye dokumentformater i 2006, og at disse vil ha en spesifikasjon som er tilgjengelig i god tid i forkant, slik at tredjeparter kan lage programmer som er klare for de nye formatene når de blir lansert. Formatene går under navnet «Microsoft Office Open XML Formats», der Microsoft sin definisjon av «Open» er at det er tilgjengelig [33]. Prosessen for å komme fram til formatet er lukket, og det blir også i de nye formatene juridiske restriksjoner på bruken av dem [34]. Microsoft blir også kritisert [35] for å være medlem av OASIS og bruke medlemskapet til å følge med på utviklingen av OpenDocument uten bidra med å gjøre det brukbart for Word, men i stedet reimplementere sammendraget av det de har lest[7].

[7] De nye formatene vil bestå av flere filer og mapper som er komprimert til én fil. Microsoft lover også å skille mellom innhold og presentasjon på en bedre måte enn i dag. Vi ser mer på hvordan disse tingene gjøres i OpenDocument, seinere i oppgaven.

2.3.4 Standardiseringsorganisasjoner

OASIS, W3C og IETF er tre store uavhengige standardiseringsorganisasjoner som deler IT-verdenen mellom seg. The Internet Engineering Task Force (IETF) er en åpen sammenslutning av nettverkskompetente mennesker som jobber med utviklingen av Internett, og er kanskje mest kjent for alle RFC-ene. World Wide Web Consortium (W3C) lager web-standarder som bygger på protokoller definert av IETF. Veldig mange av standardene som gås gjennom i denne oppgaven er W3C-anbefalinger (standarder eller recommendations). Organization for the Advancement of Structured Information Standards (OASIS), bruker gjerne W3C sine web-standarder i sine «e-business»-standarder.

2.4 OpenDocument

Vi velger å se nærmere på det dokumentformatet som er offentliggjort som en åpen standard, og som er på vei til å bli en ISO-standard.

OpenDocument er definert i XML, og ble laget fra bunnen av da StarOffice så det gamle binærformatet ble utdatert på viktige områder. De trengte et format som egnet seg bedre for dokumentutveksling, og de hadde bestemt at de ville ha formatet i Unicode [8] [36]. Dette skjedde i 1999, ett år etter at XML hadde blitt standardisert, og det var klart at å velge noe annet ville være en stor feil. Valget ble også enklere ved at XML krevde å bli kodet i Unicode. XML-gruppa til StarOffice hadde aldri noen plan om å laget et dokumentformat kun for StarOffice, og etter at StarOffice ble kjøpt opp av Sun i 2000, ble de delene av prosjektet som Sun hadde bidratt på åpnet for allmennheten under navnet «OpenOffice.org». I 2001 ble OpenOffice.org 1.0 og StarOffice 6.0 utgitt med det nye formatet som «native» filformat [31].

Dette OpenOffice.org-filformatet ble overlevert til en OASIS-arbeidsgruppe, med medlemmer fra flere store aktører i tillegg til OpenOffice.org. Gruppa brukte et par år på å standardisere formatet og gjøre det godt nok til å kunne brukes av andre, og et utkast til en spesifikasjon kom ut i desember 2004. Den endelige spesifikasjonen ble godkjent som OASIS-standard i mai 2005 [37].

2.4.1 OpenDocument og andre standarder

Det er interessant å se hvordan andre standarder brukes i spesifikasjonen [37] så langt det har vært mulig, og hvordan navnerom brukes for å holde orden. Navnerommene brukes både til å utnytte tidligere definerte standarder, og for at dokumentformatet kan utvides med nye navnerom av de som måtte ønske det.

I 2.2.3 på side 13 så vi på en rekke XML-teknologier, og OpenDocument utnytter disse og inkluderer dem i spesifikasjonen. Av disse XML-teknologiene bruker OpenDocument XSL for å transformere dokumenter mellom forskjellige formater via såkalte «filter», SVG for å kode figurtegninger og annen grafikk i dokumentene, SMIL for å kode multimediapresentasjoner, XLink for å definere lenkene internt i dokumenter og til andre dokumenter, MathML for å kode matematiske uttrykk, XForms for å definere skjemaer og brukerinteraksjon.

I tillegg til XML-teknologiene brukes Dublin Core-standarden for å kode metadata. Dublin Core er en ISO-standard laget av Dublin Core Metadata Inintiative [38]. Dublin

[8] Unicode-tegnsett prøver å tilby en universell måte å kode alle verdens skrifter og tegn, uansett språk eller plattform.

Core definerer et sett av 15 elementer for å beskrive vanlige metadata, og typiske eksempler er tittel, forfatter, emne, beskrivelse, utgiver, språk [39].

2.4.2 Andre navnerom i spesifikasjonen

Vi har nettopp sett på hvordan OpenDocument bruker andre standarder i dokumentformatet, men dokumentformatet har også hatt behov for å definere egne navnerom, både for å legge til dokumentelementer som ikke er definert i noen standard, og for å utvide allerede utnyttede standarder når disse ikke kan beskrive alle dokumentelementene innefor sitt «område».

I lista som følger ser vi navnerom som har blitt innført for å beskrive dokumentelementer som ikke er dekket av de tidligere definerte standardene. Dr3d er et eksempel på et nytt navnerom som må brukes fordi det ikke finnes noen god standard for å beskrive tredimensjonale data i XML for et dokumentformat. Vi har tidligere sett at Dulin Core blir brukt for å beskrive metadata. De 15 elementene i Dublin Core er imidlertid ikke nok for å beskrive alle typene metadata i OpenDocument, og det har derfor vært nødvendig å lage et navnerom, `meta`, for de metadataelementene som Dublin Core ikke definerer. B på side 65 viser et eksempel på metadata kodet både i Dublin Core-navnerommet og i `meta`. De ekstra navnerommene blir listet opp her:

office (urn:oasis:names:tc:opendocument:xmlns:office:1.0) Generelt navnerom for elementer som ikke er definert av andre, mer spesifikke navnerom.

meta (urn:oasis:names:tc:opendocument:xmlns:meta:1.0) For elementer og attributter som beskriver metainformasjon. Eksempler kan være:

config (urn:oasis:names:tc:opendocument:xmlns:config:1.0) Beskriver programspesifikke innstillinger.

text (urn:oasis:names:tc:opendocument:xmlns:text:1.0) Elementer og attributter som kan ligge i tekstdokumenter og tekstdeler av andre dokumenttyper, som for eksempel en celle i et regneark.

table (urn:oasis:names:tc:opendocument:xmlns:table:1.0) Elementer og attributter som kan ligge i regneark eller tabeller i tekstdokumenter.

drawing (urn:oasis:names:tc:opendocument:xmlns:drawing:1.0) Beskriver grafisk innhold, som strektegninger, sirkler, bokser og piler.

presentation (urn:oasis:names:tc:opendocument:xmlns:presentation:1.0) Beskriver presentasjonsinnhold. (Ikke i bruk i vanlige tekstdokumenter.)

dr3d (urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0) Beskriver tredimensjonalt grafisk innhold.

anim (urn:oasis:names:tc:opendocument:xmlns:animation:1.0) Beskriver animert innhold.

chart (urn:oasis:names:tc:opendocument:xmlns:chart:1.0) Beskriver diagraminnhold.

form (urn:oasis:names:tc:opendocument:xmlns:form:1.0) Beskriver skjemaer og skjemakontroller.

script (urn:oasis:names:tc:opendocument:xmlns:script:1.0) Beskriver skript og hendelser. Blant annet beskrivelse av skript, som kildekode, kompilert kode eller som lenke til ekstern skriptkode.

style (urn:oasis:names:tc:opendocument:xmlns:style:1.0) Beskriver stiler og arvingsmodell som brukes av OpenDocument-formatet. Beskriver også noen felles formateringsattributter.

number (urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0) Beskriver datastilinformasjon.

manifest (urn:oasis:names:tc:opendocument:xmlns:manifest:1.0) For elementer og attributter inneholdt i pakkemanifest.

fo (urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0) Attributter som er kompatible med attributter definert i XSL-standarden [19].

svg (urn:oasis:names:tc:opendocument:xmlns:svg-compatible:1.0) Attributter som er kompatible med SVG-standarden som beskriver vektorgrafikk [4]. Denne standarden brukes for å kode figurtegninger.

smil (urn:oasis:names:tc:opendocument:xmlns:smil-compatible:1.0) Attributter som er kompatible med attributter definert i smil-standarden [5], en standard for koding av animasjoner.

dc (http://purl.org/dc/elements/1.1/) Dublin Core-navnerommet. Beskrevet i [39].

xlink (http://www.w3.org/1999/xlink) XLink-navnerommet, som beskrevet i [21].

math (http://www.w3.org/1998/Math/MathML) MathML-navnerommet. Beskriver merkingen av matematiske uttrykk. Beskrevet i [6].

xforms (http://www.w3.org/2002/xforms) XForms-navnerommet, beskrevet i [40].

2.4.3 Spesifikasjonen og egne navnerom

Spesifikasjonen [37] sier at programmer som følger dokumentformatet *kan* inneholde elementer som ikke er definert av OpenDocument. Disse ekstra elementene må da ikke komme i navneromskonflikter med navnerom i spesifikasjonen. Programmene er også nødt til å støtte hele spesifikasjonen, både ved lesing og skriving, når alle fremmedelementer er tatt bort. Det er altså lov til å definere egne navnerom, bare de ikke kommer i konflikt med andre, og dokumentet validerer i henhold til spesifikasjonen når de ekstra navnerommene er fjernet.

Det kan være nyttig å se på hva OpenOffice som referanseimplementasjon for standarden gjør med ekstra navnerom i dokumentformatene. OpenOffice har en del navnerom som ligger utenfor spesifikasjonen, men som brukes til interne ting. Vi ser i den følgende lista tre navnerom som begynner med «ooo». Disse brukes for eksempel til å kode oppsettinformasjon for programmene, for eksempel hvilket visningsmodus som brukes eller hvilke ekstra vinduer brukeren har oppe for dokumentet. I tillegg ser vi noen navnerom som brukes til å behandle selve dokumentformatet, men som ikke trenger å være en del av det. Eksempelvis XMLSchema for å validere av dokumenter i henhold til spesifikasjonen.

ooo (http://openoffice.org/2004/office) Generelt openoffice-navnerom.

ooow (http://openoffice.org/2004/writer) Ekstra navnerom for Writer, skriveprogrammet.

oooc (http://openoffice.org/2004/calc) Ekstra navnerom for Calc, regnearkprogrammet.

dom (http://www.w3.org/2001/xml-events) Navnerom som beskriver XML-events i DOM.

xsd (http://www.w3.org/2001/XMLSchema) Navnerom for validering mot XMLSchema.

xsi (http://www.w3.org/2001/XMLSchema-instance) Navnerom for validering mot XMLSchema.

2.4.4 Filstruktur i OpenDocument

OpenDocument består av mange filer som er pakket sammen. Pakkealgoritmen er standard Zip. Her følger en liste over de obligatoriske filene som opprettes internt i et dokument:

mimetype Inneholder én linje, med dokumentets mime-type. For ren tekst vil dette være `application/vnd.oasis.opendocument.text`.

content.xml Inneholder selve dokumentet.

styles.xml Inneholder stilene som er i bruk i dokumentet.

meta.xml Inneholder metadata om dokumentet, som tittel, dato, språk og forfatter, for å nevne noen få eksempler.

settings.xml Inneholder en del informasjon om oppsett som er spesifikt for tekstbehandleren som laget det. Mesteparten er innstillinger for hvordan dokumentet skal vises i progremmet, og hvordan det bør skrives ut.

Thumbnails/thumbnail.png Et bilde av dokumentets første side, kodet i PNG-format. Dette bildet skal kunne brukes i stedet for å vise et generelt dokumentikon i filbehandlere.

META-INF/manifest.xml Inneholder informasjon om hvilke filer som er med i dokumentet, altså i den nedpakkede fila vi undersøker.

I tillegg legges råversjonen av bilder som inkluderes i dokumentet i en mappe med navnet `Pictures`. Det finnes også en mappe for ekstra innstillinger, `Configurations2`, som jeg etter utpakking av utallige dokumenter av meget forskjellig art ennå ikke har sett i bruk.

Med å legge stilinformasjonen i en egen fil, adskilt fra innholdet, følges det viktige prinsippet om å skille innhold og presentasjon. Men ikke all presentasjon er lagret helt utenom innholdet. Formateringsregler som ikke blir bestemt av de definerte stiler, og dokumentspesifikke innstillinger blir kodet i samme fil som innholdet, men da ved at presentasjonselementene blir lagret i toppen av `content.xml` og kun refereres til fra den delen av fila som koder innholdet. Dette kalles automatiske stiler. Figur 2.4 på neste side viser hvordan automatiske stiler kodes i toppen av fila, og figur 2.5 på neste side viser hvordan innholdet tar i bruk presentasjonskoden.

Et dokument uten innhold er ikke helt tomt. Dokumenter vil alltid inneholde noe markup. Alle de obligatoriske filene vil ha innhold selv om ingen tekst er skrevet.

```
<office:automatic-styles>
   <style:style style:name="T1" style:family="text">
      <style:text-properties fo:font-weight="bold" style:font-↩
         weight-asian="bold" style:font-weight-complex="bold"/>
   </style:style>
</office:automatic-styles>
```

Figur 2.4: Automatiske stiler blir lagret sammen med andre deklarasjoner i toppen av fila.

```
<text:p text:style-name="Standard">
   Tekst med
   <text:span text:style-name="T1">uthevet</text:span>
   skrift.
</text:p>
```

Figur 2.5: Automatiske stiler opprettes av tekstelementer når de trengs, og gjenbrukes når egenskapene til tekstelementer er de samme.

`content.xml` vil inneholde rammeverket for dokumentet, som deklarasjoner av navnerom, hvilken skrifttype som brukes og et tomt avsnitt merket med standard-stilen. `styles.xml` vil inneholde de stilene som er standard for alle dokumenter, eksempelvis generelle avsnitts-, grafikk- og tabellstiler, i tillegg til sideoppsettet med akrstørrelse og marger.

Kapittel 3

Bruk av dokumentformater

Dette kapittelet vil fokusere på dokumentformatene sett fra brukernes ståsted, gjennom tekstbehandlerne. Det starter med en litteraturstudie for å se hvor langt vi egentlig har kommet og hvor vi er på vei. Noe vi må vite før vi prøver å reformere formatene. Jef Raskin, grunnlegger av Apple Macintosh, blir ofte sitert i dette kapittelet, mye fordi han mener at datamaskinenes oppførsel overfor brukere må spesifiseres tidlig i utviklingsprosjekter og er en viktig del av en hver kravspesifikasjon for systemer som inkluderer brukere.

Kapittelet fokuserer også på at det er viktig at utviklingen av dokumentformater følger utviklingen av skrivebordmiljøene ellers, og bidrar til å trekke utviklingen i riktig retning.

Mot slutten av kapittelet ser vi på hvordan forskjellige typer verktøy brukes til forskjellige oppgaver, som gjerne tilhører skriveprosessen for ett og samme dokument. Hvordan et standard dokumentformat kan brukes til så mange deler av prosessen som mulig, og hva dette har å si for dokumentutveksling. Det undersøkes også hvordan et standard og stort dokumentformat kan brukes av mindre programmer med begrenset funksjonalitet, og ulike måter programmer kan velge å implementere forskjellige grader av funksjonaliteten spesifisert i spesifikasjonen.

3.1 Vi er preget av datamaskinene

Datamaskinenes utvikling har preget vår hverdag og gjort livet lettere. Men i følge brukbarhetseksperter har ikke utviklingen av brukbarheten vært like god som på andre felter, og så lenge dokumentsystemene har vært brukbare nok, så har ingen hatt motivasjon for å investere i nye brukergrensesnitt.

Vi liker å tro at det er vi som bestemmer hvordan datamaskinene skal oppføre seg, og ikke motsatt. Men når utviklingen innen dokumentorganisering og brukbarhet henger etter med 20 år [41] og [42], så er det faktisk sånn at vi i større grad enn nødvendig lærer oss vanskelige måter å gjøre enkle ting på, i stedet for å få maskinene til å gjøre tingene enklere for oss.

Christian Lagerkvist sier i [41] at å rulle vinduet ned mens man leser et dokument består av en rekke operasjoner. Først tar man blikket vekk fra teksten og flytter det til rullefeltet, flytter musepekeren bort til dette rullefeltet, mens man følger med på den. Deretter klikker man og drar i musepekeren mens man finner den linja man holdt på å lese, og drar musa sånn at dokumentet ruller helt til linja nå er øverst i vinduet. Så kan man flytte øynene bort til teksten igjen og fortsette lesinga.

Han sammenlikner det å alltid måtte se på musepekeren som å alltid måtte se på hånda når man bruker dem i dagliglivet, og har laget en liten surrealistisk historie om dette:

> When Gregor Samsa woke up one morning from unsettling dreams, he found himself changed in his bed into The Man with the World's Worst Simultaneous Capacity! His case was so bad that if he wanted to move his body in some way, he constantly had to focus his vision exactly on the body part performing the task, and keep his eyes locked onto it until the task was finished. After a three-hours shower (he didn't want to skip soaping his back), Gregor decided to skip breakfast and got in his car to drive to work. He had to cut through his trousers with a pair of scissors in order to get the keys out of the pocket, since he couldn't bring himself to controlling his hand as soon as his index finger disappeared into the pocket opening. Of course, Gregor past away some time during the first bend since his eyes were constantly locked on his hand trying to turn the steering wheel.

Lagerkvists historie setter fokus på at vi i dagliglivet gjør ting uten å følge med på alt i minste detalj. Vår simultankapasitet er langt bedre enn Gregor Samsas, og vi er i stand til å lese Donald, tygge tyggis og kjøre heis samtidig. Vi trenger nye og enklere måter å bruke datamaskinene på, som utnytter vår simultankapasitet, og som ikke er i veien for den jobben vi egentlig prøver å gjøre.

Jef Raskin, grunnleggeren av Apple Macintosh, er også en som mener at utviklingen av skrivebordsmiljøene henger etter, og at en naturlig forbedring vil være å gjøre skrivebordsmiljøene mer dokumentsentrert og mindre applikasjonssentrert [42].

3.1.1 Dokumentsentrert tankegang

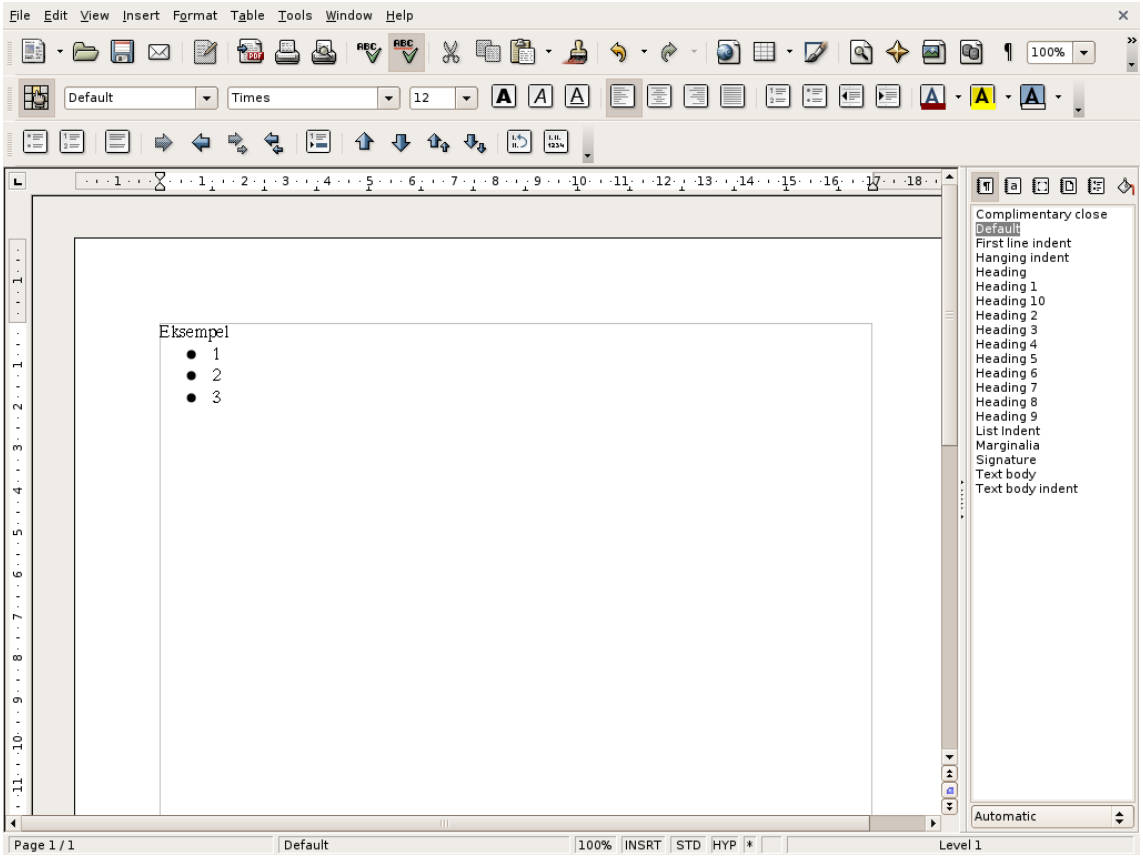
Dokumentkonseptet endrer seg sammen med endringen av det omsluttende skrivebordskonseptet. I dagens applikasjonssentrerte skrivebordsmiljøer er det slik at et program har ett eller flere dokumentformater og en mengde filer i det eller de formatene. Skal man gjøre noe fornuftig så må man på et tidspunkt åpne programmet, vente på at det starter, og når det er der, så gjør det ikke noen forsøk på å opptre diskret. Vi ser i figur 3.1 at man fort ender opp med tre knapperader [1] og et sidefelt, og da tar programmet så stor plass at det så vidt er plass til å se hele bredden av dokumentet i fullskjermvisning.

I dokumentsentrerte skrivebordsmiljøer, slik Lagerkvist beskriver det, vil alle dokumentene ligge på skrivebordet (bakgrunnen). Dokumentene vil være gruppert, ikke i mapper som må åpnes, men ved grupperinger der relaterte dokumenter ligger i nærheten av hverandre. Ved første øyekast vil man ikke se dokumentene, men navn på grupperingene, som vist i figur 3.2.

For å ta dokumentgruppene nærmere i øyesyn, vil man måtte zoome seg nærmere. Da vil nye grupper av dokumenter eller enkeltdokumenter bli synlige etterhvert som vi kommer nærmere skrivebordet. Lagerkvist bygger sine idéer på tankekart, og beskriver en verden av bobler som ligger i nærheten av hverandre. Raskin sin verden er mer strukturert, der man zoomer inn på bokser og får se nye bokser. Boksene til Raskin er heller ikke begrenset av et virtuelt skrivebord, men lar brukerne zoome ut av det virtuelle rommet og se avdelinger, etasjer og bygninger. Figur 3.3 på side 26 viser avdelinger i en etasje, med rom og dokumenter for personen i hvert rom.

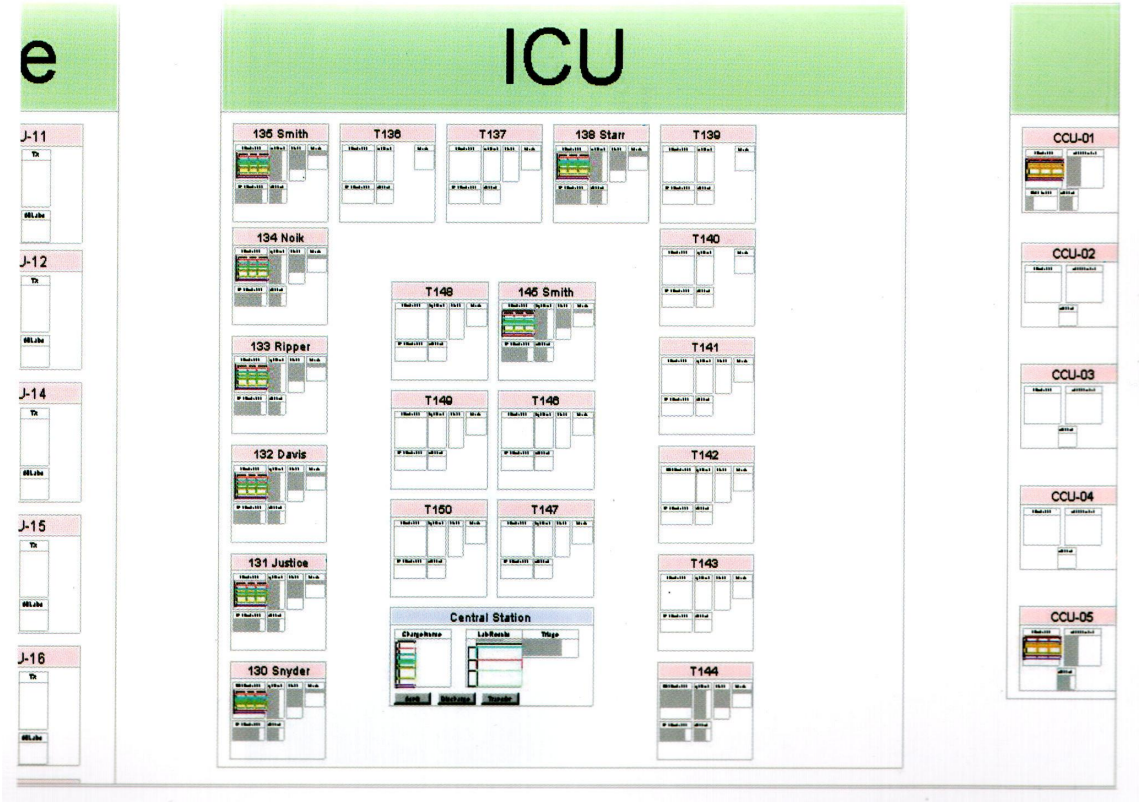
Vi åpner et dokument ved å zoome så mye inn på det at vi ser innholdet, og for å endre det ved å plassere skrivemarkøren og redigere teksten som vanlig. Programmet som

[1] Bildet viser OpenOffice 2.0 som viser en tredje menylinje for lister når brukeren har markøren i en liste. Den ekstra knapperaden forsvinner når markøren flyttes ut av lista, og dokumentet får større plass på skjermen.



Figur 3.1: Openoffice i fullskjemsvisning

Figur 3.2: Skissering av dokumentgruppering



Figur 3.3: En avdeling på et sykehus. Rom i avdelingen som har personer har fått navn og innhold i dokumentene. De andre avdelingene ligger rundt. (Bildet er hentet fra [42].)

brukeren bruker for å skrive i dokumentet vil være lite synlig, og vil kun opptre som arket brukeren skriver på.

Både Raskin og Lagerkvist beskriver en verden der man zoomer og ruller (skrolle) vinduet sideveis eller opp og ned. Dette konseptet kalles Zooming User Interface (ZUI), og er godt beskrevet i [42].

I slike arbeidsmiljøer vil dokumentene være i fokus og navigering blant dokumentene og arbeid på dem vil være nærmere den gamle måten med ark og penn.

3.1.2 Små datamaskiner

Datamaskinene blir stadig raskere og tar mindre plass. En del datamaskiner er nå så små at det er vanskelig å vise store dokumenter på dem. Det er ikke bare datastørrelsen i byte som er et problem for små datamaskiner, men også at de er så små at skjermene ikke på noen fornuftig måte kan vise et A4-ark uten alt for mye dokumentrulling. Linjelengden er i mange tilfeller for lang til at brukeren får noen særlig god oversikt på PDA-er, små datamaskiner og avanserte mobiltelefoner.

De små datamaskinene har av samme grunn også hatt problemer med å vise nettsider. Den norske nettleseren Opera har funnet en løsning på dette hvor de kutter ned på linjelengder, og forminsker bilder så sideinnholdet passer skjermbredden. Denne løsningen gjør at brukeren slipper å rulle nettsider sideveis, men den vertikale rullingen blir betraktelig forlenget.

Å rulle dokumenter i bare én retning gjør at brukeren ser alt innholdet under rulling, og ikke mister oversikten over dokumentet. Noe som ville vært tilfellet dersom brukren kun kan se en stripe av dokumentet.

Metodene som brukes for å vise nettsider kan fint bruke for å vise andre dokumenter, så vel som ved skriving til dokumentene. En mobiltelefon eller PDA må bare bryte linjene, krympe bildene, og kanskje bruke en annen lettere lesbar skrifttype.

3.2 Skriveprosess

Skriveprosess varierer fra person til person og fra dokumenttype til dokumenttype. Erfaring og rutine gjør at små dokumenter sjelden krever strukturert disponering. De færreste klarer å starte med overskriften og skrive til man når enden, og så være ferdig, og etterhvert som dokumentstørrelsen, og kompleksiteten stiger, er det færre som holder seg i denne kategorien. De fleste er nødt for å strukturere arbeidet på en eller annen måte, slik at de viktige idéene er med, og de irrelevante idéene utelates.

I dag lærer elever på videregående skoler at man ikke skal gå rett på skrivinga, men å starte med en disposisjon. På noen skoler lærer de også å bruke tankekart for å komme fram til en god disposisjon. En disposisjon som seinere skal fylles ut etterhvert som idéene kommer. Når nok idéer er satt opp og strukturert, fyller man inn tekst i rammeverket. Underveis, men mest til slutt, går man over og redigerer. Etter hvert som teksten blir redigert, blir det ofte også nødvendig å endre på hovedstrukturen, som igjen fører med seg et behov for å redigere mer. Mot slutten ser man etter at overskrifter passer med innholdet, og ferdigskriver innledning og konklusjon.

Vi går nå gjennom verktøy og formater for de forskjellige fasene av skriveprosessen. Figur 2.1 på side 10 viser oversikten over hvor i prosessen de forskjellige formatene hører hjemme. Formatene for de kreative fasene er interne for programmene sine og er derfor ikke tatt med i figuren. De ville blitt plassert helt til venstre i tekstkategorien siden de i

høyeste grad er komposisjonsformater. Av samme grunn fokuserer vi derfor på verktøyene når vi beskriver fasene, mens vi for ferdige dokumenter fokuserer mer på formatene.

3.2.1 Verktøy og formater for de kreative fasene

Det sier seg selv at et stort tekstbehandlingsprogram ikke er nødvendig for å skrive disposisjon og til å brainstorme. Til en viss grad vil et tekstbehandlingsprogram også være kreativitetshemmende, i forhold til andre typer programmer som er laget spesielt for å fremme kreativitet. Tekstbehandlere vil lett spore av brukeren til å tenke på formatering og presentasjon, som hvilken skrifttype som brukes, hvor mange sider det er i dokumentet og hvordan forsiden skal se ut, mens de viktige tankene er hva teksten skal omhandle, hvor dypt hver del skal gå og hvordan delene skal settes i sammenheng.

Det er ikke programmene som er interessante, men metodene som brukes. Man kan komme veldig langt med å skrive disposisjonen i rein tekst, og dette krever ikke noen avansert tekstbehandler. For å lenke sammen idéer kan litt mer avanserte programmer brukes i stedet, som for eksempel wikiene som vi skal se på etter hvert. Men spennet av programmer omfatter også mer avanserte metoder, og programmer for å sette sammen et tankekart er mer komplisert enn et enkelt skriveprogram.

Viktig for de kreative fasene er det at brukeren ikke blir opphengt i formatering og presentasjon. Tekst må vises pent, og det må være mulig å lime inn eller lenke opp bilder og andre elementer enkelt. Et kreativitetsfremmende program må aldri være til hinder slik at brukeren glemmer idéer.

Vi vil nå se på programmer som representerer forskjellige kategorier av programmer som kan brukes til å strukturere tanker.

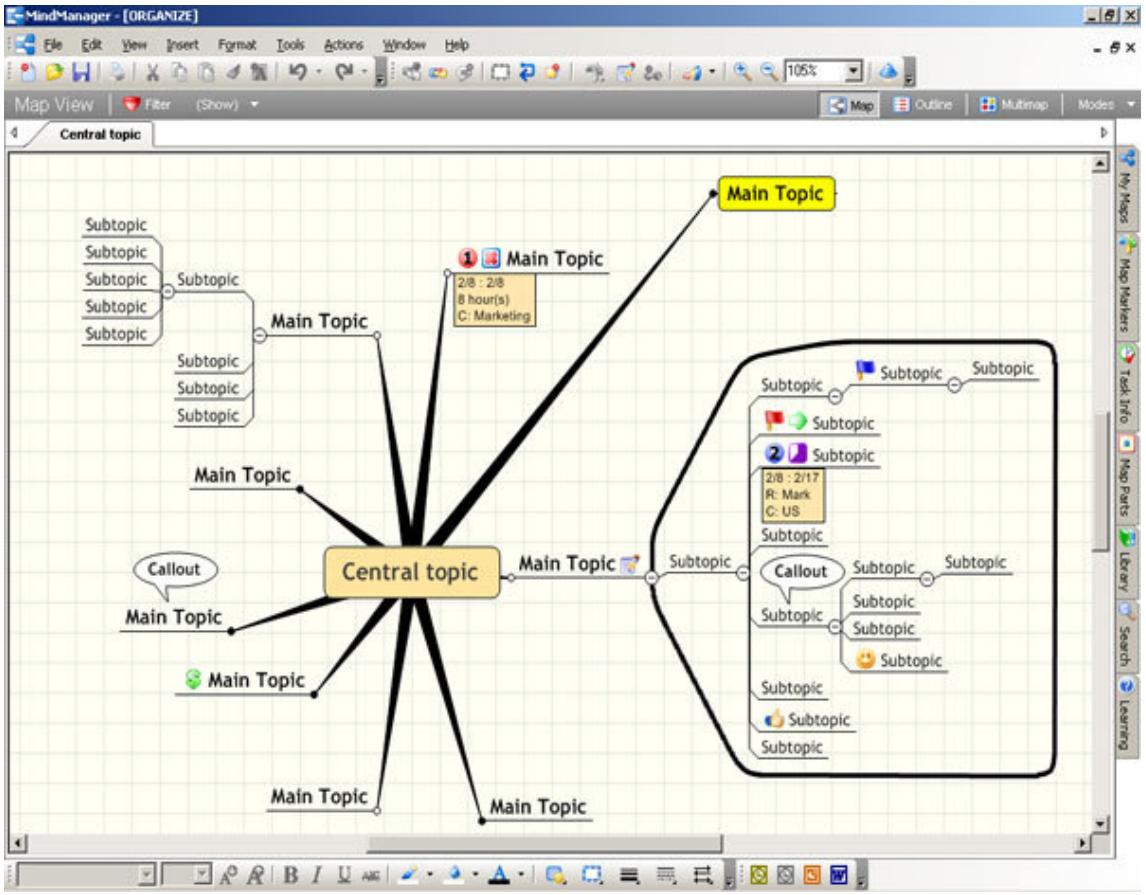
Tankekart

Tankekart (eller hjernekart) er en teknikk for å tilpasse idéer til hjernens natur, og er derfor ikke basert på linearitet [44]. Det bygges opp ved å starte med å sette sentralt hovedtema midt i tankekartet. Dette brytes ned med piler til nye emner, som igjen peker videre ned i enda mindre biter. Dette skjer i mange nivåer. Der en liten figur er enklere enn ord, brukes dette. Denne teknikken brukes av mange for å få oversikt, og den egner seg derfor godt i forbindelse med tekstskriving.

De mest tradisjonelle tankekartverktøyene kan ikke noe mer enn det man kan få til med papir og blyant, de gjør det bare litt enklere, for eksempel ved sletting eller flytting av elementer i kartet. Mindmap Manager [43] og en del andre gjør det i tillegg enkelt å lenke til eksterne dokumenter, som nettsider, bilder, regneark, for å nevne noen typer. Man får også mulighet til å merke elementer i kartet med symboler. En viktig utvidelse i forhold til papirvarianten er at det er mulig å legge til tekst, noe som kan forenkle overgangen mellom tankekart og dokument.

Et ferdig tankekart kan eksporteres til dokumentformater og brukes i programmer som Microsoft Word, Microsoft Powerpoint, Microsoft Project [2] eller LaTeX [3], på en slik måte at elementene i tankekartet blir omgjort til overskrifter så de kan brukes som et rammeverk for et dokument. Hvis det er tekst tilordnet elementene, så blir teksten lagt til som brødtekst under den aktuelle overskriften.

[2] Mindjet Mindmap manager har disse eksporteringsformatene.
[3] Kdissert eksporterer også til LaTeX [45].



Figur 3.4: Eksempel på bruk av tankekart i Mindmap Manager. (Bildet er hentet fra [43].)

Wiki

En wiki er en type redigeringsverktøy for nettsider som lar brukerne redigere innholdet på sidene med et mer lettfattelig språk enn HTML. En annen egenskap ved wikier er at det også skal være enkelt å lenke mellom dokumenter. Figur 3.5 viser eksempel på hvordan overskrifter, lister og lenker skrives i wikispråket til Moinmoin [46]. Det er også viktig for wikiene at de tar vare på gamle versjoner av sidene, fordi det er vanskelig å holde bråkmakere ute. Nettleksikonet Wikipedia er et godt eksempel på hva et wikiverktøy kan brukes til.

```
= Headers =
== Header 2 ==
=== Header 3 ===

 * item 1
 * item 2
 * item 3
   * item 3.1

MoinMoin/InstallDocs
../InstallDocs
/SubPage
```

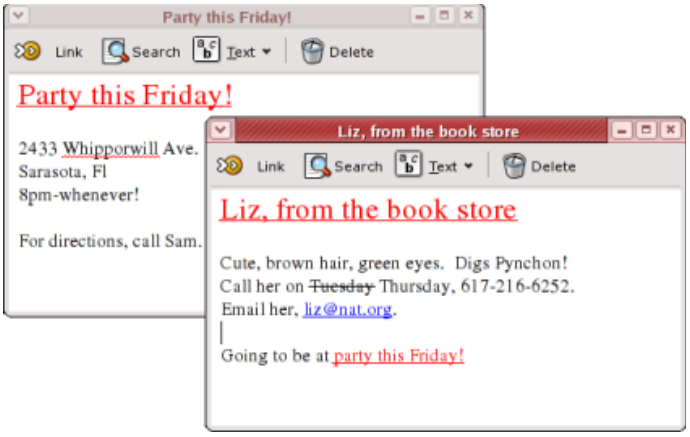
Figur 3.5: Wikitekst som definerer tre nivåer med overskrifter, en punktliste med nivåer spesifisert med innrykk, og tre måter å lenke til andre dokumenter i wikien.

Det har også dukket opp personlige wikier som ikke krever en nettjener men som installeres som et annet program, eller som en del av et annet program. Emacs-wiki [47] bruker skriveprogrammet Emacs til å lage personlige nettverk av små dokumenter.

En wiki gir ikke den samme oversikten som et tankekart, siden den grafiske framstillingen av dokumentet mangler. Men for de brukerne som liker å starte med å skrive tekst i ustrukturerte deler, for så å sette den sammen seinere, er wikiene en enkel måte å få tankene ned som tekst. Wikiene gir også den fordelen at gamle wikisider kan gjenbrukes, siden det ikke nødvendigvis trenger å være et skille mellom de dokumentene som skal og de som ikke skal med i det ferdige dokumentet.

Der tankekartene starter på toppen med dokumentoversikten og lar brukerne spesifisere detaljert nivå etterhvert, starter wikiene på bunnen og lar brukerne skrive dokumenter som seinere kan settes sammen til en større dokumentstruktur. En wiki kan også startes fra toppnivå, men da ved å opprette et dokument som er disposisjonen, som bare består av lenker til andre dokumenter. På grunn av en wikis natur, trenger ikke disse dokumentene å eksistere når de lenkes til.

Det er vanskeligere for brukerne å lage et dokument ut av et slikt dokumentnettverk. Konvertering gjøres ikke med et enkelt klikk, siden det kan være store deler av wikien som ikke skal være med i dokumentet. Brukeren blir derfor presentert med en del valg for hvilke wiki-dokumenter som det er lenket til fra disposisjonsdokumentet som skal være med, og hvilke av lenkene i disse dokumentene som skal være med. Hva som faktisk finnes av konverteringsverktøy, og hvor gode de er varierer sterkt, men det gjøres mye utvikling på dette området.



Figur 3.6: To notater lenket sammen i Tomboy

Notatprogram

Tomboy [48] er et notatskrivingsprogram, og det kan sees på som en grafisk wiki. I stedet for å skrive i et wiki-språk á la det vi så i figur 3.5 på forrige side, skriver man rik tekst som er «what you see is what you get». Siden dette er et skrivebordsprogram og ikke et tekstverktøy, har det også grafiske muligheter som gjør at det er nærmere et tankekartverktøy enn hva en tekstbasert wiki er. Figur 3.6 viser hvordan man kan lenke sammen dokumenter, og derved bygge opp et nettverk av idéer.

Tomboy er i tidlig utvikling, og støtter nå kun eksportering til HTML for øyeblikket. De samme problemene for eksport som ble ramset opp for wikier gjelder også for Tomboy.

Som nevnt i det første kapittelet har jeg studert dette programmet mye, og har også til en viss grad fått til en eksportering til OpenDocument fra dette formatet. Detaljer om dette arbeidet finnes i C på side 67.

3.2.2 Verktøy og formater for tekstbehandling

En tekstbehandler brukes naturlig til å skrive store dokumenter, enten det er helt fra bunnen av eller basert på en disposisjon eller et utkast. Da kommer tekstbehandlernes store mengde funksjonalitet til hjelp. Tradisjonelt er det tre store aktører som lar forfatterne gjøre dette på hver sin måte.

Grafisk tradisjonell tekstbehandler

Microsoft Word [49] har lenge vært det alternativet som folk flest har brukt til å lage dokumenter med. Word er er et typisk «what you see is what you get»-program, og brukerinteraksjonen er stort sett lik de andre programmene brukerne som regel har.

Word og andre tradisjonelle office-programmer har innført stiler (beskrevet i 2.1.3 på side 11) som letter formatering og gjør presentasjonen mer helhetlig. Stilene bidrar med semantisk struktur i XML-formatene, og gjør dermed uthenting av informasjon enklere fra dokumentene, samtidig som formatering av dokumenter som flere har bidratt på gjøres enklere.

Men disse office-programmene lar brukerne få gjøre akkurat hva de vil, når de vil, og de blir på ingen måte tvunget til å bruke stilene. Brukerne ser bare hvordan stilene forenkler eller gjør formatering av dokumentene vanskeligere, og får aldri med seg fordelen stilene gir i for dokumentutveksling eller automatisk uthenting av data.

Dagens dokumentsystemer viser ikke på noen måte for brukeren at dokumentkoden som blir lagret er dårligere uten bruk av stilene, for brukeren ser det helt likt ut med og uten stiler. Dokumentene blir dermed veldig ofte kodet helt uten eller delvis uten stiler, og det gjør at dokumentene mangler eller har veldig mangelfull semantikk, som igjen hindrer gjenbruk av innhold i dokumenter.

```
<w:body>
   <wx:sect>
      <w:p>
         <w:r>
            <w:t>Tekst med </w:t>
         </w:r>
         <w:r>
            <w:rPr>
               <w:b/>
            </w:rPr>
            <w:t>uthevet</w:t>
         </w:r>
         <w:r>
            <w:t> skrift.</w:t>
         </w:r>
      </w:p>
      <w:sectPr>
         <w:pgSz w:w="12240" w:h="15840"/>
         <w:pgMar w:top="1440" w:right="1800" w:bottom="1440" ↩
             w:left="1800" w:header="720" w:footer="720" w:gutter=↩
             "0"/>
         <w:cols w:space="720"/>
         <w:docGrid w:line-pitch="360"/>
      </w:sectPr>
   </wx:sect>
</w:body>
```

Figur 3.7: Dokumentstruktur i et enkelt Word-dokument. Presentasjon og innhold blandes, for eksempel ved at et avsnitt inneholder både presentasjonskode og innhold.

En viktig sak under utviklingen av XML var at det skulle være mulig å skille presentasjon og innhold, slik at innholdet er kodet for seg, og presentasjonskode for seg. Dokumenter som blander sammen presentasjon og innhold er det vanskeligere å hente ut informasjon fra. Figur 3.7 viser hvordan presentasjon og innhold blandes sammen i avsnittskoden i et Word 2003-dokument.

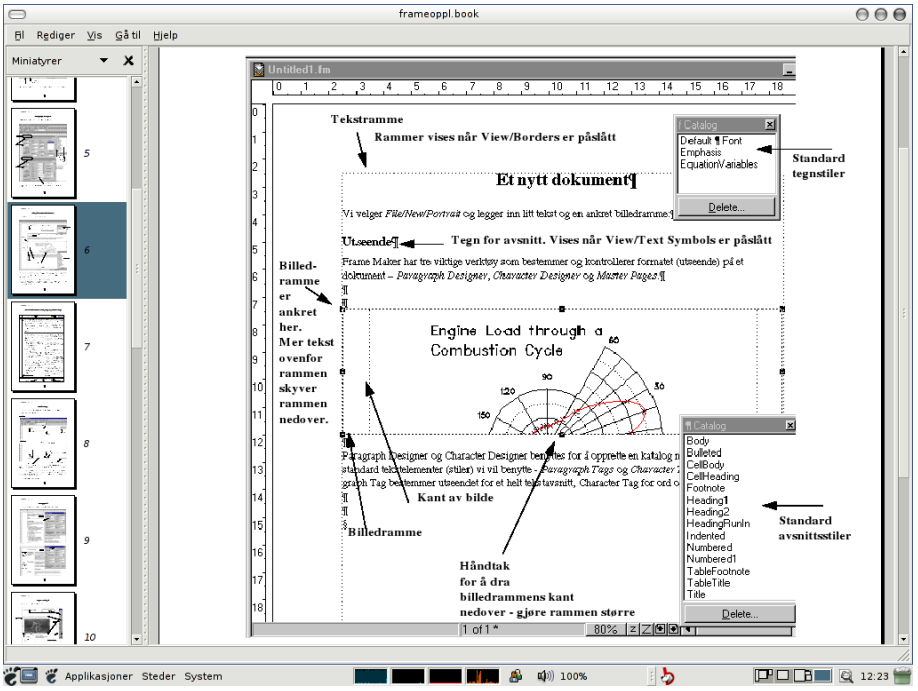
Layoutsentrert tekstbehandler

Adobe Framemaker er enkel i bruk på sin måte, men er litt vanskelig å sette seg inn i, fordi det bruker en del andre konsepter enn tradisjonell «what you see is what you get». Framemaker kan være veldig layoutorientert og kan derfor gi proffere resultater enn en tradisjonell tekstbehandler [50].

Framemaker har to brukermodi. Ett tradisjonelt og ett strukturorientert. Det er det strukturorienterte som gir mulighet til å lagre XML-formater som igjen kan konverteres til det meste, for eksempel DocBook [51]. Dette strukturerte moduset har ikke like avansert funksjonalitet når det kommer til grafisk presentasjon. Det tradisjonelle moduset, som

er layoutsentrert gjør til gjengjeld automatisk uthenting av innhold vanskelig siden dette lagrer filer i binærform. Framemaker setter brukerne her i et unødvendig valg.

Figur 3.8 viser typisk behandling av Framemaker-dokumenter.



Figur 3.8: Eksempel på bruk av Framemaker, med stiler og rammer for å definere layout og annen formatering. (Bildet er hentet fra [50].)

Tekstredigering i ren tekst

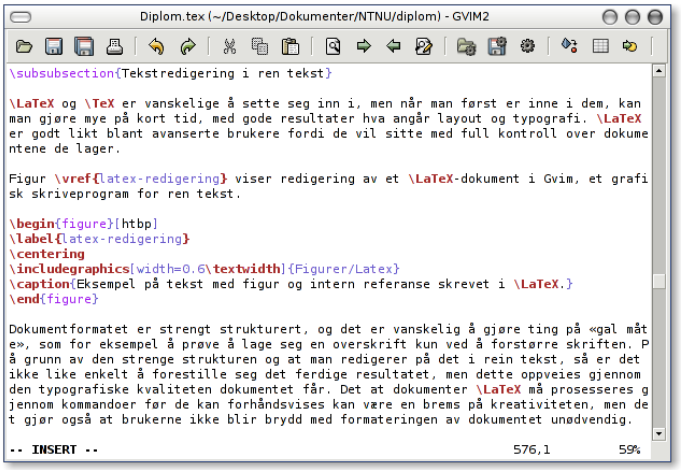
LaTeXog TeXer vanskelige å sette seg inn i, men når man først er inne i dem, kan man gjøre mye på kort tid, med gode resultater hva angår layout og typografi. LaTeXer godt likt blant avanserte brukere fordi de vil sitte med full kontroll over dokumentene de lager.

Figur 3.9 på neste side viser redigering av et LaTeX-dokument i Gvim, et grafisk skriveprogram for ren tekst.

Dokumentformatet er strengt strukturert, og det er vanskelig å gjøre ting på «gal måte», som for eksempel å prøve å lage seg en overskrift kun ved å forstørre skriften. På grunn av den strenge strukturen og at man redigerer på det i rein tekst, så er det ikke like enkelt å forestille seg det ferdige resultatet, men dette oppveies gjennom den typografiske kvaliteten dokumentet får. Det at dokumenter LaTeXmå prosesseres gjennom kommandoer før de kan forhåndsvises kan være en brems på kreativiteten, men det gjør også at brukerne ikke blir brydd med formateringen av dokumentet unødvendig.

3.2.3 Verktøy og formater for presentasjon

Verktøyene for presentasjonsformater er stort sett de samme som for komposisjonsformater, enten det er det samme formatet som brukes hele veien eller verktøyene eksporterer mellom formatene. I office-programmer er det lett å bruke det samme formatet gjennom hele skriveprosessen, selv om dette ikke alltid er det beste.



Figur 3.9: Eksempel på tekst med figur og intern referanse skrevet i LaTeX.

Format for nettsider

Nettsider lagres som regel i en eller annen variant av HTML [52], enten det er med eller uten skripting med for eksempel JavaScript i tillegg. Formater for nettsider er typiske presentasjonsformat i dag, og nettsider kan lett genereres fra de fleste tekstbehandlere, eller med XML-transformasjonsverktøy også fra XML-formater. Vi har i 2.2.2 på side 13 sett at HTML har blitt redefinert i XML, og HTML i XML har blitt kalt XHTML. Veldig få verktøy som lager HTML-dokumenter fra bunnen av klarer å følge standardene til World Wide Web Consortium, og dette gjør at HTML er et de-facto presentasjonsformat, selv om HTML med CSS nesten er fullgodt som dokumentformat [53].

Format for skrivere

PostScript er et format for skrivere som ble laget av Adobe i 1985. Dette er et format som skulle være uavhengig av skrivertype gjennom at skriverne skulle kunne motta dette formatet.

Format for det meste

Portable Document Format er en mulig arvtager til PostScript som er laget for å kunne brukes til det meste, selv om fokuset er mer presentasjonsorientert enn for office-programmene. PDF er laget for å kunne utveksles mellom ulike operativsystemer, til skrivere, for visning på prosjektør, med mer. I tillegg til å inneholde essensen av PostScript, inneholder PDF også en komponent for å lagre skrifter og en komponent for å putte dokumentets deler sammen i ett dokument [54], med mulighet for komprimering. PDF blir brukt til stadig mer, og mulighetene for å redigere på PDF-dokumenter blir også større, selv om dette ikke gjør det til et komposisjonsformat.

3.2.4 Bindeledd mellom fasene

Det finnes altså flere typer av programmer som er nyttige i de kreative fasene i starten av komposisjonsprosessen, og på samme måte forskjellige typer programmer som har hver sin måte å være god på under redigering. I tillegg finnes det en del programmer som kan vise resultatet av det ferdige dokumentet, men de ser vi ikke på her, vi nøyer oss

med å se på overgangen fra et kreativitetsfremmende program til en av de tradisjonelle tekstbehandlingsmåtene.

Mindmap Manager kan eksportere tankekartet til MSWord. Det blir da opprettet et Word-dokument hvor elementene fra tankekartet blir omgjort til overskrifter og mellomliggende tekst i dokumentet. Dette er en enveisfunksjon, så Mindmap Manager kan ikke ta inn et dokument med overskrifter og endre et tidligere laget tankekart.

Tomboy har støtte for å eksportere en sammenlenket gruppe notater til et HTML-dokument, og i C på side 67 har jeg implementert en løsning for å konvertere også til OpenDocument. Disse enkle formene for eksport eksporterer enten ett notat eller også alle notater som er lenket sammen med det notatet man eksporterer. På denne måten kan man lage seg en disposisjon med lenker til de notatene man vil ha med i det eksporterte dokumentet.

Wikiene har tilsvarende verktøy for eksportering som det Tomboy har, men med fokus på PDF. Slik eksporteringsfunksjonalitet varierer sterkt mellom de ulike og tallrike wikiene.

3.2.5 Kritikk av eksisterende løsninger

Både Word, Framemaker og LaTeXhar sine begrensninger, og et problem for alle er at de har presentasjonsfokus. Hvis de lagres i et presentasjonsformat som PDF, vil det være vanskelig å hente ut informasjon fra dem, og komposisjonsformatet må legges ved for å ha en mulighet for innholdsekstrahering.

LaTeX-filer kan utveksles enkelt, siden de består av rein tekst, men det er ikke alltid mottakeren har de samme modulene installert som avsenderen. En viss form for enighet er derfor nødvendig. Enighet er også nødvendig når flere personer redigerer filer i Words og Framemakers formater, men da om hvilken tekstbehandler og hvilken versjon av den som skal brukes. En vanlig, og i mange tilfeller kostbar praksis, er å bruke siste versjon av formatets offisielle tekstbehandler.

3.2.6 Et felles format med all nødvendig funksjonalitet

OpenDocument er en dokumentstandard som har prøvd å ta høyde for de fleste bruksområder utviklerne av office-verktøy har sett for seg at tekstdokumenter skal løse. OpenDocument har støtte for figurtegning, automatiske felter, bibliografidatabaser og avanserte formateringsmuligheter som likner det vi ser i Framemaker. I tillegg skal det være et standardisert format som flest mulig tekstbehandlere skal kunne bruke.

Med et standard og åpent format som alle kan bruke, vil alle programmer ha mulighet for å gjøre operasjoner på formatet. Dette vil formatet vil bli et eksportformat for kreative verktøy, standardformatet for tekstbehandlere, og importformat for layoutspesialiserte presentasjonsverktøy.

En slik omfattende bruk vil være avhengig av at dokumentformatet favner bruksområdene til de forskjellige programmene.

Vi ser allerede at Opendocument ikke er komplett på alle områder. Morten Welinder sier i [55] at formatet ignorerer regnearksemantikk, ved at oppbygningen av formler for bruk på celler i regneark ikke er definert. Det gir altså ingen mening å standardisere regnearkprogrammer som Gnumeric på dette formatet, siden standarden ikke er fullstendig.

3.3 Felles format og begrensninger

Det at OpenOffice sitt format ble brukt som utgangspunkt for OpenDocument, har gjort at OpenOffice.org ikke har hatt store problemer med å implementere det nye formatet full ut. OpenDocument er en stor spesifikasjon som også har blitt utvidet i forhold til det formatet OpenOffice bruker i versjon 1. tenkelige krav på dokumentfronten kan også ha negative sider for de som prøver å implementere formatet.

Dokumentformatet OpenDocument prøver som tidligere nevnt å ta høyde for alle tenkelige krav for dokumentformater. Dette har gjort OpenDocument til et stort komplekst format, som ikke bare er meningen å støtte all funksjonalitet tilgjengelig i Openoffice, men hvor intensjonen er å støtte all funksjonalitet man kan komme på å få bruk for i alle kontorstøtteprogrammer.

3.3.1 Krav til støtte for dokumentformatet

Et problem kan oppstår nå programmer med mindre funksjonalitet skal ta i bruk de samme dokumentformatene som de store bruker. Vi skal nå se på hvordan deler av et stort dokumentformat kan utelates i et begrenset skriveprogram med mindre funksjonalitet.

Data må ikke gå tapt. Dokumenter blir alltid innlest og tolket for å kunne vises. Den nye representasjonen av dokumentet som er utgangspunktet for visningen, må ikke bli brukt for lagring, for da vil ikke de elementene det begrensede verktøyet ikke klarte å tolke være med i neste lagring. Elementer et fattig verktøy ikke skjønner, må altså fremdeles være med i dokumentet.

Innholdet bør være synlig. Selv om innholdet i elementer som ikke er støttet ikke ser så bra ut som det ville gjort i et rikt verktøy, så bør innholdet likevel vises. Hvis ikke vil brukeren være nødt til å legge inn dette innholdet på nytt, og det vil bli vist dobbelt når dokumentet igjen vises i det rike verktøyet.

Elementer som vises på en begrenset måte, siden programmet ikke har støtte for formateringen, bør vises på en spesiell måte, slik at brukeren skjønner at dette er noe spesielt som ikke vises korrekt, og som derfor ikke bør endres.

Endring av elementer som er synlige må likevel være lov, siden det kan hende at brukeren vil gjøre endringer på det tekstlige innholdet eller fjerne elementet helt. Slike endringer kan medføre problemer, siden endringer gjøres på dokumentets lagringsstruktur av et program som «ikke forstår helt hva det driver med».

3.3.2 Grader av støtte i programmene

Det krever mye å lage et program som skal kunne endre på innholdet av elementer på en trygg måte, og veien fra en slik trygg lagringsstøtte til fullstendig støtte for innholdet er ikke så veldig lang. Det er forskjellige grunner til at tekstbehandlere legger seg på forskjellige steder langs denne aksen.

Full funksjonalitet

Best vil det alltid være å implementere *full funksjonalitet*, eller å ha kommet like langt i implementasjonen som de andre som bruker formatet. Dette er ikke alltid gjennomførbart. Det kan ligge tekniske problemer i veien som gjør at enkelte deler av formatet er umulig å implementere med de API-er som eksisterer, eller det kan være begrensede ressurser til gjennomføringen av implementasjonen. Vi må heller ikke glemme at ikke alle

tekstbehandlere tar mål av seg til å bli et fullstendig kontorstøtteprogram. Alternativer til å støtte full funksjonalitet finnes, og blir brukt den dag i dag.

Ikke vise dokumentet

Det enkleste et program kan gjøre, er å *nekte* å åpne et dokument som inneholder elementer det ikke forstår noe av. Dette kan være en midlertidig løsning for et program som vil ta i bruk hele dokumentformatet, men hvor ikke alle deler er ferdig, det er dette KOffice gjør i betaversjonen [4] [56]. Det kan også tenkes som en løsning for programmer som finnes i gratis lightversjon og kostbar fullversjon.

Begrenset støtte

I 3.3.1 så vi på forskjellige måter dokumentelementer kan støttes i et skriveprogram med fokus på innholdets synlighet og muligheter for å endre det. Valget av en av disse retningene er vanskelig, og forskjellige elementer kan støttes på hver sin måte. Men spesifikasjonen følges ikke hvis ikke innholdet vises, eller hvis struktur fra ikke-støttede elementer blir fjernet ved lagring.

Et eksempel på begrenset støtte kan være delvis implementasjon av innholdsfortegnelser. Programmet kan la være å vise den, eller kan si at «her er det et element av typen innholdsfortegnelse som programmet ikke støtter», innholdet kan vises uten riktig formatering, eller innholdsfortegnelsen kan vises uten kunne å endres eller oppdateres.

Ikke alle vil bruke et standardisert dokumentformat

Den største aktøren innen tekstbehandling er Microsoft Word (som vi så nærmere på i 3.2.2 på side 31). Microsoft er ikke store tilhengere av å standardisere på noe annet enn WordprocessingML [24] som ikke kan bli en standard så etter hva vi så i 2.3.2 på side 16. De har investert nok i sitt eget format, og vil holde den posisjonen de har i markedet.

Abiword [57] som jeg også analyserte i høstprosjektet [53], ville heller ikke ta i bruk OpenDocument til å starte med. Grunnen var at de med sine begrensede utviklerressurser var nødt til å fokusere på annen funksjonalitet i tekstbehandleren, for å oppnå et større antall brukere. De har imidlertid snudd, og vil tilby eksportering til og importering fra OpenDocument.

Tomboy, som vi så nærmere på i 3.2.1 på side 31 tar ikke mål av seg til å bli en stor og tung tekstbehandler, og kommer heller ikke til å ta i bruk OpenDocument som internformat. Dette programmet skal være så lite og enkelt som mulig, med kort oppstartstid og liten ressursbruk, så folk får tatt notatene sine uten å hente fram noen dinosaur. Programmet er heller ikke kommet så langt i utvikling at det er hensiktsmessig å fokusere på eksportering til OpenDocument.

Vi har her sett på noen få programmer, som ikke vil ta i bruk OpenDocument som internt format, selv om dette ville forenklet både samhandling og tekstgjenfinning betraktelig. Som vi har sett har årsakene vært politiske, av mangel på utviklingsressurser, og ønsker om at programmet bør holde seg så enkelt som mulig.

[4] KOffice lanserte 21. juni i år en versjon, der støtte for OpenDocument skal være fullstendig. OpenDocument blir etter planen internt filformat i neste versjon av KOffice.

Kapittel 4

Forslag innen modularisering

Det vil være interessant å se på videre inndeling av dokumenter, og hvordan programmodulene bedre ord bør organiseres for å gjøre både skriveprosess og bruken av maskinressurser mer effektiv.

Vi vil se på tre mulige måter å dele inn dokumenter i dokumenttyper eller dokumentklasser, og videre se på inndeling av funksjonalitet i sammenheng med disse. Til slutt vil vi se på en måte å modularisere dokumentformatene med bakgrunn i dokumentklasser.

4.1 Modularisering i dag

I dag deles dokumentene overordnet inn i «tekstdokumenter», «regneark» og «presentasjoner», og man kan spørre seg om denne inndelingen er så veldig naturlig. Vi ser at de største produsentene av tekstbehandlere tenker på denne måten i dag, men kan også skimte andre mulige inndelinger, siden OpenDocument har én stor spesifikasjon for alle disse dokumenttypene.

Det er den videre inndelingen som begynner å bli interessant, og som også gir større muligheter for den inndelingen vi nettopp har nevnt.

4.1.1 Modularisering i tekstbehandlere

I dagens systemer finner du eksempelvis tabellfunksjonalitet i en tabellmodul, listefunksjonalitet i en listemodul, bildestøtte i en bildemodul og slik deles også all annen funksjonalitet inn i moduler. Dette er en veldig grei inndeling for små dokumentsystemer og systemer som ikke blir særlig komplekse. Denne inndelingen er ineffektiv for trege maskiner, og ikke særlig god for store dokumentsystemer på grunn av den dårlige ressursbruken.

4.1.2 Ressursbruk i tekstbehandlere

Ressursbruken har ikke vært noe problem så lenge funksjonaliteten holdt seg enkel, eller filformatene skulle være lesbare for andre systemer enn bare tekstbehandleren. XML-strukturer krever mer maskinkraft enn enkle spesiallagede filformater på grunn av formatenes hierarkiske struktur, og funksjonaliteten i tekstbehandlerne har ikke akkurat blitt mindre.

En vanlig måte å redusere oppstartstiden ved åpning av et dokument, er å la deler av tekstbehandleren kjøre i minnet så lenge maskinene står på. Dette gjør at oppstarten av maskinen tar litt lengre tid, men tiden det tar å åpne dokumenter minker betraktelig.

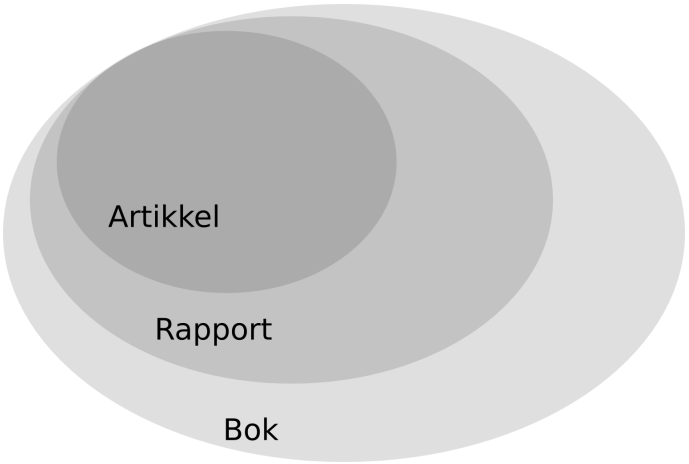
Det er også vanlig å dele så mye som mulig felles programkode mellom tekstbehandlerne og de omkringliggende vindussystemene og operativsystemene. Det gjør at programkode for eksempel til å gjøre grafisk opptegning av dokumenter allerede ligger i minnet siden vindussystemet bruker denne programkoden kontinuerlig.

Dynamisk lasting av modulene vil også utgjøre en hastighetsforbedring. I dag lastes i prinsippet all funksjonaliteten til OpenOffice inn i minnet ved oppstart, fordi programmet er kodet på en slik måte i C++ at det skal veldig mye til å gjøre dette mer effektivt.

4.2 Maler og klasser

I dag har vi maler i dokumentformatene. Et hjelpemiddel som hovedsakelig bidrar til å kontrollere layouten i dokumentet. Malene gjør det enkelt å definere et utseende én gang, og bruke utseende til å skrive mange liknende dokumenter.

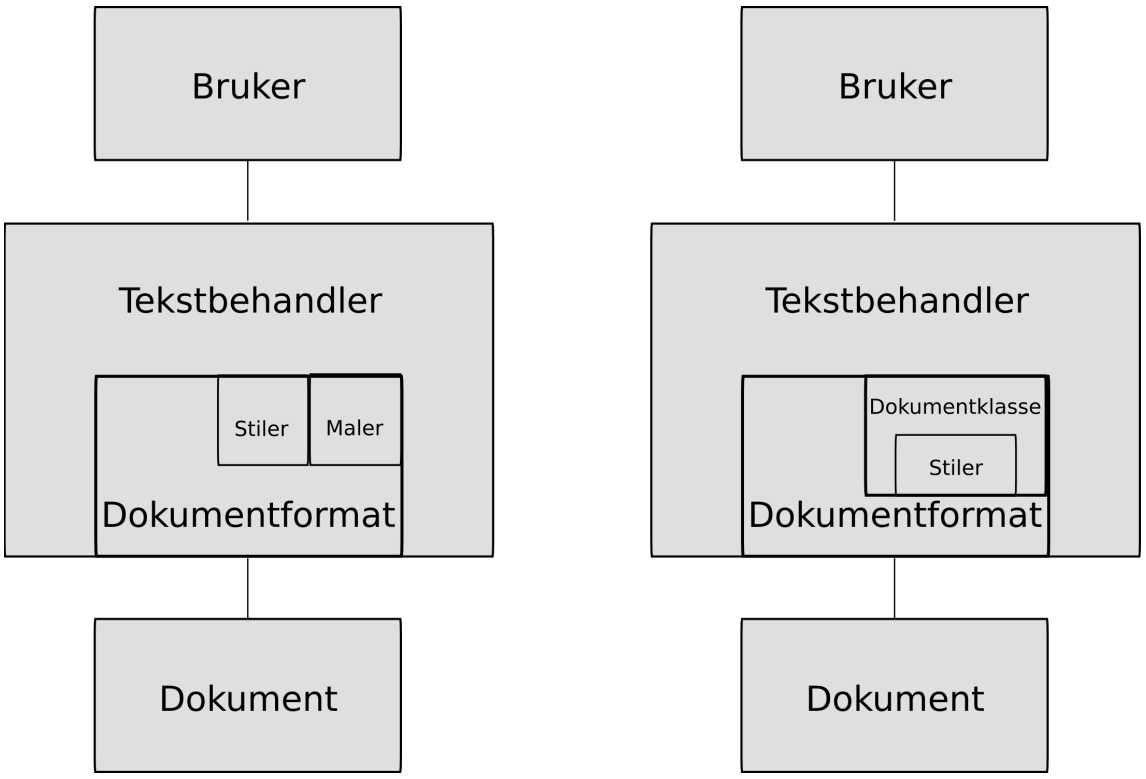
Til en viss grad brukes også malene i dag til å spesifisere hva som er lov forskjellige steder i dokumentet, ved at man kan låse felter i malene. Det kan settes opp felter som må fylles inn av brukeren, eller tekst og grafikk kan være definert slik at de alltid finnes i dokumentene som bruker malen, for å gjøre dokumentene mer ensartet. Dagens maler begrenser brukeren noe i enkelte deler av dokumentet, men brukeren har stort sett tilgang på hele office-pakka i resten av dokumentet. Feltene som er spesifisert i malen kan også utelates eller slettes.



Figur 4.1: Dokumentklassene i LaTeX, article, report og book har forskjellig funksjonalitet

I andre skrivesystemer opererer man med klasser. Klasser kan også definere felter og ensartede dokumentstrukturer, men i tillegg legger de strengere bruk for funksjonalitet. Hvis du spesifiserer at dokumentet ditt er en artikkel i LaTeX, så har du ikke muligheten til å sette inn kapitler. Tilgjengelig funksjonalitet for en artikkel er begrenset i forhold til en bok eller en rapport, og en bok har utvidet funksjonalitet i forhold til en artikkel eller en rapport. Dette vises i figur 4.1.

Figur 4.2 på neste side viser to forskjellige utvidelser av figur 1.1 på side 6, en for dokumentmaler og en for dokumentklasser. Den venstre siden beskriver forholdet i tekstbehandleren mellom grensesnitt, format, stiler og maler, der maler og stiler bidrar



Figur 4.2: Maler og klasser behandler stiler forskjellig. Venstre side: Maler bestemmer ikke over stiler. Høyre side: Dokumentklasser bestemmer over stiler.

med layout på dokumentet etter regler i dokumentformatet. Den høyre siden viser tilsvarende hvordan dokumentklasser fungerer i en tekstbehandler, men at stilene pakkes inn i og bestemmes av dokumentklassene. Dokumentklassene brukes ikke bare til å bestemme layout, men begrenser funksjonalitet gjennom å bestemme hvilke stiler som kan brukes i dokumentene. En viktig ting med dokumentklassene er at de også definerer hvor i dokumentet forskjellige stiler kan brukes.

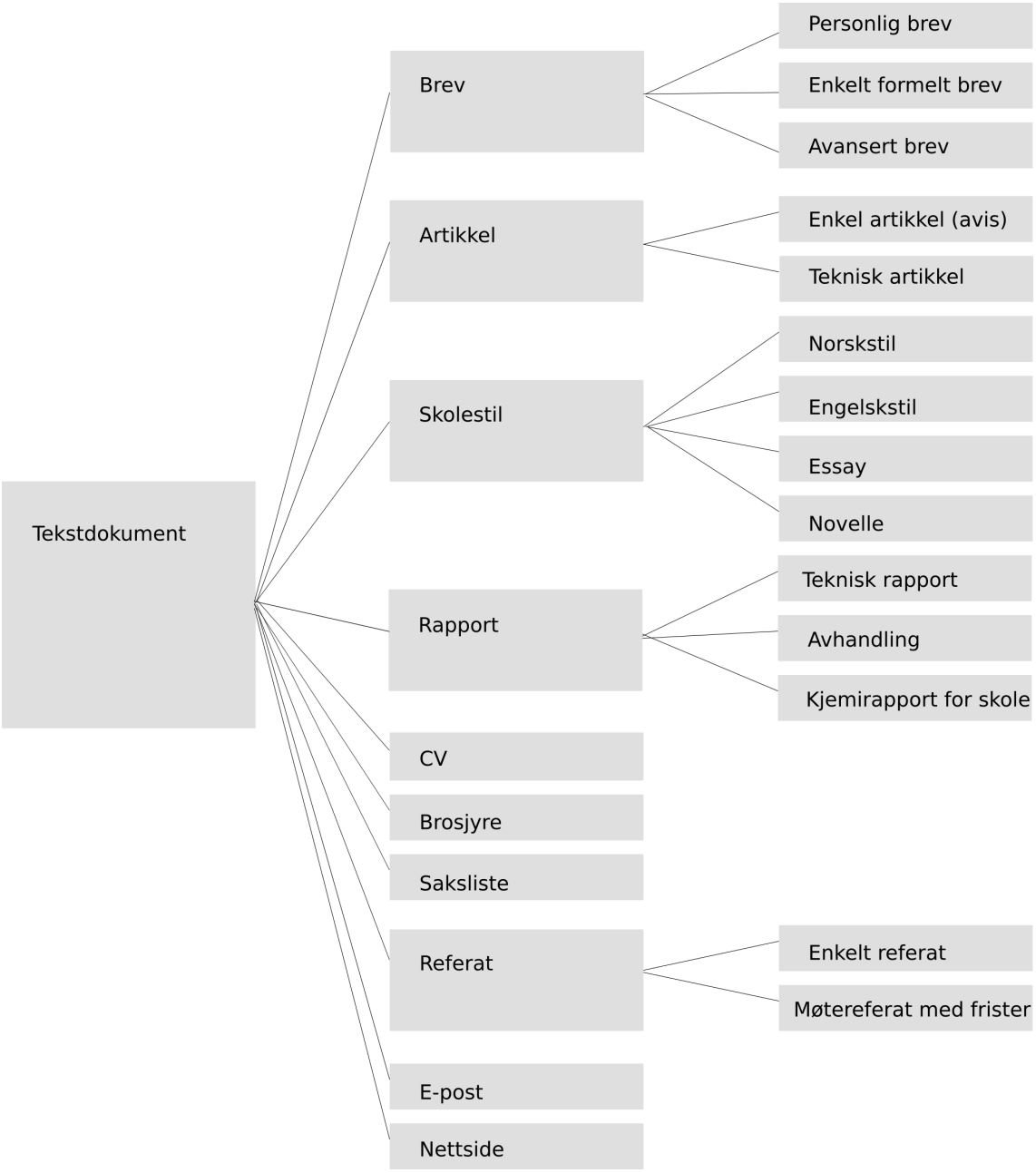
I LaTeX, som bruker klasser, får brukeren kun begrensede muligheter til å endre ting som for eksempel skriftstørrelse selv. Å endre skriftstørrelse kan enten gjøres for hele dokumentet ved å sette størrelsen på dokumentets brødtekst i starten av dokumentdefinisjonen. Størrelsen for dokumentets brødtekst brukes som utgangspunkt av LaTeXfor å sette størrelsene på de andre tekstelementene. Skriftstørrelsen kan også overstyres for deler av dokumentet, men dette er med hensikt veldig tungvint, siden det ofte forkludrer en veldefinert dokumentstil.

Maler og klasser kan også kombineres, ved at malene definerer layout, feltposisjoner og hva som er redigerbart, og klassene definerer hva som er mulig å gjøre innenfor hver enkelt stil som klassen har tilgjengelig, altså hvilke stiler som kan benyttes innen hver av de andre stilene, og hvordan hver av silene ser ut.

4.2.1 Inndeling i dokumentklasser

Alle klasser arver egenskaper fra en annen klasse, selvfølgelig med unntak av rotnoden i et slikt arvingstre som vist i figur 4.3 på neste side. Rotnoden vil naturlig være den generelle tekstklassen som ikke har noen begrensninger. Som barn av rotnoden vil vi finne de mest generelle dokumentklassene, som for eksempel Notat, Brev, Artikkel osv. Hvis brukeren er i

tvil om hva slags dokumentklasse hun skal velge, vil hun kunne prøve en av disse rot-nære dokumentklassene til å starte med, for så å endre til en mer spesialisert dokumentklasse etterhvert.



Figur 4.3: Eksempler på dokumentklasser og arvingsrelasjoner

Figuren viser noen typiske eksempler på aktuelle dokumentklasser. Trestrukturen gjør også videre spesialiseringer mulig. Det bør være enkelt å definere sine egne dokumentklasser, slik at man kan lage personlige eller firmaspesifikke dokumentklasser, i likhet med hva vi i dag har mulighet til gjennom maler. Vi vil da kunne se «Oles øvingsrapportklasse» og «Enfoldig rørs reklameavis». Eksempelklasser bør være tilgjengelig via nettjenester på lik linje med det som tilbys av maler i dag.

Tekstdokument Dette er den grunnleggende dokumentklassen, og rotnoden i arvings-

treet, og er ikke en vanlig klasse. Alle andre klasser bygger på denne.

Brev Brev er en enkel utvidelse av tekstdokumentklassen. Kan tillate bilder og enkelte andre enkle dokumentelementer.

Artikler Artikler er i utgangspunktet også enkle, men de krever en del ekstra rundt formatering av tabeller, lister og bilder, avhengig av hvor avanserte de er.

Skolestiler Dette er en enkel utvidelse av tekstklassen, med et par automatisk innfylte felter for forfatter, dokumenttittel, eventuell innleveringsfrist osv. Det finnes forskjeller på stiler, for eksempel er engelsk og norsk datoformat forskjellig, og vil derfor føre til to forskjellige, men nesten identiske dokumentklasser. Noen stilskrivingssjangere vil kanskje også kreve egne klasser.

Rapporter Rapporter og skolestiler går litt over i hverandre, men skilles på hvor teknisk innholdet er. En skolestil bør ikke trenge støtte for tabeller og figurer, men det er en kjemirapport nødt for.

CV Levnetsbeskrivelser vil være krevende med tanke på listefunksjonalitet og formatering. Her må beskrivelseslister og et stort antall nivåer og blandinger av lister støttes.

Brosjyre Det er ganske krevende grafisk å lage brosjyrer. Enkle varianter kan lages ved å ta utgangspunkt i ark med små sider, bakgrunnsbilder og flytende bilder og tekstrammer.

Saksliste Møteagendaer trenger litt avansert liste- og tabellfunksjonalitet i tillegg til vanlig tekstfunksjonalitet. Sakslister blir gjerne brukt som utgangspunkt for møtereferater, så dette blir et typisk case for at et dokument bytter dokumentklasse, noe vi ser nærmere på i 4.2.4 på side 45.

Referater Referater kan være skrevet enten helt fra bunnen av, med utgangspunkt i en saksliste, eller på bakgrunn av notater gjort andre steder. Møtereferater trenger typisk et par ekstra semantiske stiler til å markere status, aksjonspunkter og liknende.

E-post Å bruke dokumentklasser til å skrive e-post kan gi fornuftige utvidelser i forhold til de mye omdiskuterte HTML-e-postene. E-poster skrevet i tekstbehandlere må i såfall konverteres til HTML før sending.

Nettside Nettsider vil føre med seg et hav av dokumentklasser, som erstatning for dagens bruk av maler. Noen standardklasser vil også være vanskelig å lage, og vil stort sett alltid bli videreutviklet og spesialisert. Klasser for nettsider vil føre til at alle nettsidene som bruker samme klasse også får samme formatering. Også nettsider må konverteres til HTML før de publiseres på nettet.

Alle disse klassene bør kunne defineres i et ikke så altfor komplisert XML-format, og et verktøy som utvider dagens stilbegrep til å gjelde dokumentklassedefinisjoner, bør finnes tilgjengelig for avanserte brukere og administratorer. Et slikt verktøy blir behandlet i 4.2.5.

4.2.2 Klasser som begrenser

For mange muligheter for brukeren gjør at brukeren lett havner på ville veier, enten det er ved å rote seg vill i grensesnittkontroller, eller å bruke tekstelementer satt sammen på en semantisk ugyldig måte.

Et eksempel på ugyldig semantikk i dag er at det i dagens office-programmer er lov til å bruke stilen «overskrift 3» rett under «overskrift 1» uten at det er noen «overskrift 2» mellom. Office-programmene viser i dag en overskrift på øverste nivå med en overskrift på tredje nivå rett under hverandre, med de tilordnede stilene.

Selv om LaTeXogså tillater manglende overskrifter, er det en del ting som LaTeXikke tillater. I 2.1.3 på side 11 har vi sett på hvor enkelt og vanskelig det kan være å bytte skrifttyper i LaTeX, og det er også en del annen funksjonalitet som ikke skal være enkel i LaTeX, fordi det er funksjonalitet som helst ikke skal røres. LaTeXer også veldig enkel å bruke på andre ting, som at det automatisk fordeler tekst pent utover ark ved å fordele linjer jevnt nedover arket og å flytte korte ord fra fulle linjer over på mindre fulle linjer ved blokkjustering [1].

En svakhet ved LaTeXi forhold til OpenDocument som dokumentformat, er at OpenDocument prøver å være en fullstendig entydig spesifikasjon, mens LaTeXer et begrenset sett med dokumentelementer som kan utvides med nye klasser og stiler for å definere ekstra utseende. Funksjonaliteten som OpenDocument tilbyr gjør det dermed mulig å definere et grafisk brukergrensesnitt som inneholder all funksjonalitet, akkurat som OpenOffice er bygd opp, men også å lage brukergrensesnitt som inneholder begrensede subsett av den tilgjengelige funksjonaliteten.

I tekstbehandlerne kan begrensning av funksjonalitet gjøres ved gråe ut eller skjule grensesnittkontroller for elementer som dokumentklassen ikke tillater. Dette kan enten gjøres ved at tekstbehandleren sjekker hva klassen tillater og fjerner unyttige grensesnittkontroller ved lasting av dokumentet og ved klassebytter, eller at dokumentelementer blir tilgjengelige etterhvert som malen tillater at de brukes, slik at man for eksempel ikke får lov til å putte en «Overskrift 3» rett under en «Overskrift 1».

Den siste av de to løsningene, med dynamiske menystrukturer, gjør grensesnittet minimalt og veldig begrenset, og veldig enkelt å bruke. Denne løsningen bryter også et viktig brukbarhetsprinsipp om at ting ikke skal forsvinne for brukeren. I det førstnevnte alternativet vil man derimot få tilgang til mer enn man har lov til, semantisk sett. En kombinasjon av disse to, der dokumentklassene definerer mulig funksjonalitet og hvor kontrollene for ugyldige elementer gråes ut dynamisk, ivaretar brukbarhetsprinsippet som at grensesnittelementer ikke skal forsvinne.

En slik kombinert løsning vil altså i større grad kunne sørge for semantisk gyldige dokumenter, samtidig som grensesnittet mot brukeren forenkles og gjør skriveprosessen enklere. Men folk som er vant med «sin måte» å gjøre ting på, som gjerne bruker en hammer som det universelle verktøyet, vil finne det vanskeligere å bli tvunget til å følge en bestemt måte å gjøre ting på.

4.2.3 Klassenes ulemper

Ulemper vil plage brukerne, og det er et poeng å gjøre disse så små som mulig. Det vil helt klart være et problem for brukere å ikke ha tilgang til all funksjonalitet hele tiden. Med dokumentklasser av den typen som er beskrevet i oppgaven, vil brukerne bli presset inn i et annet bruksmønster. Grensesnittet mot brukeren vil være så minimalt som mulig for klassen, og brukerne vil ha problemer med å finne igjen funksjonalitet som klassene har fjernet. Ønsket funksjonalitet blir tilgjengelig så snart brukeren har byttet klasse.

[1] Vanlige tekstbehandlere fyller linjene etterhvert som man skriver dem, og flytter markøren til neste linje når en linje er full. Ved blokkjustering fordeles da ord jevnt utover de fulle linjene. LaTeXser ikke på én linje av gangen, men på hele avsnittet som helhet, og når noen linjer inneholder få og lange ord, og andre linjer inneholder mange og korte, så flyttes korte ord fra fulle linjer til mindre fulle linjer for at avsnittet skal opptre jevnere.

Det kan også være et problem for brukeren å skjønne hvordan man bytter klasse, og det må gjøres slik at når brukerne først skjønner at de må bytte dokumentklasse, må dette kunne gjøres enkelt. Hvis det også gjøres godt synlig hvor dette skjer, så vil det også være enklere å komme på at tekstbehandleren de sitter med krever bruk av dokumentklasser, og at det er viktig å være klar over hvilken klasse man sitter med.

Det er helt sikkert ikke alltid det finnes klasser som oppfyller brukernes ønsker, og det by på problemer for brukerne å redigere og lage nye dokumentklasser.

4.2.4 Bytte mellom klasser

Det er lett å tenke seg at brukeren har behov for å bytte dokumentklasse underveis på sin ferd mot et ferdig dokument. Dette kan være fordi brukeren starter med en basic rapportklasse, og seinere finner ut at den mangler noe brukeren trenger for å få det dokumentet hun ønsker.

I dag er det også vanlig å starte med et blankt dokument som bygges opp og blir noe etterhvert. Veldig mange vil helt sikkert fortsette på denne måten, da med en generell dokumentklasse som tillater det meste, for så å bytte dokumentklasse når brukeren har funnet ut hvilken dokumentklasse som egner seg.

Det er da viktig at et klassebytte informerer om hvilke deler av dokumentet som ikke kan bli vist som planlagt i det ferdige dokumentet ved bruk av en klasse som ikke støtter all funksjonalitet forfatteren av dokumentet har tatt i bruk. Likevel må brukeren få lov til å bytte til denne klassen.

Minst like viktig er det at ikke data går tapt når brukeren bytter dokumentklasse. Deler av dokumentet som ikke kan vises i den nye klassen må ikke slettes, men fremdeles være kodet inn i felter i dokumentformatet og bare være usynlig for brukeren. Det er også viktig at brukeren har tilgang til de usynlige delene av dokumentet, så de ikke opptrer som fjernet.

Et eksempel på når vi vil bytte mellom to dokumentklasser, er som vi nevnte så vidt i 4.2.1 på side 43, når vi bruker en saksliste som utgangspunkt for et møtereferat. Innholdet i de fleste dokumentelementene i møtereferatet bør da være med over i det nye dokumentet, men ikke nødvendigvis alle. Hvis referatet skal brukes som utgangspunkt for en ny saksliste, vil det kanskje også være nyttig å bruke de skjulte elementene også som utgangspunkt for den nye sakslista, kanskje som en påminnelse for ting som bør være med i den.

Klassebytte i praksis

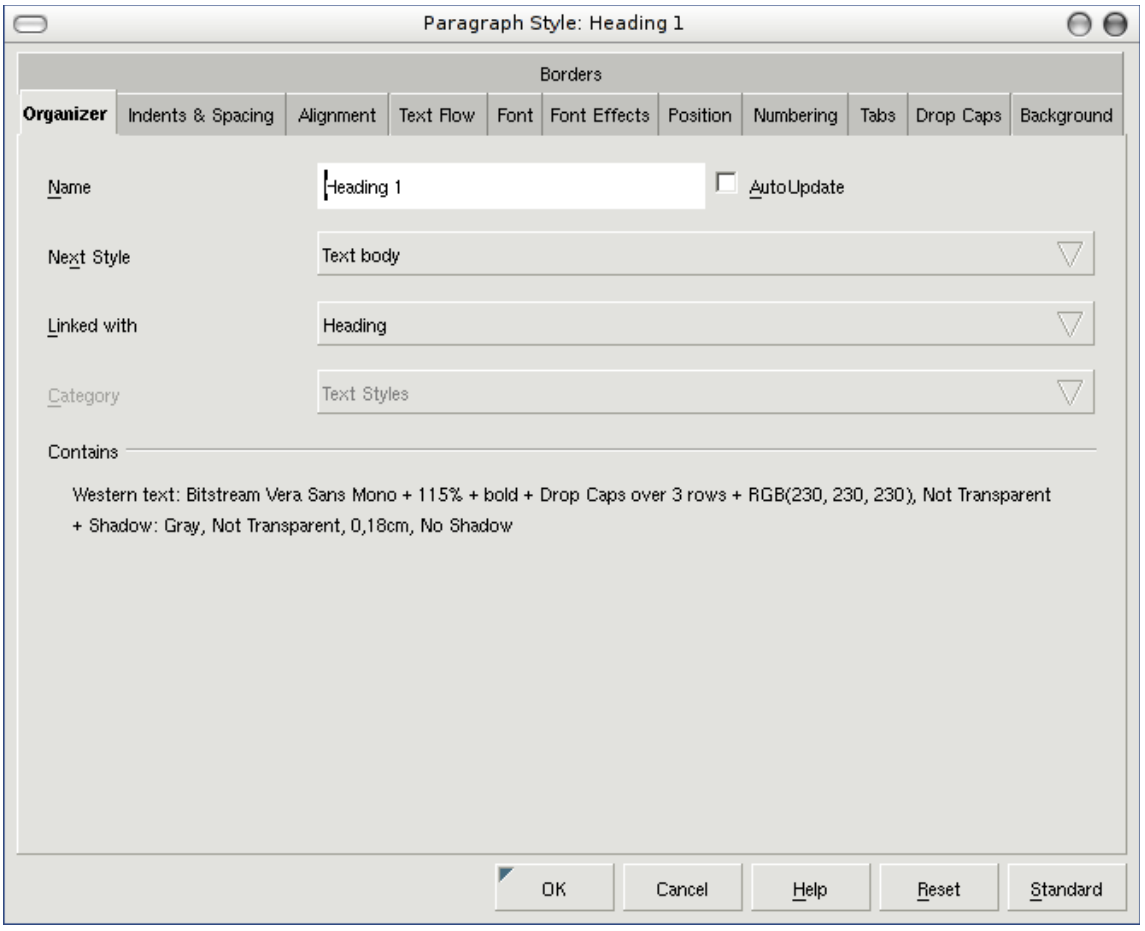
Bytting bør være lett synlig og enkelt, for eksempel med et rullegardinvalg i en verktøylinje, et valg i kontekstmenyen, eller med et eget verktøy for bytting av dokumentklasser.

Når byttet er gjort, blir grensesnittet endret, som beskrevet i del 4.2.2. Programmet endrer seg for å passe den nye klassen. Grensesnittet vil også endre seg mens brukeren skriver på et dokument og ikke tillate dokumentstrukturer som ikke er semantisk gyldige i dokumentklassen som blir brukt.

Innhold i dokumentelementer som ikke er med i den nye dokumentklassen må fremdeles være kodet i dokumentet, og grunnen til at det ikke vises er at det ikke er definert i dokumentklassen. Dataene må være lagret for å være tilgjengelige hvis brukeren bytter igjen bytter klasse tilbake til den som hadde de kodede skjulte elementene. Dataene kan også tas i bruk av en annen klasse som vil ha dem vist på en annen måte, kanskje bare deler av innholdet.

4.2.5 Klassedefinisjonsverktøy

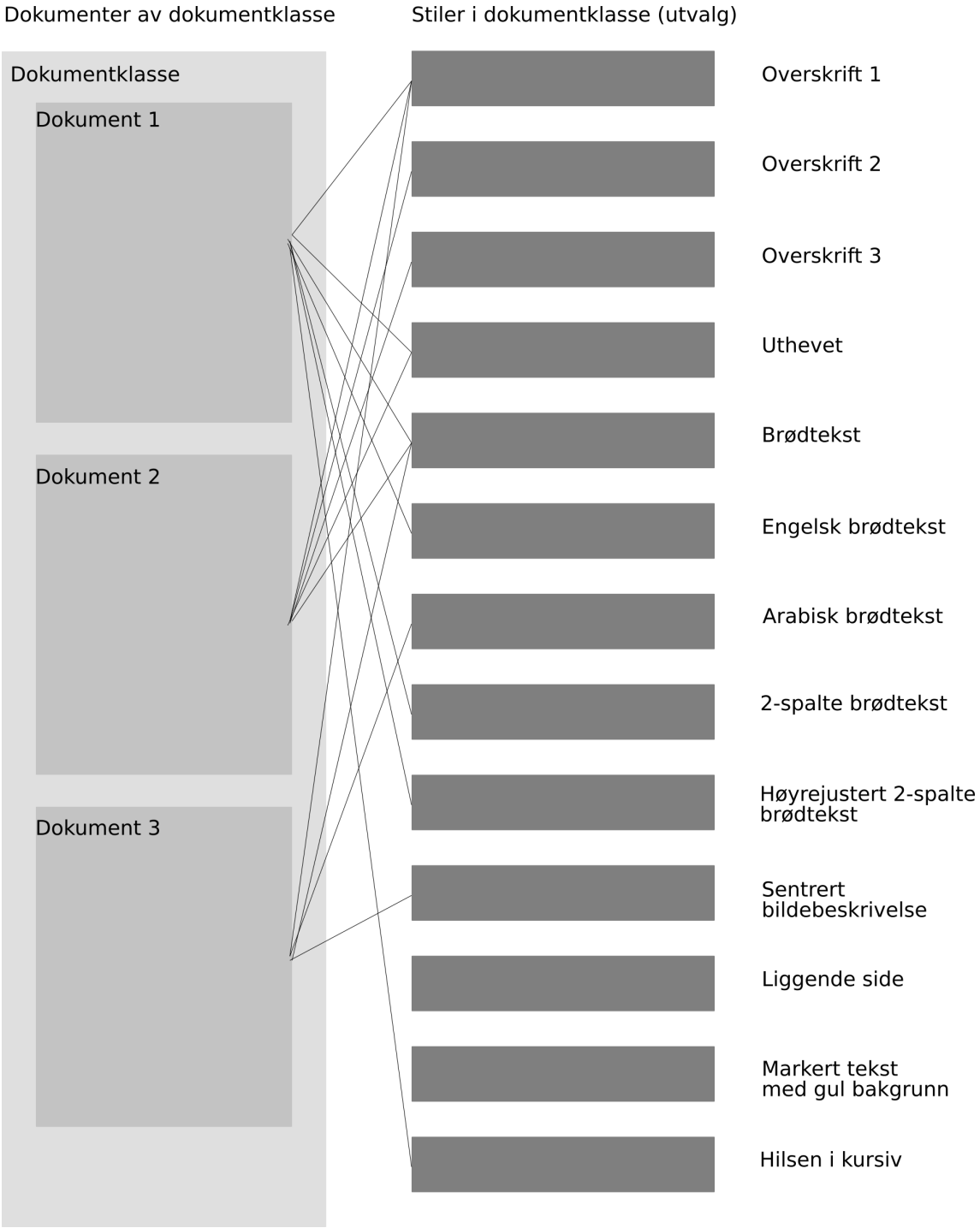
Vi har allerede sett på hvordan brukere kan bruke en klasse til å redigere dokumenter, og vil nå se på hvordan klasser kan endres og redigeres med et klassedefinisjonsverktøy. Dette vil være et supplement for de som vil definere sine klasser selv, og ikke kan bruke klasser importert fra andre dokumenter eller fra nettsteder med dokumentklassesamlinger.



Figur 4.4: Et verktøy for å endre stilene til dokumentelementer for avsnitt.

Å lage en ny klasse bør som tidligere nevnt være enkelt for brukerne, og bør gjøres i et annet verktøy eller en annen type grensesnitt enn selve skriveprosessen. Det bør fungere på samme måte som definering av stiler gjøres i dag, som vist i figur 4.4. Klassedefinering vil befinne seg på et logisk nivå over stildefinering, og naturlig innbefatte dette. Stildefinering vil da ikke lenger være en del av prosessen man gjør når man skriver et dokument. Figur 4.5 på neste side viser hvordan stilene fra figur 2.2 på side 12 brukes til å sette sammen dokumentklasser.

I dag tar det mye tid å definere stilene i et dokument. Bruk av dokumentklasser vil forenkle dette, slik at stildefinering blir gjort sammen med klassedefinering og dermed utelatt fra skriveprosessen. Dette fører til at dersom det er nødvendig å endre en stil, så vil dette bli endret i klassen, og dermed kun gjelde for denne klassen.



Figur 4.5: Dokumentklasser består av et sett med funksjonalitetsstyrende stiler.

Klassedefinering i praksis

Dokumentklassene definerer dokumentets mulige struktur, hvilke dokumentelementer som er tilgjengelige i stilen og hvilke elementer de kan stå sammen med i dokumentstrukturen. Informasjonen om dokumentklassene må være tilgjengelig sammen med dokumentet når dokumentet åpnes i en tekstbehandler. Dette utelukker bruk av nettverksløsninger og sentraladministrerte klassesamlinger, siden dette vil føre til at dokumentene ikke har tilgang på informasjonen som bestemmer deres oppbygning.

Klassedefinisjonene må lagres sammen med dokumentene for at de skal være gyldige selv med nettverksfeil eller andre tekniske problemer. En enkel løsning vil være å lagre klassedefinisjonen i dokumentene.

Dersom dokumentstrukturen, altså dokumentklassene, ikke blir lagret sammen med dokumentene, vil en støte borti problemer ved endring av dokumentklasser som er i utstrakt bruk. Det største problemet vil være å alltid sørge for at dokumentklasser som endres er bakoverkompatible med tidligere versjoner, slik at eldre dokumenter fortsatt kan åpnes. Det kan også være et prob

Denne denne informasjonen vil vi derfor lagre sammen med dokumentet, for at dokumentene skal fungere på tvers av. Dette kan gjøres ved å sette opp avanserte versjonskontrollsystemer for dokumentklassene, og ved at siste versjon alltid lastes ned fra en nettverkstjener, men den enkleste løsningen, som er mer enn god nok i dette tilfelle, vil være å lagre dokumentklassen sammen med dokumentet.

Klasser som brukes kan da automatisk lagres i skriveprogrammet dersom klassen tillater det [2], og klassene kan endres som beskrevet tidligere.

Klassedefinering i OpenDocument

Vi skal nå se på en mulighet for å definere dokumentklasser i OpenDocument med små endringer i dokumentformatet. I dag består dokumentformatet av nedpakkede filer, som vi beskrev i 2.4.4 på side 21. Vi så at `styles.xml` inneholder alle stiler som er i bruk i dokumentet. I figur 4.6 på neste side ser vi et utsnitt av innholdet i en enkel `styles.xml`, og i figur 4.7 på side 50 ser vi hva som skjer i `styles.xml` når vi legger til en stil (en overskrift på andre nivå) til i dokumentet.

En utvidelse av `styles.xml` fra å gjelde stiler som *er* i bruk i dokumentet, til å gjelde stiler som *kan være* i bruk av dokumentet, vil være en måte å lagre dokumentklassen i dokumentet. Vi har tidligere beskrevet hvorfor klasseinformasjonen må lagres sammen med dokumentet, og har nå en mulig løsning til hvordan det kan gjøres.

D på side 77 viser forslag til en enkel rapportklasse definert med `styles.xml`. Forslaget inneholder funksjonalitet som en enkel rapportklasse trenger, som for eksempel oppgavetittel og undertittel, enkle tabeller, innlimt grafikk og referanseliste.

`styles.xml` blir brukt fordi den allerede koder slik informasjon allerede, den trenger bare å brukes på flere stiler enn de som er definert i dokumentet. I tillegg til dette er tekstbehandlerne som bruker formatet nødt til å begrense utvalget av tilgjengelige stiler for brukeren til de som er definert i klassen, altså `styles.xml`. Dette har vi allerede beskrevet i 4.2.2 på side 43, der vi også beskrev hvordan stilene i klassen må definere hvilke andre stiler som skal være gyldige innenfor hver stil.

[2] Det kan være tenkelige løsninger hvor noen organisasjoner vil beskytte sine dokumentklasser fordi de inneholder malinformasjon som andre kan bruke til å skape troverdige forfalskede dokumenter.

```
<style:style style:name="Standard" style:family="paragraph" ↩
    style:class="text"/>
<style:style style:name="Text_20_body" style:display-name="Text↩
     body" style:family="paragraph" style:parent-style-name="↩
    Standard" style:class="text">
   <style:paragraph-properties fo:margin-top="0cm" fo:margin-↩
       bottom="0.212cm"/>
</style:style>
<style:style style:name="Heading" style:family="paragraph" ↩
    style:parent-style-name="Standard" style:next-style-name="↩
    Text_20_body" style:class="text">
   <style:paragraph-properties fo:margin-top="0.423cm" ↩
       fo:margin-bottom="0.212cm" fo:keep-with-next="always"/>
   <style:text-properties style:font-name="Nimbus Sans L" ↩
       fo:font-size="14pt" style:font-name-asian="Mincho" ↩
       style:font-size-asian="14pt" style:font-name-complex="↩
       Nimbus Sans L1" style:font-size-complex="14pt"/>
</style:style>
<style:style style:name="Heading_20_1" style:display-name="↩
    Heading 1" style:family="paragraph" style:parent-style-name↩
    ="Heading" style:next-style-name="Text_20_body" style:class↩
    ="text" style:default-outline-level="1">
   <style:paragraph-properties style:default-outline-level="1"/↩
       >
   <style:text-properties fo:font-size="115%" fo:font-weight="↩
       bold" style:font-size-asian="115%" style:font-weight-↩
       asian="bold" style:font-size-complex="115%" style:font-↩
       weight-complex="bold"/>
</style:style>
```

Figur 4.6: Utdrag av styles.xml med få siler.

```
<style:style style:name="Standard" style:family="paragraph" ↩
    style:class="text"/>
<style:style style:name="Text_20_body" style:display-name="Text↩
     body" style:family="paragraph" style:parent-style-name="↩
    Standard" style:class="text">
   <style:paragraph-properties fo:margin-top="0cm" fo:margin-↩
       bottom="0.212cm"/>
</style:style>
<style:style style:name="Heading" style:family="paragraph" ↩
    style:parent-style-name="Standard" style:next-style-name="↩
    Text_20_body" style:class="text">
   <style:paragraph-properties fo:margin-top="0.423cm" ↩
       fo:margin-bottom="0.212cm" fo:keep-with-next="always"/>
   <style:text-properties style:font-name="Nimbus Sans L" ↩
       fo:font-size="14pt" style:font-name-asian="Mincho" ↩
       style:font-size-asian="14pt" style:font-name-complex="↩
       Nimbus Sans L1" style:font-size-complex="14pt"/>
</style:style>
<style:style style:name="Heading_20_1" style:display-name="↩
    Heading 1" style:family="paragraph" style:parent-style-name↩
    ="Heading" style:next-style-name="Text_20_body" style:class↩
    ="text" style:default-outline-level="1">
   <style:paragraph-properties style:default-outline-level="1"/↩
       >
   <style:text-properties fo:font-size="115%" fo:font-weight="↩
       bold" style:font-size-asian="115%" style:font-weight-↩
       asian="bold" style:font-size-complex="115%" style:font-↩
       weight-complex="bold"/>
</style:style>
<style:style style:name="Heading_20_2" style:display-name="↩
    Heading 2" style:family="paragraph" style:parent-style-name↩
    ="Heading" style:next-style-name="Text_20_body" style:class↩
    ="text" style:default-outline-level="2">
   <style:paragraph-properties style:default-outline-level="2"/↩
       >
   <style:text-properties fo:font-size="14pt" fo:font-style="↩
       italic" fo:font-weight="bold" style:font-size-asian="14↩
       pt" style:font-style-asian="italic" style:font-weight-↩
       asian="bold" style:font-size-complex="14pt" style:font-↩
       style-complex="italic" style:font-weight-complex="bold"/↩
       >
</style:style>
```

Figur 4.7: Utdrag av styles.xml med en stil til.

4.3 Moduler

Kildekoden og programfunksjonaliteten til tekstbehandlerne er oppdelt i moduler, slik at funksjonalitet som henger sammen er gruppert sammen også i programmets underliggende struktur. En eller annen form for modularisering er nødvendig. Ingen vil ha støtte for *alt*. Og ingen har tid eller råd til å bære rundt på elefantprogrammer som kan gjøre alt. Modulariseringen blir gjort i flere nivåer, som ikke nødvendigvis trenger å ha noe med hverandre å gjøre.

Tradisjonen er å ha ett program per dokumentformat, altså tekstdokument, regneark, presentasjon osv, og de fleste som har laget dokumentformater til nå har også sørget for at det er et en-til-en-forhold mellom formater og program andre vei også.

I det neste nivået finner vi navnerommene. Dette er navnerom som XML-fila inneholder, og som bestemmer hvordan informasjonen som dokumentet lagrer er kodet. Hvis funksjonalitet er definert gjennom en standard, vil dokumentformatet ta i bruk denne standarden, men og kode informasjonen så den følger standarden. Dette mikser flere standarder sammen i samme dokument og dokumentformat. Navnerommene bestemmer hvilken funksjonalitet som dokumentformatet støtter.

Det er også modulariseringer på andre nivåer, som etter hvilken del av programkoden som håndterer en viss funksjonalitet. For eksempel en tegnemodul for alt som har med figurtegning å gjøre, en listemodul for alt som har med lister å gjøre, og en tabellmodul for alt som har med tabeller å gjøre.

Det vil være mulig å stykke opp modulene på andre måter, og her går jeg gjennom fordeler og ulemper ved å stykke opp modulene basert på hva man trenger i forskjellige dokumentklasser, en slags oppstykking basert på dokumentets kompleksitet.

Den første typen moduler prøver jeg videre å kalle for «navnerom» og den andre typen for «programmoduler», men kommer til å omtale moduler på generell basis for «moduler» når det ikke har noe å si hva slags moduler det er snakk om.

4.3.1 Grunner til å stykke opp

Det er flere grunner til å stykke opp dokumentformatene i moduler. På overordnet nivå er en oppstykking nyttig for å definere forskjellige dokumenttyper, gjerne i form av filtyper. OpenDocument deler, som vi så i 2.4, funksjonalitet og format med velkjente standarder, som vektorgrafikk (SVG) eller metadatamerking (Dublin Core). Dokumentformatet kan dermed blande disse kjente standardene og samtidig følge dem, på samme måte som XHTML [58] kan blandes med SVG og MathML [59] ved å blande navnerom med XML-NS [60].

Det vil også være nyttig å dele opp dokumentstrukturen i mindre enheter dersom det er mulig å dynamisk allokere de mindre enhetene i datamaskinenes minne. Der minnet er begrenset, vil dette vil få programmene mer effektive og responsive.

En oppstykking vil også gjøre det mulig for programmer å ikke implementere absolutt all funksjonalitet. Et program som vil bruke en stor spesifikasjon til å lage små dokumenter som skal lenkes sammen, vil ikke ha behov for å kunne lage innholdsfortegnelser eller figurtegninger. Det vil kanskje ikke engang ha behov for så mange som ti nivåer med overskrifter.

4.3.2 Tradisjonell oppdeling

I begynnelsen av kapittelet gikk vi gjennom hvordan modularisering gjøres i dag ved at programmene er inndelt i moduler etter hvilken funksjonalitet som hører sammen. Dette

er den tradisjonelle måten å gjøre det på, men den er ikke en særlig god løsning hvis komponenter skal lastes dynamisk, siden de fleste dokumentene stort sett trenger litt funksjonalitet fra hver programmodul, og de fleste modulene derfor må lastes likevel. En slik modularisering vil derfor ikke føre til en særlig mer effektiv ressursbruk når dokumentet har vokst seg stort enn det er å laste alle modulene, slik vi så i 4.1.2 på side 39 at OpenOffice gjør i dag. Ved oppstart av tomt dokument derimot, er få moduler lastet.

4.3.3 Inndeling i «enkel» og «avansert»

En mulig måte å dele inn funksjonalitet på er å gradvis ta i bruk mer og mer avansert funksjonalitet etterhvert som den trengs. Litt forenklet vil man da dele inn dokumenter i «enkle» og «avanserte», i likhet med dagens malsystemer, men med noen flere mellomnivåer.

Dette er en løsning som gjør det mulig å ta i bruk maskinressurser etterhvert som de blir nødvendige, og programmet vil på denne måten være enkelt og lite så lenge dokumentet er det. Forskjellene til dagens løsning med inndeling etter funksjonalitet vil ikke være veldig stor, men det er oftere at en bruker trenger litt fra hver modul enn full støtte for alt innen én modul, eksempelvis trenger man ofte enkleste typer lister og enkleste typer tabeller, men sjelden støtte for tabeller i tabeller når man skriver de enkleste typene dokumenter.

Grovinndeling av dokumentelementer

Basic
Stiler
Bilder
Lister
Mellomnivå
Topptekster og bunntekster
Faste sideskift
Spalter
Tabulatorer
Avansert
Fotnoter
Sluttnoter
Referanseliste
Tekstbokser
Figurtegninger
Kommentarer
Flere språk

En bruker er ikke interessert i å si om dokumentet skal være enkelt eller avansert, og i alle fall ikke bry seg med en mengde mellomnivåer, siden de ikke betyr noe for brukeren. Hvis en slik inndeling skal fungere, må den skje automatisk. Vi kan derfor se for oss en løsning der tekstbehandlerens funksjonalitet oppgraderes til et høyere nivå når brukeren etterspør dette ved å velge mer avansert funksjonalitet i grensesnittet.

Veien er ikke lang til å tenke på en enda finere inndeling. Dersom vi hadde delt inn funksjonalitet i et spekter av «enkle» og mer «avanserte» nivåer *per modul*, så ville vi kunne utnytte ressurser enda bedre. Tekstbehandlerens lasting av funksjonalitet ville bli enda mer dynamisk.

4.3.4 Inndeling for dokumentklasser

Med bakgrunn i den nye tankemåten rundt bruk av dokumentklasser, presentert i 4.5, ser vi på om det er mulig å tilpasse moduloppstykkinga på en ny måte. Den tradisjonelle måten å tenke modularisering på vil ikke være spesielt gunstig når vi tenker på at et gjennomsnittlig dokument kun trenger noen få prosent av funksjonaliteten som tilbys, og at det gjerne er snakk om litt funksjonalitet fra store deler av det totale antallet moduler. Det er enkelt å si at modularisering basert på dokumentenes grad av enkelhet er ønskelig, som vi så på i 4.3.3, men som vi også så vil det være vanskelig å avgjøre hvilken funksjonalitet som typisk havner inn under «enkel» og «avansert». Og i likhet med tradisjonell oppdeling vil også den gradvise oppdelingen fylle maskinenes minne mer enn nødvendig.

Modularisering for OpenDocument

En mer fingranulert modularisering basert på hvordan klassene er definert vil være en naturlig utvidelse når vi nå har funnet en måte å kode klasser på i dokumentformatet. En inndeling, for eksempel ikke bare i tekst, men i forskjellige typer tekst, som overskrifter, bildetekster, brødtekster, sitater og så videre, vil trenge en måte å mappe stilbeskrivelsene i `styles.xml` og beskrivelsene i de automatiske stilene som er lagret i `content.xml` mot de mange små modulene. Når så dette er gjort, vil vi i teorien ha en mer effektiv bruk av minnet til maskinene.

Selv om spesifikasjonen til OpenDocument i dag er streng på hva som skal til for at en tekstbehandler er i henhold til standarden, så kan det med en modularisering som dette også være mulig at enkelte mindre tekstbehandlere, som de vi så på i 3.3.1 på side 36 kunne støtte kun deler av standarden, spesifisert gjennom dokumentklasser. Dette er i alle fall mulig for tekstbehandlere som bruker et subsett av standarden eller eksporterer et subsett av standarden, da ved å bruke én eller flere dokumentklasser som utgangspunkt for lagring. Det er også mulig for tekstbehandlere som vil importere deler av standarden, ved at de er strenge på hvilke dokumentklasser de godtar som importformat, og hva som vil skje med dokumentelementer som ikke er definert i dokumentklassene.

Kapittel 5

Avslutning

5.1 Oppsummering

Denne oppgaven har tatt for seg OpenDocument som det eneste standardiserte åpne dokumentformatet, og sett på hva det kan brukes til, og mulige endringer som vil gjøre det enda mer brukbart.

5.2 Konklusjon

Det er mange fordeler knyttet til å modularisere dokumentformater med stor og krevende spesifikasjon. Dokumentklasser i seg selv vil kunne begrense tilgjengelig funksjonalitet for hver av dokumentklassene, og gjøre skriveprosessen enklere, samtidig som dokumentutveksling blir mer effektiv siden dokumentene kan følge strengere semantiske regler i dokumentenes markup. En begrensning av funksjonalitet kan også føre til mer effektive tekstbehandlere, siden programmodulene i større grad kan lastes inn dynamisk.

5.3 Videre arbeid

En naturlig videreføring er å sende inn et forslag om dokumentklasser og modularisering til OASIS-arbeidsgruppa for OpenDocument som definerer formatene, og hører om dette kan være interessant.

Det kan også gjøres forsøksimplementasjoner ved å redusere OpenOffice sin funksjonalitet på dokumentformatene til å kun gjelde stildefinisjoner fra dokumentklassen.

Det er også mulig å prøve det jeg ikke klarte, nemlig å få små tekstbehandlere til å ta i bruk et subsett av formatet, enten for konvertering begge veier, eller helst som internformat.

Bibliografi

[1] Erik T. Ray. *Learning XML*. O'Reilly & Associates, Inc, 4 edition, Jan 2001.

[2] Wikipedia. *DocBook*.
http://en.wikipedia.org/wiki/Docbook (Jun 2005).

[3] Geoff Richards. *The Big List of XML Technologies*, Jun 2002.
http://xmlsucks.org/xml_technologies/ (Jun 2005).

[4] Dean Jackson Jon Ferraiolo, Fujisawa Jun. *Scalable Vector Graphics (SVG)*. World Wide Web Consortium, Jan 2003.
http://www.w3.org/TR/2003/REC-SVG11-20030114/ (mai 2005).

[5] Thierry Michel et al. *Synchronized Multimedia Integration Language 2.0 (SMIL 2.0)*. World Wide Web Consortium, Jan 2005.
http://www.w3.org/TR/2005/REC-SMIL2-20050107/ (mai 2005).

[6] Robert Miner og Nico Poppelier David Carlisle, Patrick Ion. *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*. World Wide Web Consortium, Okt 2003.
http://www.w3.org/TR/2003/REC-MathML2-20031021/ (mai 2005).

[7] Jeremy J. Carroll og Brian McBride Graham Klyne. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. World Wide Web Consortium, Feb 2004.
http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (Jun 2005).

[8] Dave Beckett og Brian McBride. *RDF/XML Syntax Specification (Revised)*. World Wide Web Consortium, Feb 2004.
http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ (Jun 2005).

[9] Dave Winer. *RSS 2.0 Specification*. Berkman Center, Harward University, Jul 2003.
http://blogs.law.harvard.edu/tech/rss (Jun 2005).

[10] Dave Winer. *RSS History*. Berkman Center, Harward University, Apr 2004.
http://blogs.law.harvard.edu/tech/rssVersionHistory (Jun 2005).

[11] Dave Winer. *XML-RPC Specification*. UserLand Software, Jun 1999.
http://www.xmlrpc.com/spec (Jun 2005).

[12] Nilo Mitra. *SOAP Version 1.2 Part 0: Primer*. World Wide Web Consortium, Jun 2003.
http://www.w3.org/TR/2003/REC-soap12-part0-20030624/ (Jun 2005).

[13] Greg Meredith og Sanjiva Weerawarana Erik Christensen, Francisco Curbera. *Web Services Description Language (WSDL) 1.1*. World Wide Web Consortium, Mar 2001.
http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (Jun 2005).

[14] Jabber Software Foundation. *Summary of XMPP*, Mar 2005.
http://www.xmpp.org/summary.html (Jun 2005).

[15] Open Archives Initiative. *The Open Archives Initiative Protocol for Metadata Harvesting*, Jun 2002.
http://www.openarchives.org/OAI/openarchivesprotocol.html (Jun 2005).

[16] Liam Quin. *The Extensible Stylesheet Language Family (XSL)*. World Wide Web Consortium.
http://www.w3.org/Style/XSL/ ().

[17] James Clark. *XSL Transformations (XSLT) Version 1.0*. World Wide Web Consortium, Nov 1999.
http://www.w3.org/TR/1999/REC-xslt-19991116 (Jun 2005).

[18] James Clark. *XML Path Language (XPath) Version 1.0*. World Wide Web Consortium, Nov 1999.
http://www.w3.org/TR/1999/REC-xpath-19991116 (Jun 2005).

[19] World Wide Web Consortium. *Extensible Stylesheet Language (XSL)*, Okt 2001.
http://www.w3.org/TR/2001/REC-xsl-20011015/ (mai 2005).

[20] Håkon Wium Lie og Bert Bos. *Cascading Style Sheets, level 1*. World Wide Web Consortium, Jan 1999.
http://www.w3.org/TR/1999/REC-CSS1-19990111 (Jun 2005).

[21] David Orchard Steve DeRose, Eve Maler. *XML Linking Language (XLink) Version 1.0*. World Wide Web Consortium, Jun 2001.
http://www.w3c.org/TR/2000/REC-xlink-20010627/ (mai 2005).

[22] David C. Fallside og Priscilla Walmsley. *XML Schema Part 0: Primer Second Edition*. World Wide Web Consortium, Okt 2004.
http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/ (Jun 2005).

[23] World Wide Web Consortium. *XQuery 1.0: An XML Query Language*, Apr 2005.
http://www.w3.org/TR/2005/WD-xquery-20050404/ (Jun 2004).

[24] Microsoft Corporation. *Overview of WordprocessingML*, Nov 2003.
http://rep.oio.dk/Microsoft.com/officeschemas/wordprocessingml_article.htm (Jun 2005).

[25] Petter Merok. *Open Source vs. Kommersiell software - debatt med Microsoft og IBM*. Microsoft Corporation, Nov 2004.

[26] Wikipedia. *Open Document Architecture*, Feb 2004.
http://en.wikipedia.org/wiki/Open_Document_Architecture (Jun 2005).

[27] Uniform Code Council. *UCC Glossary*.
http://usnet03.uc-council.org/glossary/#S (Jun 2005).

[28] Barbara Held. *Interoperability, Standards and Open Source - Gnome User and Developer European Conference*. Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens (IDABC), Mai 2005.

[29] Trond Arne Undheim med flere. *Faktaark (Programvare)*. Teknologirådet, Aug 2003.
http://www.teknologiradet.no/files/faktaark_programvarepolitikk_copy1.pdf (Jun 2005).

[30] Bob Sutor. *Open Document Formats: "Open"must be more than a marketing term*. IBM, Jun 2005.
http://www-128.ibm.com/developerworks/blogs/dw_blog_comments.jspa?blog=384&entry=83074 (Jun 2005).

[31] Michael Brauer. *From Open Source to Open Standard: The OASIS OpenDocument Format*. Sun Microsystems, Inc., Mai 2005.
http://www.idealliance.org/proceedings/xtech05/papers/03-02-02/ (jun 2005).

[32] Microsoft Corporation. *Microsoft Office Open XML Formats Guide*, Jun 2005.
http://www.microsoft.com/office/preview/fileguide.mspx (Jun 2005).

[33] Cover Pages, OASIS. *Microsoft Announces Adoption of XML for Default File Formats in 'Office 12'*, Jun 2005.
http://xml.coverpages.org/ni2005-06-02-a.html (Jun 2005).

[34] Peter Galli. *MS Office XML Formats Not OK with GNU*. EWeek.com, Jun 2005.
http://www.eweek.com/article2/0,1759,1829355,00.asp (Jun 2005).

[35] Simon Phipps. *Defining "Open Standard", Simply*. Sun Microsystems, Jun 2005.
http://blogs.sun.com/roller/page/webmink/20050603#defining_open_standard_simply (Jun 2005).

[36] Unicode Inc. *Unicode Standard*.
http://www.unicode.org/standard/standard.html (Jun 2005).

[37] Organization for the Advancement of Structured Information Standards. *Open Document Format for Office Applications (OpenDocument) v1.0*, Mai 2005.
http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf (mai 2005).

[38] Dublin Core Metadata Initiative. *Dublin Core Metadata Initiative*.
http://dublincore.org/ (Jun 2005).

[39] Dublin Core Metadata Initiative. *Dublin Core Metadata Element Set, Version 1.1: Reference Description*, Des 2004.
http://www.dublincore.org/documents/dces/ (mai 2005).

[40] Roland Merrick og T. V. Raman Micah Dubinko, Leigh L. Klotz Jr. *XForms 1.0*. World Wide Web Consortium, Okt 2003.
http://www.w3.org/TR/2003/REC-xforms-20031014 (mai 2005).

[41] Christian Lagerkvist. *Exploring the Mnemonic user interface*.
http://cla.tc.se/mui.html (apr 2005).

[42] Jef Raskin. *The Humane Interface*. Addison Wesley, 6 edition, 2000.

[43] Mindjet. *MindManager X5 Pro*.
http://www.mindjet.com/eu/index.php (Mai 2005).

[44] Jan Gunnar Fredriksen. *Bedre læring*. Escola forlag, 1 edition, 1993.

[45] T. Nagy. *Kdissert*.
http://freehackers.org/~tnagy/kdissert/ (Mai 2005).

[46] *SyntaxReference - MoinMoin*.
http://moinmoin.wikiwikiweb.de/SyntaxReference (Jun 2005).

[47] *Emacs Wiki Site Map*.
http://www.emacswiki.org/cgi-bin/wiki (Jun 2005).

[48] Alex Graveley. *Tomboy*.
http://beatniksoftware.com/tomboy/ (Mai 2005).

[49] Microsoft Corporation. *Word 2003 Home Page*.
http://office.microsoft.com/en-us/FX010857991033.aspx (Jun 2005).

[50] Knut L. Vik. *Noen viktie begreper i Framemaker*. ITEA Desk, NTNU, Jan 2000.
http://www.itea.ntnu.no/desk/frameoppl.pdf (Jun 2005).

[51] Steve Whitlatch. *DocBook XML example projects posted. One formatted with FrameMaker 7.0. One formatted with DocBook XSL.*, Mar 2004.
http://lists.oasis-open.org/archives/docbook/200403/msg00033.html (Jun 2005).

[52] World Wide Web Consortium. *HyperText Markup Language (HTML) Home Page*.
http://www.w3.org/MarkUp/ (Jun 2005).

[53] Sigurd Gartmann. Dokumentformater. Technical report, Institutt for datateknikk og informasjonsvitenskap (IDI), NTNU, Des 2004.

[54] Wikipedia. *Portable Document Format*.
http://en.wikipedia.org/wiki/Portable_Document_Format (Jun 2005).

[55] Morten Welinder. *OpenDocument for Spreadsheets*, Jun 2005.
http://blogs.gnome.org/view/mortenw/2005/06/16/0 ().

[56] KDE e. V. *KOffice - Integrated office suite*.
http://koffice.org/ (Jun 2005).

[57] *Abiword*.
http://www.abisource.com/information/about/ (Jun 2005).

[58] World Wide Web Consortium. *XHTML (tm) 1.0 The Extensible HyperText Markup Language (Second Edition)*, Aug 2002.
http://www.w3.org/TR/2002/REC-xhtml1-20020801/ (jun 2005).

[59] Ishikawa Masayasu. *An XHTML + MathML + SVG Profile*. World Wide Web Consortium, Aug 2002.
http://www.w3.org/TR/2002/XHTMLplusMathMLplusSVG-20020809/ (jun 2005).

[60] Andrew Layman Tim Bray, Dave Hollander. *Namespaces in XML*. World Wide Web Consortium, Jan 1999.
http://www.w3.org/TR/1999/REC-xml-names-19990114/ (jun 2005).

[61] Ashley J.S Mills. *Installing And Using An XML/SGML DocBook Editing Suite*. University Of Birmingham, Jul 2002.
http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/docbooksys/segmentedhtml/index.html (Jun 2005).

Tillegg A

Strukturert XML

XML egner seg veldig godt til å strukturere dokumenter. Her ser vi en eksempelartikkel kodet i DocBook-XML. Koden er hentet fra [61].

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
<article>
  <articleinfo>
    <title>Your title here</title>

    <author>
      <firstname>Your first name</firstname>
      <surname>Your surname</surname>
      <affiliation>
        <address><email>Your e-mail address</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2002</year>
      <holder role="mailto:your e-mail address">Your name</holder>
    </copyright>

    <abstract>
      <para>Include an abstract of the article's contents here.</para↩
          >
    </abstract>
  </articleinfo>

  <sect1><title>Section 1</title>
    <para>
      blah blah blah
    </para>
  </sect1>

  <sect1><title>Section 2</title>
    <para>
      blah blah blah
    </para>
  </sect1>
</article>
```

Tillegg B

Metadata i OpenDocument

Koden viser hvordan metadata lagres i Dublin Core-navnerommet og i OpenDocument sitt utvidete `meta`-navnerom.

```
<?xml version="1.0" encoding="UTF-8"?>

<office:document-meta xmlns:office="↩
    urn:oasis:names:tc:opendocument:xmlns:office:1.0" xmlns:xlink="↩
    http://www.w3.org/1999/xlink" xmlns:dc="http://purl.org/dc/↩
    elements/1.1/" xmlns:meta="↩
    urn:oasis:names:tc:opendocument:xmlns:meta:1.0" xmlns:ooo="http:↩
    //openoffice.org/2004/office" office:version="1.0">
   <office:meta>
      <meta:generator>OpenOffice.org/1.9.{milestone}$Linux OpenOffice↩
          .org_project/680$Build-8825</meta:generator>
      <meta:initial-creator>Sigurd Gartmann</meta:initial-creator>
      <meta:creation-date>2004-12-15T10:01:00</meta:creation-date>
      <dc:date>2005-03-02T21:29:37</dc:date>
      <meta:print-date>2004-12-15T11:26:00</meta:print-date>
      <dc:language>en-US</dc:language>
      <meta:editing-cycles>8</meta:editing-cycles>
      <meta:editing-duration>PT33M0S</meta:editing-duration>
      <meta:user-defined meta:name="Info 1"/>
      <meta:user-defined meta:name="Info 2"/>
      <meta:user-defined meta:name="Info 3"/>
      <meta:user-defined meta:name="Info 4"/>
      <meta:document-statistic meta:table-count="1" meta:image-count=↩
          "1" meta:object-count="0" meta:page-count="1" ↩
          meta:paragraph-count="12" meta:word-count="46" ↩
          meta:character-count="348"/>
   </office:meta>
</office:document-meta>
```

Tillegg C

Eksportering til OpenDocument

Vi ser at XSL-transformsjonen ikke lager en eksportert versjon som er nøyaktig lik utgangspunktet fra OpenDocument. XML-transformasjonen pakker flere `text:span`-elementer i hverandre, i stedet for å bruke merker med sammenslått funksjonalitet slik OpenOffice lagde.

C.1 XSL-transformasjon

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:tomboy="http://beatniksoftware.com/tomboy" ↩
    xmlns:size="http://beatniksoftware.com/size" xmlns:office="↩
    urn:oasis:names:tc:opendocument:xmlns:office:1.0" xmlns:style="↩
    urn:oasis:names:tc:opendocument:xmlns:style:1.0" xmlns:text="↩
    urn:oasis:names:tc:opendocument:xmlns:text:1.0" xmlns:table="↩
    urn:oasis:names:tc:opendocument:xmlns:table:1.0" xmlns:draw="↩
    urn:oasis:names:tc:opendocument:xmlns:drawing:1.0" xmlns:fo="↩
    http://www.w3.org/1999/XSL/Format" xmlns:xlink="http://www.w3.org↩
    /1999/xlink" xmlns:dc="http://purl.org/dc/elements/1.1/" ↩
    xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0" ↩
    xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0↩
    " xmlns:svg="http://www.w3.org/2000/svg" xmlns:chart="↩
    urn:oasis:names:tc:opendocument:xmlns:chart:1.0" xmlns:dr3d="↩
    urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0" xmlns:math="http:↩
    //www.w3.org/1998/Math/MathML" xmlns:form="↩
    urn:oasis:names:tc:opendocument:xmlns:form:1.0" xmlns:script="↩
    urn:oasis:names:tc:opendocument:xmlns:script:1.0" xmlns:ooo="↩
    http://openoffice.org/2004/office" xmlns:ooow="http://openoffice.↩
    org/2004/writer" xmlns:oooc="http://openoffice.org/2004/calc" ↩
    xmlns:dom="http://www.w3.org/2001/xml-events" xmlns:xforms="http:↩
    //www.w3.org/2002/xforms" xmlns:xsd="http://www.w3.org/2001/↩
    XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ↩
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml" encoding="UTF-8"/>
   <xsl:preserve-space elements="*"/>
   <xsl:param name="font"/>
   <xsl:param name="export-linked"/>
   <xsl:param name="root-note"/>
   <xsl:param name="newline" select="'&#10;'"/>
   <xsl:template match="/">
```

```
<office:document-content office:version="1.0">
   <office:scripts/>
   <office:font-face-decls>
      <style:font-face style:name="Bitstream Vera Sans" ↩
          svg:font-family="'Bitstream Vera Sans'" style:font-↩
          adornments="Regular"/>
      <style:font-face style:name="Tahoma1" svg:font-family="↩
          Tahoma"/>
      <style:font-face style:name="Interface User" svg:font-↩
          family="'Interface User'" style:font-pitch="variable"↩
          />
      <style:font-face style:name="Mincho" svg:font-family="↩
          Mincho" style:font-pitch="variable"/>
      <style:font-face style:name="Tahoma" svg:font-family="↩
          Tahoma" style:font-pitch="variable"/>
      <style:font-face style:name="Times" svg:font-family="↩
          Times" style:font-family-generic="roman" style:font-↩
          pitch="variable"/>
      <style:font-face style:name="Helvetica" svg:font-family="↩
          Helvetica" style:font-family-generic="swiss" ↩
          style:font-pitch="variable"/>
   </office:font-face-decls>
   <office:automatic-styles>
      <style:style style:name="T1" style:family="text">
         <style:text-properties fo:font-size="16pt" style:font-↩
             size-asian="16pt" style:font-size-complex="16pt"/>
      </style:style>
      <style:style style:name="T2" style:family="text">
         <style:text-properties fo:font-size="14pt" style:font-↩
             size-asian="14pt" style:font-size-complex="14pt"/>
      </style:style>
      <style:style style:name="T3" style:family="text">
         <style:text-properties fo:font-size="10pt" style:font-↩
             size-asian="10pt" style:font-size-complex="10pt"/>
      </style:style>
      <style:style style:name="T4" style:family="text">
         <style:text-properties fo:background-color="#ffff00"/>
      </style:style>
      <style:style style:name="T5" style:family="text">
         <style:text-properties style:text-line-through-style="↩
             solid"/>
      </style:style>
      <style:style style:name="T6" style:family="text">
         <style:text-properties fo:font-style="italic" ↩
             style:font-style-asian="italic" style:font-style-↩
             complex="italic"/>
      </style:style>
      <style:style style:name="T7" style:family="text">
         <style:text-properties fo:font-style="italic" fo:font-↩
             weight="bold" style:font-style-asian="italic" ↩
             style:font-weight-asian="bold" style:font-style-↩
             complex="italic" style:font-weight-complex="bold"/↩
             >
      </style:style>
      <style:style style:name="T8" style:family="text">
```

```
            <style:text-properties fo:font-weight="bold" ↩
                style:font-weight-asian="bold" style:font-weight-↩
                complex="bold"/>
         </style:style>
         <style:style style:name="T9" style:family="text">
            <style:text-properties style:text-line-through-style="↩
                solid" fo:background-color="#ffff00"/>
         </style:style>
         <text:list-style style:name="L1">
            <text:list-level-style-number text:level="1" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="2" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="3" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="4" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="5" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="6" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="7" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="8" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="9" style:num↩
                -format=""/>
            <text:list-level-style-number text:level="10" ↩
                style:num-format=""/>
         </text:list-style>
      </office:automatic-styles>
      <office:body>
         <office:text>
            <text:sequence-decls>
               <text:sequence-decl text:display-outline-level="0" ↩
                   text:name="Illustration"/>
               <text:sequence-decl text:display-outline-level="0" ↩
                   text:name="Table"/>
               <text:sequence-decl text:display-outline-level="0" ↩
                   text:name="Text"/>
               <text:sequence-decl text:display-outline-level="0" ↩
                   text:name="Drawing"/>
            </text:sequence-decls>
            <xsl:apply-templates select="tomboy:note"/>
         </office:text>
      </office:body>
   </office:document-content>
</xsl:template>
<xsl:template match="tomboy:note">
   <xsl:apply-templates select="tomboy:text"/>
</xsl:template>
<xsl:template match="tomboy:text">
   <xsl:apply-templates select="node()"/>
</xsl:template>
<xsl:template match="tomboy:note/tomboy:text/*[1]/text()[1]">
   <text:h text:style-name="Heading_20_1" text:outline-level="1">
```

```
      <xsl:value-of select="substring-before(., $newline)"/>
   </text:h>
   <xsl:call-template name="replace">
      <xsl:with-param name="text" select="substring-after(., $↩
          newline)"/>
   </xsl:call-template>
</xsl:template>
<xsl:template match="tomboy:bold">
   <text:span text:style-name="T8">
      <xsl:apply-templates select="node()"/>
   </text:span>
</xsl:template>
<xsl:template match="tomboy:italic">
   <text:span text:style-name="T7">
      <xsl:apply-templates select="node()"/>
   </text:span>
</xsl:template>
<xsl:template match="tomboy:strikethrough">
   <text:span text:style-name="T5">
      <xsl:apply-templates select="node()"/>
   </text:span>
</xsl:template>
<xsl:template match="tomboy:highlight">
   <text:span text:style-name="T4">
      <xsl:apply-templates select="node()"/>
   </text:span>
</xsl:template>
<xsl:template match="size:small">
   <text:span text:style-name="T3">
      <xsl:apply-templates select="node()"/>
   </text:span>
</xsl:template>
<xsl:template match="size:large">
   <text:span text:style-name="T2">
      <xsl:apply-templates select="node()"/>
   </text:span>
</xsl:template>
<xsl:template match="size:huge">
   <text:span text:style-name="T1">
      <xsl:apply-templates select="node()"/>
   </text:span>
</xsl:template>
<xsl:template match="text()">
   <xsl:call-template name="replace">
      <xsl:with-param name="text" select="."/>
   </xsl:call-template>
</xsl:template>
<xsl:template name="replace">
   <xsl:param name="text"/>
   <xsl:choose>
      <xsl:when test="contains($text, $newline)">
         <text:p text:style-name="Text_20_body">
            <xsl:value-of select="substring-before($text, $newline↩
                )"/>
         </text:p>
         <xsl:call-template name="replace">
```

```
                <xsl:with-param name="text" select="substring-after($↩
                    text, $newline)"/>
            </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="$text"/>
        </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

C.2 Fil laget med OpenDocument

```
<?xml version="1.0" encoding="UTF-8"?>

<office:document-content xmlns:office="↩
    urn:oasis:names:tc:opendocument:xmlns:office:1.0" xmlns:style="↩
    urn:oasis:names:tc:opendocument:xmlns:style:1.0" xmlns:text="↩
    urn:oasis:names:tc:opendocument:xmlns:text:1.0" xmlns:table="↩
    urn:oasis:names:tc:opendocument:xmlns:table:1.0" xmlns:draw="↩
    urn:oasis:names:tc:opendocument:xmlns:drawing:1.0" xmlns:fo="↩
    http://www.w3.org/1999/XSL/Format" xmlns:xlink="http://www.w3.org↩
    /1999/xlink" xmlns:dc="http://purl.org/dc/elements/1.1/" ↩
    xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0" ↩
    xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0↩
    " xmlns:svg="http://www.w3.org/2000/svg" xmlns:chart="↩
    urn:oasis:names:tc:opendocument:xmlns:chart:1.0" xmlns:dr3d="↩
    urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0" xmlns:math="http:↩
    //www.w3.org/1998/Math/MathML" xmlns:form="↩
    urn:oasis:names:tc:opendocument:xmlns:form:1.0" xmlns:script="↩
    urn:oasis:names:tc:opendocument:xmlns:script:1.0" xmlns:ooo="↩
    http://openoffice.org/2004/office" xmlns:ooow="http://openoffice.↩
    org/2004/writer" xmlns:oooc="http://openoffice.org/2004/calc" ↩
    xmlns:dom="http://www.w3.org/2001/xml-events" xmlns:xforms="http:↩
    //www.w3.org/2002/xforms" xmlns:xsd="http://www.w3.org/2001/↩
    XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ↩
    office:version="1.0">
   <office:scripts/>
   <office:font-face-decls>
      <style:font-face style:name="Bitstream Vera Sans" svg:font-↩
          family="'Bitstream Vera Sans'" style:font-adornments="↩
          Regular"/>
      <style:font-face style:name="Tahoma1" svg:font-family="Tahoma"/↩
          >
      <style:font-face style:name="Interface User" svg:font-family="'↩
          Interface User'" style:font-pitch="variable"/>
      <style:font-face style:name="Mincho" svg:font-family="Mincho" ↩
          style:font-pitch="variable"/>
      <style:font-face style:name="Tahoma" svg:font-family="Tahoma" ↩
          style:font-pitch="variable"/>
      <style:font-face style:name="Times" svg:font-family="Times" ↩
          style:font-family-generic="roman" style:font-pitch="↩
          variable"/>
      <style:font-face style:name="Helvetica" svg:font-family="↩
          Helvetica" style:font-family-generic="swiss" style:font-↩
```

```
      pitch="variable"/>
</office:font-face-decls>
<office:automatic-styles>
   <style:style style:name="T1" style:family="text">
      <style:text-properties fo:font-size="16pt" style:font-size-↩
         asian="16pt" style:font-size-complex="16pt"/>
   </style:style>
   <style:style style:name="T2" style:family="text">
      <style:text-properties fo:font-size="14pt" style:font-size-↩
         asian="14pt" style:font-size-complex="14pt"/>
   </style:style>
   <style:style style:name="T3" style:family="text">
      <style:text-properties fo:font-size="10pt" style:font-size-↩
         asian="10pt" style:font-size-complex="10pt"/>
   </style:style>
   <style:style style:name="T4" style:family="text">
      <style:text-properties fo:background-color="#ffff00"/>
   </style:style>
   <style:style style:name="T5" style:family="text">
      <style:text-properties style:text-line-through-style="solid"↩
         />
   </style:style>
   <style:style style:name="T6" style:family="text">
      <style:text-properties fo:font-style="italic" style:font-↩
         style-asian="italic" style:font-style-complex="italic"/>
   </style:style>
   <style:style style:name="T7" style:family="text">
      <style:text-properties fo:font-style="italic" fo:font-weight↩
         ="bold" style:font-style-asian="italic" style:font-↩
         weight-asian="bold" style:font-style-complex="italic" ↩
         style:font-weight-complex="bold"/>
   </style:style>
   <style:style style:name="T8" style:family="text">
      <style:text-properties fo:font-weight="bold" style:font-↩
         weight-asian="bold" style:font-weight-complex="bold"/>
   </style:style>
   <style:style style:name="T9" style:family="text">
      <style:text-properties style:text-line-through-style="solid"↩
          fo:background-color="#ffff00"/>
   </style:style>
   <text:list-style style:name="L1">
      <text:list-level-style-number text:level="1" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="2" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="3" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="4" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="5" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="6" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="7" style:num-↩
         format=""/>
```

```
            <text:list-level-style-number text:level="8" style:num-↩
                format=""/>
            <text:list-level-style-number text:level="9" style:num-↩
                format=""/>
            <text:list-level-style-number text:level="10" style:num-↩
                format=""/>
        </text:list-style>
    </office:automatic-styles>
    <office:body>
        <office:text>
            <text:sequence-decls>
                <text:sequence-decl text:display-outline-level="0" ↩
                    text:name="Illustration"/>
                <text:sequence-decl text:display-outline-level="0" ↩
                    text:name="Table"/>
                <text:sequence-decl text:display-outline-level="0" ↩
                    text:name="Text"/>
                <text:sequence-decl text:display-outline-level="0" ↩
                    text:name="Drawing"/>
            </text:sequence-decls>
            <text:h text:style-name="Heading_20_1" text:outline-level="1↩
                ">Test</text:h>
            <text:p text:style-name="Text_20_body">
                <text:span text:style-name="T1">Eksempeltekst</text:span>
                <text:span text:style-name="T2">med</text:span>
                innhold
                <text:span text:style-name="T3">av</text:span>
                <text:span text:style-name="T4">forskjellige</text:span>
                <text:span text:style-name="T5">typer</text:span>
                .
                <text:span text:style-name="T6">Det </text:span>
                <text:span text:style-name="T7">er</text:span>
                <text:span text:style-name="T8"> mange</text:span>å
                mter å
                <text:span text:style-name="T9">merke tekst</text:span>å
                p.
            </text:p>
        </office:text>
    </office:body>
</office:document-content>
```


C.3 Fil eksportert fra Tomboy


```
<?xml version="1.0" encoding="UTF-8"?>

<office:document-content xmlns:office="↩
    urn:oasis:names:tc:opendocument:xmlns:office:1.0" xmlns:tomboy="↩
    http://beatniksoftware.com/tomboy" xmlns:size="http://↩
    beatniksoftware.com/size" xmlns:style="↩
    urn:oasis:names:tc:opendocument:xmlns:style:1.0" xmlns:text="↩
    urn:oasis:names:tc:opendocument:xmlns:text:1.0" xmlns:table="↩
    urn:oasis:names:tc:opendocument:xmlns:table:1.0" xmlns:draw="↩
    urn:oasis:names:tc:opendocument:xmlns:drawing:1.0" xmlns:fo="↩
    http://www.w3.org/1999/XSL/Format" xmlns:xlink="http://www.w3.org↩
    /1999/xlink" xmlns:dc="http://purl.org/dc/elements/1.1/" ↩
```

```
xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0" ↩
xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0↩
" xmlns:svg="http://www.w3.org/2000/svg" xmlns:chart="↩
urn:oasis:names:tc:opendocument:xmlns:chart:1.0" xmlns:dr3d="↩
urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0" xmlns:math="http:↩
//www.w3.org/1998/Math/MathML" xmlns:form="↩
urn:oasis:names:tc:opendocument:xmlns:form:1.0" xmlns:script="↩
urn:oasis:names:tc:opendocument:xmlns:script:1.0" xmlns:ooo="↩
http://openoffice.org/2004/office" xmlns:ooow="http://openoffice.↩
org/2004/writer" xmlns:oooc="http://openoffice.org/2004/calc" ↩
xmlns:dom="http://www.w3.org/2001/xml-events" xmlns:xforms="http:↩
//www.w3.org/2002/xforms" xmlns:xsd="http://www.w3.org/2001/↩
XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ↩
office:version="1.0">
<office:scripts/>
<office:font-face-decls>
   <style:font-face style:name="Bitstream Vera Sans" svg:font-↩
       family="'Bitstream Vera Sans'" style:font-adornments="↩
       Regular"/>
   <style:font-face style:name="Tahoma1" svg:font-family="Tahoma"/↩
       >
   <style:font-face style:name="Interface User" svg:font-family="'↩
       Interface User'" style:font-pitch="variable"/>
   <style:font-face style:name="Mincho" svg:font-family="Mincho" ↩
       style:font-pitch="variable"/>
   <style:font-face style:name="Tahoma" svg:font-family="Tahoma" ↩
       style:font-pitch="variable"/>
   <style:font-face style:name="Times" svg:font-family="Times" ↩
       style:font-family-generic="roman" style:font-pitch="↩
       variable"/>
   <style:font-face style:name="Helvetica" svg:font-family="↩
       Helvetica" style:font-family-generic="swiss" style:font-↩
       pitch="variable"/>
</office:font-face-decls>
<office:automatic-styles>
   <style:style style:name="T1" style:family="text">
      <style:text-properties fo:font-size="16pt" style:font-size-↩
          asian="16pt" style:font-size-complex="16pt"/>
   </style:style>
   <style:style style:name="T2" style:family="text">
      <style:text-properties fo:font-size="14pt" style:font-size-↩
          asian="14pt" style:font-size-complex="14pt"/>
   </style:style>
   <style:style style:name="T3" style:family="text">
      <style:text-properties fo:font-size="10pt" style:font-size-↩
          asian="10pt" style:font-size-complex="10pt"/>
   </style:style>
   <style:style style:name="T4" style:family="text">
      <style:text-properties fo:background-color="#ffff00"/>
   </style:style>
   <style:style style:name="T5" style:family="text">
      <style:text-properties style:text-line-through-style="solid"↩
          />
   </style:style>
   <style:style style:name="T6" style:family="text">
```

```
      <style:text-properties fo:font-style="italic" style:font-↩
          style-asian="italic" style:font-style-complex="italic"/>
   </style:style>
   <style:style style:name="T7" style:family="text">
      <style:text-properties fo:font-style="italic" fo:font-weight↩
         ="bold" style:font-style-asian="italic" style:font-↩
         weight-asian="bold" style:font-style-complex="italic" ↩
         style:font-weight-complex="bold"/>
   </style:style>
   <style:style style:name="T8" style:family="text">
      <style:text-properties fo:font-weight="bold" style:font-↩
         weight-asian="bold" style:font-weight-complex="bold"/>
   </style:style>
   <style:style style:name="T9" style:family="text">
      <style:text-properties style:text-line-through-style="solid"↩
          fo:background-color="#ffff00"/>
   </style:style>
   <text:list-style style:name="L1">
      <text:list-level-style-number text:level="1" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="2" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="3" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="4" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="5" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="6" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="7" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="8" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="9" style:num-↩
         format=""/>
      <text:list-level-style-number text:level="10" style:num-↩
         format=""/>
   </text:list-style>
</office:automatic-styles>
<office:body>
   <office:text>
      <text:sequence-decls>
         <text:sequence-decl text:display-outline-level="0" ↩
            text:name="Illustration"/>
         <text:sequence-decl text:display-outline-level="0" ↩
            text:name="Table"/>
         <text:sequence-decl text:display-outline-level="0" ↩
            text:name="Text"/>
         <text:sequence-decl text:display-outline-level="0" ↩
            text:name="Drawing"/>
      </text:sequence-decls>
      <text:h text:style-name="Heading_20_1" text:outline-level="1↩
         ">Test</text:h>
      <text:p text:style-name="Text_20_body">
         <text:span text:style-name="T1">Eksempeltekst</text:span>
```

```
            <text:span text:style-name="T2">med</text:span>
            innhold
            <text:span text:style-name="T3">av</text:span>
            <text:span text:style-name="T4">forskjellige</text:span>
            <text:span text:style-name="T5">typer.</text:span>
            <text:span text:style-name="T7">
            Det
            <text:span text:style-name="T8">er</text:span>
            </text:span>
            <text:span text:style-name="T8"> mange</text:span>å
            mter å
            <text:span text:style-name="T5">
            <text:span text:style-name="T4">merke tekst</text:span>
            </text:span>å
            p.
         </text:p>
      </office:text>
   </office:body>
</office:document-content>
```

Tillegg D

Rapportstil som dokumentklasse

Her vises `styles.xml` til et dokument som inneholder eksempler på dokumentelementer som en enkel rapportklasse trenger. Eksempler på innhold er oppgavetittel og undertittel, fem nivåer med overskrifter, punktlister og nummererte lister, enkle tabeller, innlimt grafikk og referanseliste.

```
<?xml version="1.0" encoding="UTF-8"?>

<office:document-styles xmlns:office="↩
    urn:oasis:names:tc:opendocument:xmlns:office:1.0" xmlns:style="↩
    urn:oasis:names:tc:opendocument:xmlns:style:1.0" xmlns:text="↩
    urn:oasis:names:tc:opendocument:xmlns:text:1.0" xmlns:table="↩
    urn:oasis:names:tc:opendocument:xmlns:table:1.0" xmlns:draw="↩
    urn:oasis:names:tc:opendocument:xmlns:drawing:1.0" xmlns:fo="↩
    http://www.w3.org/1999/XSL/Format" xmlns:xlink="http://www.w3.org↩
    /1999/xlink" xmlns:dc="http://purl.org/dc/elements/1.1/" ↩
    xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0" ↩
    xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0↩
    " xmlns:svg="http://www.w3.org/2000/svg" xmlns:chart="↩
    urn:oasis:names:tc:opendocument:xmlns:chart:1.0" xmlns:dr3d="↩
    urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0" xmlns:math="http:↩
    //www.w3.org/1998/Math/MathML" xmlns:form="↩
    urn:oasis:names:tc:opendocument:xmlns:form:1.0" xmlns:script="↩
    urn:oasis:names:tc:opendocument:xmlns:script:1.0" xmlns:ooo="↩
    http://openoffice.org/2004/office" xmlns:ooow="http://openoffice.↩
    org/2004/writer" xmlns:oooc="http://openoffice.org/2004/calc" ↩
    xmlns:dom="http://www.w3.org/2001/xml-events" office:version="1.0↩
    ">
   <office:font-face-decls>
      <style:font-face style:name="StarSymbol" svg:font-family="↩
          StarSymbol" style:font-charset="x-symbol"/>
      <style:font-face style:name="Tahoma1" svg:font-family="Tahoma"/↩
          >
      <style:font-face style:name="Interface User" svg:font-family="'↩
          Interface User'" style:font-pitch="variable"/>
      <style:font-face style:name="Mincho" svg:font-family="Mincho" ↩
          style:font-pitch="variable"/>
      <style:font-face style:name="Tahoma" svg:font-family="Tahoma" ↩
          style:font-pitch="variable"/>
      <style:font-face style:name="Times" svg:font-family="Times" ↩
          style:font-family-generic="roman" style:font-pitch="↩
          variable"/>
```

```
    <style:font-face style:name="Helvetica" svg:font-family="↩
        Helvetica" style:font-family-generic="swiss" style:font-↩
        pitch="variable"/>
</office:font-face-decls>
<office:styles>
    <style:default-style style:family="graphic">
        <style:graphic-properties draw:shadow-offset-x="0.3cm" ↩
            draw:shadow-offset-y="0.3cm" draw:start-line-spacing-↩
            horizontal="0.283cm" draw:start-line-spacing-vertical="↩
            0.283cm" draw:end-line-spacing-horizontal="0.283cm" ↩
            draw:end-line-spacing-vertical="0.283cm" style:flow-with↩
            -text="false"/>
        <style:paragraph-properties style:text-autospace="ideograph-↩
            alpha" style:line-break="strict" style:writing-mode="lr-↩
            tb" style:font-independent-line-spacing="false">
            <style:tab-stops/>
        </style:paragraph-properties>
        <style:text-properties style:use-window-font-color="true" ↩
            fo:font-size="12pt" fo:language="en" fo:country="US" ↩
            style:font-size-asian="12pt" style:language-asian="none"↩
             style:country-asian="none" style:font-size-complex="12↩
            pt" style:language-complex="none" style:country-complex=↩
            "none"/>
    </style:default-style>
    <style:default-style style:family="paragraph">
        <style:paragraph-properties fo:hyphenation-ladder-count="no-↩
            limit" style:text-autospace="ideograph-alpha" ↩
            style:punctuation-wrap="hanging" style:line-break="↩
            strict" style:tab-stop-distance="1.251cm" style:writing-↩
            mode="page"/>
        <style:text-properties style:use-window-font-color="true" ↩
            style:font-name="Times" fo:font-size="12pt" fo:language=↩
            "en" fo:country="US" style:font-name-asian="Interface ↩
            User" style:font-size-asian="12pt" style:language-asian=↩
            "none" style:country-asian="none" style:font-name-↩
            complex="Tahoma" style:font-size-complex="12pt" ↩
            style:language-complex="none" style:country-complex="↩
            none" fo:hyphenate="false" fo:hyphenation-remain-char-↩
            count="2" fo:hyphenation-push-char-count="2"/>
    </style:default-style>
    <style:default-style style:family="table">
        <style:table-properties table:border-model="collapsing"/>
    </style:default-style>
    <style:default-style style:family="table-row">
        <style:table-row-properties fo:keep-together="auto"/>
    </style:default-style>
    <style:style style:name="Standard" style:family="paragraph" ↩
        style:class="text"/>
    <style:style style:name="Text_20_body" style:display-name="Text↩
         body" style:family="paragraph" style:parent-style-name="↩
        Standard" style:class="text">
        <style:paragraph-properties fo:margin-top="0cm" fo:margin-↩
            bottom="0.212cm"/>
    </style:style>
    <style:style style:name="Signature" style:family="paragraph" ↩
        style:parent-style-name="Standard" style:class="text">
```

```
    <style:paragraph-properties text:number-lines="false" ↩
        text:line-number="0"/>
  </style:style>
  <style:style style:name="Heading" style:family="paragraph" ↩
     style:parent-style-name="Standard" style:next-style-name="↩
     Text_20_body" style:class="text">
    <style:paragraph-properties fo:margin-top="0.423cm" ↩
        fo:margin-bottom="0.212cm" fo:keep-with-next="always"/>
    <style:text-properties style:font-name="Helvetica" fo:font-↩
        size="14pt" style:font-name-asian="Mincho" style:font-↩
        size-asian="14pt" style:font-name-complex="Tahoma" ↩
        style:font-size-complex="14pt"/>
  </style:style>
  <style:style style:name="Heading_20_1" style:display-name="↩
     Heading 1" style:family="paragraph" style:parent-style-name↩
     ="Heading" style:next-style-name="Text_20_body" style:class↩
     ="text" style:default-outline-level="1">
    <style:paragraph-properties style:default-outline-level="1"/↩
        >
    <style:text-properties fo:font-size="115%" fo:font-weight="↩
        bold" style:font-size-asian="115%" style:font-weight-↩
        asian="bold" style:font-size-complex="115%" style:font-↩
        weight-complex="bold"/>
  </style:style>
  <style:style style:name="Heading_20_2" style:display-name="↩
     Heading 2" style:family="paragraph" style:parent-style-name↩
     ="Heading" style:next-style-name="Text_20_body" style:class↩
     ="text" style:default-outline-level="2">
    <style:paragraph-properties style:default-outline-level="2"/↩
        >
    <style:text-properties fo:font-size="14pt" fo:font-style="↩
        italic" fo:font-weight="bold" style:font-size-asian="14↩
        pt" style:font-style-asian="italic" style:font-weight-↩
        asian="bold" style:font-size-complex="14pt" style:font-↩
        style-complex="italic" style:font-weight-complex="bold"/↩
        >
  </style:style>
  <style:style style:name="Heading_20_3" style:display-name="↩
     Heading 3" style:family="paragraph" style:parent-style-name↩
     ="Heading" style:next-style-name="Text_20_body" style:class↩
     ="text" style:default-outline-level="3">
    <style:paragraph-properties style:default-outline-level="3"/↩
        >
    <style:text-properties fo:font-size="14pt" fo:font-weight="↩
        bold" style:font-size-asian="14pt" style:font-weight-↩
        asian="bold" style:font-size-complex="14pt" style:font-↩
        weight-complex="bold"/>
  </style:style>
  <style:style style:name="Heading_20_4" style:display-name="↩
     Heading 4" style:family="paragraph" style:parent-style-name↩
     ="Heading" style:next-style-name="Text_20_body" style:class↩
     ="text" style:default-outline-level="4">
    <style:paragraph-properties style:default-outline-level="4"/↩
        >
    <style:text-properties fo:font-size="85%" fo:font-style="↩
        italic" fo:font-weight="bold" style:font-size-asian="85%↩
```

```
        " style:font-style-asian="italic" style:font-weight-↩
        asian="bold" style:font-size-complex="85%" style:font-↩
        style-complex="italic" style:font-weight-complex="bold"/↩
        >
</style:style>
<style:style style:name="Heading_20_5" style:display-name="↩
    Heading 5" style:family="paragraph" style:parent-style-name↩
    ="Heading" style:next-style-name="Text_20_body" style:class↩
    ="text" style:default-outline-level="5">
   <style:paragraph-properties style:default-outline-level="5"/↩
       >
   <style:text-properties fo:font-size="85%" fo:font-weight="↩
       bold" style:font-size-asian="85%" style:font-weight-↩
       asian="bold" style:font-size-complex="85%" style:font-↩
       weight-complex="bold"/>
</style:style>
<style:style style:name="List" style:family="paragraph" ↩
    style:parent-style-name="Text_20_body" style:class="list">
   <style:text-properties style:font-name-complex="Tahoma1"/>
</style:style>
<style:style style:name="Table_20_Contents" style:display-name=↩
    "Table Contents" style:family="paragraph" style:parent-↩
    style-name="Standard" style:class="extra">
   <style:paragraph-properties text:number-lines="false" ↩
       text:line-number="0"/>
</style:style>
<style:style style:name="Table_20_Heading" style:display-name="↩
    Table Heading" style:family="paragraph" style:parent-style-↩
    name="Table_20_Contents" style:class="extra">
   <style:paragraph-properties fo:text-align="center" ↩
       style:justify-single-word="false" text:number-lines="↩
       false" text:line-number="0"/>
   <style:text-properties fo:font-style="italic" fo:font-weight↩
       ="bold" style:font-style-asian="italic" style:font-↩
       weight-asian="bold" style:font-style-complex="italic" ↩
       style:font-weight-complex="bold"/>
</style:style>
<style:style style:name="Caption" style:family="paragraph" ↩
    style:parent-style-name="Standard" style:class="extra">
   <style:paragraph-properties fo:margin-top="0.212cm" ↩
       fo:margin-bottom="0.212cm" text:number-lines="false" ↩
       text:line-number="0"/>
   <style:text-properties fo:font-size="10pt" fo:font-style="↩
       italic" style:font-size-asian="10pt" style:font-style-↩
       asian="italic" style:font-name-complex="Tahoma1" ↩
       style:font-size-complex="10pt" style:font-style-complex=↩
       "italic"/>
</style:style>
<style:style style:name="Illustration" style:family="paragraph"↩
     style:parent-style-name="Caption" style:class="extra"/>
<style:style style:name="Index" style:family="paragraph" ↩
    style:parent-style-name="Standard" style:class="index">
   <style:paragraph-properties text:number-lines="false" ↩
       text:line-number="0"/>
   <style:text-properties style:font-name-complex="Tahoma1"/>
</style:style>
```

```
<style:style style:name="Contents_20_Heading" style:display-↩
    name="Contents Heading" style:family="paragraph" ↩
    style:parent-style-name="Heading" style:class="index">
   <style:paragraph-properties fo:margin-left="0cm" fo:margin-↩
       right="0cm" fo:text-indent="0cm" style:auto-text-indent=↩
       "false" text:number-lines="false" text:line-number="0"/>
   <style:text-properties fo:font-size="16pt" fo:font-weight="↩
       bold" style:font-size-asian="16pt" style:font-weight-↩
       asian="bold" style:font-size-complex="16pt" style:font-↩
       weight-complex="bold"/>
</style:style>
<style:style style:name="Contents_20_1" style:display-name="↩
    Contents 1" style:family="paragraph" style:parent-style-↩
    name="Index" style:class="index">
   <style:paragraph-properties fo:margin-left="0cm" fo:margin-↩
       right="0cm" fo:text-indent="0cm" style:auto-text-indent=↩
       "false">
      <style:tab-stops>
         <style:tab-stop style:position="16.999cm" style:type="↩
             right" style:leader-style="dotted" style:leader-↩
             text="."/>
      </style:tab-stops>
   </style:paragraph-properties>
</style:style>
<style:style style:name="Contents_20_2" style:display-name="↩
    Contents 2" style:family="paragraph" style:parent-style-↩
    name="Index" style:class="index">
   <style:paragraph-properties fo:margin-left="0.499cm" ↩
       fo:margin-right="0cm" fo:text-indent="0cm" style:auto-↩
       text-indent="false">
      <style:tab-stops>
         <style:tab-stop style:position="16.499cm" style:type="↩
             right" style:leader-style="dotted" style:leader-↩
             text="."/>
      </style:tab-stops>
   </style:paragraph-properties>
</style:style>
<style:style style:name="Contents_20_3" style:display-name="↩
    Contents 3" style:family="paragraph" style:parent-style-↩
    name="Index" style:class="index">
   <style:paragraph-properties fo:margin-left="0.998cm" ↩
       fo:margin-right="0cm" fo:text-indent="0cm" style:auto-↩
       text-indent="false">
      <style:tab-stops>
         <style:tab-stop style:position="16cm" style:type="↩
             right" style:leader-style="dotted" style:leader-↩
             text="."/>
      </style:tab-stops>
   </style:paragraph-properties>
</style:style>
<style:style style:name="Contents_20_4" style:display-name="↩
    Contents 4" style:family="paragraph" style:parent-style-↩
    name="Index" style:class="index">
   <style:paragraph-properties fo:margin-left="1.498cm" ↩
       fo:margin-right="0cm" fo:text-indent="0cm" style:auto-↩
       text-indent="false">
```

```
        <style:tab-stops>
            <style:tab-stop style:position="15.501cm" style:type="↩
                right" style:leader-style="dotted" style:leader-↩
                text="."/>
        </style:tab-stops>
    </style:paragraph-properties>
</style:style>
<style:style style:name="Contents_20_5" style:display-name="↩
    Contents 5" style:family="paragraph" style:parent-style-↩
    name="Index" style:class="index">
    <style:paragraph-properties fo:margin-left="1.997cm" ↩
        fo:margin-right="0cm" fo:text-indent="0cm" style:auto-↩
        text-indent="false">
        <style:tab-stops>
            <style:tab-stop style:position="15.002cm" style:type="↩
                right" style:leader-style="dotted" style:leader-↩
                text="."/>
        </style:tab-stops>
    </style:paragraph-properties>
</style:style>
<style:style style:name="Bibliography_20_Heading" style:display ↩
    -name="Bibliography Heading" style:family="paragraph" ↩
    style:parent-style-name="Heading" style:class="index">
    <style:paragraph-properties fo:margin-left="0cm" fo:margin-↩
        right="0cm" fo:text-indent="0cm" style:auto-text-indent=↩
        "false" text:number-lines="false" text:line-number="0"/>
    <style:text-properties fo:font-size="16pt" fo:font-weight="↩
        bold" style:font-size-asian="16pt" style:font-weight-↩
        asian="bold" style:font-size-complex="16pt" style:font-↩
        weight-complex="bold"/>
</style:style>
<style:style style:name="Title" style:family="paragraph" ↩
    style:parent-style-name="Heading" style:next-style-name="↩
    Subtitle" style:class="chapter">
    <style:paragraph-properties fo:text-align="center" ↩
        style:justify-single-word="false"/>
    <style:text-properties fo:font-size="18pt" fo:font-weight="↩
        bold" style:font-size-asian="18pt" style:font-weight-↩
        asian="bold" style:font-size-complex="18pt" style:font-↩
        weight-complex="bold"/>
</style:style>
<style:style style:name="Subtitle" style:family="paragraph" ↩
    style:parent-style-name="Heading" style:next-style-name="↩
    Text_20_body" style:class="chapter">
    <style:paragraph-properties fo:text-align="center" ↩
        style:justify-single-word="false"/>
    <style:text-properties fo:font-size="14pt" fo:font-style="↩
        italic" style:font-size-asian="14pt" style:font-style-↩
        asian="italic" style:font-size-complex="14pt" style:font ↩
        -style-complex="italic"/>
</style:style>
<style:style style:name="Numbering_20_Symbols" style:display-↩
    name="Numbering Symbols" style:family="text"/>
<style:style style:name="Bullet_20_Symbols" style:display-name=↩
    "Bullet Symbols" style:family="text">
```

```
    <style:text-properties style:font-name="StarSymbol" fo:font-↩
        size="9pt" style:font-name-asian="StarSymbol" style:font ↩
        -size-asian="9pt" style:font-name-complex="StarSymbol" ↩
        style:font-size-complex="9pt"/>
</style:style>
<style:style style:name="Emphasis" style:family="text">
    <style:text-properties fo:font-style="italic" style:font-↩
        style-asian="italic" style:font-style-complex="italic"/>
</style:style>
<style:style style:name="Citation" style:family="text">
    <style:text-properties fo:font-style="italic" style:font-↩
        style-asian="italic" style:font-style-complex="italic"/>
</style:style>
<style:style style:name="Strong_20_Emphasis" style:display-name↩
    ="Strong Emphasis" style:family="text">
    <style:text-properties fo:font-weight="bold" style:font-↩
        weight-asian="bold" style:font-weight-complex="bold"/>
</style:style>
<style:style style:name="Frame" style:family="graphic">
    <style:graphic-properties text:anchor-type="paragraph" svg:x↩
        ="0cm" svg:y="0cm" fo:margin-left="0.201cm" fo:margin-↩
        right="0.201cm" fo:margin-top="0.201cm" fo:margin-bottom↩
        ="0.201cm" style:wrap="parallel" style:number-wrapped-↩
        paragraphs="no-limit" style:wrap-contour="false" ↩
        style:vertical-pos="top" style:vertical-rel="paragraph-↩
        content" style:horizontal-pos="center" style:horizontal-↩
        rel="paragraph-content" fo:padding="0.15cm" fo:border="↩
        0.002cm solid #000000"/>
</style:style>
<style:style style:name="Graphics" style:family="graphic">
    <style:graphic-properties text:anchor-type="paragraph" svg:x↩
        ="0cm" svg:y="0cm" style:wrap="dynamic" style:number-↩
        wrapped-paragraphs="no-limit" style:wrap-contour="false"↩
         style:vertical-pos="top" style:vertical-rel="paragraph"↩
         style:horizontal-pos="center" style:horizontal-rel="↩
        paragraph"/>
</style:style>
<text:outline-style>
    <text:outline-level-style text:level="1" style:num-format=""↩
        />
    <text:outline-level-style text:level="2" style:num-format=""↩
        />
    <text:outline-level-style text:level="3" style:num-format=""↩
        />
    <text:outline-level-style text:level="4" style:num-format=""↩
        />
    <text:outline-level-style text:level="5" style:num-format=""↩
        />
    <text:outline-level-style text:level="6" style:num-format=""↩
        />
    <text:outline-level-style text:level="7" style:num-format=""↩
        />
    <text:outline-level-style text:level="8" style:num-format=""↩
        />
    <text:outline-level-style text:level="9" style:num-format=""↩
        />
```

```
        <text:outline-level-style text:level="10" style:num-format="↩
            "/>
      </text:outline-style>
      <text:notes-configuration text:note-class="footnote" style:num-↩
          format="1" text:start-value="0" text:footnotes-position="↩
          page" text:start-numbering-at="document"/>
      <text:notes-configuration text:note-class="endnote" style:num-↩
          format="i" text:start-value="0"/>
      <text:bibliography-configuration text:prefix="[" text:suffix="]↩
          " fo:language="nb" fo:country="NO"/>
      <text:linenumbering-configuration text:number-lines="false" ↩
          text:offset="0.499cm" style:num-format="1" text:number-↩
          position="left" text:increment="5"/>
   </office:styles>
   <office:automatic-styles>
      <style:page-layout style:name="pm1">
        <style:page-layout-properties fo:page-width="20.999cm" ↩
            fo:page-height="29.699cm" style:num-format="1" ↩
            style:print-orientation="portrait" fo:margin-top="2cm" ↩
            fo:margin-bottom="2cm" fo:margin-left="2cm" fo:margin-↩
            right="2cm" style:writing-mode="lr-tb" style:footnote-↩
            max-height="0cm">
           <style:footnote-sep style:width="0.018cm" style:distance-↩
               before-sep="0.101cm" style:distance-after-sep="0.101↩
               cm" style:adjustment="left" style:rel-width="25%" ↩
               style:color="#000000"/>
        </style:page-layout-properties>
        <style:header-style/>
        <style:footer-style/>
      </style:page-layout>
   </office:automatic-styles>
   <office:master-styles>
      <style:master-page style:name="Standard" style:page-layout-name↩
          ="pm1"/>
      <style:master-page style:name="First_20_Page" style:display-↩
          name="First Page" style:page-layout-name="pm1" style:next-↩
          style-name="Standard"/>
   </office:master-styles>
</office:document-styles>
```