Høgskolen i Østfold

XML

Et utdrag fra: http://www.ia.hiof.no/~borres/ml

Dette utdraget fra et relativt dynamisk vevsted vil alltid være litt ute av synk. Det er bare ment som litt trøst for de som vil ha noe å lese på senga før de sovner.

Sidene er produsert ved hjelp av WXT (http://www.ia.hiof.no/~borres/wxtdoc/), Python og PrinceXML (http://www.princexml.com/)

Rettighetene er de samme som for vevstedet materialet er hentet fra:

- Bruk det til hva du vil så lenge du refererer til vevstedet
- Det følger ikke med noen som helst garanti for at koden og teksten er riktig i en hver forstand.



Høgskolen i Østfold

Innhold

- Markup
- XML
- Noen begreper
- Strukturer
- Validering
- DTD
- Schema
- Formattering
- CSS
- Media
- Sider i CSS
- Skripting
- XSL-FO
- Noen XML-språk
- SVG
- MathML
- X3D

Markup

Hva:

Bakgrunn for dagens XML-teknologi. Noen viktige begreper og sammenhengen mellom dem, SGML, XML, HTML, XHTML, CSS

På den ene siden ønsker vi å tilføre dokumenter **utseende** eller layout og på den andre siden ønsker vi å gi dokumenter en **struktur**.

Behovet for utseende er åpenbart fordi vi ønsker å lage pene dokumenter, med overskrifter, innrykk, ulike fonter og fontstørrelser etc.

Struktur ønsker vi i hovedsak av to grunner:

- Vi vil bruke dokumentene som datasett og skrive eller bruke programmer som kan lese og endre innholdet som strukturerte data. F.eks ekstrahere ingressen fra en serie dokumenter og samle dem i et annet dokument. For at dette skal være mulig må begrepet ingress være definert, identifiserbart, i dokumentene.
- Vi vil lett kunne endre dokumentets layout ved å endre egenskapene til de enkelte elementene i dokumentet. F.eks. endre fonten på alle overskrifter, eller endre linjeavstand i all brødtekst. Dette er viktig også fordi vi ønsker å framstille dokumentet på forskjellige medier som papir, skjermer, små skjermer, overheadkanoner. For at dette skal være mulig må de enkelte elementtypene merkes, det er ikke effektivt å gi individuell form til alle deler av et dokument.

Denne to-sidigheten ved et dokument, utseende og struktur, er nøkkelen til å forstå dokumentbehandling og alle de ulike teknologiene som er under utvikling på dette området. Dette er på mange måter noe av det mest spennende som skjer innen databehandling for tiden. Behovet for å finne, presentere og utveksle innhold i den stadig voksende informasjonsjungelen som kalles Internettet er selvsagt hovedgrunnen til at denne utviklingen er sentral nå.

Markup av tekst er ikke noe nytt fenomen. Det er lett å tenke seg at behovet for å sette merker i en standardisert form i en tekst for å angi hvordan den skulle settes har vært nyttig og nødvendig i trykkeribransjen i lang tid. Elektronisk behandling av dokumenter gjør det selvsagt mulig å sette slike merker i teksten på en slik måte at framstillingsteknologien, skjermdriver eller printerdriver, kan tolke merkene. Begrepet WYSIWYG (What You See Is What You Get) som ble introdusert på 80-tallet i forbindelse med introduksjon av rastergrafiske skjermer er en god illustrasjon på dette. Det WYSIWYG egentlig betyr er at det du lager på skjermen er det samme som det du får ut på skriveren. Som bruker av tekstbehandlingsprogrammer, som f.eks. MSWorld, blir du tilbudt funksjonalitet som i tillegg til å behandle selve teksten, gjør det mulig å bestemme nøyaktig hvordan teksten skal se ut. For at dette skal være mulig må det legges igjen (skjult) informasjon i teksten som skriveren og skjermdriveren kan forstå og tolke. Slik informasjon om dataene kan vi kalle metainformasjon. Det har blitt introdusert en rekke formater, både proprietære og åpne, for å realisere dette.

Øvelse

Lage et enkelt dokument i Microsoft Word, med en overskrift og et par linjer tekst.

- Spar dokumetet som .doc format. Forsøk å åpne det i Ultraedit, og se om du kan finne igjen den teksten du har skrevet. doc er et proprietært binærformat fra Microsoft og du er avhengig av at Utraedit forsøker å tolke binærdataene som tegn.
- Spar dokumentet som .rtf (rich tekst format). Forsøk å åpne det i Ultraedit eller Notepad, og se om du kan finne igjen den teksten du har skrevet. rtf er et format som er introdusert av Microsoft. rtf har fått en viss utbredelse (og et litt frynsete rykte).
- Spar dokumentet som .html. Forsøk å åpne det i Ultraedit eller Notepad, og se om du kan finne igjen den teksten du har skrevet.

Forsøk å endre teksten slik du finner den i Utraedit (eller Notepad) i de forskjellige formatene og forsøk deretter å åpne den i Word.

Skriv dokumentet ved å velge en tilgjengelig skriver, men merk av at du vil skrive til fil. Forsøk å åpne den .prn- fila som lages og se om du kan finne igjen den teksten du har skrevet. prn fila vil for de fleste printere være i Postscript. Postscript er til forskjell fra eksemplene ovenfor et programmeringsspråk som utføres på skriveren.

Hovedfokus i WYSIWYG-strategien er layout, altså hvordan dokumentet skal se ut. Digital behandling av strukturerte data har historisk hatt fokus mot databaser, regnearkformater eller "kommasepararerte" filer. Framstilling av strukturerte data har i stor grad vært behandlet av diverse rapportgeneratorer eller spesialprogrammer i de enkelte tilfellene. Nå er ikke verden fullt så enkel: Microsoft har lagt ned en god del arbeid i å utvikle proprietære teknologier for å integrere verktøyene i Office-familien på en slik måte at data kan sirkulere mellom ulike verktøy i familien.

Mangelen på ryddighet og forståelse av de to perspektivene på et dokument, layout og struktur, er et problem som stikker dypt, både systemteknisk og mentalt. De fleste brukere av tekstbehandlere har ikke dette skillet klart for seg og bruker strukturerelementer og formatspesifikasjoner om hverandre for å oppnå visuelle effekter. Det samme skjer ved skriving av HTML.

XML-teknologien er interessant fordi:

- det gjør det mulig og naturlig å skille klart mellom struktur og layout
- det gjør det mulig å bearbeide begge aspektene ved et dokument hver for seg, og samtidig
- den gjør det teknisk enkelt å definere felles struktur for dokumenter innen en organisasjon eller et fagområde.
- det finnes en rekke produkter og programbiblioteker som gjør behandling av XML-dokumenter sikkert og enkelt.
- den er i motsetning til de fleste av de andre markup-språkene befriende enkel. Det er så enkelt at det er vanskelig å korrumpere selve ideen med proprietære varianter
- den er anvendbar både til tradisjonell dokumentbehandling og til veldefinerte mellomformater i systemløsninger, f.eks. til konfigurasjonsbeskrivelser.
- den trolig vil erstatte de enkleste databaseløsningene.
- den passer som hånd i hanske som mellomformat mot databaseløsninger.

Historie i lommeformat

Historien om World Wide Web og tilhørende teknologier er skrevet i utallige versjoner og finnes en eller annen form i de fleste lærebøker i relevante emner. Denne historien tolkes og analyseres på forskjellig måte, alt etter hvilket fokus forfatteren har. Det du finner her er en svært kort versjon som har fokus på å forstå hvorfor XML er et svar på en utvikling som delvis har vært styrt av et felles mål, og dels har gått av sporet på noen avgjørende punkter.

SGML

(Standard Generalized Mark-up Language) [1]

SGML var opprinnelig et initiativ fra IBM for å muliggjøre utveksling av dokumenter mellom systemer. Arbeidet ble påbegynt så tidlig som på slutten av 60-tallet av Goldfarb, med hjelp av Mosher og Lorie (GML). Arbeidet endte opp som ISO-standard (ISO 8879) i 1986. SGML er en svært omfattende standard som har hatt en viss utbredelse innen organisasjoner med stort behov for dokumentadministrasjon. Typiske anvendelser er forlag som ønsker å kunne publisere i forskjellige formater og f.eks. flyfabrikker med stort behov for å kunne dokumentere produksjon, der kvalitetssikring og sporbarhet er nøkkelord.

SGML har de samme karakteristika som vi forbinder med XML. SGML er utvidbart, det vil si at det ikke er et endelig språk, men et regelsett for å definere språk, et metaspråk. SGML-språk var basert på definisjon av dokumenttyper, DTD (Document Type Definition). Det ble skrevet parsere for SGML og det ble definert et språk for å definere stilsett for SGML-dokumenter, DSSSL (Document Style Semantics and Spesification Language). [2]

Det er alltid vanskelig å vurdere hva som gjør at en teknologi "tar av" mens en annen ikke gjør det. Det finnes en god del logiske argumenter for at SGML kunne slått gjennom som en viktig nøkkelteknologi i et større marked. Hovedgrunnene til at det ikke skjedde var trolig for det første at det er for omfattende og for det andre at behovet for dokumentstrukturering ikke var allment annerkjent som en generell utfordring før bruken av Internett hadde modnet.

HTML

(Hyper Text Markup Language) [3]

HTML ble utviklet ved CERN (opprinnelig: Conseil Européen pour la Recherche Nucléaire, nå:Organisation Européenne pour la Recherche Nucléaire) i 1989 av Tim Berners-Lee. Hensikten var å kunne utveksle og binde sammen dokumenter innen organisasjonen. Arbeidet ble basert på SGML og HTML kan betraktes som et språk definert i SGML. HTML skiller seg altså fra SGML, og XML, ved at det er et definert språk med et fast ordforråd. HTML har et endelig sett med elementer.

HTML ble utviklet uten at noen var i stand til å forutse hvor stor utbredelsen av dette språket skulle bli. Word Wide Web eksploderte og vi fikk generelle nettlesere som kunne parse og framstille HTML.

Det er mange som i ettertid har fundert på hvordan HTML burde ha vært og hvordan nettleserne burde ha utviklet seg for at vi idag skulle hatt en litt enklere verden enn vi har. Slike spekulasjoner er interessante i den grad vi kan bruke dem til å belyse aktuelle problemstillinger i dagens utvklingsarbeid. Det er to forhold som fortjener oppmerksomhet som grunnlag for forståelse.

1. HTML bidrar ikke til å gjennomføre et klart skille mellom struktur og layout. I hovedsak ble HTML designet med elementer som er av strukturell art, men vi vet at mangelen på stilsett tidlig i utviklingen har ført til att de strukturelle elementene brukes til å tvinge fram layout.

2. Nettleserne ble alt fra starten laget med en betydelig toleranse for misformede dokumenter. SGML har til en viss grad en slik toleranse. Dette har opp gjennom mange generasjoner av HTML og nettlesere ført til en ganske uoversiktlig situasjon der dokumenter blir tolket og framstilt på forskjellig og ofte uforutsette måter. Det har dessuten ført til at den dokumentstrukturen som legges til grunn ved parsing blir tvetydig. Scripting, dynamisk HTML på klientsiden, blir en risikosport.

CSS

(Cascading Style Sheets). [4]

Vi må minne om at en av hovedpersonene bak CSS, Håkon Wium Lie, kommer fra Halden og har sin grunnutdanning i informatikk fra HiØ.

CSS er et språk som gjør det mulig å definere hvordan ulike elementer i et HTML- eller XML-dokument skal vises. Hensikten med dette er altså at dokumentet kan lages (i hovedsak) med tanke på struktur, mens utseendet kan defineres og endres uten å endre i selve dokumentet. Dette er interessant også fordi ett og samme dokument kan vises etter de behov ulike medier krever (Store skjermer, små skjermer, papir). CSS er et svar på noe av det som ble ivaretatt med DSSSL i forhold til SGML, men som ikke ble ivaretatt når HTML opprinnelig ble definert.

Det er mange forfattere som mener at CSS ikke er en fullgod løsning på layoutproblemet og de framholder XSL-FO (eXtensible Stylesheet Language-Formatting) som en alternativ eller komplementerende teknologi. I dette materialet vil du finne stoff om CSS og XSLT (eXtensible Stylesheet Language Transformations), og litt om XSL-FO.

XML

(eXtensible Markup Language) [5]

XML er i en viss forstand en ny start, der erfaringene fra historien er ivaretatt. XML er bygget på samme lest som SGML, og er i stor utstrekning et subsett av SGML. Det er vesentlig enklere enn SGML, det har ingen toleranse for ufullstendige formuleringer og det er altså, i motsetning til HTML, ikke et språk men et metaspråk. Det vil si: et regelsett for å definere språk. XML mangler noe viktig for å være et språk, nemlig ord.

Det betyr at når vi skriver et XML-dokument kan vi kontrollere at det er velformet i henhold til de grunnleggende generelle syntaksreglene, og vi kan kontrollere at det er validerbart som dokument av en bestemt type. Det siste forutsetter at vi har definert en dokumenttype i form av en DTD (Document Type Defnition) eller et SCHEMA. Å definere en DTD betyr egentlig at vi definerer vårt eget språk. En validering betyr at vi sjekker at et dokument er skrevet i henhold til dette spåket. XHTML er et slikt språk. Det er i hovedsak HTML4 med utvidede formelle krav til riktig syntaks som blandt annet innebærer at en del attributter som kontrollerer utseende er fjernet og formateringen forutsettes beskrevet som inkluderte eller lenkede stilark.

Når det i mange sammenhenger understrekes at XML er enkelt, så er det en sannhet med modifikasjoner. De grunnleggende syntaksreglene i XML er befriende enkle, men bildet kompliseres en god del av en del praktiske nødvendigheter som dukker opp når fragmenter fra ulike dokumenttyper skal blandes osv. Mer om slike komplikasjoner andre steder i dette materialet.

Andre markup-språk

Det er lett å se at det generelle behovet for å standardisere dokumenter slik at de kan utveksles mellom applikasjoner har ført til en rekke mer eller mindre utbredte standardformater opp gjennom databehandlingens historie. De som er nevnt nedenfor har nesten utelukkende blitt utformet med tanke på å beskrive layout. Det finnes sikkert en rekke andre som er kjente i ulike miljøer.

RTF

Rich Text Format er et format som ble introdusert av Microsoft. Se f.eks.: [6] for en beskrivelse. Du kan se hvordan det ser ut ved å lagre en MS Word fil som rtf og åpne den i Ultraedit eller Notepad eller en lignende teksteditor.

LaTeX

ble opprinnelig introdusert som TeX av Donald Knuth fordi han trengte et verktøy for å kontrollere teksten i en bok han skrev. LaTeX har hatt og har fortsatt utbredelse og tilhengere i akademiske miljøer. Mer om både LaTeX og Knuth på sidene til TeX Users Group: [7]

troff

Utdrag fra innledningen fra Troff User's Manual by Jospeh F. Ossanna and Brian W. Kernighan, se [8]

"troff is a text-formatting program for driving the Graphic Systems phototypesetter on the UNIX and GCOS operating systems. This device is capable of producing high quality text.... troff allows the user full control over fonts, sizes, and character positions, as well as the usual features of a formatter: right-margin justification, automatic hyphenation, page titling and numbering, and so on. It also provides macros, arithmetic variables and operations, and conditional testing, for complicated formatting tasks."

Postscript

Postscript representerer en litt annen innfallsvinkel til formatering. Vi markerer ikke for Postscript i dokumentet, men det finnes oversettere som genererer Postscript basert på annen markup. Hele dokumentet omgjøres til et program som så kan transporteres og kjøres på den enheten som skal presentere det. Printere er typiske enheter som kan utføre postscriptkode. [9]

Referanser og eksempler:

- [1] SGML (Standard Generalized Mark-up Language):
  www.w3.org/MarkUp/SGML/
- [2] DSSSL(Document Style Semantics and Spesification Language):
  www.jclark.com/dsssl/
- [3] HTML 4.01(HyperText Markup Language):
  www.w3.org/TR/html4/
- [4] CSS (Cascading Style Sheets):
  www.w3.org/Style/CSS/
- [5] XML (Extensible Markup Language):
  www.w3.org/XML/
- [6] RTF (Rich text Format):
  www.biblioscape.com/rtf15_spec.htm
- [7] LaTeX:
  www.tug.org/
- [8] troff:
  www.kohala.com/start/troff/troff.html
- [9] Postscript:
  www.adobe.com/products/postscript/main.html

XML

eXtensible Markup Language

Hva:

Forklaring av anatomien, de enkelte delene, i et XML-dokument

XML er ikke et språk Det er et metaspråk eller om vi vil, et sett med regler for å lage språk.

XML er på den ene siden enkelt. Det er intuitivt enkelt å lage et formelt riktig XML-dokument. På den annen side er det en del nødvendige mekanismer som gjør det litt vanskeligere.

I grove trekk kan vi si at et XML-dokument består av tre hoveddeler: En prolog, datadelen (rotelementet) og en epilog. Noe av dette er nødvendig og noe er valgfritt.

Et XML-dokument kan se slik ut, med tre hoveddeler:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!-- dette er en velformet fil uten mening -->
<!DOCTYPE litteratur SYSTEM "bokdok.dtd" [<!ENTITY gj "God Jul">]>
<?xml-stylesheet type="text/xsl" href="a_trans.xslt"?>

<litteratur/>

<!-- epilog -->
```

Prolog

Den første linja har flere komponenter:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Den første delen, *xml version=1.0"* er obligatorisk og signaliserer til det programmet som skal lese fila, og oss, at dette er en XML-fil. Det kan ikke være blanke linjer eller tegn foran denne begynnelsen.

Den andre delen *encoding="UTF-8"* sier oss hvordan innholdet er kodet, hva slags tegnsett som er brukt. Dette åpner for stor frihet til å skrive i en rekke ulike språk med tegnsett som er helt annerledes en det vi er vant til å bruke. Det finnes en rekke definerte tegnsett som har kjente betegnelser. For å skrive norsk bruker vi ofte "ISO-8859-1", som er en standard som ivaretar våre nordeuropeiske tegn. Tegnsett og nødvendig tilhørende teknologi er en ganske omfattende historie. Unicode er en viktig del av dette. UTF-8 er 8-bits unicode som expanderer i antall byte etter behov. UTF-8 er "standard" og default koding i alt som har med XML å gjøre

Den tredje delen *standalone="no"* sier oss hvorvidt dokumentet er selvforsynt, eller om andre filer er involvert, f.eks. om det finnes en ekstern DTD. Denne delen trenger vi ikke ta med, og det litt uavklart hvordan ulike programmer og parsere skal forholde seg til dette.

Den andre linja:

```
<!-- dette er en velformet fil uten mening -->
```

er en kommentarlinje som kan settes inn hvor som helst i et xml-dokument.

Den tredje linja:

```
<!DOCTYPE litteratur SYSTEM "bokdok.dtd">
```

er en dokumentdeklarasjon, *DOCTYPE*.

Den andre delen *litteratur* angir at deklarasjonen gjelder dokumenter med rotelement litteratur.

Den tredje delen *SYSTEM "bokdok.dtd"* henger sammen og sier oss at selve dokumentdefinisjonen ligger på en URI som heter bokdok.dtd. Det vil i dette tilfellet si en fil i samme katalogen som selve XML-dokumentet. SYSTEM sier oss at dette ikke er noen generell, allment kjent DTD. Alternativet er PUBLIC. PUBLIC antyder at den aktuelle dokmenttypen er slik at det behandlende programmet kan gjenkjenne den og kan behandle den uten slå opp definisjonen. I så fall tjener URI'en som en identifikasjon, snarere enn en adresse til en definisjon.

En dokumentdeklarasjon kan inneholde både en PUBLIC-del og en SYSTEM-del. F.eks. kjenner vi deklarasjonen av XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

der den første delen *-//W3C//DTD XHTML 1.0 Strict//EN* er tilstrekkelig informasjon for nettlesere til å vite hva de skal gjøre mens den andre URI'en er en implisitt SYSTEM-deklarasjon og angir hvor DTD'en faktisk befinner seg.

Det er også lovlig å lage følgendedede konstruksjon:

```
<!DOCTYPE litteratur SYSTEM "bokdok.dtd" [<!ENTITY gj "God Jul">]>
```

der vi har lagt til en spesifikasjon av en entitet til den aktuelle DTD'en.

Den fjerde linja

```
<?xml-stylesheet type="text/xsl" href="a_trans.xslt"?>
```

er et eksempel på en prosesserings instruksjon (PI). I dette tilfellet angivelse av et stilsett som skal brukes, her en XSL-transformasjon. Et annet eksempel er angivelse av et CSS-stilsett:

```
<?xml-stylesheet type="text/css" href="a_stil.css"?>
```

Roteelementet

Hoveddelen, den delen som vi hovedsaklig har fokus på er i dette perspektivet svært enkel. Den består av et rotelement som kan inneholde alle de komponentene som kan inngå i lovlig, velformet XML. (elementer, tekst, attributter, CDATA). I eksempelet over er rotelementet tomt.

```
<litteratur/>
```

Siden et XML-dokument skal ha en ren trestruktur, så er det altså ikke lov å ha flere "rotelementer".

Epilog

Dette er en litt underlig del av et XML-dokument. En epilog kan inneholde alt som kan inneholdes i en prolog, untatt selve XML-linja og dokumenttypedeklarasjonen. Det vil si at vi kan legge inn kommentarer og prosesserings instruksjoner. Det er ikke åpennbart at dette er en nødvendig del av et XML-dokument (?).

Referanser og eksempler:

www.w3.org: www.w3.org/XML

Noen begreper

Hva:

En kortfattet sammenhengende innføring i de viktigste begrepene knyttet til XML, velformethet, validerbarhet, elementer, attributter, CDATA

Velformet XML

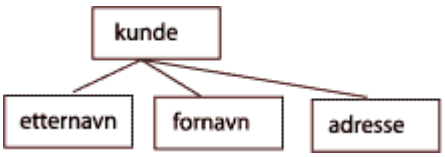
Følgende er en velformet XML-fil:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<kunde>
<etternavn/>
<fornavn/>
<adresse/>
</kunde>
```

DEMO: http://www.ia.hiof.no/~borres/ml/xml/dok1.xml

Du ser sannsynligvis ikke så mye dersom du kjører en nettleser som ikke tolker XML og som ignorerer elementer som ikke er HTML.

Vi ser at den første linja angir at dette er en XML-fil slik at programvare, f.eks. en nettleser, som leser fila vet hva den kan forvente. Det angis dessuten hva slags koding som er benyttet. I dette tilfellet er det ISO-8859-1 som er en ISO-standard (International Standard Organisation) som ivaretar lagring og presentasjon av våre nordiske tegn (æ,ø,å). Vi går ikke mer inn på dette her. De neste linjene er elementer som er ordnet i en trestruktur.



Betingelsen for velformethet i dette eksempelet er at elementene lukkes før neste begynner. Merk at elementer kan skrives på en to former.

- Vi kan skrive <fornavn/> for et komplett element uten innhold. Slik som det står her har det ikke mye mening siden det ikke har hverken attributter, underelementer eller innhold.
- Vi kan skrive <kunde> innhold </kunde>, altså en åpningsdel og en lukkedel med innhold i mellom.

Denne er ikke velformet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<kunde>
  <etternavn>
  </etternavn>
  <fornavn>
  <adresse>
  </fornavn>
  </adresse>
</kunde>
```

DEMO: http://www.ia.hiof.no/~borres/ml/xml/dokfeil.xml

Vi ser at fila ovenfor (den velformede versjonen) ikke inneholder informasjon om noen spesiell person. Den bærer ikke semantisk informasjon. Vi kan skrive den om slik:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<kunde>
<etternavn>
 Nordmann
</etternavn>
<fornavn>
Ola
</fornavn>
<adresse>
Trollheimen
</adresse>
</kunde>
```

DEMO: http://www.ia.hiof.no/~borres/ml/xml/dok2.xml

Vi ser at fila beskriver de fire aktuelle elementene: adresse, etternavn, fornavn og kunde. For de tre første er det angitt at innholdet skal våre PCDATA (Parsed Character Data). Det vil i vår foreløpige analyse si tekst. For elementet kunde ser vi at et kunde-element skal inneholde en forekomst av de tre andre elementene og ikke noe "løs" tekst.

Vi kopler dokumenttypen til selve dokumentet på følgende måte:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE kunde SYSTEM "dok2.dtd">
<kunde>
<etternavn>
Nordmann
</etternavn>
<fornavn>
Ola
</fornavn>
<adresse>
Trollheimen
</adresse>
</kunde>
```

DEMO: http://www.ia.hiof.no/~borres/ml/xml/dok22.xml

Der den andre linja angir dokumenttypen, kunde, og en referanse, URL, til en fil som inneholder DTD'en. Dokumenttypen, kunde, er det samme som navnet på rotelementet i XML-fila. Slik dokumentet nå er formulert kan programvare sjekke det både med tanke på velformethet og med tanke på validitet.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE kunde SYSTEM "dok2.dtd">
<kunde>
<fornavn>
Ola
</fornavn>
<hårfarge>
Grønn
</hårfarge>
<adresse>
Trollheimen
</adresse>
</kunde>
```

Nettleserne sjekker ikke validitet. (etternavnet er erstattet med hårfarge)

DEMO: http://www.ia.hiof.no/~borres/ml/xml/dok22feil.xml

Det finnes andre verktøy for å validere XML-dokumenter. Noen kan du laste ned og kjøre offline og noen ligger som en tjeneste på nettet. XMLSpy kan validere og er et greitt laboratorium for å eksperimentere med XML. Du finner referanse til noen nettbaserte validatorer under referanser på denne siden.

Attributter

I eksempelet ovenfor har vi konsentrert oss om elementer og tekstinnhold. I XML kan elementer ha attributter. Vi kan kople egenskaper til elementene. F.eks. kan vi skrive:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<kunde kundeid="a123">
<etternavn>
 Nordmann
</etternavn>
<fornavn>
Ola
</fornavn>
<adresse>
Trollheimen
</adresse>
</kunde>
```

Her har vi sagt at en kunde har egenskapen, attributtet, kundeid. Denne måten å uttrykke seg på er kjent fra HTML, der vi f.eks. skriver:

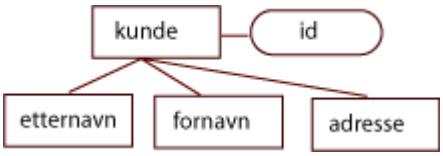
```
<img src="bilde.gif" alt="oversikt" width="40" height="20"/>
```

Reglene for velformethet i XML krever at det skal være en blank mellom navn-verdi parene i attributtlista og at alle verdier skal skrives i "". Vi vet at tidligere versjoner av HTML, som er SGML-basert, tolererer at vi utelater "". Slik er det ikke i XML, eller i nye HTML varianter som er definert som et XML-språk.

En dokumenttype definisjon må også kunne spesifisere hvilke attributter som er lovlige og hvor de skal kunne forekomme. En utvidet DTD for vårt superenkle eksempel ser slik ut:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT adresse (#PCDATA)>
<!ELEMENT etternavn (#PCDATA)>
<!ELEMENT fornavn (#PCDATA)>
<!ELEMENT kunde (etternavn, fornavn, adresse)>
<!ATTLIST kunde
    kundeid CDATA #REQUIRED
>
```

Når vi har plassert kundeid som en attributt til kundelementet har vi truffet et valg. Vi kunne like gjerne innført kundeid som et nytt element på linje med etternavn, fornavn og adresse. Det er ofte en avveining hvorvidt opplysninger skal plasseres som attributter eller elementer. Mange forfattere drøfter dette uten at det er etablert noen entydige gode regler for hvordan en DTD bør designes med tanke på denne avveiningen. En tommeregel er kanskje å anvende samme type resonnement og vurdering som når vi definererer klasser når vi programmerer. Attributter er kanskje sammenlignbare med attributter i et objekt, mens elementer er selvstendige objekter. Analogien halter nok på noen punkter, men som utgangspunkt for resonneringen er den kanskje brukbar? Vi vil møte igjen denne problemstillingen når vi skal skrive transformasjoner. Det er dessuten ført noen resonnementer om dette i modulen Strukturer



Struktur med attributt.

CDATA

Det er et problem i behandlingen av XML-filer at vi av og til ønsker at dokumentet skal innholde tekst som ikke skal tolkes av det programmet som leser dokumentet. F.eks.:

```
<?xml version="1.0"?>
<eksamen>
<kandidat>
 ole
</kandidat>
<besvarelse>
<html>
<h1>Flott</h1>
<p>webside</p>
</html>
</besvarelse>
</eksamen>
```

Vi ser at innholdet i elementet besvarelse, som er elevens besvarelse består av elementer som vil bli tolket og som vil føre til at dokumentet i sin helhet misforstås.

Vi kan løse dette ved å angi at en tekst ikke skal parses. Vi innfører en CDATA-seksjon. CDATA står for Character Data, i motsetning til PCDATA som står for Parsed Character Data. Meldinga til det programmet som leser fila er altså: Ikke bry deg om innholdet i denne teksten.

```
<?xml version="1.0"?>
<eksamen>
<kandidat>
 ole
</kandidat>
<besvarelse>
<![CDATA[
   <html>
   <h1>Flott</h1>
   <p>webside</p>
   </html>
]]>
</besvarelse>
</eksamen>
```

DEMO: http://www.ia.hiof.no/~borres/ml/xml/dok41.xml

Namespace

Det er mulig at namespace burde oversettes med navnerom på norsk, men jeg beholder det engelske ordet for ikke å skape mer forvirring enn nødvendig.

Namespaces er et fenomen som er konstruert for å løse opp i noen problemer som dukker opp ganske raskt når vi resonnerer litt omkring XML og hensikten med XML.

Det er viktig å ha klart for seg at:

- namespace er helt uavhengig av DTD. Et dokument kan ha en DTD og ikke noe (eksplisitt) namespace og omvendt.
- namespace ble opprinnelig ikke definert som en del av XML, men er tilpasset på et senere tidspunkt.

En av hensiktene med XML er jo at vi skal kunne utveksle og tolke data innen ulike organisasjoner og mellom systemer. Det er en umulig tanke at noen skulle kunne så og si ta patent på elementnavn. Det er grunn til å tro at elementer som f.eks. employee, address, customer, article, book finnes i noen tusen systemer rundt i verden og at de er definert forskjellig, med forskjellig betydning, forskjellige attributter og forskjellige underelementer. Vi må altså ha en måte å angi hvilken definisjon som gjelder i et konkret dokument eller datasett. Denne informasjonen er selvsagt oftest implisitt gitt ved at et program vet hvor datasettet det arbeider med kommer fra. Verre blir det når vi får datasett eller dokumenter som består av fragmenter fra flere dokumenttyper. Selv om vi implisitt vet hvor dataene kommer fra, må vi ha en måte å løse opp i navnekonflikter i datasettet. Vi må da ha en mulighet for å angi, i selve dokumentet, hvilket namespace, navnerom, et element hører til.

Anta følgende, ganske sære, eksempel:

```
<?xml version="1.0"?>
<bok>
   <markedsføring>
      <tittel>
      Verdens beste bok
      </tittel>
      <innhold>
      Solgt i uttallige eksemplarer
            </innhold>
   </markedsføring>

   <tittel>Ole Brum</tittel>
   <ingress>
   Om en teddybjørn og vennene hans
   </ingress>
   <innhold>
   En dag Kristoffer Robin...
   </innhold>
</bok>
```

Vi ser at elementene tittel og innhold inngår både i selve bokbeskrivelsen og i den informasjonen som skal brukes i markedsføring. De tilhører forskjellige namespaces. Vi må markere denne forskjellen på en eller annen måte. XML har syntaksregler for å angi namespace. Vi kan f.eks. skrive:

```
<?xml version="1.0"?>
<bok>
   <markedsføring>
      <marked:tittel>
      Verdens beste bok
      </marked:tittel>
      <marked:innhold>
      Solgt i uttallige eksemplarer
      </marked:innhold>
   </markedsføring>
   <tittel>
   Ole Brum
   </tittel>
   <ingress>
   Om en teddybjørn og vennene hans
   </ingress>
   <innhold>
   En dag Kristoffer Robin...
   </innhold>
</bok>
```

Fila overfor er velformet, men vi ser at angivelsen av namespace marked er av tvilsom verdi. Det kan være mange som har kommet på denne ideen. Vanligvis ønsker vi derfor å forsøke å gi et namespace et unikt navn. En måte å gjøre dette på er å angi en URN (Uniform Resource Name). Feks. kan vi skrive om fila slik:

```
<?xml version="1.0"?>
<bok>
   <markedsføring>
      <marked:tittel  xmlns:marked="http://www.forlaget.no/std/marketing.html">
      Verdens beste bok
      </marked:tittel>
      <marked:innhold  xmlns:marked="http://www.forlaget.no/std/marketing.html">
      Solgt i uttallige eksemplarer
      </marked:innhold>
   <!-- </marked> -->
   </markedsføring>

   <tittel>
   Ole Brum
   </tittel>
   <ingress>
   Om en teddybjørn og vennene hans
   </ingress>
   <innhold>
   En dag Kristoffer Robin...
   </innhold>
</bok>
```

Nå ser vi at for selve boka er det ikke angitt noe namespace. Vi kan godt gjøre dette ved å definere en default namespace, og samtidig definerer marked:

```
<?xml version="1.0"?>
<bok  xmlns="http://www.forlaget.no/std/boker.html"
      xmlns:marked="http://www.forlaget.no/std/marketing.html">
   <markedsføring>
      <marked:tittel>
      Verdens beste bok
      </marked:tittel>
      <marked:innhold>
      Solgt i uttallige eksemplarer
      </marked:innhold>
   </markedsføring>

   <tittel>
   Ole Brum
   </tittel>
   <ingress>
   Om en teddybjørn og vennene hans
   </ingress>
   <innhold>
   En dag Kristoffer Robin...
   </innhold>
</bok>
```

De bibliotekene som vi som programmerere benytter for å analysere XML-dokumenter har funksjoner for å identifisere elementer med angitte namespaces.

Entity

De tegnene som inngår i formateringen av XML er vanskelige å bruke dersom vi skriver dem rett fram. F.eks. vil det være problematisk å skrive <. Programmet som skal tolke XML-fila vil oppfatte dette som betynnelsen på et element. For å unngå dette er det definert et sett med såkalte entities.

En entity er en slags definisjon som vi kan bruke, og få det tolket som et spesielt tegn. Vi skriver **&navn;** dersom vi vil framstille tegnet som har navnet navn.

I XML er følgende entities predefinerte:

- < som <
- > som >
- & som &
- " som "
- ' som '

I tillegg til dette kan vi definere våre egne entiteter som en del av DTD. Vi kan skrive:

```
<?xml version="1.0"?>
<!DOCTYPE bok [<!ENTITY skribent "Børre Stenseth, HIØ">]>
<bok>
  <tittel>
    XML på alle fat
  </tittel>
  <forfatter>
    &skribent;
  </forfatter>
</bok>
```

Hvis vi ser på spesielle XML-språk, vil vi se at disse ofte har predefinert langt flere entities enn de 5 standard tegnene. F.eks. XHTML har definert entities som angitt i refreransene nedenfor.

Vi kan f.eks. skrive ♣ ♦ ♥ ♠ som ♣ ♦ ♥ ♠ og vi kan skrive € som €

Referanser og eksempler:

XML-sjekking på nettet

- Oversikt over valideringstjenester:
  www.oasis-open.org/cover/check-xml.html
- The Brown University Scholary Technology Group's XML Validation Form:
  www.stg.brown.edu/pub/xmlvalid/
  (Denne aksepterer ikke relative dtd angivelser)
- Multipage Validator: www.validator.ca/

Namespace

- Hos W3C: Namespaces in XML www.w3.org/TR/REC-xml-names/
- På norsk Navnerom i XML : www.net-blast.com/Webdesk/documents/w3.org-TR/REC/REC-xml-names-no-bkm.html

Entities

- Entities i XHTML: www.cookwood.com/html/extras/entities.html
- XHTML-definisjonen hos W3C: www.w3.org/TR/xhtml1/

Strukturer

Hva:

Noen grunnleggende resonnementer omkring strukturering av informasjon i XML-format

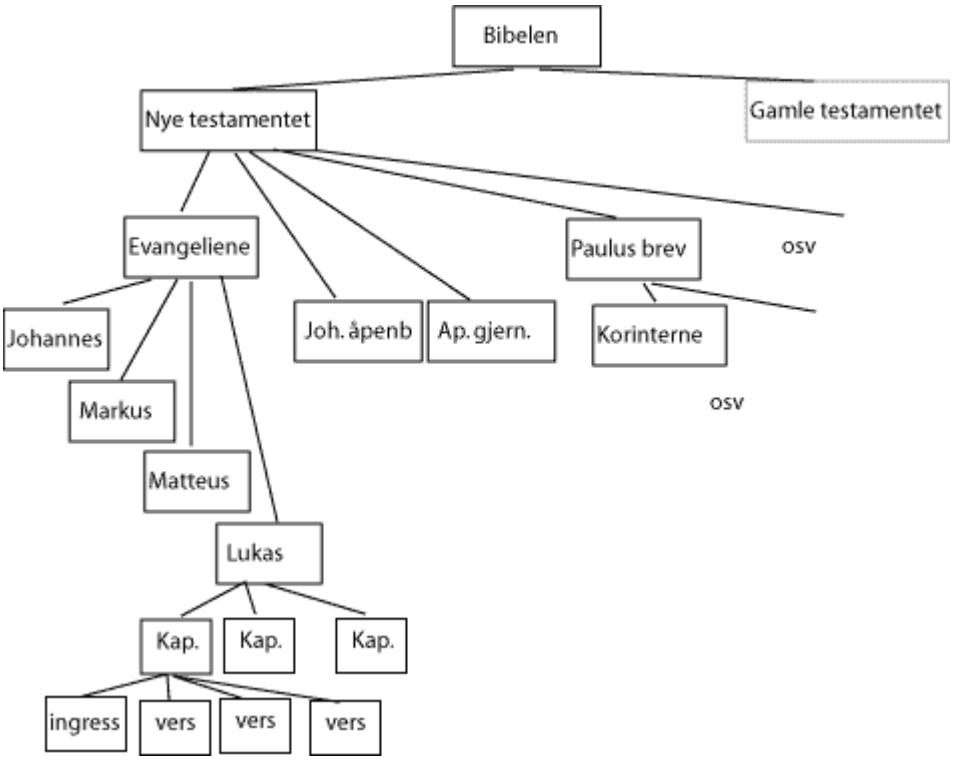
Ved planlegging av XML-dokumenter er det mange avveininger som må gjøres. Vi må vurdere hele trestrukturen og vi må ta stilling til hva som skal være elementer og hva som skal være attributter. Det er ikke noen enkle svar eller tommeregler som hjelper oss med dette. Hensikten med denne modulen er å resonnere litt omkring noen avveininger som kan være aktuelle.

Vi gjør dette ved hjelp av et par eksempler.

Bibelen

La oss tenke oss at vi ønsker å lage en XML-struktur av bibelen. Hensikten kan være at vi ønsker å lage et vevsted for raske oppslag eller at vi ønsker å lage utskrifter av vers, bøker eller andre utdrag. Altså et generelt og litt upresist formål.

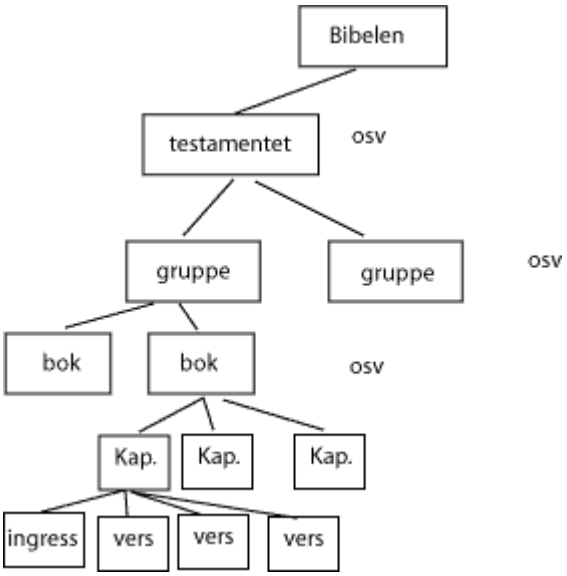
Hvis vi tar utgangspunkt i den strukturen som finnes i bibelen, finner vi noe slikt:



Vi kan lage en XML-struktur som avbilder dette direkte, skissert slik:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bibelen>
<gamle_testamente>
...
</gamle_testamente>
<nye_testamente>
<evangeliene>
<johannes>...</johannes>
<markus>...</markus>
<matteus>...</matteus>
<lukas>
<kap>
<vers>...</vers>
<vers>...</vers>
...
</kap>
</lukas>
<johannes_åenbaring>
</johannes_åenbaring>
osv...
</evangeliene>
</nye_testamente>
</bibelen>
```

Dette er neppe særlig lurt. Vi får en grisete dokumentbeskrivelse og den koden vi må skrive for å lete i og manipulere strukturen må forholde seg til en rekke "spesialtilfeller". Det første vi bør gjøre er å generalisere, det vil si finne en generell struktur som kan bære all informasjonen. Kanskje vi kan skissere det slik:



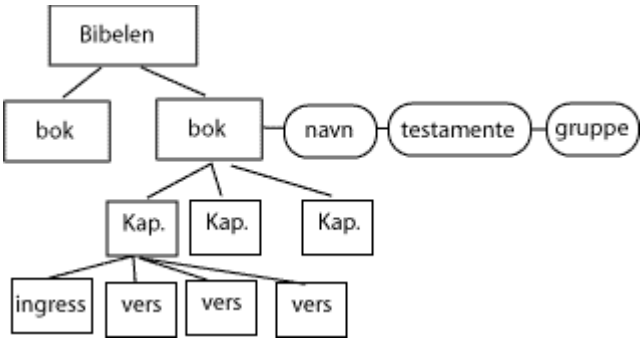
I tillegg til denne grovstrukturen må vi hekte opp de aktuelle navnene enten som elementer under de respektive elementene (testamente, grupp, bok) eller vi må angi navnet som egenskaper, attributter. Nummerering av kapitler og vers følger av elementets plass i barneflokken.

Element eller attributt? er en stadig tilbakevendende problemstilling. Det finnes neppe noe allment svar på dette, men jeg vil prøve ut et resonnement.

Det går i korthet ut på at jeg lokaliserer det "viktigste" elementet. I dette tilfellet velger jeg bok.

Argumenter for å velge bok er for det første at det er det øverste elementet som har en ens substruktur. Gruppe er tvilsom fordi ikke alle bøker er grupperte. Vi må da innføre kunstige grupper med kun en bok. For det andre tror jeg at bok vil være et naurlig fokus for avsøking og ekstrahering. Alt som er over dette valgte elementet, unntatt rotelementet, angir jeg som attributter.

Konsekvensen av dette blir:



En XML-skisse blir slik:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bibelen>
<bok navn="Lukas evangelium" testamente="NT" gruppe="evangeliene">
<kap>
<vers>
</vers>
</kap>
</bok>
</bibelen>
```

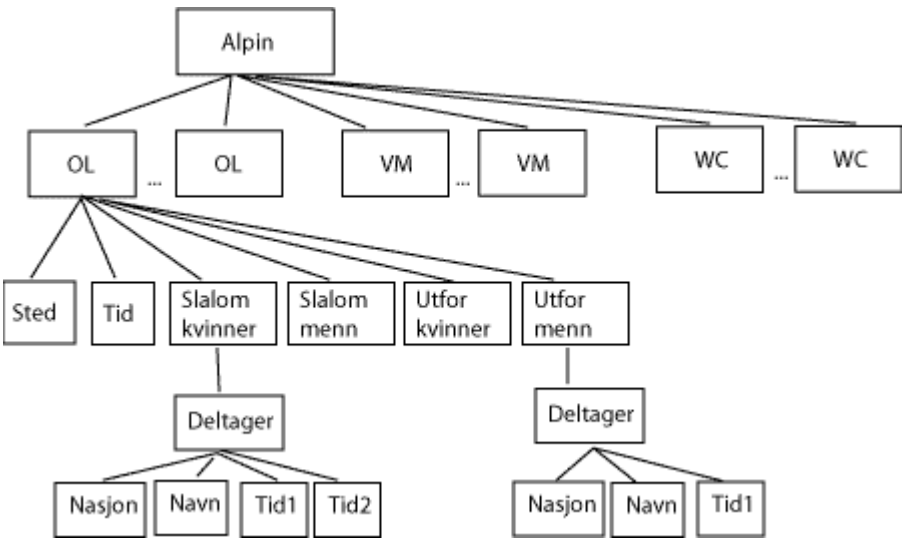
Hvis vi skal karakterisere denne øvelsen kan vi si at vi har "brettet" strukturen rundt det fokuserte elementet og plassert elementene over, untatt rotelementet, som attributter.

Nå er det ikke sikkert at vi stopper her. Det kan finnes argumenter for å innføre andre elementer og attributter av årsaker som henger sammen med den funksjonaliteten vi planlegger. En forsker på bibeltekster vil definitvt ha behov for innføre en rekke andre egenskaper ved bøkene.

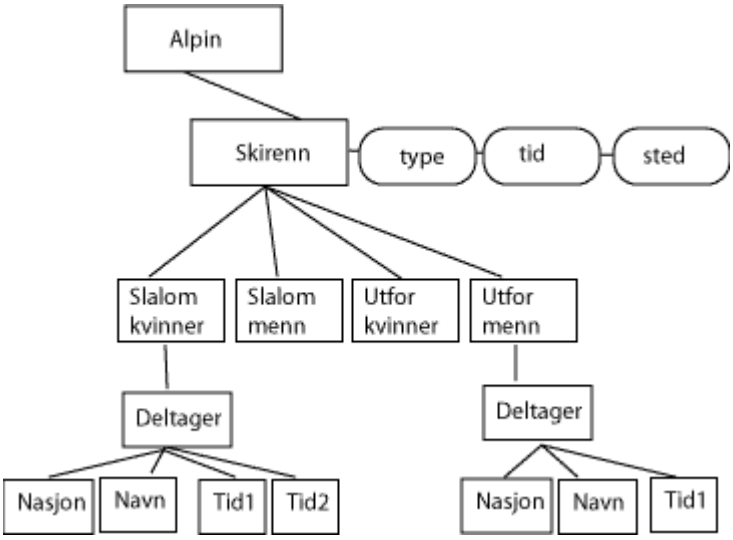
Det som er gjort ovenfor er ikke på noen måte noen oppskrift på hvordan en skal gå fram ved design av en XML-struktur. Det er bare ett av mange mulige resonnementer som kan prøves ut.

Alpint

La oss tenke oss at vi ønsker å lage en oversikt over resultater i internasjonale alpinrenn. Hvis vi betrakter problemet "ovenfra" vet vi at slike renn arrangeres i mange kategorier. Vi konsentrerer oss om Olymiader(OL), Verdensmesterskap(VM) og World Cup(WC). For enkelhets skyld nøyer vi oss med slalom og utfor. Vi kan da tenke oss en slik struktur, hvis vi som sagt ser det "ovenfra":



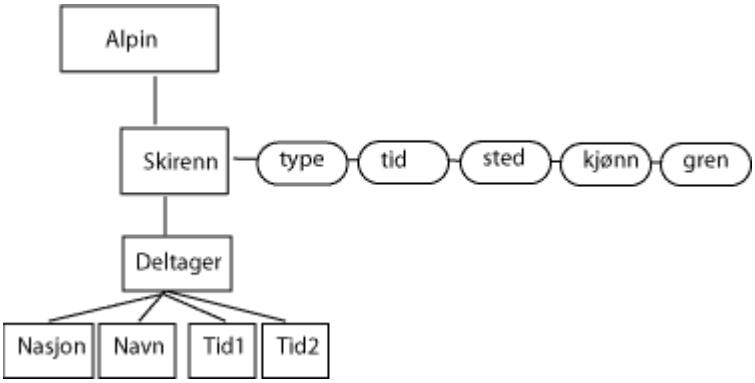
Igjen har vi en struktur som neppe bør avbildes direkte. Vi må søke fokus i materialet. Det resonnementet vi førte for bibelen er litt vanskeligere å få øye på. Vi må tro at siden det er ønskelig å sammenligne resultater fra alle de tre typene arrangementer, så er det selve arrangementet eller rennet som må stå i fokus. Vi kan da i hvert fall delvis gjennomføre resonnementet fra bibel-eksempelet og si at arrangemenetstypen (OL, VM, WC) blir en egenskap ved et skirenn. Det samme kan vi gjøre med nodene tid og sted. En "bretteøvelse" lik den vi gjorde i bibeleksempelet bringer oss til følgende struktur:



Det gir oss fire typer renn: slalom/menn, slalom/kvinner, utfor/menn og utfor/kvinner. Her kan vi tenke på flere forskjellige måter:

- Det er trolig ikke slik at vi skal sammenligne menn og kvinner i de programmene som skal bruke denne informsjonen. Vi kan kanskje skille de to helt fra hverandre og lage en fil for alpint-menn og en for alpint-kvinner.
- Strukturen for de to typene renn (slalom og utfor) er litt forskjellige siden slalom består av to omganger. Vi kan derfor kanskje skille slalom og utfor som separate barn under alpin.
- Vi kan se bort fra forskjellen i struktur mellom slalom og utfor og si at renntypen skal avgjøre om tid2 er signifikant eller ikke.

Hvis vi velger det siste alternativet og lar kvinner og menn opptre i samme fil, kan vi ende opp med følgende:



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<alpin>
   <skirenn type="WC" tid="2001" sted="Wengen" kjønn="menn" gren="utfor">
   <deltager>
      <navn>Kjuus</navn>
      <Nasjon>Norge</Nasjon>
      <tid1>130.012</tid1>
      <td2>na</td2>
   </deltager>
   ...
   </skirenn>
   ...
</alpin>
```

Nærmere ettertanke kan kanskje lede oss på andre spor ? Tenk gjennom noen alternativer.

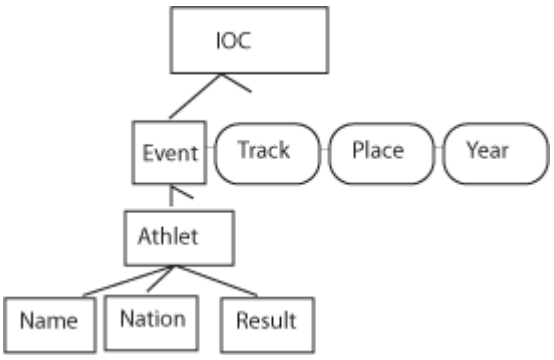
Olympiade

I modulen Noen datasett er det beskrevet en XML-fil som inneholder sprintresultater for de seneste sommerolympiadene. Denne fila er bygd opp slik:




```
<?xml version="1.0" encoding="ISO-8859-1"?>
<IOC>
  <OlympicGame place="Barcelona" year="1992">
    <event track="100m">
      <athlet>
        <name>Dennis Mitchell</name>
        <nation>USA</nation>
        <result>10.04</result>
      </athlet>
            ...
    </event>
    <event track="200m">
            ...
    </event>
        ...
 </OlympicGame>
 <OlympicGame place="Atlanta" year="1996">
    ...
 </OlympicGame>
    ...
<IOC>
```

Dersom vi skal følge resonnementet somer brukt under alpinekesempelet ovenfor vil vi få noe slikt.




```
<?xml version="1.0" encoding="ISO-8859-1"?>
<IOC>
  <event track="100m" place="Sidney"
           year="2000">
    <athlet>
      <name>Maurice Greene</name>
      <nation>USA</nation>
      <result>9.87</result>
    </athlet>
        ...
    </event>
    <event track="200m" place="Sidney"
             year="2000">
    ...
    </event>
    ...
<IOC>
```

Overgangen fra den ene strukturen til den andre kan vi enelt realisere med en transformasjon i XSLT. Dette er nærmere omtalt i eksempelet Olympiade. Transformasjonen kan se slik ut:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"
    doctype-system="olympreorg.dtd" encoding="ISO-8859-1"/>

<xsl:template match="/">
<IOC>
   <xsl:apply-templates select="/IOC/OlympicGame/event"/>
</IOC>
</xsl:template>

<xsl:template match="event">
    <xsl:element name="event">
      <xsl:attribute name="track">
         <xsl:value-of select="@track"/>
      </xsl:attribute>
      <xsl:attribute name="place">
         <xsl:value-of select="ancestor::OlympicGame/@place"/>
      </xsl:attribute>
      <xsl:attribute name="year">
         <xsl:value-of select="ancestor::OlympicGame/@year"/>
      </xsl:attribute>
      <xsl:apply-templates select="athlet"/>
   </xsl:element>
</xsl:template>

<xsl:template match="athlet">
 <xsl:copy-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

Referanser og eksempler:

Olympiadedata i modulen: Noen datasett

Mer om transformasjoner i modulen: XSLT

Eksempelet Olympiade inneholder flere transformasjoner av det aktuelle datasettet: Olympiade

Validering

Velformethet er en generell egenskap ved XML-filer, uavhengig av vilket XML-språk vi snakker om. Vi kan alltid sjekke om en XML-fil er velformet basert de generelle reglene for hvordan elementer skal settes opp osv.

Validering av en XML-fil forutsetter at vi har et spesielt XML-språk og at vi har beskrevet dette språket. Det er to muligheter for å lage slike beskrivelser

- Document Type definition (DTD). Dette er den tradisjonelle formen.
- Schema. Dette er en beskrivelsesform som er mer presis enn DTD. Schema har dessuten den egenskapen at de i seg selv er XML_filer og derfor kan behandles med XML-verktøy. Merk at det er flere Schema-versjoner i bruk. Jeg forholder meg til Schema fra W3C.

DTD

Document Type Definition

Hva:

DTD som formell beskrivelse av et XML-språk for å kunne validere et dokument, Document Type Definition

En DTD-fil beskriver et ML-språk. Det vil i korthet si hvilke elementer og attributter som inngår, hvilke elementer som kan være inkludert i andre elementer og hvilke elementer/attributter som er nødvendige og/eller mulige. Vi trenger en slik beskrivelse for å kunne **validere** et XML-dokument.

Skjema er et alternativ til DTD.

DTD-filer kan fort bli ganske kompliserte. Det er til stor hjelp å ha et verktøy som gjør det mulig å validere forløpende og det er til enda større hjelp å ha et verktøy som generer DTD filer for XML-eksempler. XMLSpy kan gjøre begge deler.

Selv om vi bruker et slikt verktøy eller ikke, så har vi bruk for å gjøre håndarbeid på DTD-fila. Nedenfor finner du litt enkel førstehjelp

Elementer

Et element defineres som en gruppe av subelementer eller subgrupper:

```
<!ELEMENT A (B,C)>

<!ELEMENT A (#PCDATA)>
<!ELEMENT A EMPTY>
<!ELEMENT A ANY>
```

EMPTY betyr at det ikke skal være noen barn til noden. ANY betyr at noden kan inneholde ingen eller flere noder av en hvilken som helst type som er dfinert i dokumentet. PCDATA står for Parsed Character DATA og er en terminologi hentet fra SGML. I praksis betyr dette tekst, i en eller annen form.

Grupper kan være sekvenser eller de kan være valgfrie

```
<!ELEMENT A (B)>                # B
<!ELEMENT A (B,C)>              B før C
<!ELEMENT A (B | C)>            B eller C
<!ELEMENT A (B, (C | D), E>     B før C eller D
<!ELEMENT A (B | C | (D,E))>    B eller C eller sekvensen D E
```

Grupper kan merkes som valgfrie (?), en-eller-flere (+), ingen-eller-noen (*).

```
<!ELEMENT A (B?, C)>
<!ELEMENT A (B, (C | D)+, E>
<!ELEMENT A (B,C)*>
```

Character data kan sidestilles med subgrupper:

```
<!ELEMENT A (#PCDATA | B | C)*>
```

Atributter

Elementer kan ha ingen eller flere atributter

```
<!ATTLIST A a CDATA #IMPLIED>
```

Flere atributt angivelser kan skrives i samme angivelse eller hver for seg:

```
<!ATTLIST A a CDATA #IMPLIED b CDATA #IMPLIED>
..
<!ATTLIST A a CDATA #IMPLIED>
<!ATTLIST A b CDATA #IMPLIED>
```

Atributter kan være obligatoriske (REQUIRED), valgfrie (IMPLIED)eller et valg mellom spesifiserte verdier (FIXED). Valgfrie atributter kan ha en default verdi, spesifiserte må ha en default verdi.

```
<!ATTLIST A a CDATA #IMPLIED>
<!ATTLIST A a CDATA "aaa">
<!ATTLIST A a CDATA #REQUIRED>
<!ATTLIST A a CDATA #FIXED "aaa">
```

Entities

Entities kan vi betrakte som enheter som kan defineres og plasseres inn i en XML-fil når den parses. F.eks. kan vi lage følgende XML-fil:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE bok SYSTEM "dok1.dtd">
<bok>
<tittel>
En bok
</tittel>
<forfatter>
&meg_selv;
</forfatter>
</bok>
```

Med følgende DTD:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT tittel (#PCDATA)>
<!ELEMENT bok (tittel,forfatter)>
<!ELEMENT forfatter (#PCDATA)>
<!ENTITY meg_selv  "Børre Stenseth, Høgskolen i Østfold">
```

Dette går bra i MSIE:

DEMO: http://www.ia.hiof.no/~borres/ml/dtd/dok1.xml

Dersom vi skal he det til å fungere i Mozilla, må vi plassere entiteten eksplisitt i selve XML-fila:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE bok SYSTEM "dok1.dtd"[<!ENTITY meg_selv "Børre Stenseth">]>
<bok>
<tittel>
En bok
</tittel>
<forfatter>
&meg_selv;
</forfatter>
</bok>
```

Dersom vi gjør begge deler, virker det i både MSIE og Mozilla:

DEMO: http://www.ia.hiof.no/~borres/ml/dtd/dok2.xml

Det fungerer altså som en replace, og kan typisk sammenlignes med en konstantdefinisjon i et programmeringsspåk som C. Vi kan definere eksterne entities og importere hele filer ved denne teknikken.

Standard entities

I XML er det definert følgende standard, eller predefinerte, entities:

- < for mindre enn, eller åpningsparentesen i elementer: <
- > for større enn, eller avslutningsparentesen i elementer: >
- & for &
- " for doble anførselstegn: "
- ' for enkel apostrof: '

Referanser og eksempler:

En kort beskrivelse: www.rpbourret.com/xml/xmldtd.htm

Eksempler på DTD'er:

- Beskrivelsen av HTML4:
  www.w3.org/TR/REC-html40/sgml/dtd.html
- Format for nyheter:
  www.xmlnews.org/docs/xmlnews-story.html

Schema

Hva:

Introduksjon av skjemaer, under arbeid

Schema er funksjonelt sett å betrakte som et alternativ til Document Type Definition, se DTD. DTD har flere svakheter. Det har lite presisjonsnivå når det gjelder å angi hvilke typer av data som er tillatt og det har lite fleksibilitet når det gjelder alternative muligheter i en XML-fil.

Disse svakhetene har ført til at det har dukket opp forskjellige varianter av alternative formater for dokumentvalidering. I denne modulen holder vi oss til W3C's Schema.

Det ser ut til at Schemaer er i ferd med å overta som hovedverktøy i dokumentvalidering. Årsaken til dette er både at Schema tillater større presisjon og fleksibilitet og at det er enklere å transformere og sammenligne Schemaer enn DTDer.

Et skjema fyller altså samme funksjon som en DTD. Vi kan validere XML-dokumenter mot skjemaer. La oss se på et lite eksempel med en DTD-fil og en Schema-fil.

Vi tar for oss en enkel bokliste. Starten på den er slik:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<booklist>
    <book isbn="0-596-00292-0" pages="600">
        <title>XML in a nutshell</title>
        <course>Programmering for web</course>
        <category>XML</category>
        <author>Elliot Rusty Harold and W. Scott Means</author>
        <publisher>OReilly</publisher>
        <year>2002</year>
        <comment>Ganske mer omfattende. God bok </comment>
    </book>
    <book isbn="0-201-76990-5" pages="200">
        <title>The web wizards guide to XML</title>
        <course>Programmering for web</course>
        ....
```

En enkel DTD-fil for å validere filer av denne typen. Merk at vi kan bedre presisjonsnivået i denne DTD'en betraktelig, men hensikten med dette eksemplete er først og fremst Schema-formatet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT author (#PCDATA)>
<!ELEMENT book (title, course, category, author, publisher, year, comment)>
<!ATTLIST book
    isbn CDATA #REQUIRED
    pages CDATA #REQUIRED
>
<!ELEMENT booklist (book+)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT course (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

En Schema-fil med samme hensikt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="author" type="xs:string"/>
    <xs:element name="book">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="title"/>
                <xs:element ref="course"/>
                <xs:element ref="category"/>
                <xs:element ref="author"/>
                <xs:element ref="publisher"/>
                <xs:element ref="year"/>
                <xs:element ref="comment"/>
            </xs:sequence>
            <xs:attribute name="isbn" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string"/>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="pages" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:short"/>
                </xs:simpleType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
    <xs:element name="booklist">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="book" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="category">
        <xs:simpleType>
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
    </xs:element>
    <xs:element name="comment" type="xs:string"/>
    <xs:element name="course" type="xs:string"/>
    <xs:element name="publisher" type="xs:string"/>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="year">
        <xs:simpleType>
            <xs:restriction base="xs:short"/>
        </xs:simpleType>
    </xs:element>
</xs:schema>
```

Vi ser for det første at Schema-fila har et ekte XML-format, og er således et "ekte medlem av XML-familien".

Vi ser at Schemaet inneholder struktur informasjon på samme måte som en DTD. F.eks.:

```
<xs:element name="book">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="title"/>
            <xs:element ref="course"/>
            <xs:element ref="category"/>
            <xs:element ref="author"/>
            <xs:element ref="publisher"/>
            <xs:element ref="year"/>
            <xs:element ref="comment"/>
        </xs:sequence>
        ...
     </xs:complexType>
</xs:element>
```

uttrykker det samme som vi finner i denne linja i DTD-fila:

```
<!ELEMENT book (title, course, category, author, publisher, year, comment)>
```

I begge formatene har vi mulighet for sette alternativer slik at vi ikke har en låst sekvens med obligatoriske elementer.

Selv om Schemaet ovenfor er svært enkelt, ser vi at vi har et langt bedre utbygd apparat for å angi hva slags dataformat vi forlanger på attributter og elementinnhold. F.eks. ser vi at sidetallet skal angis som et heltall:

```
<xs:attribute name="pages" use="required">
    <xs:simpleType>
        <xs:restriction base="xs:short"/>
    </xs:simpleType>
</xs:attribute>
```

Følgende linje, fra et annet Schema, angir at attributten location er obligatorisk og skal ha format av en URI:

```
<xs:attribute name="location" type="xs:anyURI" use="required"/>
```

Referanser og eksempler:

- Schema: www.w3.org/XML/Schema
- Filene fra bok-eksempelet: bokfiler.zip

Formattering av XML

[CSS][XSL-FO]

XML er et verktøy for å stukturere data. De fleste XML-språk inneholder ikke informasjon om hvordan dataene skal framstilles av programmer. I en ideell verden er dette skillet absolutt.

XHTML er et XML-språk som rktig nok går ganske langt i å gjennomføre dette skillet men det henger fortsatt litt formateringsinformasjon i språket. F.eks. cellspacing og cellpadding i table-elementer.

Vi ser at når en nettleser skal framstille en vilkårlig XML-fil så treffer de ulike nettleserne et valg om hvordan de skal vise fram fila. Vi har ingen kontroll over dette dersom vi ikke bidrar med formateringsinstruksjoner i form av et stilsett.

Det er to aktuelle formateringsmekanismer: CSS og XSL-FO. Både CSS og XSL-FO er nærmere beskrevet i egne moduler.

CSS

Cascading Style Sheets (CSS) som er det vi bruker når vi skal gi format til XML-filer i nettlesere. Vi vet dessuten at nettleserne har et standard CSS-stilark som de anvender dersom den fila den skal vise fram er HTML eller XHTML. Det stilsettet vi bidrar med, komplementerer eller overstyrer egenskaper i dette standard stilsettet. Men vi kan anvende CSS på andre XML-språk enn HTML og XHTML.

Det skjer stadig utvikling av CSS og det er beskrevet et medie-begrep som gjør det mulig å skifte stilsett avhengig av framstillingsmåte. For at dette skal fungere smertefritt må det programmet som skal framstille dokumentet kople til den riktige mediestilen. I øyeblikket kjenner Firefox, Internett Explorer og Opera mediene **screen** og **print**. Det vil du se dersom du ber om en print-preview (eller print) av den siden du leser nå, menyer og diverse fjernes. Opera kjenner i tillegg mediet **projection**, og kopler det til når du trykker F11 (full skjerm). Dette er drøftet nærmere i modulen Media og i eksempelet Presentasjon

CSS har vært lite egnet til å framstille dokumenter som skal vises eller printes som formaterte sider. F.eks. har det vært problematisk å ta en vilkålig vellformet XHTM-side og få den ut på printer som PDF. Versjon 2.1 introduserer regelen **@page** som åpner for sideformatering.

Vi må forvente at dette implementeres når CSS 2.1 blir en anbefalt "standard". Det finnes, i hvert fall, ett interessant verkøy som bruker dette: Prince. Prince lager PDF fra XHTML på en robust og fleksibel måte. Prince kan lastes ned og kjøres fritt på din egen harddisk. Det etterlater et merke i øvre høyre hjørne på første side når du ser dokumentet i en nettleser, men dette forsvinner når du printer.

Bruk av Prince kan se slik ut:

Mer om dette på modulen Sider i CSS

XSL-FO

XSL-FO. Dette er formateringsangivelser som primært er rettet mot andre programmer enn nettlesere. XSL_Fo's styrke er sideadministrasjon og den typiske anvendelsen er å preparere for utskrift. Bruk av XSL-FO er litt annerledes enn bruk av CSS. Den vanligste bruken er å kjøre et XML-dokument gjennom to transformasjoner, typisk slik:

Der T1 er en XSLT-transformasjon og T2 er programmet fop.

Mer om dette på modulen XSL-FO

Cascading Style Sheet

Hva:

En kort innføring i de mest sentrale delene av CSS, Cascading Style Sheets. Stilark eller stilsett

I modulen Markup er det forsøkt å legge vekt på forskjellen mellom struktur i et dokument og dokumentets utseende. Når vi betrakter stilark seter vi fokus på utseendet. Stilark lar seg anvende på HTML- og XML-dokumenter.

CSS forekommer i tre versjoner. CSS1 og CSS2 er begge anbefalinger fra W3C. CSS3 er under utarbeidelse.

CSS holdes ofte opp mot XSL_FO, XSL-Formattering. Noen betrakter disse som konkurrerende teknologier og noen ser dem som supplerende. Jeg har ingen erfaring med XSL-FO, men det er vel slik at dette er en mer ambisiøs teknologi som dessuten har en syntaks og en definisjon som faller bedre på plass i XML-familien. På den annen side er CSS vesentlig enklere, i alle fall har CSS en slakkere læringskurve, enkle ting er intuitivt enkelt.

I de fleste eksemplene nedenfor bruker jeg minimalistisk HTML.

Cascading

Hva i all verden betyr det ?

Utgangspunktet er at det er mulig å ha flere konkurerende stilsett, siden stilsett kan koples til på forskjellig måte, se nedenfor. Det er dessuten slik at det er tre roller som kan ha meninger om et dokuments utseende:

- **Forfatteren** som skriver og angir et stilsett
- **Leseren** som kan ha spesielle ønsker som hun forsøker å realisere ved å påvirke framvisningsprogrammet, f.eks. et ønske om større tekst
- **Framvisningsprogammet**, f.eks. en nettleser. En nettleser må treffe noen beslutninger basert på det mediet dokumentet skal vises på. Det må dessuten komme opp med en default stil når ingen av de andre to har uttrykt noen mening.

Disse tre kan ha overlappende ønsker. Derav navnet kaskade. Disse tre ønskene kan komme i konflikt og det må være noen regler for å løse opp i dette. Som hovedregel er det forfatteren som har kommandoen og legger premissene for de andre.

Kopling

Vi vet at når vi lager en enkel HTML-fil som f.eks.:

```
<html>
   <head>
      <title>dok1</title>
   </head>
   <body>
      <h1>Velkommen</h1>
      <p>Dette er en demoside</p>
      <p>
      <a href="javascript:history.back()">Tilbake</a>
      </p>
   </body>
</html>
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/dok1.html

så gis den et utseende som er bestemt av nettleseren. Dokumentet har fått en default stil. Vi vet videre at vi kan manipulere dette i noen grad ved å gjøre noen valg i nettleseren. F.eks. kan vi endre skriftstørrelse, farge på linker etc. Nedenfor skal vi gjøre fire forsøk på å sette ulike farger på overskriften ved hjelp av stilark.

1 stil på stedet

Vi skriver stilen direkte inn i h1-elementet:

```
<html>
   <head>
      <title>dok1</title>
   </head>
   <body>
      <h1 style="color: red">
      Velkommen
      </h1>
      <p>Dette er en demoside</p>
      <p>
      <a href="javascript:history.back()">Tilbake</a>
      </p>
   </body>
</html>
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/dok11.html

2 stilark i head-elementet

Vi skriver et stilark inn i head-elementet. <style> </style>

```
<html>
   <head>
      <title>dok1</title>
      <style type="text/css">
      h1 { color: green }
      </style>
   </head>
   <body>
      <h1>
      Velkommen
      </h1>
      <p>Dette er en demoside</p>
      <p>
      <a href="javascript:history.back()">Tilbake</a>
      </p>
   </body>
</html>
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/dok12.html

3 importert stilark

Vi importerer et stilark fra en egen fil,url:

```
<html>
   <head>
      <title>dok1</title>
      <style type="text/css">
      @import url(stil13.css);
      </style>
   </head>
   <body>
      <h1>
      Velkommen
      </h1>
      <p>Dette er en demoside</p>
      <p>
      <a href="javascript:history.back()">Tilbake</a>
      </p>
   </body>
</html>
```

der stil13.css ser slik ut

```
h1{color:blue}
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/dok13.html

4 link til et stilark

Vi setter opp en link til et stilark på en egen fil,url:

```
<html>
   <head>
      <title>dok1</title>
      <link rel=stylesheet
      type="text/css"
      href="stil14.css">
   </head>
   <body>
      <h1>
      Velkommen
      </h1>
      <p>Dette er en demoside</p>
   </body>
</html>
```

der stil14.css ser slik ut

```
h1{color:yellow}
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/dok14.html

Vi kan blande disse mekanismene friskt og fort komme ut i en litt forvirret situasjon som framfor alt er vond å vedlikeholde. For de fleste vanlige publikasjonsformål er det min erfaring at den siste versjonen med å linke til et stilsett er å foretrekke. Samme stilsett kan brukes for mange sider, det trenger ikke lastes mer enn en gang for flere sider i et vevsted og det er bare ett sted å gjør endringer dersom vi ønsker å endre utseende på en gruppe dokumenter. Det forhindrer selvsagt ikke at vi kan overstyre en standard stil ved å benytte stilangivelse i et element dersom det er spesielle ting vi er ut etter.

Stilsettet som jeg bruker for dette materialet og i litt forskjellige varianter i en del andre vevsteder ser slik ut: Børres stilsett. (Det kan se ut som det trengs litt rydding og forenkling.)

CSS gir selvsagt langt flere muligheter enn å endre fargen på overskrifter. Denne modulen har ikke som ambisjon å dekke alle disse mulighetene. Vi skal konsentrere oss om noen av de mest sentrale mekanismene, valgt ut fra et praktisk publiseringsformål.

Identifikasjon av elementer

Det er enkelt sagt følgende måter å kople stilbeskrivelser til elementer på:

- Ved angivelse av element-type
- Ved angivelse av klasse
- Ved angivelse av pseudoklasse
- Ved identifikasjon, id
- Kombinasjoner av disse

Ved element-type

Dette er den enkle, rett-fram metoden. Vi setter f.eks. stilen på alle overskrifter på nivå 4 ved å skrive:

```
  h4{color:red}
```

Ved klasse

Vi kan angi stil for en klasse. Det vil si at vi beskriver en stil som skal gjelde for alle elemnter som er av den klasse. Vi skriver f.eks.:

```
.redfont{color:red}
```

Dette betyr at alle elementer som har verdien "redfont" til attributten class. F.eks.:

```
<p class="redfont"> tekst</p>
<div class="redfont"> mere tekst </div>
```

Ved pseudoklasse

Vi kan angi stil for det W3.org kaller pseudo klasser. Syntaksen er slik:

```
selector:pseudo-class {property: value}
```

I praksis er det pseudoklasser knyttet til anker-elementer som er aktuelle. Vanligvis finner vi noe som ligner på dette i et stilsett:

```
a:link {color: #blue}
a:visited {color: #green}
a:hover {color: #red}
a:active {color: #black}
```

Merk at rekkefølgen av en eller annen grunn er viktig for å få den ønskede effekten.

Ved identifikasjon

Vi kan angi stil for et bestemt element. Vi skriver f.eks.:

```
#the_red_par{color:red}
```

Dette betyr at elementet som har verdien "the_red_par" til attributten id. F.eks.:

```
<p id="the_red_par"> hilited </p>
```

En id-verdi er forutsatt å være entydig på en side. I en rekke andre sammenhenger, som DOM-programmering, er effekten av like id'er at den først forekomsten brukes og de andre ignoreres. Anvendelsen av stilsett-definisjoner synes ikke å være like konsekvent i nettleserne.

Kombinasjoner

Vi kan kombinere og vi kan liste flere elementidentifikasjoner i samme stilangivelse. Vi skriver f.eks.:

```
h1,h2,h3{color:red}
h2,h3{font-size:18px}
h1,.bigTxt{font-size:24px}
p{margin-top:10px}
p.luftig{margin:20px}
.mlcode,.pcode{color:blue}
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/ex-ident.html

Noen egenskaper

Nedenfor finner du, uten nærmere forklaring, en opplisting av noen av egenskapene du kan bruke når du lager stilsett. Egenskapene er i henhold til CSS1. Hensikten er bare å minne om mulighetene. Slå opp detaljer hos W3C.

Fonter

```
font-family    : proritert liste av fonter
font-style     : normal |italic | oblique
font-variant   : normal | small-caps
font-weight    : normal | bold | bolder |lighter |100 | 200 |... | 900
font-size      : xx-small | x-small | small | medium |
                 large | x-large | xx-large | smaller | larger |
                 % | absolutt størrelse
font           : alle font-egenskaper kan settes i en definisjon
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/ex-font.html

Tekst

```
color           : farge | #hex
word-spacing    : normal | absolutt størrelse
letter-spacing  : normal | absolutt størrelse
text-decoration : none | underline | overline | line-through | blink
text-transform  : none | capitalize | uppercase | lowercase
text-align      : left | right | center | justify
text-indent     : absolutt størrelse | %
line-height     : normal | faktor | absolutt størrelse | %
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/ex-text.html

Bakgrunnsbilder

```
background-color      : farge | #hex | transparent
background-image      : url | none
background-repeat     : repeat | repeat-x | repeat-y | no-repeat
background-attachment : scroll | fixed
background-position   : top left | top center | top right |
                        center left | center center | center right |
                        bottom left | bottom center | bottom right |
                        x-% y-% | x-pos y-pos
background            : alle egenskapene kan settes i en definisjon
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/ex-back.html

Lister

```
list-style-type     : none | disc | circle | square | decimal |
                      decimal-leading-zero | lower-roman | upper-roman |
                      lower-alpha | upper-alpha | lower-greek | lower-latin |
                      upper-latin | hebrew | armenian | georgian |
                      cjk-ideographic |hiragana | katakana | hiragana-iroha |
                      katakana-iroha
list-style-image    : url | none
```

```
list-style-position : inside | outside
list-style          : alle egenskapene kan settes i en definisjon
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/ex-list.html

Border/margin/padding

```
margin-top
margin-right
margin-bottom
margin-left          : auto | absolutt størrelse | %
margin               : margin-top | margin-right | margin-bottom | margin-left

padding-top
padding-right
padding-bottom
padding-left         : absolutt størrelse | %
padding              : padding-top | padding-right | padding-bottom | padding-left

border-top-width
border-right-width
border-bottom-width
border-left-width
border-width         : thin | medium | thick | absolutt størrelse

border-top-style
border-right-style
border-bottom-style
border-left-style
border-style         : none | hidden | dotted | dashed | solid | double |
                       groove | ridge | inset | outset

border-top-color
border-right-color
border-bottom-color
border-left-color    : farge | #hex
border-color         : farge | #hex

border-top           : border-top-width | border-style | border-color
border-right         : border-right-width | border-style | border-color
border-bottom        : border-bottom-width | border-style | border-color
border-left          : border-left-width | border-style | border-color

border               : border-width | border-style | border-color

width
height               : auto |absolutt størrelse | %
line-height          : normal | antall | absolutt størrelse | %


display              : none | inline | block | list-item | run-in |
                       compact | marker | table | inline-table |
                       table-row-group | table-header-group |
                       table-footer-group |
                       table-row | table-column-group | table-column |
                       table-cell | table-caption

cursor                : url | auto | crosshair | default | pointer | move |
                        e-resize | ne-resize | nw-resize | n-resize |
                        se-resize |sw-resize |
                        s-resize | w-resize | text | wait | help
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/ex-border.html

Posisjonering

```
float               : left | right | none
clear               : left | right | both | none
position            : static | relative | absolute | fixed | inherit
```

DEMO: http://www.ia.hiof.no/~borres/ml/css/ex-box.html

DEMO: http://www.ia.hiof.no/~borres/ml/css/position/posfixed.html

Lengdeenheter

```
px    - pixel
em    - høyden av elementets font
ex    - høyden ab bokstaven x

in    - inch, tomme
cm
mm
pt    - point, 1pt=1/72 inch
pc    - pica,  1pc=12 pt

%
```

Dynamisk

Vi kan endre style-elementer dynamisk ved hjelp av javascript. Se position/mover.html. Dette har ikke vært (er ikke ?) helt problemfritt. Det ser ut til at følgende virker i de siste versjonene av Opera, FireFox og IE (på Windows):

```
im.style.left=nextx+'px';
im.style.top=nexty+'px';
im.style.display='block';
```

Altså: element.style.name=value

Alternativ formulering i IE:

```
im.style.left.setAttribute("12px")
```

eller i Mozilla/Opera, f.eks:

```
stylevalue='z-index:'+ix+';position:absolute;left:'+nextx+'px;top:'+nexty+'px';
im.setAttribute('style',stylevalue);
```

Se også eksempelet Picasso der vi bytter stilsett ved hjelp av Javascript.

I XML

Det er ikke noe spesielt annerledes å definere stilsett for en generell XML-fil enn det er for HTML. Den eneste forskjellen er at vi selv definerer elementene. Det har blandt annet den konsekvensen at nettleseren ikke har noe godt default stilark å stille opp med. I beste fall kan den lage noe som er helt generelt. MSExplorer og Mozilla gjør det slik, mens Opera og Navigator ignorerer XML-taggene og framstiller teksten som en sammenhengende streng. Vi prøver oss med et enkelt eksempel.

XML	CSS

`<?xml version="1.0" encoding="ISO-8859-1"?>`
`<?xml-stylesheet type="text/css" href="stil3.css"?>`
`<resultater>`
`  <deltager>`
`  <navn>Ole Brumm</navn>`
`  <tid>24.7</tid>`
`  </deltager>`
`  <deltager>`
`  <navn>Nasse Nøff</navn>`
`  <tid>24.8</tid>`
`  </deltager>`
`  <deltager>`
`  <navn>Tigergutt</navn>`
`  <tid>37.5</tid>`
`  </deltager>`
`</resultater>`
	`navn{ font-size:14px;`
	`  font-weight: bold;`
	`  display:block}`
	
	`tid{ font-size:14px;`
	`  font-weight: normal}`

Se resultatet: dok3.xml Merk at display:block er nødvendig for å få linjeskift.	

Fordelen med å ha HTML som format ved framstilling framfor XML, er at vi får en rekke stilegenskaper gratis. Det er dessuten et problem å få framstilt attributter som ofte er en viktig del av datainnholdet i en XML-fil. XSL-transformasjoner er det naturlige svaret på å bringe generelle XML-filer over på HTML-format. Se modulen XSL og spesielt eksempelet Olympiade for en serie med transformasjoner mot samme datagrunnlag.

Referanser og eksempler:

- Web Design Group, CSS: www.htmlhelp.com/reference/css/

Mediavalg i CSS

Hva:

Stilsett for forskjellige media

CSS versjon 2 introduserer mediaspesifikasjon som en av flere interessante muligheter. Utgangspunktet er at de sidene vi produserer skal kunne fungere på ulike media. W3C foreslår følgende mediadefinisjoner og forutser at det vil bli behov for flere:

all	generell, brukes til alle media
aural	stemme syntesizere
braille	enheter som gir blindeskrift
embossed	enheter som gir sideorientert blindeskrift
handheld	små, håndholdte enheter, typisk med liten "skjerm"
print	sideorienterte enheter
projection	projiserte presentasjoner
screen	"normale" dataskjermer
tty	enkle, karakterorienterte enheter
tv	lav oppløsning, begrenset scroll, tilgang på lyd

Hva betyr så dette i praksis. Det vi er ute etter er selvsagt at vi som forfattere, eller innholdsgeneratorer, ønsker at det vi lager skal fungere best mulig i det mediet leseren velger å bruke. Dette kan vi selvsagt gjøre ved å generere sidene spesielt for ulike medier, men det ville i mange sammenhenger være en fordel om vi kunne lage en side som så og si tilpasset seg mediet. Ofte vet vi ikke på forhånd hvilket medium siden vil bli framstilt i.

Nettlesere på vanlige PC'er er en av de programmene som skal kunne handtere deler av dette. Det viser seg at implementasjonen av media-tilpassing er i varetatt på litt forskjellig måte. Vi kan bruke et eksempel til å studere dette nærmere.

Vi tar for oss en HTML-fil med de 10 første av Shakespeares sonetter. (De som vi se alle 154 kan se på eksempelet Sonetter. Fila, mediademo.html, ser slik ut, forkortet:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>Mediademo</title>
  <link rel="stylesheet" href="shakeprintstil.css">
 </head>
 <body>
  <h1>Shakespeares sonetter</h1>

   <h2>I.</h2>
   <div class="sonett">
   From fairest creatures we desire increase, <br/>
   That thereby beauty's rose might never die, <br/>
   But as the riper should by time decease, <br/>
   His tender heir might bear his memory: <br/>
   But thou, contracted to thine own bright eyes, <br/>
   Feed'st thy light's flame with self-substantial fuel, <br/>
   Making a famine where abundance lies, <br/>
   Thyself thy foe, to thy sweet self too cruel. <br/>
   Thou that art now the world's fresh ornament <br/>
   And only herald to the gaudy spring, <br/>
   Within thine own bud buriest thy content <br/>
   And, tender churl, mak'st waste in niggarding. <br/>
   Pity the world, or else this glutton be, <br/>
   To eat the world's due, by the grave and thee.<br/>
   </div>
   ...
</html>
```

Som det gå fram av koden er det koplet til et stilsett, shakeprintstil.css. Dette stilsettet definerer stiler for tre forskjellige medier: **screen**, **print** og **projection**. Stilsettet er i sin helhet slik:

```
/* Stylesheet for changing media
   Børre Stenseth, nov 2003
*/

@media screen{
   h1{font-size: 24px; font-weight: bold; color:black;
     }
   h2{font-size: 20px; font-weight: bold; color:red;
      margin-left:100px;
     }
   div.sonett{ font-size: 14px; margin-top:2px;
     }
   div.sonett:first-letter { font-size: 200%;
                             font-weight: bold;
                             padding-right:5px;
                             color:black
     }
}


@media print{
h1{font-size: 24px; font-weight: bold; color:blue;}

h2{   font-size: 20px; font-weight: bold; color:gray;
    margin-left:100px;
    page-break-before:always}
div.sonett{ font-size: 14px; padding-top:20px;}
div.sonett:first-letter { font-size: 200%;
                          font-weight: bold;
                          padding-right:10px;
                           color:blue }
}

@media projection{
   h1{font-size: 36px; font-weight:bold; color:blue;
      margin-left:100px; margin-top:100px
      }

   h2{font-size: 12px; font-weight: bold; color:black;
       margin-left:0px; page-break-before:always
     }
   div.sonett{ font-size: 30px;
               margin-left:100px; margin-top:100px
     }
   div.sonett:first-letter { font-size: 200%;
                             font-weight: bold;
                             padding-right:10px;
                             color:blue
     }
}
```

Stilsettet er altså forberedt for å ta signaler om mediaskifte. Hvis vi skal se på HTML-fila i en nettleser så er vi avhengige av at nettleseren kan gjenkjenne og reagere på skifte i medium, eller med andre ord kan vi provosere fram en situasjon som tolkes som mediaskifte. I de fleste nettlesere vil vi kunne se print-varianten når vi printer eller når vi ber om å få se print-preview.

DEMO: http://www.ia.hiof.no/~borres/ml/cssmedia/mediademo.html

Eller du kan forsøke å be om File - Print Preview av den siden du leser nå.

Opera versjon 7+ har i tillegg til screen og print implementert projection slik at dette "skrus på" når vi trykker F11 (full screen). Vi kan da bla fra side til side med PgUp og PgDn.

DEMO: http://www.ia.hiof.no/~borres/ml/cssmedia/mediademo.html

Mediavalg er interessant fordi vi kan vedlikeholde en fil for mange formål. Vi kan for eksemple tenke oss at vi har en artikkel som skal kunne leses på skjermen (screen) og vises i et foredrag på storskjerm (projection), samt at den skal kunne printes med kontrollerbare sideskift (print). Det siste er interessant fordi en slik printstil også vil fungere når vi printer til en fil, slik som vi kan gjøre når vi lager pdf. Det betyr at vi kan kontrollere sideskift, i tillegg til all annen layout.

Referanser og eksempler:

www.w3.org/TR/CSS2/

Du kan hente ned de to involverte filene, sonettene og stilsettet, her: tofiler.zip

Eksempelet Presentasjon beskriver et forsøk på å få til en screen/print/projection løsning i alle de tre nettleserne (IE, FireFox, Opera)

Sider i CSS

Hva:

Sidebegrepet i CSS og PDF

Versjon 2.1 av CSS annonserer sidebegret, regelen @page. Dette gjør det mulig for oss å lage sidelayout, med sidenummerering, header, footer, høyre og venstre oppslag osv. Kort sagt vi kan kontrollere utskrift på en helt annen måte enn tidligere. Et område der dette er intressant er produskjon av PDF. Vi holder oss til dette i denne modulen.

Mens vi venter på at nettleserne skal fikse konvertering av XHTML til PDF med sidebegrep, kan vi bruke programmet Prince. Prince er åpent tilgjengelig i den forstand at vi kan laste ned en versjon og bruke den fritt på vår egen maskin. Som en del andre programmer med en slik begrenset fri lisens, setter Prince igjen sitt bumerke øverst til høyre på første side. Dette forsvinner når vi printer.

Eksempel 1

Vi ser på denne siden som eksempel. Slik du betrakter den nå er den styrt av et stilsett som beskriver utseendet for **@screen**. Dersom du ber nettleseren vise en print-preview, eller du ber om en print, vil du se at siden ser annerledes ut. Nettleseren kopler inn stiler som jeg har beskrevet med regelen **@print**. Forskjellen er i grove trekk at jeg har tatt vekk (display:none) menyene og banneret og jeg har lagt til en enkel footer. Jeg har dessuten lagt til sideskift (page-break-after:allways) for headinger av typen h1. Dette er omtrent så lang jeg kommer i bruk av tradisjonell CSS. Du kan se mitt stilsett som gjelder for denne siden her: stiler (jada, jeg vet jeg burde rydde litt)

Nå ønsker jeg å tilføre stiler for bedre side kontroll. Jeg lager et stilsett som ser slik ut:

```
@page { size: A4;
        margin: 60pt 30pt 40pt 30pt;
        @top-left {
            content:url(printlogo.gif);
          border-bottom-style:solid;
          border-bottom-width:thin;
        }
        @top-right {
            content:"Markup og Web";
            font-size:24px;
          border-bottom-style:solid;
          border-bottom-width:thin;
        }
        @bottom-right {
          content: counter(page);
            font-style: italic
        }
}
```

Så gjør jeg følgende, enten ved GUI'en til prince eller via commandolinja:

DEMO: http://www.ia.hiof.no/~borres/ml/paging/p-paging1.pdf

Ikke særlig imponerende, men vi har i hvertfall greidd å sette opp A4 med marger, og vi har greidd å legge til en header og sidenummerering som footer.

Det er imidlertid en ting som er verdt å merke seg: Vår definisjon av @print legges til grunn. Det kan vi se siden menyer og annet er fjernet. Dette har den konsekvensen at vi kan preparere for så pen som mulig XHTML-utskrift med "tradisjonell" CSS og så kan vi legge til effekter ved hjelp av @page dersom vi ønsker pdf.

Vi forsøker å bygge ut stilsettet vårt litt:

```
@page { size: A4;
        margin: 60pt 30pt 40pt 30pt;
        @top-left {
            content:url(printlogo.gif);
          border-bottom-style:solid;
          border-bottom-width:thin;
        }
        @top-right {
            content:"Markup og Web";
            font-size:24px;
          border-bottom-style:solid;
          border-bottom-width:thin;
        }
        @bottom-right {
          content: counter(page);
            font-style: italic
        }
}
table,fieldset,
.javascriptcode,.pythoncode,.javascriptcode,.javacode,.mlcode
{page-break-inside: avoid}

fieldset{border-left:1px solid gray;
         padding:10px;
         padding-left:20px
         }
legend {
  display:block;
  margin-bottom:10px;
  font-weight:bold;
  font-size:110%;
  text-align:left;
  }
```

og prøver igjen:

DEMO: http://www.ia.hiof.no/~borres/ml/paging/p-paging2.pdf

Vi har ryddet litt i fieldsets og legends, og vi har forsøkt å unngå sideskift inne i kodeblokker og tabeller. Merk at de siste definisjonene ligger utenfor @page - blokken. De gis likevel prioritet i den transformasjonen Prince gjør.

Prince inneholder en rekke flere muligheter enn det som er brukt i dette enkle eksempelet, se dokumentasjonen og eksemplene på vevsiden til Prince.

Referanser og eksempler:

Prince: www.princexml.com/
CSS 2.1: www.w3.org/TR/CSS21/
CSS www.w3.org/Style/CSS/
Modulen På papir viser noen utskriftsmuligheter fra dette materialet.

Endre stilsett fra Javascript

Hva:

Hvordan vi kan programmere endring av stilsett i Javascript

Vi kan ommøblere en HTML-side stort sett slik vi ønsker ved hjelp av DOM-programmering i Javascript, se modulen DOM. I noen tilfeller kan det være vel så effektivt å gjøre dette ved hjelp av stilsett. Det er mulig å endre hvilket stilsett en HTML-side benytter fra Javascript. Nedenfor er et eksempel der en fransk artikkel om Picassos surrealistiske periode kan vises med engelsk eller tysk oversettelse. Oversettelsen er gjort med Babel Fish og er vel heller tvilsom, men det er ikke det som er hovedpoenget her.

En alternativ løsning finnes i modulen: DOM.

DEMO: http://www.ia.hiof.no/~borres/ml/scriptstyles/language.html

Inspiser kildekoden for å se hvordan siden er bygd opp.

Stilsettene er referert i HTML-fila slik:

```
<link rel="stylesheet" href="picstilFrench.css">
<link rel="stylesheet" href="picstilEnglish.css">
<link rel="stylesheet" href="picstilGerman.css">
```

De tre stilsettene er slik:

- picstilFrench.css

```
.rightimage{float:right; margin: 0 0 10px 10px;padding:2px;}
.german{display:none}
.english{display:none}
.french{display:visible;}
```

- picstilEnglish.css

```
.rightimage{float:right; margin: 0 0 10px 10px;padding:2px;}
.german{display:none}
.english{display:visible;}
.french{display:none;}
```

- picstilGerman.css

```
.rightimage{float:right; margin: 0 0 10px 10px; padding:2px;}
.german{display:visible;}
.english{display:none}
.french{display:none;}
```

Vi ser at det eneste som er forskjellig er hvorvidt det er engelsk, tysk eller fransk versjonen skal vises.

Javascriptet som bytter stilsett er slik:

```
 function chooseStyle(theChoice){
  if(document.styleSheets)
    for(i=0;i<document.styleSheets.length;i++)
      if(i==theChoice)
        document.styleSheets[i].disabled=false;
      else
        document.styleSheets[i].disabled=true;
}
```

Javascriptet kalles slik:

```
<a href="Javascript:chooseStyle(1)">English</a> |
<a href="Javascript:chooseStyle(2)">German</a> |
<a href="Javascript:chooseStyle(0)">Francais</a>
```

Vi kan selvsagt kalle funksjonen på andre måter, f.eks. som respons på et museklikk:

```
<div onClick="chooseStyle(2)">
  klikk her for å bytte stilsett til tysk
</div>
```

Referanser og eksempler:

Filene i eksemplet, 1 html-fil, 3 stilsett og et bilde, femfiler.zip

Eksempelet Picasso sammeligner metoden med programmering av stilsett med to andre metoder.

XSL-FO

eXtensible Stylesheet Language FOrmatting

Hva:

Formattering av XML-filer

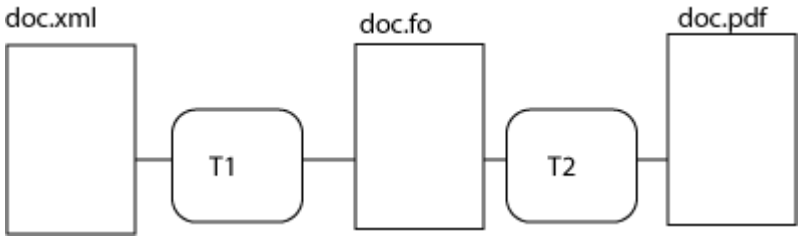
XSL er egentlig to ting: Et transformasjonsspråk (XSLT)og et formateringsspråk (XSL-FO). Det vil si at XSL har i seg ambisjoner om å i vareta begge de interessante perspektivene ved XML. Transformasjonsdelen, XSLT, er behandlet i modulen: XSLT. Her ser vi litt på XSL for formattering.

Denne modulen er ikke noen komplett eller systematisk inføring i XSLT-FO. Det er all grunn til å lese litteraturen og å bruke W3C.

Det har en periode vært noe diskusjon i fagmiljøet om hvilken av de to formatteringsverktøyene som er viktigst: CSS eller XSL-FO. CSS er enkelt, har en lav bruker terskel og fungerer godt for web-presentasjoner og i mange situasjoner godt nok for presentasjon på andre media, som print eller håndholdte enheter. XSL-FO er mer omfattende og er ment å være det verktøyet som kan ivareta profesjonelle krav til dokumentformatering i forbindelse med trykking spesielt.

Jeg slår fast at i øyeblikket trenger vi begge, og jeg velger å betrakte dem som komplementerende. Merk også at de har en god del fellestrekk i måten å beskrive detaljerte blokk- eller paragraf formater på.

La oss ta utgangspunkt i følgende skisse av en to-stegs dokumenttransformasjon:



Dette beskriver en normal arbeidssituasjon når vi arbeider med XSL-FO. Vi antar at vi har et dokument, doc, som er et XML-dokument av en eller annen type. Så gjør vi følgende:

1. Vi transformerer doc.xml til doc.fo. Transformasjon, T1, er en vanlig XSLT-transformasjon. Resultatet, doc.fo, er et xml dokument som inneholder det vi måtte ønske fra opprinnelige dokumentet pluss formateringsinformasjon. Det er vanlig å bruke .fo som suffix på slike dokumenter.
2. Vi transformerer doc.fo til det formatet vi måtte ønske. Typiske formater er, foruten pdf som her, er XML, RTF, tekst eller egentlig hva som helst. Transformasjonen T2 kan være ganske komplisert og krever selvsagt full kontroll over det formatet vi skal transformere til. Det vanlige er å bruke et av flere standard programmer til dette formålet. Det vanligste i øyeblikket ser ut til å være FOP fra http://xml.apache.org/fop/

Det er selvsagt mulig å hoppe over den første transformasjonen og skrive doc.fo direkte for hånd. I det første eksempelet nedenfor gjør vi det. XMLSpy er en egnet editor for å editere både fo-filer og xslt-filer. XMLSpy kan settes opp til å foreta FOP-transformasjoner dersom du har installert FOP på maskinen din.

Merk også at det finnes publiserte XML-formater som har sine egne ferdige T1-transformasjoner. Det mest kjente eksempelet er DocBook som er XML-formater for bøker og artikler. Til DocBook er det utviklet flere transformasjoner som tar dokumenter fra .xml til .fo. Disse transformasjonene er parameterstyrte og muliggjør tilpassing på forskjellige nivåer. Det betyr altså at dersom vi lager dokumenter i DocBook format så kan vi bruke ferdige transformasjoner til både T1 og T2 jobene på figuren ovenfor. Og vi kan få ut alle de formatene som den siste transformasjonen kan handtere. FOP greier eller planlegger blandt annet PDF, HTML, RTF, SVG, XML, PS, TXT. PDF er hovedformatet.

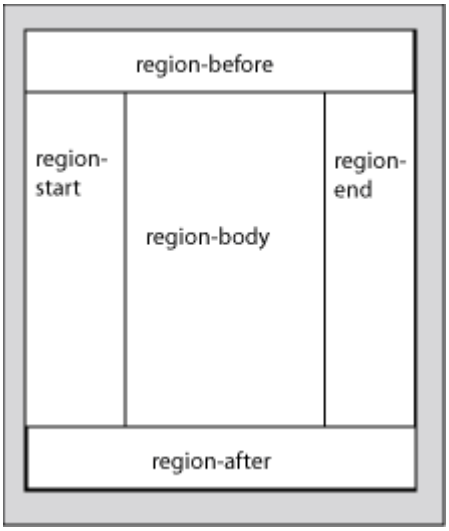
Et håndskrevet eksempel

Vi tar utgangspunkt i følgende enkle skjelett:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
  </fo:layout-master-set>
  <fo:page-sequence>
  </fo:page-sequence>
</fo:root>
```

Vårt skjelett består av to deler. I den første delen, layout-master, setter vi opp de sidene vi vil bruke. I den andre delen, page-sequence, fyller vi sidene. Vi begynner veldig enkelt med å definere en sidemaster:

```
  <fo:layout-master-set>
    <fo:simple-page-master master-name="side"
    page-height="29.7cm"
    page-width="21cm"
    margin-top="7cm"
    margin-bottom="2cm"
    margin-left="2.5cm"
    margin-right="2.5cm">
      <fo:region-body  margin-top="3cm" margin-bottom="1.5cm"/>
      <fo:region-before extent="2cm"/>
      <fo:region-after extent="1.5cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
```



Først angir vi papirstørrelsen og så angir vi absolutte marger. Margene vill ikke kunne fylles med noe innhold. Det vi da sitter igjen med er den delen av arket vi kan disponere. Denne delen (page-reference-area) kan vi dele inn i 5 regioner. Margene er vist grå.

De fire regionenen som omgir region-body vil måtte plasseres forskjellig avhengig av hvilken skrivemåte vi har som referanse. For oss som skriver ovenfra og nedover, fra venstre mot høyre, vil de oppfattes som på tegningen.

Innhold kan rutes til de forskjellige regionene. Den løpende teksten vil typisk rutes til region-body, mens sidenummerering vil havne i region-after.

Vi legger den første av Shakespeares sonetter inn i page-sequence elementet i vår enkle fil og prøver oss med en .fo fil som i sin helhet ser slik ut:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="side"
    page-height="29.7cm"
    page-width="21cm"
    margin-top="7cm"
    margin-bottom="2cm"
    margin-left="2.5cm"
    margin-right="2.5cm">
      <fo:region-body margin-top="3cm"
                      margin-bottom="1.5cm"/>
      <fo:region-before extent="2cm"/>
      <fo:region-after extent="1.5cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="side">
    <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="12pt" line-height="14pt"
                font-family="Times"
                white-space-collapse="false">
FROM fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel.
Thou that art now the world's fresh ornament
And only herald to the gaudy spring,
Within thine own bud buriest thy content
And, tender churl, mak'st waste in niggarding.
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.
          </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Vi ser at strukturen i page-sequence elementet er slik:

```
<fo:page-sequence master-reference="side">
  <fo:flow flow-name="xsl-region-body">
    <fo:block>
    ...
    </fo:block>
  </fo:flow>
</fo:page-sequence>
```

Vi angir hvilken sidemaster vi skal bruke: "side", vi angir hvor innhold skal rutes: "xsl-region-body" og vi pakker innholdet i en (eller flere) block elementer. Det er et stort antall attributter som kan brukes i alle disse elementene for å kontrollere utseende og "flyt".

Resultatet av FOP-transformasjon av fila ovenfor til PDF blir slik: shake1.pdf. Ikke særlig imponerende eller nyttig, men du kan prøve å endre fila for å se effektene.

Vi tar eksempelet et steg til. Nå ønsker vi ta med to sonetter på hver sin side og vi vil ha en egen forside med et bilde. Vi vil dessuten nummerere de to sidene med sonetter på. File kan være slik:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="forside"
    page-height="29.7cm"  page-width="21cm"  margin="2.5cm">
      <fo:region-body   margin="1.5cm"/>
      <fo:region-before extent="2cm"/>
      <fo:region-after extent="3cm"/>
    </fo:simple-page-master>

 <fo:simple-page-master master-name="side"
    page-height="29.7cm"  page-width="21cm"  margin="2.5cm">
      <fo:region-body
          margin-left="3cm"
          margin-top="3cm"
          margin-bottom="1.5cm"/>
      <fo:region-before extent="2cm"/>
      <fo:region-after extent="1.5cm"/>
</fo:simple-page-master>
</fo:layout-master-set>

 <fo:page-sequence master-reference="forside">
    <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="32pt" font-family="Times"
                text-align="center">
        To utvalgte sonetter av Shakespeare
     </fo:block>
     <fo:block text-align="center" margin="20px">
        <fo:external-graphic src="theman.gif"/>
     </fo:block>
    </fo:flow>
  </fo:page-sequence>

  <fo:page-sequence master-reference="side">

     <fo:static-content flow-name="xsl-region-after">
      <fo:block font-size="12pt" font-family="Times"
                text-align="center">
        side <fo:page-number/>
     </fo:block>
     </fo:static-content>

    <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="12pt" font-family="Times"
        white-space-collapse="false">
From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel.
Thou that art now the world's fresh ornament
And only herald to the gaudy spring,
Within thine own bud buriest thy content
And, tender churl, mak'st waste in niggarding.
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.
     </fo:block>

    <fo:block font-size="12pt" font-family="Times"
      white-space-collapse="false" break-before="page">
When forty winters shall besiege thy brow,
```

```
And dig deep trenches in thy beauty's field,
Thy youth's proud livery, so gaz'd on now,
Will be a tatter'd weed, of small worth held:
Then being ask'd where all thy beauty lies,
Where all the treasure of thy lusty days,
To say, within thine own deep-sunken eyes,
Were an all-eating shame and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer, 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it
cold.

      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

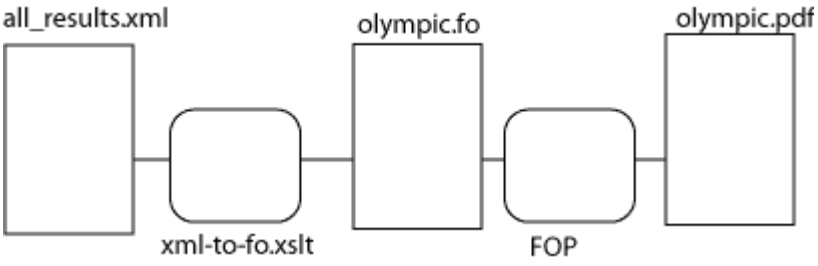


Som du ser har vi introdusert en ny sidemaster som heter forside. De to block-elementene som legger ut sonetter er helt like bortsett fra at den andre bestiller et sideskift. Vi oppdager fort at det er lite effektivt å skrive fo-filer for hånd. I praksis er dette en ganske masete øvelse som krever mye detaljarbeid og mye gjentagelse av detaljer. fo-filene blir ofte ganske omfattende.

Resultatet av FOP-transformasjon av fila ovenfor til PDF blir slik: shake2.pdf. Transformasjonen forutsetter et bilde, theman.gif, i samme katalog som fo-fila.

Eksempel 2

Vi tar fatt i resultatene fra alle sprint øvelsen i de siste olympiadene og vi ønsker å lage en PDF-utskrift sortert på olympiade, øvelse og løperens tid. Oppsettet blir slik:



all_results.xml er en XML-fil som beskriver resultatene. Starten av fila ser slik ut:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<IOC>
  <OlympicGame place="Sidney" year="2000">
    <event track="100m">
      <athlet>
        <name>Maurice Greene</name>
        <nation>USA</nation>
        <result>9.87</result>
      </athlet>
      <athlet>
        <name>Obadele Thompson</name>
        <nation>BAR</nation>
        <result>10.04</result>
      </athlet>
      ...
```

xml-to-fo.xslt er en XSLT-fil. Siden vi har definert våre olympiske resultater i vårt eget format må vi skrive denne selv:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.1"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:output method="xml" version="1.0"
              encoding="ISO-8859-1" indent="yes"/>

<xsl:template match="/">

  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>

  <!-- layout for the first page -->
  <fo:simple-page-master master-name="coverpage"
                page-height="29.7cm"
                page-width="21cm"
                margin-top="7cm"
                margin-bottom="2cm"
                margin-left="2.5cm"
                margin-right="2.5cm">
    <fo:region-body margin-top="3cm"  margin-bottom="1.5cm"/>
    <fo:region-before extent="2cm"/>
    <fo:region-after extent="1.5cm"/>
  </fo:simple-page-master>

  <!-- layout for all other pages -->
  <fo:simple-page-master master-name="pages"
                page-height="29.7cm"
                page-width="21cm"
                margin-top="1cm"
                margin-bottom="2cm"
                margin-left="2.5cm"
                margin-right="2.5cm">
    <fo:region-body margin-top="1cm"  margin-bottom="1cm"/>
    <fo:region-before extent="2cm"/>
    <fo:region-after extent="1.5cm"/>
  </fo:simple-page-master>

  </fo:layout-master-set>

  <!-- filling the front page -->
  <fo:page-sequence master-reference="coverpage">
  <fo:flow flow-name="xsl-region-body">
  <fo:block font-weight="bold" font-size="28pt"
            line-height="38pt" font-family="Times">
```

```
 Olympiske resultater
  </fo:block>
  <fo:block font-weight="normal" font-size="13pt"
            line-height="15pt" font-family="Times">
  Sprintøvelsene i de siste olympiadene
  </fo:block>
    </fo:flow>
    </fo:page-sequence>

  <!-- doing all olympics in turn -->
  <xsl:apply-templates select="/IOC/OlympicGame">
    <xsl:sort select="@year"/>
  </xsl:apply-templates>

</fo:root>
</xsl:template>

<xsl:template match="//OlympicGame">
    <fo:page-sequence  master-reference="pages">
    <fo:flow flow-name="xsl-region-body">
      <fo:block font-weight="bold" font-size="18pt"
                line-height="28pt" font-family="Times"
                padding-top="0cm"
                border-bottom-color="black"
                border-bottom-style="solid">
        <xsl:element name="fo:external-graphic">
           <xsl:attribute name="src">
           <xsl:value-of select="@place"/>.gif
           </xsl:attribute>
        </xsl:element>
      </fo:block>
              <xsl:apply-templates select="event">
              <xsl:sort select="@track"/>
              </xsl:apply-templates>
      </fo:flow>
    </fo:page-sequence>
</xsl:template>

<xsl:template match="//event">
  <fo:block font-weight="bold" font-size="12pt"
            line-height="14pt" font-family="Times"
             padding-top="1cm"  >
         <xsl:value-of select="@track"/>
  </fo:block>
  <xsl:apply-templates select="athlet">
  <xsl:sort  data-type="number" select="result"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="//athlet">
  <fo:block font-size="10pt" line-height="14pt"
            font-family="Times" >
      <xsl:value-of select="name"/>,
      <xsl:value-of select="nation"/>   :
      <xsl:value-of select="result"/>
  </fo:block>
</xsl:template>

</xsl:stylesheet>
```

Vi ser at transformasjonen består av to hovedkomponenter. En del som setter opp layout. Så ser vi at de ulike outputdelene, faste tekster og verdier hentet fra XML-fila pakkes inn i blokker med format-attributter.

olympic.fo som er resultatet av den første transformasjonen er en ganske omfattende og tunglest fil. Du kan se denne i et eget vindu: olympic_fo.html

FOP-transformasjonen fram til det endelige PDF-formatet gjør vi (på MS-Windows)ved å kjøre en bat-fil som tar olympic.fo som input og som produserer olympic.pdf som output. Installasjon og oppsett av FOP er ganske rett fram og greitt dokumentert, se referanser.

olympic.pdf kan du se her: olympic.pdf

Referanser og eksempler:

- Alle filene i eksempel 2: allefiler.zip
- FOP: xml.apache.org/fop/
- DocBook: docbook.sourceforge.net/

Noen anvendelser av XML

Hva:

Noen eksempler på XML-språk med åpne definisjoner

XML kan brukes til mange forskjellige ting. Vi kan bruke XML til å definere dataformater internt i systemløsninger og vi kan bruke XML til å definere språk som er nyttige for å lage enhetlige dokumenter eller datasett for allmenheten eller for grupper med felles interesser.

Cover Pages, [1] , har en samling av referanser til XML-anvendelser

Nedenfor er listet noen kjente XML-språk som har en åpen definisjon og som har ambisjoner om å skape nyttige, allmenne standarder. Lista er selvsagt ikke uttømmende. Utvalget kan virke ganske tilfeldig og er vel i bunn og grunn basert på forfatterens erfaringer og interesser. Gi meg et tips dersom du finner andre kandidater til denne oversikten.

Noen av anvendelsene er drøftet mer inngående i egne moduler.

XHTML

Extensible Hypertext Markup Language, [2]

Dette er en reformulering av HTML 4 i XML 1.0. Det vil si at det er laget en DTD som beskriver HTML 4 som et XML-språk. Dersom vi nullstiller oss helt og ser bort fra at HTML er en de-facto "standard" måte å publisere på, så har vi strengt tatt ikke bruk for HTML. Vi kan tenke oss at all publisering skjer i XML, med tilhørende stilsett (XSL eller CSS). Generelle nettlesere vil måtte kunne parse XML og anvende stilsettet for framstilling. Nå er det flere grunner til at en slik idealisert situasjon er ganske urealistisk. For det første er HTML etablert, for det andre er kunnskapen om stilsett lite utbredt og for det tredje er de færreste forfattere i stand til, eller villige til, å skrive med den disiplinen som XML krever. XHTML er et forsøk på et slags kompromiss som forlanger velformethet, muliggjør validering og lar nettleserne operere med et default stilsett.

De formelle kravene vil trolig begrense anvendelsen av XHTML betraktelig i en lang periode. Det ville være selvmord for en nettleserprodusent å stramme inn på den fleksibiliteten vi kjenner i dagens nettlesere. Selv om en slik innstramming åpenbart ville være svært ønskelig med tanke på en konsistent funksjonalitet.

MathML

Mathematical Markup Language, [3]

Dette er et språk for å definere matematiske uttrykk, og må primært betraktes som et språk for å lage layout.

På [4] finner du en oversikt over en del nyttig programvare for å editere og framstille MathML.

Se også modulen: [5] MathML

SVG

Scalable Vector Graphics, [6]

Språk for å beskrive vektorbasert 2D-grafikk på vevsider.

Se også modulen: [7] SVG

RDF

...kommer snart

X3D

Extensible 3D graphics files [8]

Språk for å beskrive 3D-grafikk på vevsider.

Se også modulen: [9] X3D

XMLNews

For å beskrive nyheter slik at de kan utveksles mellom byråer og framstilles på ulike medier.

Se oversiktsside: [10] og spesifikasjoner: [11]

GML

Geography Markup Language, lansert og vedlikeholdt av Open Gis Consortium som er gruppe av bedrifter og universiteter som arbeider for åpne standarder i utveklsing av geografisk informasjon.

Se [12] GML-spesifikasjon finner du på [13]

GML er tema i avdelingens prosjekt OneMap som drives av Gunnar Misund, se [14]

DocBook

For å skrive dokumentasjon. DocBook vedlikeholdes både som en SGML-definisjon og som en XML-definisjon, [15]

Ant

Apache Ant er et format for å beskrive oppgaver som kompilering, administrasjon av filer (fjerning, flytting, sletting), komprimering av filer etc. Ant er skrevet i Java og kan utvides med Java. Ant er et fleksibelt alternativ til ymse "make"-kommandoer. Ant er utmerket for å administrere prosjekter av ulikt slag, også prosjekter som ikke er Java-baserte. [16]

WSDL

Web Services Description Language er et språk for å beskrive Web Services slik at tilbyder og forbruker av tjenesten kan forstå hverandre. [17]

KML

Keyhole Markup Language

Lansert av Google for å handtere geografiske data, også 3D. [18]

WXT

Web XML Tool

WXT fortjener ikke noen plass på denne lista dersom en skal basere den på utbredelse. WXT er kommet med på lista fordi det er laget og vedlikeholdt av forfatteren. WXT er blandt annet et format for å beskrive vevsteder. De sidene du leser nå bygges og vedlikeholdes ved hjelp av WXT. Se [19]

Referanser og eksempler:

- [1] Cover Pages (en samling med XML-prosjekter):
  xml.coverpages.org/xmlApplications.html
- [2] XHTML (Extensible Hypertext Markup Language):
  www.w3.org/TR/xhtml1
- [3] MathML (Mathematical Markup Language):
  www.w3.org/Math
- [4] MathML (Implementasjoner):
  www.w3.org/Math/implementations.html
- [5] MathML:
  Noen demonstrasjoner av MathML, Matematical Markup Language
  MathML
- [6] SVG (Scalable Vector Graphics):
  www.w3.org/Graphics/SVG
- [7] SVG:
  Noen demonstrasjoner av SVG, Scalar Vector Graphics
  SVG
- [8] X3D (Extensible 3D graphics files):
  www.web3d.org/x3d/specifications/
- [9] X3D:
  Noen korte demonstrasjoner av X3D, eXtensible 3D graphics. Kommer snart
  X3D
- [10] XMLNews:
  www.xmlnews.org/index.html
- [11] XMLNews (spesifikasjon):
  www.xmlnews.org/docs/xmlnews-story.html
- [12] GML (Geography Markup Language):
  www.opengis.org/
- [13] GML (Geography Markup Language):
  www.opengis.org/techno/specs/02-009/GML2-11.html
- [14] Prosjekt OneMap, HiØ:
  www.onemap.org/
- [15] DocBook:
  www.docbook.org/
- [16] Ant:
  ant.apache.org/
- [17] WSDL (Web Services Description Language):
  www.w3.org/TR/wsdl
- [18] KML(Keyhole Markup Language):
  code.google.com/apis/kml/documentation/whatiskml.html
- [19] WXT (Web XML Tool):
  www.ia.hiof.no/~borres/wxtdoc/

SVG

Hva:

Noen demonstrasjoner av SVG, Scalar Vector Graphics

> *Statusen til SVG er i skrivende stund, august 2006, noe uklar. Fra betydelig optimisme for noen år siden ser det ut til at ulike og manglende installasjoner i nettleserne har lagt en viss demper på entusiasmen. Mozilla og Opera har installert SVG i selve nettleseren, mens MSIE fordrer en plugin. Noen av eksemplene på disse sidene kan feile i noen nettlesere.*

Tradisjonelt har illustrasjoner på vevsider vært rastergrafikk i form av gif, jpg eller png bilder. Dersom en er ute etter illustrasjoner som er bygget opp av tegningselementer i form av geometriske figurer, rotert tekst eller lignende er trolig Flash fra Macromedia det mest nærliggende alternativet for de fleste. Det samme gjelder animasjon av geometriske elementer. Plugin for flash har vært og er en kurant løsning med enkel installasjon.

SVG er en åpen standard støttet av W3C som er et alternativ i mange situasjoner. I dag er det slik at også SVG fordrer en plugin. Du må installere denne eller du kan forsøke nettleseren Amaya, som er den nettleseren W3C benytter for å teste blandt annet SVG. Du trenger en plugin for å arbeide med eksemplene nedenfor. Se referanser.

Vi ønsker selvsagt å inludere SVG i vevsider som også består av andre språkelementer. Typisk vil vi ha en side med XHTML og SVG, eller XHTML, SVG og MathML, eller XHTML, SVG og RDF. Mer om dette senere. Vi starter med noen rene SVG-dokumenter for å illustrere noen grunnleggende prinsipper.

Et rektangel

Et enkelt SVGdokument, svgdok1.svg, kan se slik ut:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="5cm"  viewBox="0 0 50 50"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Et enkelt eksempel</desc>
  <rect x="10" y="10" width="20" height="20"
    fill="yellow" stroke="blue" stroke-width="2"/>
</svg>
```

DEMO: http://www.ia.hiof.no/~borres/ml/svg/svgdok1.svg

Vi har etablert en fil med riktig dtd, og vi har et rotelement: svg. Dette elementet er kvalifisert via et namspace og vår SVG-tegning beslaglegger et område på 5x5 cm. Det adresseområdet vi har definer er 50 X 50. Det vil si at når vi beskriver figurer i dette adresseområdet, så mappes dette til det 5x5 cm store området på siden (skjermen). Det eneste vi tegner ut er et rektangle som er fylt med gult og med blå kant(, så svenskene får sitt). Du kan eksperimentere med koordinatene til rektangelet og til svg-elementet for å se hvordan mappingen via viewBox fungerer.

Default er origo oppe til venstre i det koordinatsystemet som etableres.

Flagget

Det norske flagget kan lages slik:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="300px" height="300px"  viewBox="0 0 300 300"
  xmlns="http://www.w3.org/2000/svg">
  <desc>
  Flaggets proposjoner:
  Horisontalt: 6R 1H 2B 1h 12R=22
  Vertikalt:   6R 1H 2B 1H 6R=16
  </desc>
    <g transform="translate(20,50)">
      <rect x="0" y="0" width="220" height="160"
      fill="red" stroke="none" stroke-width="0"/>

      <rect x="0" y="60" width="220" height="40"
      fill="white" stroke="none"/>
      <rect x="60" y="0" width="40" height="160"
      fill="white" stroke="none"/>

      <rect x="0" y="70" width="220" height="20"
      fill="blue" stroke="none"/>
      <rect x="70" y="0" width="20" height="160"
      fill="blue" stroke="none"/>

      <g transform="translate(0,225)">
        <text id="TextElement" x="0" y="0"
          font-family="Verdana" font-size="25"
          fill="black" stroke="none">
          17.mai
        </text>
      </g>

   </g>
</svg>
```

DEMO: http://www.ia.hiof.no/~borres/ml/svg/svgnoflag.svg

Vi har tatt i bruk elementet g og vi bruker det til å flyttet origo mens vi "tegner", translate. Teksten som kommer under flagget er lagt inn i et eget g-element. Siden dette er omsluttet av det andre vil hele tegningen, flagget og teksten, kunne forskyves ved å endre parametrene til translate i det først, ytterste, g-elementet.

Smiley

En blid smiley har litt mer komplisert form enn de rektanglene vi har brukt ovenfor. Vi vil se at den runde hodeformen og øynene lar seg realisere med elementer som heter circle. Uten munn kan en smiley lages slik, svgsmiley1.svg:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="200px" height="200px"  viewBox="0 0 400 400"
  xmlns="http://www.w3.org/2000/svg">
  <desc>En smiley</desc>
  <g transform="translate(200,200)">
  <circle cx="0" cy="0" r="100" fill="yellow"/>
  <circle cx="40" cy="-40" r="20" fill="black"/>
  <circle cx="-40" cy="-40" r="20" fill="black"/>
  </g>
</svg>
```

DEMO: http://www.ia.hiof.no/~borres/ml/svg/svgsmiley1.svg

Hva så med munnen. Dette er litt mer utfordrende. Vi må på en eller annen måte kunne forme en kurve. Den enklsete måten å beskrive en kurve på er å angi punktene på kurven. I SVG kan vilkårlige polygoner (mangekanter) beskrives ved den vi kan kalle en blyant-metafor. Vi har en kommando for å løfte blyanten og flytte den og en kommando for å trekke en strek. Elementet

```
<path d="M 0,0 L 10,0 5,10 z"
      fill="none" stroke="black" stroke-width="1"/>
```

vil tegne en trekant. Attributten d(ata) sier at vi skal M(ove) blyanten til 0,0, så skal vi lage L(ine) fra punkt til punkt i lista som følger. z til slutt sier at kurven skal lukkes. På grunnlag av dette kan vi foreslå følgende smiley, svgsmiley2.svg

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="200px" height="200px"  viewBox="0 0 400 400"
  xmlns="http://www.w3.org/2000/svg">
  <desc>En smiley</desc>
  <g transform="translate(200,200)">
  <circle cx="0" cy="0" r="100" fill="yellow"/>
  <circle cx="40" cy="-40" r="20" fill="black"/>
  <circle cx="-40" cy="-40" r="20" fill="black"/>
  <path d="M-40,10 L -20,30 -10,35 0,40  10,35 20,30 40,10"
        fill="black" stroke="none"/>
  </g>
</svg>
```

DEMO: http://www.ia.hiof.no/~borres/ml/svg/svgsmiley2.svg

Dette er ikke helt vellykket. Vi har mange muligheter for forbedring. Vi kan lage flere og tettere punkter, eller vi kan bruke en matematisk måte å beskrive en glattere kurve. For de som er kjent med grafisk databehandling så er det f.eks. mulig å lage Bezierkurver. Vi forfølger ikke dette her.

Tekst

Vi brukte tekst ovenfor i flaggeksempelet uten å kommentere det noe videre. Det er relativt kurant å sett ut tekst i SVG-områder. F.eks. slik, svgtext1.svg:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="400px"  viewBox="0 0 400 400"
  xmlns="http://www.w3.org/2000/svg">
  <desc>litt ymse tekst</desc>
<g transform="translate(200,200)">
   <text id="TextElement" x="0" y="0"
      transform="rotate(-30)"
      font-family="Verdana" font-size="20"
      fill="blue" stroke="none">
      Hallo dette ble skjevt
   </text>
   <text id="TextElement" x="-70" y="50"
      transform="rotate(30)"
      font-family="Verdana" font-size="20"
      fill="red" stroke="black" stroke-width="0.2">
      Ja du store
   </text>
</g>
</svg>
```

DEMO: http://www.ia.hiof.no/~borres/ml/svg/svgtext1.svg

Animasjon

Vi kan skape tidsstyrte animasjoner i SVG. Det er tre elementer som styrer slik animasjon. Nedenfor er de satt opp med noen typiske attributter. Du må selvsagt slå opp på SVG sidene i W3C for å få full oversikt.

```
<animateMotion from="0,0" to="50,50"
     begin="0s" dur="24s" repeatCount="indefinite"/>
<animateTransform attributeName="transform" attributeType="XML"
     type="rotate" from="1" to="1440"
     begin="0s" dur="24s" repeatCount="indefinite"/>
<animateColor attributeName="fill" attributeType="CSS"
     from="blue" to="red"
     begin="0s" dur="24s" repeatCount="indefinite"/>
```

Vi forsøker oss på litt tekst,svgtext2.svg.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="500px"  viewBox="0 0 500 400"
  xmlns="http://www.w3.org/2000/svg">
  <desc>litt dynamisk tekst</desc>
<g transform="translate(200,200)">
   <text id="TextElement" x="-10" y="5"
      font-family="Verdana" font-size="5"
      fill="blue" stroke="none">
      Hallo
      <animateMotion from="0,0" to="50,50"
         begin="0s" dur="12s"  fill="freeze" />
         <animateTransform attributeName="transform"
         attributeType="XML"
         type="rotate" from="1" to="1440"
         begin="0s" dur="6s" fill="freeze" />
      <animateTransform attributeName="transform"
         attributeType="XML"
         type="scale" from="1" to="15" additive="sum"
         begin="0s" dur="12s" fill="freeze" />
      <animateColor attributeName="fill"
         attributeType="CSS"
         from="blue" to="red"
         begin="0s" dur="24s" repeatCount="indefinite"/>
   </text>
</g>
</svg>
```

DEMO: http://www.ia.hiof.no/~borres/ml/svg/svgtext2.svg

DEMO: http://www.ia.hiof.no/~borres/ml/svg/p-jungle.svg

Dersom vi skal animere, f.eks. rotere, vår smiley må vi lage en konstruksjon som roterer alle delene. Dette kan vi oppnå ved å lage et g-element som inneholder det ønskede animasjonselementet og smileybeskrivelse, svgtext3.svg:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="200px" height="200px"  viewBox="0 0 400 400"
xmlns="http://www.w3.org/2000/svg">
<desc>En smiley</desc>
<g transform="translate(200,200)">
   <g>
      <animateTransform attributeName="transform"
      attributeType="XML"
      type="rotate" from="1" to="1440"
      begin="0s" dur="6s" fill="freeze" />

      <circle cx="0" cy="0" r="100" fill="yellow"/>
      <circle cx="40" cy="-40" r="20" fill="black"/>
      <circle cx="-40" cy="-40" r="20" fill="black"/>
      <path d="M-40,10 L -20,30 -10,35 0,40  10,35 20,30 40,10"
      fill="black" stroke="none"/>
   </g>
</g>
</svg>
```

DEMO: http://www.ia.hiof.no/~borres/ml/svg/svgsmiley3.svg

Kartdata

SVG er ganske raskt.

DEMO: http://www.ia.hiof.no/~borres/ml/svg/norge.svg

Interaksjonen i kartet er realisert ved hjelp av de eventene som er definert for SVG og litt DOM-programmering. Et utdrag av kildekoden er slik: (Polygondataene er redusert):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="100%" height="100%" viewBox="0 0 11939 15026"
      xmlns="http://www.w3.org/2000/svg">
<desc>Norgeskart</desc>

<!-- ECMAScript to identify a kommune after mouseover -->
<script type="text/ecmascript"> <![CDATA[

var lastPath=null;
var markcolor="red";
var unmarkcolor="white";

function kommune_click(evt) {
  var path = evt.target;
  path.setAttribute("fill",markcolor);
  if (lastPath!=null)
    lastPath.setAttribute("fill",unmarkcolor);
  lastPath=path;
  var kommunename = path.getAttribute("id");
  var komElement=document.getElementById("KommuneElement");
  var komTextNode=document.createTextNode(kommunename);
  komElement.replaceChild(komTextNode,komElement.getFirstChild());
}
]]> </script>

<g transform="translate(250,1250)">
<text id="TextElement" x="0" y="0" font-family="Verdana"
         font-size="500" fill="black">
  Norge
</text>
</g>

<g transform="translate(7250,7250)">
<text id="KommuneElement" x="0" y="0" font-family="Verdana"
         font-size="500" fill="black">
  Move mouse on map
</text>
</g>
<g transform="translate(0,15026)">
  <g transform="scale(1.0,-1.0)">
    <path onmouseover="kommune_click(evt)" id="Halden"
      d="M 3839,1188  L 3834,1257 ... z" fill="none" stroke="black"/>
    <path onmouseover="kommune_click(evt)" id="Sarpsborg"
      d="M 3599,1314  L 3594,1324 ... z" fill="none" stroke="black"/>
    <path onmouseover="kommune_click(evt)" id="Fredrikstad"
      d="M 3509,1154  L 3509,1171 ... z" fill="none" stroke="black"/>
    ...
  </g>
</g>
</svg>
```

Den komplette kildekoden finner du her: norge_svg.html .

Merk at dataene er gamle (før siste kommunesammenslåing) og det er noen feil i polygonene.

Som del av et dokument

Utgangspunktet er at vi ønsker å legge SVG-komponenter inn på en HTML-side. Det er flere måter å tenke seg dette løst på:

Vi kan tenke oss en løsning der vi rett og slett legger SVG-elementene på plass og kvalifiserer dem med namespace, De fleste lærebøkene i XML bruker slike filer om eksempel:

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>mixed SVG</title>
</head>
<body>
<h1>En side</h1>
<svg xmlns="http://www.w3.org/2000/svg"
   width="5cm" height="5cm"  viewBox="0 0 50 50">
  <desc>Et enkelt eksempel</desc>
  <rect x="10" y="10" width="20" height="20"
    fill="yellow" stroke="blue" stroke-width="2"/>
</svg>
<p>med blandet innhold</p>
</body>
</html>
```

Dette forutsetter at programmet som leser fila er forberedt på å generere SVG når den treffer på en slk node i DOM-treet. Nettleserne synes ikke være klare for det, og greier ikke å kople plug-ins til slik SVG-rendering.

Dersom SVG blir akseptert som en del av det normale nettlesere skal kunne handtere uten plug-in vil denne koden være den naturlige formen å inkludere SVG på.

Alternativet er å bruke det generelle <embed> </embed> elementet, slik vi gjør når vi vil legge inn andre komponenter som fordrer plug-in. Følgende element vi vise fila etbilde.svg, eventuelt tilby seg å installere plug-in først:

```
<embed width="100px" width="100px" src="supersmiley.svg"
       pluginspace="http://www.adobe.com/svg/viewer/install/">
```

DEMO: http://www.ia.hiof.no/~borres/ml/svg/smiley4.html

Referanser og eksempler:

- Scalable Vector Graphics: www.w3.org/Graphics/SVG
- Plugin for SVG.
  Generelt:
  www.w3.org/Graphics/SVG/SVG-Implementations.htm8#viewer
  eller direkte til Adobe:
  www.adobe.com/svg/viewer/install/
  Merk at det er litt problemer med Mozilla og Adobe SVG-plugin. Se plugindoc.mozdev.org/windows.html for en forklaring på hva du kan gjøre.
- Noen gode råd når du skal generere SVG-filer fra et CGI-script: www.svgopen.org/2004/papers/ServerSideSurvey/
- Du kan tegne og få SVG-output i flere verktøy. Høgsskolen har lisens for disse:
  - Abode Illustrator: www.adobe.com/products/illustrator/main.html
  - WebDraw: www.jasc.com/products/webdraw/

MathML

Hva:

Noen demonstrasjoner av MathML, Matematical Markup Language

Merk at MathML er designet for å dekke to formål, markup for strukturering av matematiske uttrykk og markup for presentasjon av matematiske uttrykk. I denne modulen tar bare hensyn til den andre egenskapen: Hvordan få presentert matematiske uttrykk på en vevside.

Det har vært et problem å framstille matematiske uttrykk på vevsider. En måte å løse dette på er som følger:

1. Åpne MS Word og lag et nytt dokument
2. Sett inn et objekt av type MS Equation Editor på siden
3. Rediger formelen ved hjelp av Equation Editor
4. Spar dokumetet som html, formelen blir lagret som gif i en subkatalog som kalles images.
5. Plukk opp det riktige bildet og lim det inn i vevsiden

... og vi kan få Taylors formel slik: $f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + R(x)$

Dette er ikke noen god løsning. Dels gjør vi oss avhengig av MS Equation editor, dels er det stor sannsynlighet for at dokumentet med tilhørende formler forsvinner, dels har vi ingen mulighet til å endre stil eller størrelse av noe slag når bildet først er laget og dels tar slike bilder av formler stor plass.

Enkle uttrykk som involverer eksponenter og indekser kan vi få til i vanlig HTML, ved hjelp av elementene <sup> og <sub>. F.eks slik:

$X^2+Y^2=R^2$	`X<sup>2</sup>+Y<sup>2</sup>=R<sup>2</sup>`
$Y:\{w_1,v_3,\}$	`Y:{w<sub>1</sub>,v<sub>3</sub>}`

Dette gir oss alt for lite spillerom til å formulere matematisk uttrykk.

MathML gjør det mulig for oss å editere matematikken direkt på siden eller vi kan bruke en WYSIWYG editor som i motseting til MS Equation Editor genererer åpen XML-kode.

I dag er det slik at Mozilla skal kunne vise fram sider med MathML-innhold uten noen Plugin. MSIE fordrer en Plugin. Amaya kan vise (og editere) rene MathML-sider uten Plugin. Opera er ikke særlig glad i MathML i det hele tatt(?). Nedenfor skal vi se på et svært enkelt matematisk uttrykk preparert for de tre nettleserne:

MS Internet Explorer

Jeg bruker en plugin, MathPlayer, fra Design Science: www.dessci.com. En typisk HTML-side for MSIE kan da se slik ut:

```
<?xml version="1.0"?>

<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:m="http://www.w3.org/1998/Math/MathML" >
<head>
   <OBJECT ID="MathPlayer"
      CLASSID="clsid:32F66A20-7614-11D4-BD11-00104BD3F987">
   </OBJECT>
   <?IMPORT NAMESPACE="m" IMPLEMENTATION="#MathPlayer" ?>
</head>

<body>
<p>Et uttrykk skrevet i MathML</p>
<p style="font-size:25px;color:red">
<m:math>
  <m:msup>
    <m:mi>r</m:mi>
    <m:mn>2</m:mn>
  </m:msup>
  <m:mo>=</m:mo>
  <m:msup>
    <m:mi>x</m:mi>
    <m:mn>2</m:mn>
  </m:msup>
  <m:mo>+</m:mo>
  <m:msup>
    <m:mi>y</m:mi>
    <m:mn>2</m:mn>
  </m:msup>
</m:math>
</p>
</body>
</html>
```

Forutsatt at du har installert mathPlayer:

DEMO: http://www.ia.hiof.no/~borres/ml/mathml/smallIE.html

Vi ser her en ganske spesiell konstruksjon. Vi finner et objekt som har ID=MathPlayer og så koples namespace m til dette objektet. Det er lett å se at dette gir en logisk kopling mellom de aktuelle elementene og den programkoden (plug-in) som skal tolke dem, men det er en uvanlig løsning.

Mozilla

Mozilla skal greie seg uten Plugin. Den samme siden preparert for Mozilla kan se slik ut:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN"
                "http://www.w3.org/TR/MathML2/dtd/xhtml-math11-f.dtd"
[
  <!ENTITY mathml "http://www.w3.org/1998/Math/MathML">
]>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>MathML-1</title>
</head>
<body>
<p>Et uttrykk skrevet i MathML</p>
<p style="font-size:25px;color:red">
<math xmlns="&mathml;">
  <msup>
    <mi>r</mi>
    <mn>2</mn>
  </msup>
  <mo>=</mo>
  <msup>
    <mi>x</mi>
    <mn>2</mn>
  </msup>
  <mo>+</mo>
  <msup>
    <mi>y</mi>
    <mn>2</mn>
  </msup>
</math>
</p>
</body>
</html>
```

DEMO: http://www.ia.hiof.no/~borres/ml/mathml/smallMoz.xhtml

Amaya

Amaya innholder både en enkle MathML-editor og en framviser. Filene lagres som rene mml-filer. En slik fil kan se slik ut:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
      "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<!-- Created by amaya 8.7, see http://www.w3.org/Amaya/ -->
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <msup>
    <mi>r</mi>
    <mn>2</mn>
  </msup>
  <mo>=</mo>
  <msup>
    <mi>x</mi>
    <mn>2</mn>
  </msup>
  <mo>+</mo>
  <msup>
    <mi>y</mi>
    <mn>2</mn>
  </msup>
</math>
```

DEMO: http://www.ia.hiof.no/~borres/ml/mathml/smallAmaya.mml

Et eksperiment

Ta for deg fila som er laget for Mozilla og åpne den i MSIE:

DEMO: http://www.ia.hiof.no/~borres/ml/mathml/smallIE.html

Hvis du har installert MathPlayer skal du se sida her også. Se på kildekoden til den siden du åpner. MSIE har tydeligvis funnet ut hva slags innhold vi har og modifisert fila i nettleseren. Summen av dette er altså at vi kan lage filer for Mozilla, og håpe at de vises både i Mozilla og MSIE, dersom du har installert MathPlayer i sistnevnte.

Eksempel 1

Vi lager en enkel kvadratrot med summen av to kvadrater. Merk at det er koplet til et enkelt stilsett som gjør matematikken blå:

```
<math xmlns="">
   <mi>r</mi>
   <mo>=</mo>
   <msqrt>
      <mrow>
         <mo>(</mo>
         <msup>
            <mi>x</mi>
            <mn>2</mn>
         </msup>
         <mo>+</mo>
         <msup>
            <mi>y</mi>
            <mn>2</mn>
         </msup>
         <mo>)</mo>
      </mrow>
   </msqrt>
</math>
```

math	introduserer det matematiske uttrykket
mi	identifier
mn	tall
mo	operator, parentes eller separator
msqrt	et rottegn med innhold
mrow	ordner innholdet horisontalt, en rad
msup	eksponesiering, superscript

DEMO: http://www.ia.hiof.no/~borres/ml/mathml/math2.xhtml

Eksempel 2

Vi lager en 3X3 matrise (som framstiller en rotasjonstransformasjon i planet).

```
<math xmlns="">
<mrow>
  <mi>R</mi>
  <mo>=</mo>
  <mfenced open="[" close="]">
    <mtable>
      <mtr>
         <mtd>
            <mi>-cos</mi>
            <mo>
            (</mo>
            <mi>v</mi><mo>)
            </mo>
         </mtd>
         <mtd>
            <mi>sin</mi>
            <mo>
            (</mo><mi>v</mi><mo>)
            </mo>
         </mtd>
         <mtd>
            <mi>0</mi>
         </mtd>
      </mtr>
      <mtr>
         <mtd>
            <mi>sin</mi>
            <mo>
            (</mo><mi>v</mi><mo>)
            </mo>
         </mtd>
         <mtd>
            <mi>cos</mi>
            <mo>(</mo><mi>v</mi><mo>)
            </mo>
         </mtd>
         <mtd>
            <mi>0</mi>
         </mtd>
      </mtr>
      <mtr>
         <mtd><mi>0</mi></mtd>
         <mtd><mi>0</mi></mtd>
         <mtd><mi>1</mi></mtd>
      </mtr>
    </mtable>
  </mfenced>
</mrow>
</math>
```

mfenced	sett gjerde rundt innholdet
mtable	tilsvarer table i html
mtr	tilsvarer tr i html
mtd	tilsvarer td i html

DEMO: http://www.ia.hiof.no/~borres/ml/mathml/math3.xhtml

Referanser og eksempler:

- Mathematical Markup Language, www.w3.org/Math
- MathPlayer, en MathML plugin fra Design Science: www.dessci.com
- En MathML editor: www.soft4science.com/
  ... og det finnes flere.
- Amaya: www.w3.org/Amaya/
- Å forfatte for Mozilla: www.mozilla.org/projects/mathml/authoring.html

X3D

Hva:

Noen korte demonstrasjoner av X3D, eXtensible 3D graphics. Kommer snart

Med 3D-grafikk forstår vi tredimensjonale modeller som lar seg betrakte fra flere synsvinkler. Det er mulig å lage 2D-grafikk som gir en illusjon av 3D (dybde, skygger osv). Det vi er ute etter er modeller som tolkes av nettleseren, eller tilhørende programmer, og som rendres, framstilles hos klienten. Det er grovt sett to vanlige muligheter for å vise slik 3D-grafikk på vevsider.

- Applets. det finnes en rekke løsninger som er skrevet som JavaApplets. Noen av disse fordrer at vi har installert et Java-bibliotek på klientmaskinen og noen laster ned komplett javakode sammen med nettsiden.
- Plugins som tolker VRML (Virtual Reality Modelling Language). VRML har en historie med flere versoner og varianter og det finnes flere aktuelle plugins.

X3D er en videreføring og standardisering av VRML. X3D trenger en plugin for å kjøre.

...kommer snart

Referanser og eksempler:

- Extensible 3D graphics files www.web3d.org/x3d/specifications/