XML - indholdsfortegnelse

Introduktion til XML 2
Kom i gang med at skrive XML 3
Specialtegn – de fem indbyggede entiteter 3
Undgå at parseren læser og tolker det strevende 3
DTD 4
Oprettelse af DTD'er 4
To måder at oprette schema på: DTD'er og XML Schema 4
Begrænsning ved DTD 4
Definition af elementer og attributter i DTD'er 4
Entiteter og notationer i DTD'er 5
Generelle entiteter 5
- Der findes 5 indbyggede af den slags Specialtegn – de fem indbyggede entiteter (kap.1) 5
Parameterentiteter 5
XML Schema 6
To måder at oprette schema på: DTD'er og XML Schema 6
Begrænsning/ulemper ved DTD 6
XML Schema muliggøre 6
Lokal/global erklæring – s. 71 6
Schema.xsd 6
Definition af simple typer 7
Simple typer 7
Taltyper s. 80 7
Afledning af simple typer s. 81 7
Angiv værdimængde 8
Angiv mønster 8
XML Schema - Definition af komplekse typer 10
Elementer der kun må indeholde elementer og evt. attributter 10
Min- og maxOccurs – s. 101 11
Elementer der kun må indeholde tekst 11
Tomme elementer 11
Komplekse typer baseret på andre komplekse typer – s. 105 12
Atributter – s. 109 12
XML Schema – Anvendelse af namespaces 12
XSLT 14
Eksempler 14
3 måder at udfører transformationen på 14
Transformationen sker kun når data ændre sig 14
Hvorfor? 14
To typer komponenter i et XSLT style sheet – s. 140 15
Start 15
Rodskabelon – s. 139 15
Givne node 15
Skabelonregler 15
Batchbehandling af noder – s. 146 15
Alternative betinkelser 16
Sortering af noder inden de behandles – s. 150 16
Xpath: Mønstre og udtryk – s. 153 16
XSTL: Test med udtryk og funktioner 17
Sammenligning at to værdier 17
XML – del 2, kr s. 117 18
ASP script der udfører transformeringen – skrivTilFil 18
DOM 20
Specifikation 20
Forskeld ml. XSLT og DOM 20

Introduktion til XML
Kr, kap. 2
XML dokument: Data og metadata – databeskrivelse
XML
Oprindeligt afløser for html
Kommer ud over HTML's begrænsninger
Nedstammer fra SGML – teknikken med at beskrive data sammen med data
Udbyttet med XML er poportinalt med evnen til at abstrahere fra HTML
XML = HTML + navngive egne elementer og attributter + grundlæggende styrke: Hierarkier
XML lige så let som HTML med mulighederne er større
Dataformatete kendetegn: Tekstformat bestående af markup organiseret i hierarkier

- Tekstformat: Unicode **<?xml version="1.0" encoding="ISO-8859-1"?>**
- XML dokumenter i ascii kræver encoding

Sproget skal overholde syntaktiske og semantiske regler (Semantik – hvordan de forståes)
Data pakket ind i markup – elementer organiseret i hieraki

Krav: Læses direkte af mennesker
Fondamentale ligheder: Tekstbaseret, markup
Forskeld
Tags: HMLT – unsende, XML – indhold

Markup
Mængden af markup-elementer er forholdsvis overskuelig for HTML
Mængden af markup-elementer er uendelig for XML
- Denne frihed kaldes **vokabularer** / semantisk markup
(Vokabular: De regler man definere, navn + indhold
Data, beskrivelse at data og kommentarer i et og samme dokument
Tags = elementer < >
Noder = elementer + attributter + kommentarer
Syntaktisk krav: Start og slut-tag
Metadata: Data der følger med data og fortæller noget om disse data
Velformet når XML dokumentet overholder de syntaktiske krav (fra st.brev 2) kan det vises i en browser

- Start/sluttag – elementer skal afsluttes
- Elementer må ikke overlappe hinanden
- Start/sluttag – samme case
- Kun et element på yderste niveau / dokumentelement / rodelement (- kommentarer og procesinstruktioner)
- Tegnene < og & må ikke forekomme i elementer (CDATA sektioner undtaget)
- Attributværdier i ' ' eller " " citationstegn
- Et element på ikke have to attributter med samme navn
- Kommentarer og procesinstruktioner ikke indeni tags
- Entiteter skal erklæres i DTD

Hierarki: Uomtvisteligt, opnås ved indryk – lettere at udvikle og læse

- X-diagram kan visualisere et XML-hierarki – s. 30-31

Velformethedstet: Kr s. 59
Problem med internet: Ustrukturerede data – kan ikke se træer for bare træer

Hvorfor XML:

- Åbent tekstformat
- Selvbeskrivende data
- Adskillelse af data og præsentation (største forskel på XML og HTML)
- Letter kommunikation med andre systemer (A2A – B2B)
- Mindre antal kodelinjer da tungt arbejde flyttes til standardiserede, gennemtestede, robuste og skalerbare XML-parser – giver mere focus på regler og data

Kom i gang med at skrive XML
Ca, kap. 1

HTML's mangler: Enkel og tolerant men dette begrænser effektivitet. Tags kun vedr. formatering ikke indhold hvilket vanskeliggøre genbrug. Kræver mere af browseren. Begrænsning i formatering og dynamik.

XML´s styrke: Ikke kun til oprettelse af websider men også sprog. Information bliver vha. tag anvendelige for andre sammenhænge. Mere striks.

Specialtegn – de fem indbyggede entiteter
& &
< <
> >
" "
' '

mellemrum

Undgå at parseren læser og tolker det strevende
Omring det med <!CDATA[ ]]>

Visning af XML med CSS
Kr. s. 62.
Simpel måde at vise XML = stylesheet (CSS og XML transformationer)
- CSS: Regler for hvordan elementer viser (deklarativt sprog)
- CSS: Ingen mulighed for: tilføje, oprette som tabel – kan gøres med data island (simpel) men andre ord vise xml dokument i html-dokumenter

DTD

Oprettelse af DTD'er
Ca, kap. 2

XML er ikke blot sprog, det kan bruges til at oprette nye sprog
Et regelsæt til et sprog kaldes **schema**
Ikke obligatorisk at anvende men nyttigt værktøj
Hvis et dokument overholder reglerne i skemaet er dokumentet **gyldigt**
To måder at oprette schema på: DTD'er og XML Schema
DTD'er er gammeldags men udbredt, det anvender en speciel og begrænset syntaks

Intern erklæring: s. 36 – dokumenttypeerklæring
Ekstern erklæring: s. 37 – gem filen som *.dtd fil

Navngivning af eksterne DTD´er: s. 38 – personlig : url

Begrænsning ved DTD
- Man kan ikke angive af bestemte data skal være tal, dato, tekst, ...(altså ingen typeangivelser)
- Ingen god måde til at angive et bestemt antal, se ? + *
- Understøtter ikke direkte namespaces – ca s. 119

Definition af elementer og attributter i DTD'er
Ca, kap. 3

Rækkefølgen er ligemeget

<!ELEMENT navn **(element)>**
<!ELEMENT navn **EMPTY>** - tomp ofte billeder, men så attributter
<!ELEMENT navn **ANY>** - alle, vær forsigtig med dennes brug – meningen?
<!ELEMENT navn **(#PCDATA)>** - parsed character data : tal, bogstaver, symboler og entiteter
- Kan ikke indeholde andre elementer
- Kan være en del af valg, men ikke del af sekvens
<!ELEMENT navn **(element1, element2, element3)>** - sekvens(her kan #PCDATA ikke benyttes)
- Kan godt oprette valgmuligheder som en del af en sekvent
<!ELEMENT navn **(element1 | element2 | element3)>** - valgmuligheder
<!ELEMENT navn **(element1 | element2 | (element3, element4))*>** -
- Kan indeholde et vilkårligt antal valgmuligheder. Ikke-ordnet liste i et ordnet element
- Den første valgmulighed kan være #PCDATA på den måde kan leves sekvens med #PCDATA
<!ELEMENT navn **(element?)> -** element kan højst optræde en gang
<!ELEMENT navn **(element+)> -** element skal optræde min. en gang
<!ELEMENT navn **(element*)> -** element kan optræde 0, 1 eller mange gange

Attributter
- Inf om sidens indhold frem for dele af indhold
- Informationer om informationer
<!ELEMENT navn **(#PCDATA)>** - s. 50
<!ATTLIST navn attributnavn **CDATA #IMPLIED>** - uden #P !!
<!ATTLIST navn attributnavn **CDATA (A | B) standard >** - uden #P !!
- Attributværdien skal være A eller B, A el. B kan også angives som standardværdi
- #FIXED X – værdien skal sættes til X,
- #REQUIRED – skal tildeles en værdi
- #IMPLIED – ingen standardværdi og kan undlades helt
<!ATTLIST navn attributnavn **ID #REQUIRED>** - Kræver et unikt id
- ID inderholder kun gyldige xml-navne, start med bogstav el _

<!ATTLIST navn attributnavn **IDREFS #REQUIRED>** - Bestemt id s. 53
- Angiv en eller attributværdier som eksisterer
<!ATTLIST navn attributnavn **NMTOKEN #IMPLIED>** - s. 54
- Eneste restrektion i DTD – attributten skal være gyldigt XML-navn
<!ATTLIST navn attributnavn **NMTOKENS #IMPLIED>** - hvis liste

Entiteter og notationer i DTD'er
Ca, kap. 4

Entiteter bruges til at definere en forkortelse (entitetshenvisningen)
Der findes flere forskellige entiteter, forskeld:
- Hvor defineres den?
- Hvilken information indeholder den?

2 hovedgrupper: **Generelle entiteter** og **parameterentiteter**
- Generelle entiteter: Indlæser data i selve XML-dokumentet
- Parameterentiteter: Henviser til data der så bliver en del af DTD'en

Generelle entiteter
- Data som anvendes i XML-dokumentets indhold
Underopdeles i interne og eksterne
- Angiver om de er defineret inden for DTD'en eller i en ekstern fil

Interne: Tekstforkortere (s. 56-57)
- Der findes 5 indbyggede af den slags **Specialtegn – de fem indbyggede entiteter (kap.1)**
(de er ikke entiteter – skal ikke erklæres i DTD'en)
Eksterne: Parsede og ikke-parsede
Eksterne generelle entiteter – entiteter bestående af XML-kode eller tekst
(Tekstforkortere i eksterne filer – s. 58)
(Fordel: Hvis store, deling med andre, oprette dokument på baggrund af andre) **<description>**
- Parsede analyseres af XML-parseren – entiter indeholdende tekst
- Ikke-parsede analyseres ikke (ofte binære eller ikke tekst data – behøves ikke at indeholde tekst)
(s. 62-65) – billede, lyd, file el. Andre former for multimediefiler
(kun som en del af XML-dokumentets indhold

Parameterentiteter
Underopdeles i interne og eksterne
- Angiver om de er defineret inden for DTD'en eller i en ekstern fil
- Parses altid
- Ca mener interne har begrænset anvendelse (kan kun indeholde komplette erklæringer)

Eksterne parameterentiteter
- Forkortelser for dele af selve DTD'en (s.60)
- Bruge en anden persons standard-DTD
- Bruge standardiserede liste over entiteter (specialtegn med accent)

Interne parameterentiteter (superliga – hjemme og udehold)
<!ENTRY % entitetsnavn "(elementdefinition)">
<!ELEMENT hjemmehold %hold;>
- Elementer med samme elementdefinition kan således referere til en parameter entitet, der indeholder elementdefinitionen

XML Schema
Ca, kap. 5 (s.69)

XML er ikke blot sprog, det kan bruges til at oprette nye sprog
Et regelsæt til et sprog kaldes **schema**
Ikke obligatorisk at anvende men nyttigt værktøj
Hvis et dokument overholder reglerne i skemaet er dokumentet **gyldigt**
To måder at oprette schema på: DTD'er og XML Schema
DTD'er er gammeldags men udbredt, det anvender en speciel og begrænset syntaks
DTD er et schema hvis "schema" forsås meget generelt

Begrænsning/ulemper ved DTD
- Man kan ikke angive af bestemte data skal være tal, dato, tekst, ...(altså ingen typeangivelser)
- Ingen god måde til at angive et bestemt antal, se ? + *
- DTD syntaks har ikke meget med XML at gøre
- Alle DTD'er er globale, - kan ikke definere to med samme navn
= XML Schema er et forsøg på at løse alle disse problemer

XML Schema muliggøre
- Datatyper
- Lokale og globale elementer

- Dokument opdeles i to forskelling typer: **simpel og kompleks**
- Alle typer kan være navngivne eller anonyme (kun bruges indenfor erklæringselement)
Simple: Elementer der kun indeholder tekst (integer, string, dato m.v. – brugerdefinerede typer)
Komplekse: Elementer der indeholder andre elementer el. Attributter
- Elementer af den komplekse type vil ofte beskrive dokumentets struktur frem for indhold
- Attributter altid simple typer da de kun kan indeholde tekst
Der findes 4 grundlæggende former for komplekse typer:
- Elementer der kun indeholder andre elementer
- Elementer der kun indeholder tekst
- Elementer der både indeholder andre elementer og tekst
- Tomme elementer

Lokal/global erklæring – s. 71
- I Schema er konteksten meget vigtig
- Global erklæret: Alt der erklæres på schemas øverste niveau – kan anvendes overalt, fastsætter kun hvordan elementet ser ud ikke hvor det optræder (ekskl. rodelement!)
- Lokal erklæret: Indenfor et element (kontekst vigtig)

Schema.xsd
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsd:stylesheet xmlns:xs**l**="http://www.w3.org/1999/XSL/Transform" version="1.0">
:
</xsd:stylesheet>
```
(ex. xsl el. xsd – erklære namespace for "schemaet for schemaet")

- Angiv schemaets placering – nødvendig for nogle valideringsprogrammer – s. 73
- Kommentarer – til sig selv og andre s. 74
```
<xsd:annotation>
        <xsd:documentation>
                :
        </xsd:documentation>
</xsd:annotation>
```

Definition af simple typer
Ca. kap. 6 (s. 75)

- Kan kun indeholde elementer af en bestemt type (tekst). Anvend:
  - Foruddefineret simpel type (streng, boolean, url, tal, dato)
  - Afled egen brugerdefineret typedefinition (ex. tekst med mønster)

Simple typer
<xsl:element name=”elementnavn” type=”xsd:**string**”>
<xsl:element name=”elementnavn” type=”xsd:**decimal**”> - decimaltal
<xsl:element name=”elementnavn” type=”xsd:**boolean**”> - true | false | 1 | 0
<xsl:element name=”elementnavn” type=”xsd:**date**”> - s. 78-79 – mange typer
<xsl:element name=”elementnavn” type=”xsd:**time**”> - s. 78-79 – mange typer
<xsl:element name=”elementnavn” type=”xsd:**uri-Reference**”> - URL (der star uri!)
<xsl:element name=”elementnavn” type=”xsd:**language**”> - ISO639 2 cifret sprogkode
<xsl:element name=”elementnavn” type=”xsd:**brugerdef**”>
- Der finder andre indbyggede simple typer (s. 77) – ex. list datatyper
- Brugerdefinerede kan oprettes
- Indbyggede typer starter normalt med xsd: (der er undtagelser s. 128), dette går det nemt at skelne dem fra brugerdefinerede typer
- Elementnavn skal starte med bogstav el. _ for at være gyldigt!
Hjertekir -lysestager
Taltyper s. 80
<xsl:element name=”elementnavn” type=”xsd:**decimal**”> - 4,26, -100 el 0 – valgfri decimaltegn
<xsl:element name=”elementnavn” type=”xsd:**integer**”> - positive el. negative heltal
<xsl:element name=”elementnavn” type=”xsd:**positivInteger**”> - positive heltal
<xsl:element name=”elementnavn” type=”xsd:**negativInteger**”> - negative heltal
<xsl:element name=”elementnavn” type=”xsd:**nonPositivInteger**”> - positive heltal + 0
<xsl:element name=”elementnavn” type=”xsd:**nonNegativInteger**”> - negative heltal + 0
<xsl:element name=”elementnavn” type=”xsd:**float**”> - 32-bit flydende decimaltal, 0, m.m.!
<xsl:element name=”elementnavn” type=”xsd:**double**”> - 32-bit flydende decimaltal - præcision

<u>Brugerdefinerede typer</u>

Afledning af simple typer s. 81
<xsd:**simpleType** name=”typenavn”>
- Grundlæggende type :
<xsd:restriction base=”type ex. xsd:string”>
  - Angiv nogle *facetter*
</ xsd:restriction>
<xsd:simpleType>
- OBS: Skal henvises til som typenavn og ikke xsd:typenavn

- **Typenavn behøves ikke** hvis den kun skal benytted én gang:

```
<xsd:element name=”elementnavn”
    <xsd:simpleType>
        <xsd:restriction base=”xsd:string”>
                <xsd:pattern value=”\d{5}(-\d{4})?”/>
        </ xsd:restriction>
    <xsd:simpleType>
</xsd:element>
```

Angiv værdimængde

<xsd:simpleType name=”typenavn”>
    <xsd:restriction base=”xsd:string”> - kan bruges med alle typer pånær boolean
        <xsd:enumeration value=”Asia”/> - hver enkel værdi skal være unik
        <xsd:enumeration value=”Europe”/> - kan indeholde luft, blanke tegn, tab
    </ xsd:restriction>
<xsd:simpleType>

Angiv mønster

<xsd:simpleType name=”typenavn”>
    <xsd:restriction base=”xsd:string”>
        <xsd:pattern value=”\d{5}(-\d{4})?”/>
    </ xsd:restriction>
<xsd:simpleType>

/d vilkårligt ciffer
/D vilkårligt ikke-ciffer
/s vilkårligt blank tegn (tab, retur, mellemrum m.m.)
/S vilkårligt ikke blank tegn
x* 0 eller flere x - (xy)*
x? 0 eller 1 x – (xy)?
x+ 1 eller flere x – (xy)+
[abc] repræsentere a, b eller c
[0-9] repræsentere intervallet af værdier fra 0 til 9
[0-255] repræsentere intervallet af værdier fra 0, 1 el. 2 samt et 5 tal og et 5 tal
x|y repræsentere x ellery
x{5} repræsentere nøjagtig 5 xér
x{5,} repræsentere mindst 5 xér
x{5,8} repræsentere mindst 5 og højest 8 xér
x{5} repræsentere nøjagtig 5 xér
(ab){2}repræsentere nøjagtig 2 ab´er

- Angiv acceptable værdier – s. 86

<xsd:simpleType name=”typenavn”>
    <xsd:restriction base=”xsd:integer”>
        <xsd:minInclusive value =”8000”/>
        <xsd:maxInclusive value =”8999”/>
    </ xsd:restriction>
<xsd:simpleType>

- Samt **xsd:maxExclusive og xsd:minExclusive**
- Kan også bruges på xsd:date – s. 87

- Begræns længden af en simpel type

<xsd:simpleType name=”typenavn”>
    <xsd:restriction base=”xsd:string”>
        <xsd:length value=”4”/> - skal indeholde 4 tegn, bedre er et mønster
    </ xsd:restriction>
<xsd:simpleType>

- **<xsd:minLength value=”4”/> -** skal indeholde mindst 4 tegn
- **<xsd:maxLength value=”4”/> -** skal indeholde max. 4 tegn

- Begræns cifre i tal

<xsd:simpleType name=”typenavn”>
    <xsd:restriction base=”xsd:decimal”>
        <xsd:precision value=”4”/> - max. antal cifre i tallet
        <xsd:scale value=”2”/> - max. antal cifre til højre for decimaltegnet
    </ xsd:restriction>
<xsd:simpleType>

- Værdierne må ikke være negativ, 0 eller for precision
- Kan bruges for enhver taltype men ikke strenge

- Listetyper (liste af datoer, tal m.v.) – s. 90

<xsd:simpleType name=”typenavn-list”>
    <xsd:list base=”xsd:date>
        - evt. Begrænsninger: xsd:length, xsd:minLength, maxLength el. enumeration
<xsd:simpleType>

- Værdierne adskilles af mellemrum: <heltal>35 356 5 23</heltal>

- **Foruddefinering af elementers indhold – s. 91**

2 måder: Fastsætte en værdi eller tildele en værdi

<xsl:element name=”elementnavn” type=”xsd:string” **fixed=”fastværdi”**>

- Skal indeholde den faste værdi eller tomt (= implicit samme værdi!)

<xsl:element name=”elementnavn” type=”xsd:string” **default=”defaultværdi”**>

- Vil få tildelt defaultværdien selvom det ikke optræder

XML Schema - Definition af komplekse typer
Ca. kap. 7 (s. 93)

- Dokument opdeles i to forskelling typer: **simpel og kompleks**
- Alle typer kan være navngivne eller anonyme (kun bruges indenfor erklæringselement)

Komplekse: Elementer der indeholder andre elementer el. Attributter
- Elementer af den komplekse type vil ofte beskrive dokumentets struktur frem for indhold
- Attributter altid simple typer da de kun kan indeholde tekst

Der findes 4 grundlæggende former for komplekse typer:
- Elementer der kun indeholder andre elementer eller atributter (ikke tekst)
- Elementer der kun indeholder tekst = rene tekstelementer
- Elementer der både indeholder andre elementer og tekst = blandede elementer
- Tomme elementer evt. med attributter

Elementer der kun må indeholde elementer og evt. attributter

<xsd:**complexType** name=”typenavn”>
:
<xsd:**complexType**>
- Skal være: en sekvens, en valgmulighed, en ikke-ordnetgruppe el. en navngiven gruppe

Skal optræde i en sekvens
- Hvis et element skal indeholde andre elementer i en bestemt rækkefølge
- Kan indeholde (el. være indeholdet i) andre sekvenser, valgmuligheder el henvisninger til navngiven gruppe

```
<xsd:complexType name=”typenavn”>
        <xsd:sequence>
                <xsl:element name=”elementnavn” type=”xsd:string”>
                <xsl:element name=”elementnavn” type=”xsd:decimal”>
        </xsd:sequence>
<xsd:complexType>
```

Gruppe af valgmuligheder – s. 96
- Kan indeholde (el. være indeholdet i) andre valgmuligheder, sekvenser el. henvisninger til en navngiven gruppe

```
<xsd:complexType name=”typenavn”>
        <xsd:choice>
                <xsl:element name=”elementnavn” type=”xsd:string”>
                <xsl:element name=”elementnavn” type=”xsd:decimal”>
        </xsd:choice>
<xsd:complexType>
```

- <xsd:**choice minOccurs=”0” maxOccurs=”unbounded”**> - s. 101
- Default er min- og maxOccurs = 1

Ikke-ordnet gruppe
- Hvis et element må indeholde andre elmenter i vilkårlig rækkefølge

```
<xsd:complexType name=”typenavn”>
        <xsd:all>
                <xsl:element name=”elementnavn” type=”xsd:string”>
                <xsl:element name=”elementnavn” type=”xsd:decimal”>
        </xsd:all>
<xsd:complexType>
```

- <xsd:**choice minOccurs=”0” maxOccurs=”1”**> - kun 0 og 1 - s. 101
- Kan kun indeholde elementerklæringer el. henvisninger – ikke andre grupper!
- Intet element kan optræde mere end én gang
- Kan kun være indeholdt i en kompleks type el. navngiven gruppe og skal være det eneste underordnede element

Navngiven gruppe – s. 98
- Hvis en gruppe elementer altid optræder sammen
- En gruppe kan kun optræde på øverste niveau i et schema

```
<xsd:schema>
        <xsd:group name=”gruppenavn”>
                <xsl:element name=”elementnavn” type=”xsd:string”>
                <xsl:element name=”elementnavn” type=”xsd:decimal”>
        </xsd:group>
```

- Henvisning til en navngiven gruppe:

```
<xsd:complexType name=”typenavn”>
        <xsd:all>
                <xsl:group ref=”gruppenavn”/>
        </xsd:all>
<xsd:complexType>
```

- Henvisning til allerede definerede grupper

<xsd:element **ref=”navn”**>

- Man kan kun henvise til allerede definerede elementerklæringer i: sekvens, valgmulighed, ikke-ordnet gruppe el. i en navngivengruppe
- Du kan alle steder henvise til globalt erklærede elementer

Min- og maxOccurs – s. 101
- Fastsættelse af antal gange for element, sekvens, gruppe, ikke-ordnet gruppe eller navngiven gruppe opstår
- maxOccurs kan sættes til uendeligt – unbounded
- Kan ikke bruges sammen med globalt erklærede elementer
- Kan anvendes i: xsd:sequence, xsd:choice, xsd:all og i henvisning til navngiven gruppe

Elementer der kun må indeholde tekst
- Hvis man har brug for en bestemt simple type (tekst) som skal have atributter

```
<xsd:complexType name=”typenavn”>
        <xsd:simpleContent>
                - her kan tilføjes xsd:restriction, eller extension for at udbygge en type
                <xsd:extension base=”xsd:integer>
                        <xsd:attribute name=”attributnavn” type=”xsd:integer”/>
                <xsd:extension base=”xsd:integer>
        </xsd:simpleContent>
</xsd:complexType>
```

Tomme elementer
- Kan indeholde attributer ellers intet indhold mellem start og slut tag

```
<xsd:complexType name=”typenavn”>
        <xsd:complexContent>
                <xsd:restriction base=”xsd:anyType> - angiver basseret på ingen anden type
                        <xsd:attribute name=”attributnavn” type=”xsd:integer”/>
                <xsd:restriction>
        </xsd:complexContent>
</xsd:complexType>
```

Elementer med blandet indhold

```
<xsd:complexType name=”typenavn” mixed=”true”>
        :
</xsd:complexType>
```

- mixed=”true” for at angive at elementet kan indeholde elementer, attributer **og tekst**

Komplekse typer baseret på andre komplekse typer – s. 105
- Bygger videre på en anden kompleks type

```
<xsd:complexType name=”typenavn”>
        <xsd:complexContent>
                <xsd:extension base=”anden komplekt type”>
                        :
                <xsd:extension>
                - eller angiv <xsd:restriction>
        </xsd:complexContent>
</xsd:complexType>
```

- **Typenavn behøves ikke** hvis den kun skal benytted én gang: = **anonyme komplekse typer** - 107

```
<xsd:element name=”elementnavn”
    <xsd:compleType>
        :
    <xsd:compleType>
</xsd:element>
```

Atributter – s. 109
- Altid en simple type og skal erklæres til sidst i en kompleks type
- Attribut obligatorisk: use=”required – kan også tilføje value=”tvang” for fast værdi
- Attribut ikke optræder: use=”prohibited” – doc kun gyldigt hvis attribut ikke optræder (use=”optional” er standardværdi)

```
<xsd:complexType name=”typenavn”>
        <xsd:complexContent>
                :
                <xsd:attributename=”attributnavn” type=”xsd:string use=”required”>
        </xsd:complexContent>
</xsd:complexType>
```

- Attributværdi fastsættes på forhånd:

<xsd:attributename=”attributnavn” type=”xsd:string use=”fixed” value=”hej”>
<xsd:attributename=”attributnavn” type=”xsd:string use=”default” value=”hej”>

Attributgrupper – s. 111

XML Schema – Anvendelse af namespaces
Ca. kap. 8 (s. 113)

- Kombinere dokumenter med andres hvor man har erklæret nogle ens navne for globale elementer
- Hvis man kombinere vil data i dette element blive blandet og meningsløst
- **Løsning:** Oprette et superlable (unikt!? – registreret vedvarende URL)
- Opbygning af namespace s. 114

- Erklære standardnamespace – s. 115

<source id=”2” xmlIns=”din unike url”>

- Erklære specielle individuelle elementer uden at påvirke underordnede elementer – s. 116

<source id=”2” xmlIns=”din unike url” **xmlIns:navn=”din anden url”>**
- Prefix kan ikke starte med xml...
- XML-processoren betrakter det som en integreret del af navnet

<din unike url:element></din unike url:element>

<din anden url:element></din anden url:element>

- Attributter og namespace – s. 119
- Næsten altid irrelevant, **men** standardnamespaces gælder ikke for attributter (lokalt virkefelt – identificeres af det element der indeholder den

XSLT
Ca. kap. 10 (s. 135) – emne oprette og anvende skabeloner

- Visning af XML dokumenter
- Oprindeligt via. XSL (Extensible Stylesheet Language) men tog for lang tid, derfor opdelt i to dele XSLT (Transformation) og XSL-FO (Formatting Objects)

Simpel måde at vise XML = stylesheet (CSS og XML transformationer – ofte sammen)
- CSS: Regler for hvordan elementer viser (deklarativt sprog) (den egentlige formatering)
- CSS: Ingen mulighed for: tilføje, oprette som tabel – kan gøres med data island (simpel) men andre ord vise xml dokument i html-dokumenter
- CSS: Ikke styre forløbene i rekursion – kr s. 95
- Med XSLT får man langt flere muligheder, kan generere et hvilket som helst dokument
- Ikke kun transformationer men også XPath og XSL-FO (output i trykt format)
- Have lige så mange hierarkier og elementer man har lyst til (eks. kr, s. 85)

XSLT kan bruges til at omstrukturere outputtet efter forskellige kriterier
- træ-til-træ transformationen (til HTML, WML m.m.)

Eksempler
- kr, s. 88 – mappestruktur (mappe | fil – rekursivt)
- kr, s. 99 - produkter
- kr, s. 107 – produkt via. server

Transformere et XML-dokument indeholder:
- Analyser indhold
- Udfør forskellige handlinger alt efter hvilke elementer
- **Til at udføre denne transformation skal bruges en XSLT-processor**
    1. Analysere indhold
    2. Konvertere til et nodetræ (hierarkisk repræsentation af XML-dokumentet
    3. Ser i et XSLT style sheet for at se hvad den skal gøre med noderne (skabeloner)
    4. Søger efter rod-skabelonen
    5. Nodesæt og skabeloner identificeres og udvælges vha. **udtryk og mønstre**

(bevæger sig gennem stylesheet og XML-dokument en node ad gangen)

Xsl:apply-templates – identifiserer en nodegruppe (nodesæt)
- XSLT kan bruges til at konvertere stort set alle dokumenter (til andre)

3 måder at udfører transformationen på
Anvende browser (skal understøtte XML og XSL)
Udføre et script på klientsiden (ex. JavaScript) (brower skal indeholde en XML-parser)
Udføre ASP script på serversiden (localhost/cd.asp)

- Hvis vi lader browseren klare transformeringen af et document med given XSLT-stylesheet
  <?xml-stylesheet type=”text/xsl” href=”til_html_vis_alt.xsl”?>
- Ellers klarere serveren det, så modtager browseren resultatet, 3 fordele:
  Browser skal ikke kunne understøtte XML og XSL
  Serveren vil kunne detektere hvilken type browser der anvendes
  Sikring af source – at bruger ikke bare kan vælge ’vis kilde’ og se alt
  Transformationen sker kun når data ændre sig
  - dette kræver DOM (transformering i ASP-script – kr s. 104

Hvorfor?
- Transformation til præsentationsformål
- Adskil data og præsentation
- Lav en transformation af data til modtagerens format

- Tilføj/fjerne elementer

To typer komponenter i et XSLT style sheet – s. 140
- Instruktioner – Angiver hvordan XML-dokumentet bliver transformeret
- Konstanter – Typisk HTML, optræder i nøjagtig samme form i outputtet

Start
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method=”html”> - kr s. 91: ex. encoding og indent

Andre
<xsl:apply-template select=”disk”/> - Angiver at template til disk skal kaldes
<xsl:value-of select=”et eller andet”> - indsætter værdi fra source dokumentet

Rodskabelon – s. 139
<xsl:template match=”/”> - / er mønstret til rodelementet
- XSLT-processoren er ligeglad hvor i dit XSLT style sheet skabelonen er placeret
- Hvis rodskabelon ikke finden anvendes en indbygget skabelon
    <xsl:template><xsl:applytemplates/></xsl:template>
- I rodskabelonen oprettes strukturen (HTML) for det færdige transformerede dokument
- HTML skal overholode XML-regler for velformethed

Givne node
- Når HTML-koden er oprettet for en given node skal indholdet/strengværdien indsættes
    <xsl:value-of select=”udtryk”>
- **select=”.”** For at skrive den aktuelle nodes indhold
- Hvis det nodesæt der identificeres indeholder flere end en node er det kun den første nodes strengværdi der udskrives
- Har noden underelementer omfatter srengværdien også denne tekst
- Boolsk udtryk retuneres som ”true” el. ”false”
- Der kan oprettes udtryk der beregner

Skabelonregler
- Moduler der beskriver hvordan en bestemt XML-kode skal udskrives
- Består af tre dele:
    Starttag – beskriver hvilke dele af XML-dokumentet skabelonen skal udføres på
    Midterste del beskriver hvad som skal ske
    Sluttag – afslutter skabelonen
- Oprettelse:

<xsl:template match=”mønster”>
    - Angiver hvad som skal ske
</xsl:template>
- Kun rodskabelonen kaldes automatisk andre skabeloner skal kaldes manuelt
- Rækkefølge af skabeloner er underordnet, bestemmes af **xsl:apply-templates-elemenernes** rækkefølge
- Ikke nok at oprette skabelon – den skal også kaldes, - sådan kaldes/anvendes en skabelog

<xsl:apply-templates select=”udtryk”/>
- Angives selectdelen ikke anvendes skabelonen på hver eneste efterkommer-node
- apply-templates søger efter mest passende skabelonregel

Batchbehandling af noder – s. 146
<xsl:for-each select =”udtryk”>
</xsl:for-each>

- Bruges typisk til at oprette HTM-tabeller

Betinget behandling af noder – s. 148
<xsl:if test="udtryk">
</xsl:if>
- Et nodesæt betraktes som sant hvis ikke det er tomt

Alternative betinkelser
<xsl:**choose**>
    <xsl:when test=="udtryk">
        :
    </xsl:when>
    <xsl:otherwise>
        :
    </xsl:otherwise>
</xsl:choose>

Sortering af noder inden de behandles – s. 150

Generering af attributter – s. 151

Xpath: Mønstre og udtryk – s. 153
Ca, kap. 11

XPath er en syntaks til at adressere bestemte elementer
- Når skabeloner anvendes sker det ved at man angiver at en gruppe noder skal behandles af **de relevante skabeloner**
- Når skabelonen oprettes angives mønster for at udvælge bestemte noder
- Både mønstre og udtryk skrives vha. XPath
- **Forskel:** - Mønstre er kontekstfri, dvs. Mønster "name" svare til ethvert nameelement (placering underordnet. **Modsat** kan udtryk kun evalueres ved at undersøge den kontekst de tilhører
- Den aktuelle node angives i skabelonens match-attribut
- select="." For at skrive den aktuelle nodes indhold

- Valg af **underordnet node**, skriv blot dens navn **uden fuldstændig sti** (/ kan også angives) – s. 156 -
- Valg af **overordnede/sideordnede noder** – s. 157
- Kun hvis forholdet ml. aktuelle node og ønskede node er klart
<xsl:value-of select="../[@language="English"]"/> - aflæs overordnede nodes attribut
<xsl:value-of select="/*/u_sideordnet"/> - udvælg alle alle underordnede elementer for alle sideordnede elementer
// - nyttig hvis du har behov for at vælge **alle efterkommere** af en bestemt node
.// - Alle **aktuelle nodes efterkommere**
- Ovenover benyttes aktuelle node, det kan ignoreres ved at angeve **fuldstændig sti**
- **Attributter angives med @ - s. 160**
- **Delmængde** udvælges ved at benytte **[og]** efter navnet på det ønskede
<xsl:value-of select="../[@language="English"]"/> - aflæs overordnede nodes attribut
<xsl:value-of select="../[@language="English"][position()=last()]"/>
- Flere prædikater: - her findes den sidste node som har languagr="English"

XSTL: Test med udtryk og funktioner
Ca, kap. 12

- Der kan udføres en eller flere handlinger på en streng inden den udskrives

Sammenligning at to værdier
= =
<> !=
< <
<= <=
> >
>= >=

```
<xsl:template macth="alder">
    <xsl:choose>
        <xsl:when test=".=0">
            :
        <xsl:when test=".&gt;0 and .&lt; 50">
```

Nodes placering
<xsl:when test="**position()=last()**"> - den sidste underordnede node
<xsl:when test="**undernode[1]**"> - den første underordnede node

Beregn sum
<xsl:value-of select=**"sum(antal)"**/>

Optælling af noder
<xsl:value-of select=**"count(undernoder)"**/>

+ - * div
<xsl:value-of select=**". div sum(../../subspec>antal)*100/>**

Formatering at tal – s. 169
<xsl:value-of select=**"format-number(. div sum(../../subspec>antal)*100), '##0.0%')/>**
- 0 for hver ciffer der skal vises
- # for hver ciffer der kun skal vises hvis der er <> 0
- #.##0,00 – mindst et tal og punktum for hver tredje ciffer

Afrunding af tal – s. 170
<xsl:value-of select=**"celling**(./@x div 2)"/> - rund op til nærmeste heltal
<xsl:value-of select=**"floor**(./@x div 2)"/> - runde ned til nærmeste heltal
<xsl:value-of select=**"round**(./@x div 2)"/> - afrunde til nærmeste heltal

Delstrenge – s. 171
<xsl:value-of select=**"substring**(.,1fra,1antal)"/> - udskriver første bogstav af aktuel node
- substring-after
- substring-before

Store og små bogstaver s. 173

XML – del 2, kr s. 117
- Data-til-data transformation / struktureret transformation
- Lave output-dokument med samme vokabularer, hvis fx. værdier ændres ud fra andre værdier
- To hovedområder
    1. Mapping af vokabularer, transformation fra et til et andet
    2. Logik uden ændring af vokabular, fjerne/tilføje bestemte elementer

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

<!-- Match dokument-roden. -->
<xsl:template match="/">
  <xsl:apply-templates select="*|processing-instruction()"/>
```
- **Angiver** at XSLT-parseren her skal udfører templates for alle elementer (*) **og processing-instructions** (noder = elementer + PI, kommentarer, attributter)
```
 </xsl:template>

<!-- Match alle andre noder. -->
<xsl:template match="*|@*|text()|comment()|processing-instruction()">
  <xsl:copy>
    <xsl:apply-templates select="*|@*|text()|comment()|processing-instruction()"/>
```
- Processing-Instructions*; ex. <?xml-stylesheet type= …>
```
  </xsl:copy>
 </xsl:template>
```

- Mulige templates søges fra bunden, første match medfører at parseren betrakter elementet som transformeret og gør ikke mere (FORKERT)
- Når XSLT-processoren læser element <xsl:apply-templates> fremfindes hele sættet at secetede noder, hvorpå kun den bedste (mest specifikke) skabelon benyttes
    - Findes en matchende ikke benyttes en indbygget skabelon

ASP script der udfører transformeringen – skrivTilFil
- problem hermed er at der sker tilføjelse hver gang scriptet udføres (se tilføjelse og fjernelse)

- Følgende template udførres for alle kampe!
```
<xsl:template match="kamp">
  <xsl:copy>
    <xsl:apply-templates select="*|@tilskuere"/>
    <xsl:element name="{name()}"><xsl:value-of select="."/></xsl:element>
  </xsl:copy>
 </xsl:template>
```
- Hvis kun der ønskes for nogle kan angivet [og]
```
<!-- Match kamp-elementer. -->
<xsl:template match="kamp[antal &gt;= 500]">
  <xsl:copy>
    <xsl:apply-templates select="*|@tilskuere"/>
    <xsl:element name="{name()}"><xsl:value-of select="."/></xsl:element>
  </xsl:copy>
 </xsl:template>
```

Tilføjelse og fjernelse af elementer
- kr s. 126-127 – godt pga. deklarativ regelfortolkning

XSLT-stylesheet til generering af X-diagrammer – s. 131 kr

Egen funktioner – s. 136
- De regler og den business-logik man kan lave med XPath er ikke altid nok
- I stedet for at bruge value-of til udregninger kan laves funktioner
- Selvom XSL-transformationer egentlig er deklarative kan laves procedural udvikling via. funk.
- Eks. kr s. 136

- For at kunne lave funktioner skal <xsl:stylesheet> udvides med
  xmlns:msxsl=”urn:schemas-microsoft-com:xslt”
  xmlns:user=”urn:mit.eget.namespace”
- Fortæller parseren at den skal kunne arbejde med udvidelser til XSLT

Brug af funktion
<xsl:value-of select=”user:minFunktion(pris)”/>

Variable i XSLT
<xsl:variable name=”varnavn” select=”varværdi”/>
- brug:
<xsl:if test=”**$varnavn**”> - reference til variabel angives med $
- Var kan kun bruges i det scope den er oprettet i, men kan oprettes globalt

DOM
kr kap.5

- XSLT er deklarativ, dvs. vi deklarere hvad vi ønsker udført
- Sommetider er direkte manipulering nødvendig – ex. bestilling
- **Direkte manipulering kan ske med DOM – Document Object Model**

- DOM er en **objectmodel** (et givent XML-dokuments objektmodel)
- Består af objekter med metoder og properties
- Et hierarki af objekter kaldes noder

Specifikation
Et platform- og sprogneutralt interfase, som muliggøre dynamisk ændring af indhold, stryktur og udseende

- DOM giver adgang til standardiseret set af objekter der kan repræsentere HTML- og XML-doc.
- Standardiseret interface til at tilgå og manipulere dem
- Giver progarmatisk adgang til dokuments node-hierarki samt til API

Forskeld ml. XSLT og DOM
- XSLT er deklarativt, DOM er procedural
- Med XSLT laves et helt nyt dokument, med DOM manipuleres
- Direkte manipulering med DOM

- XML-dokument består og tekst og kan manipuleres som sådan, men ikke særlig hensigtsmæssigt kræver masser af arbejde og risikere stor risiko for at velformethed forsvinder

- **Med DOM kan XML-doc manipuleres på en abstrakt model** der indeholder hierarki af noder
- Tænk på hver node som et object med programerbar grænseflade (via properties og metoder)
- Ala API - standardiseret og dokumenteret grænseflade

- Det er XML-parseren som doc indlæses i der sørger for at hver node kan tilgår via DOM-API'et
- Appendiks indeholder en liste over properties og metoder
- XML-parseren opbygger hierarki i hukommelsen hvilket manipulere via p og m

- X-diagram med DOM – s. 149

Læsning af værdier
XPath fungere på samme måde men direkte 'springe rundt' – s. 155
- Eksempel på s. 156 (fordel: arbejder direkte på det refererede objekt)

```
var strXPath = "/produkt"
var oNode = oSource.selectSingleNode(strXPath);
```

Referencer til flere noder – s. 157
- Noder kan behandles som i et array

```
var strXPath = "/produkt"
var oNode = oSource.selectNode(strXPath);

for (var iNode=0; iNode<oNoder.length; iNoder++) {
  Response.Write("<br>" + oNoder[iNode].text);
}
```

Ændring af node-værdier – s.158
- Kan ændres direkte da selectSingleNode() og selectNodes() retunere referenser og ikke en kopi

Ændre struktur – tilføj node s. 161
- Først tilføjes et nyt element med appendChild()
var oNyNode = oSource.createNode(1, ”kommentar”, ””)
- parameter – 1. = type 2. = nodenavn 3. = værdi
oNyNode.text = ”abc”;
oNode.appendChild(oNyNode)
- Placeres under det selectede

Ændre struktur – slet node s. 165
oNode.removeChild(oSletNode)

Andre strukturændringer – s. 167
- replaceChild()

Spangsbjerg møllevej 62, lej. F7 – gårdkollegiet
Tlf. 97115157
79

11L1574 ibm corp c 1998
Ibm copyright 1998

Stik
Foxconn
Dvi 1208
88.9042