

## XSL

---

### Slide 1

- Varför XSL?
- **XSL**: tre standarder
- **XMLs dokumentmodell**:  
träd, noder, noduppsättningar  
och relationer
- Traversering av XML-trädet
- XSLT i praktiken
  - XSL-element
  - XPath-uttryck
  - Några goda råd

## Transformation: varför

- TEI är endast ett uppmärkningsschema. Det är inte specificerat hur de olika TEI-elementen skall placeras ut på en bildskärm eller på en bit papper.
- För (X)HTML har däremot varje webbläsare stöd och en möjlig lösning är att transformera TEI-kodningen till XHTML. Den resulterande presentationen är då ett *derivat* av originalet (TEI-källan).
- **XSL** är en teknik för detta ändamål.

### Slide 2

# XSL

## eXtensible Stylesheet Language

- **XPath** – en teknik för att *referera* till specifika noder i ett dokumentträd

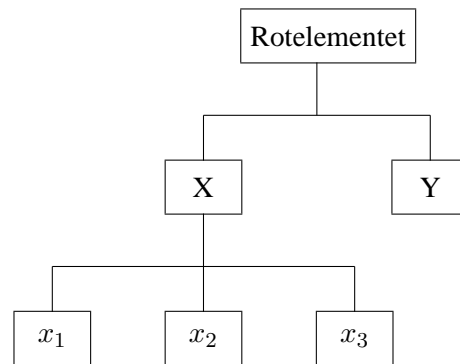
### Slide 3

- **XSLT** (XSL Transformations)– en teknik för att *omvandla* ett dokumentträd
- **XSL-FO** (XSL Formatting Objects) – en teknik för att precisera hur noder skall *presenteras* i olika medier (jfr. CSS)

## XMLs dokumentmodell

### Slide 4

- Ett dokument modelleras primärt som ett träd och inte som en sekvens, dvs som en struktur av element med relationer (barn, förälder, ättlingar) och inte som en ordnad följd av element
- Ett dokument är en struktur ur vilken innehåll kan hämtas och vidare bearbetas

**Slide 5**

- 1)  $x_1, x_2, x_3$  är barn (*child*) till  $X$
- 2)  $X$  är förälder (*parent*) till  $x_1, x_2, x_3$
3.  $x_1, x_2, x_3$  är syskon (*sibling*)
- 4)  $x_1$  är "äldre" syskon (*preceding-sibling*) till  $x_2$
- 5) Rotelementet är förfader (*ancestor*) till  $x_1, x_2, x_3$
- 6)  $x_1, x_2, x_3$  är ättlingar (*desendant*) till Rotelementet och till  $X$

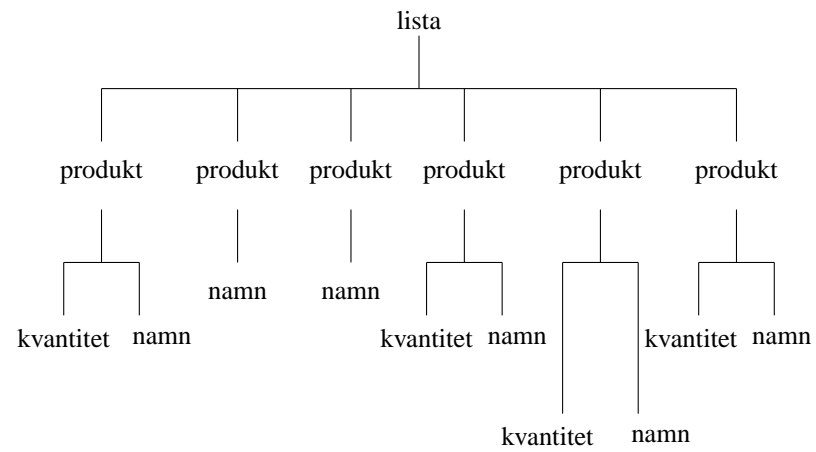
## Exempel

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE lista SYSTEM "lista.dtd">
<lista>
  <produkt><kvantitet>1 l</kvantitet>
    <namn>Mjölk</namn></produkt>
  <produkt><namn>Kaffe</namn></produkt>
  <produkt><namn>Bröd</namn></produkt>
  <produkt><kvantitet>3 dl</kvantitet>
    <namn>Grädde</namn></produkt>
  <produkt><kvantitet>1 paket</kvantitet>
    <namn>jäst</namn></produkt>
  <produkt><kvantitet>2 kg</kvantitet>
    <namn>Mjöl</namn></produkt>
</lista>
```

**Slide 6**

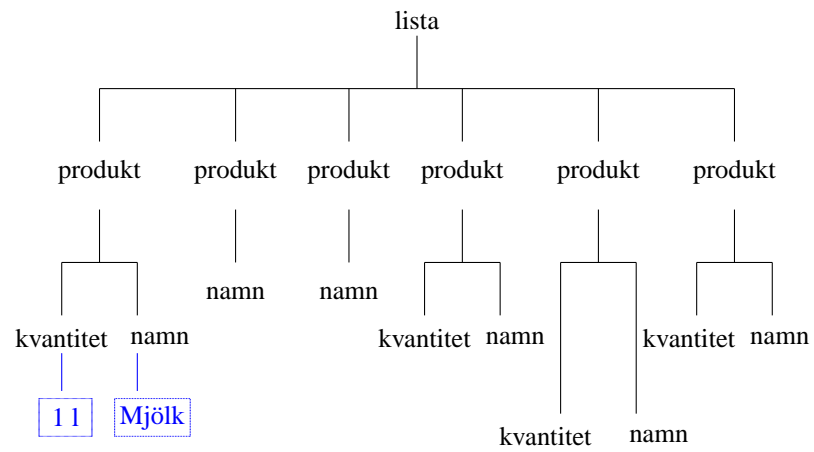
## Dokumentträd (endast elementnoder)

Slide 7



## Dokumentträd (med tillägg av två textnoder)

Slide 8



## 7 typer av noder

### Slide 9

- Element `<lista> ... </lista>`
- Text ...
- Attribut `<lista type='shopping'> ... </lista>`
- Kommentarer `<!-- ... -->`
- Bearbetningsinstruktioner `<? ... ?>`
- Namnrymdeklaration `<lista xmlns='...'> ... </lista>`
- Rotnoden

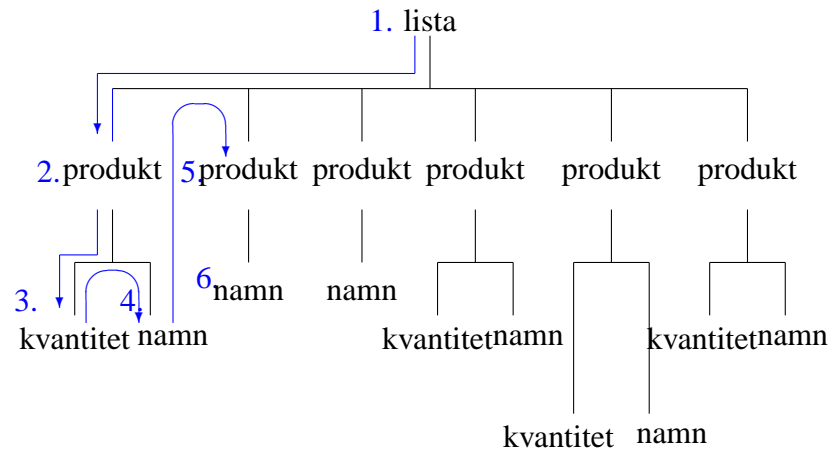
# Traversering

## Slide 10

- XSL förutsätter en xml-processor (MSXML, XALAN, XT etc) – ett program som kan *traversera* xml-trädet och göra något med de noder det stöter på, exempelvis omvandla ett tei-element till ett xhtml-element.
- Givetvis måste detta ske i en speciell ordningsföljd: xml-processorn gör en på-djupet-först (depth-first) sökning.

# Traversering av dokumentträd

Slide 11



## XSLT och XPath

### Slide 12

- **XSLT** förser oss med medel för att göra vissa saker med innehållet i ett xml-träd. XSLT är en xml-tillämpning med drygt 30 fördefinierade element.
- **XPath** i kombination med XSLT gör det möjligt att precisera vad i xml-trädet som skall bearbetas, dvs vilka noder som det skall hända något med.
- Som regel vill vi med ett visst XPath-uttryck tillsammans med ett xsl-element antingen 1) 'grabba' tag i en uppsättning noder, och/eller 2) generera innehåll utifrån källdokumentet.

## Tre metoder att tillämpa XSL på

### Slide 13

- **on-the-fly** – xsl-mallen appliceras på xml-dokument i samma stund som en webbläsare läser in xml-dokumentet. (Internet Explorer)
- **batch** – xsl-mallen associeras med och appliceras på ett xml-dokument via en fristående xsl-processor (Xalan i JEdit)
- En server hämtar xml-fil och xsl-fil och genererar utdata till en webbläsare.

## XSLT-mall

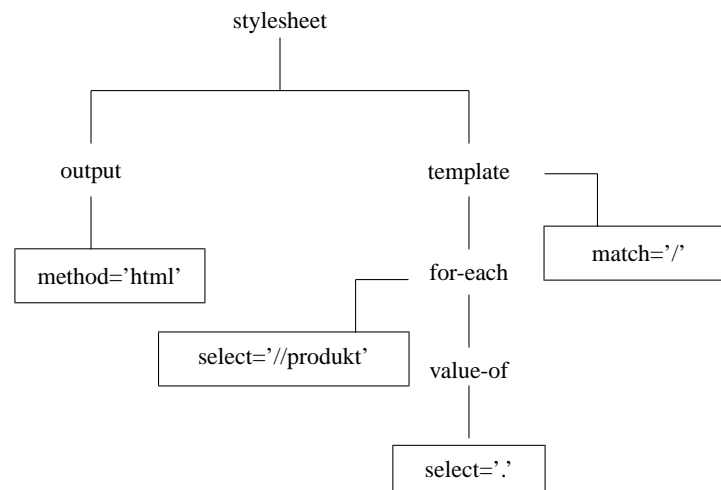
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:for-each select="//produkt">
      <p>
        <xsl:value-of select="."/>
      </p>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

### Slide 14

OBS!!! Notera de tre XPath-uttrycken och att varje xslt-märke inleds med xsl :

## XSLT-mall element och attribut

Slide 15



## Resultande XHTML

```
<p>1 l
      Mj&ouml;l k</p>
<p>Kaffe</p>
<p>Br&ouml;d</p>
<p>3 dl
      Gr&auml;dde</p>
```

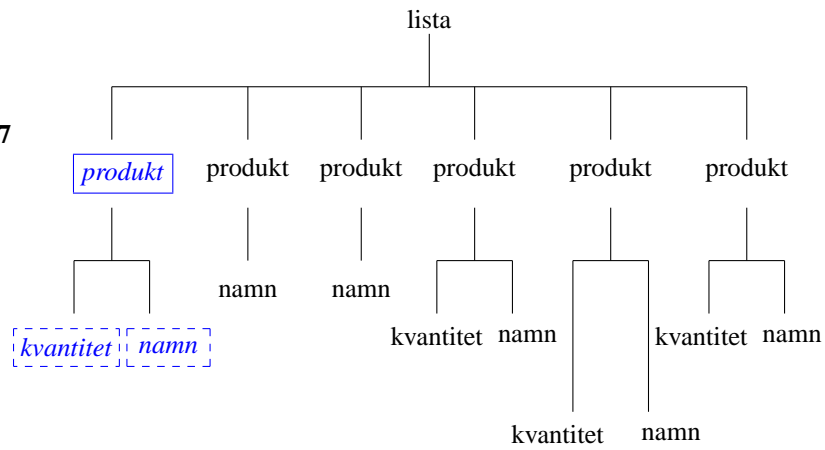
**Slide 16** <p>1 paket

```
      j&auml;st</p>
<p>2 kg
      Mj&ouml;l</p>
```

Notera hur å, ä och ö transformerats till entitetsreferenser. Notera också hur XPath-uttrycket `//produkt` i xslt-mallen resulterar i en noduppsättning om 6 stycken noder, varför vi också får sex stycken p-element. Om vi ersätter `//produkt` med `//produkt[position()=1]` resulterar det i att vi endast genererar `<p>1 l Mj&ouml;l k</p>`.

//produkt[position()=1]

Slide 17



## Hands-on (10 min)

### Slide 18

1. Starta JEDIT
2. Skapa två filer `x.xml` och `x.xsl` med xml-dokumentet respektive xslt-mallen som innehåll. Kopiera texten från pdf-presentationen.
3. Aktivera xslt-processorn och applicera xslt-mallen på xml-dokumentet

**Slide 19**

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3   version="1.0">
4   <xsl:output method="html"/>
5   <xsl:template match="/">
6     <html>
7       <head>
8         <title>
9           Inköpslista
10        </title>
11      </head>
12      <body>
13        <xsl:apply-templates/>
14      </body>
15    </html>
16  </xsl:template>
17  <xsl:template match="produkt">
18    <p>
19      <xsl:value-of select="."/>
20    </p>
21  </xsl:template>
22 </xsl:stylesheet>
```

## Nodtyper med respektive XPath-uttryck

Slide 20

Nodtyp	XPath	Matchar
Element	<code>/lista</code>	<code>&lt;lista&gt; ... &lt;/lista&gt;</code>
Text	<code>lista/text()</code>	<code>...</code>
Attribut	<code>lista/@type</code>	<code>&lt;lista type='shopping'&gt; ... &lt;/lista&gt;</code>
Kommentarer	<code>comment()</code>	<code>&lt;!-- ... --&gt;</code>
Bearbetningsinstruktioner	<code>processing-instruction()</code>	<code>&lt;? ... ?&gt;</code>
Namnrymdeklaration	<code>//namespace::node()</code>	<code>&lt;lista xmlns='...'&gt; ... &lt;/lista&gt;</code>
Rotnoden	<code>/</code>	

## Funktioner

“Som regel vill vi med ett visst XPath-uttryck tillsammans med ett xsl-element antingen 1) ’grabba’ tag i en uppsättning noder, och/eller 2) generera innehåll utifrån källdokumentet.”

### Slide 21

- `<xsl:template match="//TEI.2/text"> ... </xsl:template>`  
ger en uppsättning noder, en del av trädet, men ...
- `<xsl:value-of select="count(//text)"/>`  
`<xsl:text>,</xsl:text> <xsl:value-of`  
`select="substring(5678910,3,2)"/>`  
ger sannolikt 1,78

## Några goda råd

### Slide 22

- Börja med att definiera en “template” för rotnoden där du lägger den grundläggande xhtml-strukturen och ett eller flera `apply-templates` i xhtml-strukturens `body`
- Definiera sedan “templates” för varje element du räknar med att använda och som du vill ge en särskild presentation.
- Infoga ett `apply-templates` i templates för alla element som har ett innehåll du vill generera. `<xsl:apply-templates/>` får samma effekt som `<xsl:value-of select="."/>` i det tidigare exemplet

## Två alternativ.

### Slide 23

1. Spel utan handikapp: Skriv en mall från början till slut
2. Ett visst handikapp: Utgå ifrån

`http://www.adm.hb.se/~mg/dig/digtext.xsl`