Tehnici avansate de programare Curs 2

Cristian Frăsinaru

acf@infoiasi.ro

Facultatea de Informatică

Universitatea "Al. I. Cuza" laşi

Transformarea documentelor XML

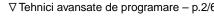
Ce este XSL ?

- Ce este XSL ?
- Limbajul XSLT

- Ce este XSL ?
- Limbajul XSLT
- Limbajul XPath

- Ce este XSL ?
- Limbajul XSLT
- Limbajul XPath
- Exemple de utilizare XSLT şi XPath

- Ce este XSL ?
- Limbajul XSLT
- Limbajul XPath
- Exemple de utilizare XSLT şi XPath
- Limbajul XQuery



- Ce este XSL ?
- Limbajul XSLT
- Limbajul XPath
- Exemple de utilizare XSLT şi XPath
- Limbajul XQuery
- Java şi XSLT: TrAX

XSL

Ce este XSL

Extensible Stylesheet Language

Limbaj pentru definirea de stylesheet-uri pentru documente XML

CSS styleheet-uri pentru HTML ⇔ XSL styleheet-uri pentru XML

Componentele XSL sunt:

- XSLT (Transformation)
- XPath
- XSL FO (Formatting Objects)

Limbajul XSL

- Un limbaj de transformare XML în HTML.
- Un limbaj capabil să filtreze şi să ordoneze date XML.
- Un limbaj ce poate formata un document XML pe baza datelor conţinute, cum ar fi afişarea numerelor negative cu roşu, etc.
- Un limbaj ce poate transmite datele XML către diverse dispozitive de ieşire.

XSL este un standard W3C



Ce este XSLT?

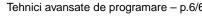
Limbaj funcţional folosit pentru a procesa informaţii structurate.

Rolul său:

- Transformare: XML XML, respectiv
- Prezentare: XML HTML, PDF, text, ...

Arbore sursă → Arbore destinaţie

XSLT + XML ⇔ Proceduri stocate + BD



XSLT ca limbaj de programare

Ca limbaj de programare, XSLT permite:

- Efectuarea de calcule
- Rearanjarea sau sortarea datelor
- Combinarea datelor din surse multiple
- Personalizarea afişarii, etc.

Specificaţiile complete ale limbajului:

http://www.w3.org/TR/xslt



XSLT este în format XML

Structura generală a unui document este:

Unui fisier XML îi poate fi asociată o transformare XSLT astfel:

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

"Pattern matching"

Paradigma de procesare este "pattern matching".

Un document XSLT = mulţime de *reguli de potrivire* (şabloane, template-uri) de forma:

"dacă intrarea este de această formă - generează următorul rezultat".

Ordinea de aplicare a regulilor depinde de implementare.

Procesarea nu se face linie cu linie, ca în cazul procesoarelor de texte, ci la nivel de nod.

Procesoare XSLT

- JAXP (Sun)
- Xalan (Apache)
- MSXML (Microsoft parserul inclus în suita Internet Explorer)
- System.xml: (Microsoft parserul inclus în .NET)
- Oracle-J (Oracle)
- XT (James Clark)

Locul procesării poate fi atât la nivelul clientului cât și la nivelul serverului.



Sintaxa XSLT

Limbajul XSLT pune la dispoziție elemente pentru:

- Definirea regulilor de transformare (şabloane)
- Crearea arborelui rezultat
- Definirea de variabile şi parametri
- Procesare repetitivă, condiţională, sortare
- Includerea altor fişiere cu transformări

XSLT foloseşte expresii XPath



Şabloane

xsl:template

```
<xsl:template match="fragment-intrare">
  <!-- fragment-iesire -->
  </xsl:template>
```

xsl:apply-templates

```
<xsl:apply-templates/>
<xsl:apply-templates select="persoana"/>
<xsl:apply-templates select="nume|prenume|adresa"/>
```

Şabloane

xsl:call-template

Crearea arborelui rezultat

xsl:value-of <xsl:value-of select="expresieXPath" /> xsl:copy-of <xsl:value-of select="expresieXPath" /> Transformarea identica: <xsl:template match="/"> <xsl:copy-of select="."/> </xsl:template>

xsl:copy

Crearea de noi elemente

xsl:element, xsl:attribute

```
<xsl:element name="tag">
  <xsl:attribute name="atribut1">
    <!-- Valoare atribut1>
  </xsl:attribute>
  <!-- Continutul elementului>
</xsl:element>
<xsl:template name="htmLink">
  <xsl:param name="dest" select="UNDEFINED"/>
  <xsl:element name="a">
    <xsl:attribute name="href">
      <xsl:value-of select="$dest"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

Crearea de texte și comentarii

xsl:text

<xsl:text> Java si XML </xsl:text>

xsl:comment

<xsl:comment>
Acest fisier este generat automat.
</xsl:comment>

va crea următorul comentariu:

<!--Acest fisier este generat automat.-->

Crearea de instrucțiuni de procesare

xsl:processing-instruction

```
<xsl:processing-instruction name="xml-stylesheet">
   href="book.css" type="text/css"
</xsl:processing-instruction>
```

va crea instrucţiunea de procesare:

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

Definirea de variabile

xsl:variable

```
<xsl:variable name="numeVariabila" select="expresieXPath" />
```

Referirea variabilelor: \$numeVariabila

Procesarea repetitivă

xsl:for-each

```
<xsl:for-each select="expresieXPath">
    ...
</xsl:for-each>
```

Să considerăm un document XML cu structura:

Exemplu de utilizare for-each

```
<xsl:template match="/">
 <html><body>
 Nr.
 Nume
 Data nasterii
 <xsl:for-each select="/agenda/persoana">
  <t.r>
  <xsl:value-of select="position()"/>
  <xsl:value-of select="nume"/>
  <xsl:value-of select="datan"/>
  </xsl:for-each>
 </body></html>
</xsl:template>
```

Sortarea

xsl:sort

```
<xsl:sort select="element" order="ascending|descending"/>
```

Putem sorta agenda după diverse criterii:

```
<xsl:for-each select="/agenda/persoana">
    <!-- Sortare dupa nume -->
    <xsl:sort select="nume" order="ascending"/>
        ...
</xsl:for-each>

<xsl:for-each select="/agenda/persoana">
        <!-- Sortare dupa data nasterii -->
        <xsl:sort select="substring(datan,7,2)" order="descending"/>
        <xsl:sort select="substring(datan,4,2)"/>
        <xsl:sort select="substring(datan,1,2)"/>
        ...
</xsl:for-each>
```

Procesarea condițională: if

xsl:if

```
<xsl:if test="conditie">
    ...
</xsl:if>
```

Colorăm tabelul din două în două linii:

Procesarea condițională: choose

xsl:choose - xsl:when - xsl:otherwise

```
>
<xsl:variable name="zi" select="substring(datan,1,2)" />
<xsl:choose>
  <xsl:when test="$zi=13">
    <font color="red"> <xsl:value-of select="nume"/> </font>
  </xsl:when>
  <xsl:when test="starts-with(nume, 'A')">
    <br/><b> <xsl:value-of select="nume"/> </b>
 </xsl:when>
  <xsl:otherwise>
    <xsl:value-of select="nume"/>
  </xsl:otherwise>
</xsl:choose>
```

Alte elemente XSLT

xsl:include

```
<xsl:include href="fisier.xsl"/>
```

xsl:import

```
<xsl:import href="fisier.xsl"/>
```

xsl:output

```
<xsl:output method="xml" indent="yes"/>
<xsl:output method="html"/>
<xsl:output method="text"/>
```

Exemplu

Să considerăm următorul document XML:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="note prez3.xsl"?>
<note an="3">
  <student>
  <nume> Ionescu </nume>
  <nota disciplina="Java"> 9 </nota>
  <nota disciplina="C"> 5 </nota>
</student>
<student>
  <nume> Popescu </nume>
  <nota disciplina="Java"> 8 </nota>
  <nota disciplina="C"> 6 </nota>
</student>
<student>
  <nume> Gigi </nume>
  <nota disciplina="Java"> 4 </nota>
  <nota disciplina="C"> 4 </nota>
</student>
</note>
```

Transformare: mărirea notelor

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="*">
  <xsl:copy> <xsl:apply-templates/> </xsl:copy>
</xsl:template>
<xsl:template match="nota">
  <xsl:variable name="n" select="."/>
  <xsl:variable name="d" select="@disciplina"/>
  <xsl:element name="nota">
    <xsl:attribute name="disciplina"> <xsl:value-of select="$d"/> </xsl:attribute>
    <xsl:choose>
      <xsl:when test="$n &lt; 10 and $n &gt; 4">
        <xsl:value-of select="$n + 1"/>
      </xsl:when>
      <xsl:otherwise> <xsl:value-of select="$n"/> </xsl:otherwise>
    </xsl:choose>
```

</xsl:element>
</xsl:template>
</xsl:transform>

XPath

Ce este XPath?

XPath este un limbaj de interogare conceput special pentru modelul ierarhic al unui document XML.

Expresiile XPath sunt folosite pentru:

- Selectarea nodurilor ce vor fi procesate
- Testarea unor condiţii pe parcursul procesării
- Calcularea valorilor ce vor fi inserate în arborele rezultat



XPath ca limbaj de programare

e

Include facilități pentru lucrul cu șiruri, calcule numerice și operații logice.

Exemplu:

Specificațiile complete ale limbajului XPath:

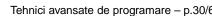
http://www.w3.org/TR/xpath

Sintaxa XPath

XPath folosește multe din notațiile consacrate pentru directoare.

Regulile de bază pentru descrierea nodurilor sunt:

- / reprezintă separatorul pentru locaţii (căi)
- O cale absolută de la rădăcină începe cu /
- O cale relativă începe cu orice altceva în afară de /
- .. reprezintă părintele nodului curent
- reprezintă nodul curent



Specificarea locațiilor

O locație (cale) XPath definește o mulțime de noduri. Locațiile pot fi:

- absolute: $/pas1/pas2/.../pas_n$
- relative: $pas1/pas2/.../pas_n$

Structura unui pas este:

directie::nod[predicat]

- directie: relaţia dintre nodul curent şi nodurile selectate
- nod: nodurile ce vor fi selectate
- predicat: condiție pentru filtrarea nodurilor rezultat



Specificarea direcțiilor

ancestor
ancestor-or-self
attribute
child
descendant
descendant-or-self
following
following-sibling
namescpace
parent
preceding
preceding-sibling
self

Wildcard-uri

Următoarele notații au semnificație de wildcard:

*	orice nod de tip element		
text()	orice node tip text		
node()	() orice nod, indiferent de tip		
@*	orice atribut		



Abrevieri

In XPath sunt permise următoarele abrevieri:

nimic	child::
	(nod este schivalent cu child::nod)
@	attribute::
	self::node()
• •	parent::node()
//	descendant-or-self::node()/

```
//persoana Toate elementele 'persoana'
//* Toate elementele din document
../@id Atributul id al parintelui
@* Toate atributele nodului curent
}
```

Expresii

XPath oferă suport pentru expresii numerice, relaţionale şi logice. Operatorii permişi sunt:

```
+ - * div mod = != < <= > >= or and
```

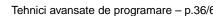
Exemple:

```
persoana[@id != "0"]
  atributul id trebuie sa nu fie 0
persoana[@id]
  atributul id trebuie sa fie prezent
persoana[@id and @activ]
  ambele atribute id si activ trebuie sa fie prezente
persoana[@activ='da']
  selecteaza doar persoanele in activitate
persoana[salariu > 5000000]
persoana[nume='Popescu']
```

Funcții XPath

Funcţiile disponibile în XPath pot fi grupate astfel:

- La nivel de nod
- Pe şiruri de caractere
- Numerice
- Logice

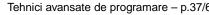


Funcții la nivel de nod

```
count
id
last
local-name
name
name
position
```

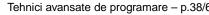
Exemple:

```
persoana[last()] Selecteaza ultima persoana din document
persoana[1] Selecteaza prima persoana din document, echivalent cu
persoana[position()=1]
```



Funcții pe șiruri de caractere

concat
contains
normalize-space
starts-with
string
string-length
substring
substring-after
substring-before
translate
position



Funcții numerice

```
ceiling
floor
number
round
sum
```

Exemplu:

Funcții logice

boolean false lang not true

Exemple de utilizare XSLT şi XPath

Fişierul XML

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="note_prez1.xsl"?>
<note an="3">
  <student>
  <nume> Ionescu> </nume>
  <nota disciplina="Java"> 9 </nota>
  <nota disciplina="C"> 5 </nota>
</student>
<student>
  <nume> Popescu </nume>
  <nota disciplina="Java"> 8 </nota>
  <nota disciplina="C"> 6 </nota>
</student>
<student>
  <nume> Gigi </nume>
  <nota disciplina="Java"> 4 </nota>
  <nota disciplina="C"> 4 </nota>
</student>
</note>
```

Rezultatul dorit 1



- 1. Ionescu
 - o Java: 9
 - $\circ C:5$
- 2. Popescu
 - o Java: 8
 - o C:6
- 3. Gigi
 - Java: 4
 - o C:4

Transformarea 1

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="note">
 <html>
 <body>
 <h1>Note anul <xsl:value-of select="@an"/></h1>
 <01>
 <xsl:apply-templates/>
 </body>
 </html>
</xsl:template>
<xsl:template match="student">
 <b>
 <xsl:value-of select="nume"/> 
 <l
   <sl:value-of select="nota[1]/@disciplina"/> :
        <xsl:value-of select="nota[1]"/> 
    <xsl:value-of select="nota[2]/@disciplina"/> :
        <xsl:value-of select="nota[2]"/> 
 </b>
</xsl:template>
</xsl:transform>
```

Rezultatul dorit 2

Note anul 3

Nume	Notal	Nota2	Media
Ionescu	9	5	7
Popescu	8	6	7
Gigi	4	4	4

Transformarea 2

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="note">
 <html><body>
 <h1>Note anul <xsl:value-of select="@an"/></h1>
 Nume Nota1 Nota2 Media
   <xsl:apply-templates/>
  </body></html>
</xsl:template>
<xsl:template match="student">
  <xsl:value-of select="nume"/>
  <xsl:value-of select="nota[1]"/> 
  <xsl:value-of select="nota[2]"/> 
 <xsl:variable name="medie" select="(nota[1]+nota[2])div 2" />
 <t.d>
 <xsl:choose>
   <xsl:when test="$medie &lt; 5"> <xsl:value-of select="$medie"/> </xsl:when>
   <xsl:otherwise> <xsl:value-of select="$medie"/> </xsl:otherwise>
 </xsl:choose>
 </xsl:template>
</xsl:transform>
```

Rezultatul dorit 3

Note anul 3

Nume	Nota C	Nota Java	Media
Ionescu	5.	9.	7
Popescu	6	8	7
Gigi	4	4	4

Transformarea 3

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="note">
 <html>
 <body>
 <h1>Note anul <xsl:value-of select="@an"/></h1>
 Nume
 <xsl:for-each select="student[1]/nota">
   <xsl:sort select="@disciplina" order="ascending"/>
   <xsl:value-of select="concat('Nota ', @disciplina)"/>
   </t.h>
 </xsl:for-each>
 Media
 <xsl:apply-templates/>
 </body>
 </html>
</xsl:template>
```

Transformarea 3 (continuare)

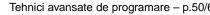
```
<xsl:template match="student">
 <xsl:value-of select="nume"/> 
 <xsl:for-each select="nota">
   <xsl:sort select="@disciplina"</pre>
             order="ascending"/>
    <xsl:value-of select="."/> 
 </xsl:for-each>
 <xsl:variable name="medie" select="sum(nota) div count(nota) "/>
 <xsl:choose>
   <xsl:when test="$medie &lt; 5">
     <font color="red"> <b> <xsl:value-of select="$medie"/> </b> </font>
    </xsl:when>
   <xsl:otherwise>
     <b><xsl:value-of select="$medie"/> </b>
   </xsl:otherwise>
 </xsl:choose>
 </xsl:template>
</xsl:transform>
```

Rezultatul dorit 4

Statistica anul 3

Media notelor la Java: 7

Media notelor la C:5



Transformarea 4

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="note">
  <html>
  <body>
  <h1>Statistica anul <xsl:value-of select="@an"/></h1>
  <xsl:variable name="tot" select="student/nota/@disciplina" />
  <xsl:variable name="dif" select="$tot[not(.=preceding::student/nota/@disciplina)]" />
  <xsl:for-each select="$dif">
    <q>
    <xsl:variable name="this" select="." />
    <xsl:variable name="note" select="//student/nota[./@disciplina=$this]" />
    <br/>b> Media notelor la
    <xsl:value-of select="$this"/> :
    <xsl:value-of select="sum($note) div count($note) "/>
    </b> <hr/> 
  </xsl:for-each>
  </body>
  </html>
</xsl:template>
</xsl:transform>
```

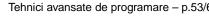
XQuery

Ce este XQuery?

- Este un limbaj pentru interogarea datelor în format XML.
- Foloseşte sintaxa XML.
- XQuery + XML ⇔ SQL + BD relaţionale.
- XQuey şi Xpath sunt construite pe acelaşi model sintactic.
- Nu este încă un standard W3C

Exemplu:

Selectează toți studenții din fișierul "note.xml" care au o nota de 10



Funcția doc

Funcţia doc returnează rădăcina unui document XML:

```
doc("note.xml")
  Returneaza tot arborele

doc("note.xml")//student[nota=10]
  Returneaza studentii cu macar o nota de 10
```

FLWOR

FLWOR = For, Let, Where, Order, Return.

for \$x in doc("note.xml")/student
where \$x/sum(nota) div count(nota)>5
order by \$x/nume
return \$x/nume

Java şi XSLT

TrAX

TrAX este acronimul de la *Transformation API for XML*. Este parte integrantă din distribuţia JAXP.

XML sursa \rightarrow Arbore DOM sursă \rightarrow XSLT \rightarrow Arbore DOM destinație \rightarrow XML destinație

Pachetele care oferă suport pentru transformări sunt:

- javax.xml.transform
- javax.xml.transform.dom
- javax.xml.transform.stream

Etapele transformării

- 1. Stabilirea sursei şi a rezultatului
- 2. Crearea unui obiect Transformer
- 3. Transformarea propriu-zisă

Interfața Source descrie sursa unei transformări. Implementări: DOMSource, SaxSource, StreamSource

Interfața Result descrie destinația unei transformări. Implementări: DOMResult, SaxResult, StreamResult



Secvența de transformare

```
// Stabilim sursa si destinatia
Source source = new DOMSource(sourceDoc);
Result result = new DOMResult(resultDoc);

// Specificam fisierul cu regulile
Source style = new StreamSource("fisier.xsl");

// Cream objectul Transformer
TransformerFactory transFactory =
   TransformerFactory.newInstance();

Transformer transformer =
   transFactory.newTransformer(style);

transformer.transform(source, result);
```

Afişarea unui arbore DOM

```
String filename = "fisier.xml";
 DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
 DocumentBuilder db = dbf.newDocumentBuilder();
 Document doc = db.parse(new File(filename));
 DOMSource source = new DOMSource(doc);
 StreamResult result = new StreamResult(System.out);
 TransformerFactory transFactory = TransformerFactory.newInstance();
 Transformer transformer = transFactory.newTransformer();
 transformer.setOutputProperty(OutputKeys.INDENT, "yes");
//transformer.setOutputProperty("indent", "yes");
 transformer.transform(source, result);
```

Afişarea unui subarbore

```
// Selectam primul nod 'persoana' din arbore
NodeList list = document.getElementsByTagName("persoana");
Node node = list.item(0);

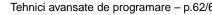
// Sursa o contruim folosind nodul selectat
DOMSource source = new DOMSource(node);
StreamResult result = new StreamResult(System.out);
...
transformer.transform(source, result);
```

Uilitarul Process

Utilitarul Process din distribuţia JAXP permite efectuarea de transformări de la linia de comandă:

```
java org.apache.xalan.xslt.Process
```

- -IN intrare.xml
- -XSL tranformare.xsl
- -OUT rezultat.xml



Asta-i tot...

Ne vedem la laborator.