### Introducere în XSL

Pe Web apar tot mai des documente scrise în XML, autorii acestora având posibilitatea să le definească după necesităţi structura, utilizând diverse unelte XML pentru crearea, verificarea şi interschimbarea de date între diverse formate şi XML. Documentelor sau fragmentelor de documente XML li se pot asocia diverse descrieri (metadate) prin intermediul *RDF* (*Resource Description Framework*). Datorită faptului că documentelor XML le lipseşte o semantică interpretabilă de către calculator, limbajul XML dictând de fapt numai sintaxa elementelor (*tag*-urilor) şi nu înţelesul lor, se propune **rescrierea** sursei unui document XML (conţinând elemente ale căror semantici au fost stabilite de autorul acelui document) generându-se un alt document, de cele mai multe ori tot XML (cuprinzând elemente având semantici care vor putea fi procesate de programul ce va prelucra acel document). Această asignare (*XML mapping*) este realizată de **XSL** (**eXtensible Stylesheet Language**), specificaţie folosind, la nivel sintactic, limbajul XML.

XSL se bazează pe *DSSSL* (*Document Style Semantics and Specification Language*), standard internaţional pentru definirea foilor de stiluri (ISO/IEC 10179:1996). XSL oferă următoarele:

* formatarea bazată pe moştenire sau poziţie;
* crearea de construcţii de formatare incluzând text şi grafică;
* definirea de macro-uri de formatare;
* foi de stiluri independente de direcţia de scriere (suport pentru internaţionalizare);
* un set extensibil de obiecte de formatare;
* publicare independentă de medii.

Fiecare foaie de stiluri XSL constă, simplist vorbind, din reguli compuse dintr-un model (*pattern*) şi o acţiune, modelul identificând un element sau grup de elemente XML asupra căruia se va aplica acţiunea. Modelul poate specifica toate elementele de un anumit tip, toate elementele dintr-un subarbore de noduri XML, toate elementele având un anume atribut specificat etc. Expresiile modelului sunt scrise într-un limbaj derivat din XML, denumit *XPath*, iar acţiunile (transformările efectuate asupra elementelor documentului) sunt specificate în *XSLT* (*XSL Transformations*), ambele recomandări ale Consorţiului Web începând cu sfârşitul anului 1999. Expresiile vehiculate în cadrul XSL sunt inspirate din *ECMAScript*, limbaj script standardizat de Asociaţia Europeană a Producătorilor de Calculatoare care va fi prezentat în una dintre secţiunile viitoare ale acestui text.

Principiile de proiectare a limbajului XSL au fost cele legate de suportul pentru aplicaţiile de navigare, listare şi editare interactivă, pentru prezentări vizuale şi non-vizuale de date, de posibilitatea de extindere cu noi elemente şi declaraţii, de usurinţa de parcurgere atât din punctul de vedere computaţional, cât şi din punctul de vedere uman.

Desigur, specificaţia XSLT este mult mai complexă, putându-se realiza combinarea şi includerea mai multor foi de stiluri pentru un acelaşi document, conceperea unor şabloane de reguli, procesarea condiţionată a elementelor XML, totul bazându-se pe *DOM* (*Document Object Model*), model obiectual descris într-unul dintre capitolele acestui material. Procesorul XSL va aplica în mod recursiv transformări asupra tuturor nodurilor care satisfac modelul sau şabloanele de reguli.

Pentru a putea observa flexibilitatea şi puterea XSL-ului, vom realiza pentru început o foaie de stiluri XSL care dintr-un document XML (specificând o lucrare ştiinţifică sau un articol) construieşte un document XHTML conţinând cuprinsul acestuia (în cuprins vor apare doar numele capitolelor lucrării). Aşadar, documentul XML iniţial va avea forma:

<article>

 <meta>

 <title>Documente XML active</title>

 <author email="gigel@ee.tuiasi.ro">

 GIGEL

 </author>

 </meta>

 <body>

 <intro>...</intro>

 <chapter>

 <title>Documente active</title>

 ...

 </chapter>

 <chapter>

 <title>Displet-uri</title>

 ...

 </chapter>

 <chapter>

 <title>Concluzii</title>

 ...

 </chapter>

 </body>

</article>

Cuprinsul va fi precedat de adresa de e-mail a autorului lucrării, deci va trebui să construim un element <a href="mailto:..."> pornind de la informaţiile deţinute de atributul email al elementului <author> din secţiunea <meta> a articolului. Construcţia XSL va fi:

<a>

 <xsl:attribute name="href">mailto:

 <xsl:value-of select="article/meta/author/@email" />

 </xsl:attribute>

 <xsl:value-of select="article/meta/author" />

</a>

Tag-ului <a> i se va ataşa atributul href a cărui valoare va fi dată de expresia XPath article/meta/author/@email (se porneşte de la elementul rădăcină article până se întâlneşte un nod descendent author şi se selectează valoarea atributului email). La fel, între <a> şi </a> se va include numele autorului.

Cuprinsul propriu-zis va fi o listă construită astfel:

<ol id="cuprins">

 <xsl:for-each select="article/body/chapter">

 <li>

 <h5 style="color:green">

 <xsl:value-of select="title" />

 </h5>

 </li>

 </xsl:for-each>

</ol>

Fragmentul XSL va genera o bucată de cod XHTML, pentru fiecare element <chapter> inclus în elementul <body> al articolului construindu-se un element de listă numerotată. Între tag-urile de început şi de sfârşit ale elementului <li> se va adăuga nodul text asociat elementului <title>, adică tocmai titlul fiecărui capitol al lucrării (am utilizat pentru formatare marcatorul <h5> şi proprietatea color din CSS1). Desigur, fiecare membru al listei va putea conţine o legătură internă la începutul capitolului desemnat, lăsăm acest exerciţiu în sarcina cititorului. Cele de mai sus pot fi procesate de navigatoarele Web din ultima generaţie: Netscape Communicator 6.x şi Internet Explorer 5.x. Există de asemeni şi editoare specializate, precum *XRay* (Architag). Procesarea foilor de stiluri XSL se poate realiza şi pe partea de server (de exemplu, în cadrul serverului IIS (Internet Information Server) există o extensie ISAPI pentru prelucrarea fişierelor XSL).

#### Declararea foilor de stiluri XSL

Toate foile de stiluri vor trebui să aparţină spaţiului de nume xsl definit de Consorţiul Web. Forma generală a declaraţiei unei foi de stiluri este următoarea:

<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

 ...

</xsl:stylesheet>

Cele mai uzuale elemente care pot compune o foaie de stiluri sunt cele din tabelul de mai jos:

|  |  |
| --- | --- |
| **Element** | **Descriere** |
| xsl:stylesheet  | Utilizat ca element rădăcină al unei foi de stiluri, poate conţine şabloanele care vor fi aplicate pentru generarea altui document XML |
| xsl:template  | Defineşte o serie de transformări/formatări care vor fi aplicate pentru un anumit model particular |
| xsl:apply-templates  | Invocă şabloanele definite anterior |
| xsl:comment  | Inserează un comentariu în cadrul documentului generat |
| xsl:pi  | Inserează o intrucţiune de procesare în documentul XML de la ieşire |
| xsl:element  | Inserează un element XML în documentul generat de foaia de stiluri |
| xsl:attribute  | Ataşează unui element un anumit atribut |
| xsl:value-of  | Inserează în cadrul documentului de la ieşire valoarea unui nod (element, atribut, comentariu etc.) particular |
| xsl:for-each  | Realizează o buclă de aplicare a unui şablon mai multor noduri respectând un model XPath |
| xsl:if  | Utilizat pentru aplicări condiţionale de tipul *if-then* |
| xsl:choose  | Folosit pentru condiţii multiple (de tipul *switch* din C), în conjuncţie cu elementele xsl:when şi xsl:otherwise |
| xsl:copy  | Copie conţinutul valorilor unor elemente în documentul rezultat |

#### Modele

Modelele folosite în cadrul foilor XSL vor fi specificate în limbajul XPath şi ele vor fi contextuale, potrivirile de elemente sau atribute XML realizindu-se în funcţie de plasarea în cadrul ierarhiei arborelui de noduri asociat unui document XML (mai multe amănunte despre arborii de noduri vom putea găsi în capitolul dedicat modelului DOM).

Sintaxa XPath este una declarativă, putând fi folosiţi operatorii descrişi în continuare:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Nume** | **Descriere** | **Exemplu** |
| /  | Descendent  | Utilizat să definească o ierarhie de elemente (o cale de parcurgere a nodurilor din arborele asociat documentului)  | //article/metachapter/title  |
| \*  | *wildcard*  | Folosit pentru elemente şi atribute, oferă toate elementele/atributele de pe un anumit nivel al arborelui  | /article/\*body/\*/title  |
| //  | traversare recursivă  | Traversează în adincime întreg sub-arborele unui anumit element | //article//body  |
| @  | atribut  | Utilizat să specifice un anumit atribut particular  | author/@emailtable/@\*  |
| :  | spaţiu de nume  | Folosit să separe elementele bazate pe spaţiul lor de nume  | xsl:\*//namespace:\*  |
| []  | filtru/index  | Aplică un filtru sau localizează elementele pe baza unui index  | table[tr]author[@email="..."]book[@title="Web"]/author[@age="33"]ul/li[1]  |

În continuare vom da o serie de exemple XPath împreună cu explicaţii:

* paragraph[@align="center"] selectează toţi descendenţii (nodurile copil) care sunt elemente <paragraph> având valoarea "center" pentru atributul align.
* //rdf:\* selectează toate elementele aparţinând spaţiului de nume *rdf* din cadrul unui document.
* ulist/img[position() = 33] selectează al 33-lea element <img> care face parte dintr-un element listă <ulist>. Funcţia position() este o funcţie ECMAScript standard.
* li[position() mod 2 = 1] va fi adevărată pentru fiecare element <li> aflat la poziţie impară în cadrul elementului părinte.
* books[@subject="Web"]/author[@name="Sabin"][@address="gigel@ee.tuiasi.ro"] va funcţiona ca un filtru, furnizând toate elementele incluse într-un element <books> care are atributul subject="Web" şi care are în componenţă elementul <author> având specificate atributele name="Sabin" şi address="gigel@ee.tuiasi.ro". Filtrele vide nu sunt permise (e.g. articol[]).

Pentru filtre pot fi folosiţi operatorii logici, relaţionali şi referitori la manipularea mulţimilor de elemente, prezenţi în limbajele de programare obişnuite (i.e. and, or, not, =, !=, <, >, <=, >, $all$, $any$, | etc.).

Câteva exemple:

* table/tr[@align="center" or @valign="top"]/\* va selecta elementele dintr-un <tr> având specificate atributele align="center" sau valign="top" din cadrul unui <table>.
* body[text != "white"] va selecta toate elementele dintr-un <body> care nu conţin o construcţie de forma <text>white</text>.
* library/book | library/magazine (echivalentă cu library/(book | magazine) va furniza toate elementele <book> şi <magazine>, descendenţi ai elementului <library>.

Modele utilizate de foile de stiluri pot fi *informative* (referitoare la tipul, numele şi valoarea unui element/atribut, putând fi folosite metode precum nodeName, nodeType, index, end ori number) sau *de grupare* (returnează colecţii de noduri de tipuri particulare: text, textNode, element, attribute, comment etc.).

#### Obiecte de formatare

Formatarea diferitelor elemente în XSL urmează modelul foilor de stiluri CSS, existând o multitudine de obiecte de formatare, fiecare dintre ele având o serie cuprinzatoare de proprietăţi pentru a oferi proiectanţilor un grad foarte mare de flexibilitate. Lista tuturor proprietăţilor poate fi consultată în specificaţia Consorţiului Web. Comparaţia cu foile de stiluri CSS - nivelele 1 şi 2 se poate parcurge în următoarea tabelă:

|  |  |  |  |
| --- | --- | --- | --- |
| **Caracteristica** | **XSL** | **CSS** | **Comentarii** |
| Utilizare cu HTML  | nu  | da  | XSL poate fi utilizat să convertească XML în XHTML. XSL nu funcţionează pentru documente HTML care nu respectă standardul XHTML. CSS se poate folosi ca modalitate de ataşare de stiluri independentă de documentul HTML.  |
| Suport pentru *scripting*  | da  | da  | XSL şi CSS oferă mecanisme de programare a foilor de stiluri via ECMAScript.  |
| Sintaxa bazată pe XML  | da  | nu  | Foile CSS nu folosesc o gramatică XML, pe când XSL este bazat pe XML.  |
| Afişarea de informaţii XML  | da  | da  | Se pot asocia unor documente XML sau bazate pe XML.  |
| Suport al navigatoarelor  | parţial  | da  | În prezent, Internet Explorer 5 sau ulterior oferă suport pentru XSL. Se aşteaptă suport şi din partea Netscape/Mozilla 6. CSS nivelul 1 şi nivelul 2 sunt suportate aproape total de navigatoarele actuale.  |
| Transformări de structură  | da  | nu  | CSS nu poate modifica structura documentelor, fiind doar o modalitate de a ataşa stiluri de formatare a unor elemente. Pentru HTML, soluţia modificărilor de structură este oferită de modelul DOM via JavaScript.  |
| Transformări asupra conţinutului  | da  | nu  | XSL poate modifica şi conţinutul documentelor XML, într-o manieră bazată pe reguli de rescriere, folosind modele/filtre.  |
| Manipularea datelor XML (căutare, filtrare, sortare)  | da  | nu  | XSL poate fi văzut ca limbaj simplu de interogare pentru documentele XML. CSS nu oferă nici o facilitate în acest sens, nefiind limbaj de programare.  |

### 7. XML Schema

După cum am văzut până acum, XML este un meta-limbaj definind reguli pentru definirea altor limbaje de adnotare. O **schemă** reprezintă o specificaţie formală a gramaticii asociate unui limbaj definit în XML, utilă pentru validarea documentelor scrise în acel limbaj.

Desigur, fiecărui document îi puteam asocia, în mod explicit, un DTD folosit pentru validarea acestuia, reminiscenţă de la SGML.

Schemele utilizează sintaxa XML şi sunt mai natural de definit decât DTD-urile, specificaţiile fiind încă în faza preliminară, bazate pe *Document Content Description* (*DCD*) şi *XML-Data*, ambele în lucru la Consorţiul Web. În prezent, schemele XML sunt suportate în formă limitată de Internet Explorer 5 şi Netscape 6.

#### Definirea elementelor şi atributelor

Ca şi la DTD, cea mai importantă parte dintr-o schemă XML o reprezintă specificarea elementelor şi atributelor care pot compune un document, incluzând tipul şi ordinea lor de apariţie. Elementele şi atributele sunt definite în XML Schema prin tag-urile <ElementType> şi, respectiv, <AttributeType>, iar instanţele lor prin <element> şi <attribute>, respectiv. Această abordare este similară declaraţiei typedef din limbajul C.

Un exemplu:

<?xml version="1.0"?>

<Schema name="BookSchema" xmlns="schemas-microsoft-com:xml-data">

 <ElementType name="title" content="textOnly" />

 <ElementType name="author" content="textOnly" />

 <ElementType name="description" content="textOnly" />

 <ElementType name="book" order="many">

 <element type="title" maxOccurs="1" />

 <element type="author" minOccurs="1" maxOccurs="\*" />

 <element type="description" />

 <AttributeType name="isbn" required="yes" />

 <attribute type="isbn" />

 </ElementType>

</Schema>

Am definit astfel patru elemente <title>, <author>, <description> şi <book>, ultimul fiind considerat element rădăcină. Elementul <book> va avea atributul obligatoriu isbn.

Schema specificată în acest exemplu va fi referită prin numele "BookSchema" stabilit prin intermediul atributului name al tag-ului <Schema>. Fiecare schemă are asociat un spaţiu de nume specificat prin xmlns.

Atributul order din definiţia unui tip de element dictează regulile de apariţie a elementelor în cadrul unui element părinte, putând avea valorile:

* seq - sub-elementele apar în secvenţa specificată în cadrul schemei;
* many - sub-elementele pot apare în orice ordine, în orice cantitate;
* one - un singur sub-element din lista celor specificate poate fi utilizat;
* empty - nu se permite apariţia nici unui sub-element (elementul va fi vid).

Fiecărui <ElementType> îi putem asocia atributul content care dictează tipul conţinutului acestuia:

* textOnly - numai text, fără alte sub-elemente;
* eltOnly - doar sub-elemente;
* mixed - conţinut eterogen, şi sub-elemente, şi text.

Implicit, pentru content="eltOnly" se consideră că order este setat pe seq, iar pentru content="mixed" atributul order are valoarea many.

XML Schema ne dă posibilitatea să aplicăm constrângeri unei mulţimi de (sub-)elemente prin <group>:

...

<ElementType name="author" order="seq">

 <group order="one">

 <element type="phone" />

 <element type="email" />

 </group>

 <element type="firstname" />

 <element type="lastname" />

</ElementType>

...

În acest exemplu, elementul <author> va putea conţine trei descendenţi: <phone> sau <email> (nu pot apare ambele), apoi <firstname> şi <lastname>, în această ordine.

Atributele se definesc asemănător elementelor, dar nu pot conţine sub-elemente şi nu pot apare într-o anumită ordine în componenţa unui marcator.

Ca şi la DTD, un atribut poate fi obligatoriu (required="yes" într-un <AttributeType>) sau nu (required="no"). Valoarea implicită se specifică prin default (e.g. default="33").

Pentru definirea tipului de date se utilizează fie tipurile primitive din specificaţia XML (de exemplu ID, NMTOKEN etc.), fie un spaţiu de nume definind tipurile de date permise. În implementarea IE 5, în cadrul elementului <Schema> putem include atributele xmlns="urn:schemas-microsoft-com:xml-data" şi xmlns:dt="urn:schemas-microsoft-com:datatypes" pentru a accesa o serie de tipuri de bază. Se observă faptul că lipsa lui xmlns="urn:schemas-microsoft-com:xml-data" duce la imposibilitatea validării documentului de către analizorul XML inclus în IE 5.

Aceste tipuri de date pot fi specificate într-un <ElementType> sau <AttributeType> în două moduri: prin atributul dt:type ori utilizând elementul <datatype>.

Astfel, linia următoare:

<ElementType name="figures" dt:type="int" />

este echivalentă cu construcţia:

<ElementType name="figures">

 <datatype dt:type="int" />

</ElementType>

Tipurile de date acceptate pot fi scalare ("int"), enumerări ("enumeration"), identificator sau referinţă la un identificator ("id", "idref"), entităţi ("entity") sau şiruri de caractere ("string"). Specificaţia Microsoft pentru tipurile de date permise defineşte o multitudine de alte sub-tipuri (e.g. "bin.hex", "boolean", "date", "dateTime", "float", "time", "uri" sau "uuid").

Un exemplu de tip enumerativ utilizat în cadrul declarării unui atribut:

<AttributeType name="stare"

 xmlns="urn:schemas-microsoft-com:datatypes"

 dt:type="enumeration"

 dt:values="în\_lucru depanare definitiv" />

De asemeni, un element poate avea ataşat tipul enumerare:

<ElementType xmlns="urn:schemas-microsoft-com:datatypes"

 name="year"

 dt:type="enumeration"

 dt:values="1 2 3 4" />

Între <year> şi </year> nu vor fi permise decât valorile "1", "2", "3" sau "4". Astfel, construcţia <year>7</year> va fi considerată eronată.

Pentru fiecare element se pot stabili, de asemeni, numărul minim şi maxim de apariţii ale acestuia prin intermediul atributelor minOccurs şi maxOccurs, respectiv. Valorile valide ale lui MaxOccurs pot fi numerele şi semnul "\*" (semnifică orice cantitate, fără restricţii). În mod obişnuit, valoarea implicită a lui maxOccurs este 1, cu excepţia cazului în care content="mixed" când va fi 0. Aceste atribute se utilizează în declaraţiile <element> şi <group>.

Putem fixa şi lungimea în caractere a valorilor conţinute de un atribut de tip "number", "int" sau "string", prin minLength şi maxLength:

<AttributeType name="password"

 xmlns="urn:schemas-microsoft-com:datatypes"

 dt:type="string"

 dt:minLength="5" dt:maxLength="8" />

#### Utilizarea schemelor

O schemă XML poate fi folosită via declaraţia unui spaţiu de nume prin includerea URI-ului la care este definită acea schemă. Acest lucru se realizează cu ajutorul atributului xmlns, prefixând URI-ul schemei cu şirul x-schema.

Vom utiliza schema definită în exemplul de mai sus pentru validarea urmatorului document:

<book xmlns="x-schema:http://www.ee.tuiasi.ro/schemas/book-schema.xml"

 isbn="973-97706-3-0">

 <title>Reţele de calculatoare</title>

 <author>Andrew S.Tanenbaum</author>

 <description>Un best-seller clasic</description>

</book>

De asemeni, putem defini scheme în interiorul documentului XML (maniera de definire *inline*):

<?xml version="1.0"?>

<Studenti>

 <Schema name="stud"

 xmlns="urn:schemas-microsoft-com:xml-data"

 xmlns:dt="urn:schemas-microsoft-com:datatypes">

 <ElementType name="an" dt:type="int" />

 <ElementType name="nume" dt:type="string" />

 <ElementType name="student" order="seq">

 <element type="an" />

 <element type="nume" />

 </ElementType>

 </Schema>

 <student xmlns="x-schema:#stud">

 <!-- Referire la schema de mai sus -->

 <an>4</an>

 <nume>Ştefan Tanasă</nume>

 </student>

 <student xmlns="x-schema:#stud">

 <an>3</an>

 <nume>Radu Filip</nume>

 </student>

</Studenti>

Dacă sunt specificate două scheme cu acelaşi nume, prima schemă va fi utilizată pentru validarea documentului. Acest lucru însă depinde de analizorul XML folosit.

În cadrul unui document pot apare construcţii de elemente şi atribute care vor fi validate utilizându-se scheme multiple, identificarea schemelor realizându-se de cele mai multe ori prin spaţiile de nume.

#### Extinderea schemelor

Spre deosebire de DTD-uri, în XML Schema se pot extinde schemele în funcţie de necesităţile proiectanţilor documentelor XML care pot adăuga propriile lor elemente şi atribute. Am definit în unul dintre exemplele precedente elementul <figures> de tip "int" stabilind numărul de figuri (ilustraţii) care apar într-o carte. Putem să impunem ca acest număr să nu depăşească 50. Astfel, extindem schema *BookSchema* în maniera de mai jos:

<ElementType name="figures" xmlns="urn:ee.tuiasi.ro:ext-book-schema">

 <datatype dt:type="int" />

 <extb:max>50</extb:max>

</ElementType>

Spaţiul de nume "extb" este definit la URN-ul specificat de atributul xmlns.

După ce documentul va fi validat conform regulilor descrise de schema originală, aplicaţia va putea să verifice dacă valoarea conţinută între <figures> şi </figures> depăşeşte sau nu valoarea 50.

#### XML Schema vs. DTD

De reţinut următoarele deosebiri între XML Schema şi DTD:

1. Schemele şi DTD-urile utilizează moduri diferite de specificare a conţinutului documentelor XML. În cadrul unui model XML Schema deschis, un element poate include definiţii <ElementType> şi <AttributeType> din alte scheme. Modelul XML Schema închis şi DTD sunt similare.
2. În XML Schema tipurile posibile de date pentru elemente şi atribute pot fi variate şi complexe, la specificaţia XML-Data încă lucrându-se în prezent. În cazul DTD, această flexibilitate nu există.
3. Includerea declaraţiilor de spaţii de nume XML nu poate fi modelată în DTD. Schemele pot fi extinse, pe când DTD-urile prin natura lor sunt fixe.
4. <ElementType> poate avea un unic element descendent, ceea ce în DTD nu se poate defini în mod strict.
5. Datorită folosirii sintaxei XML, schemele pot fi mai uşor create, validate şi procesate, prin intermediul modelului DOM.