UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI
Instrumente Structurale 2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Interacțiunea om-calculator

27. Proiectare de documente XML (XML, XHTML, DTD, XML SCHEMA)

XML
Prezentare

1. Despre XML – scurta istorie

XML – (eXtensible Markup Language) este un limbaj de adnotare/structurare a datelor. Istoria aparitiei sale porneste de la SGML (Standard Generalized Markup Language) aparut in anii 80. Acest limbaj isi propunea la aparitie sa creeze documente ce pot fi analizate de catre masini prin introducerea de marcaje (sau taguri). SGML nu a fost un succes deoarece era foarte complex si astfel crearea de programe care sa-l analizeze.
In anii 90 la CERN a aparut HTML-ul (HyperText Markup Language). La baza acestui limbaj a fost SGML-ul si scopul sau a fost marcarea documentelor astfel incat sa poata fi transmise prin retea. Unul din primele documente care au stat la baza WWW-ului si respectiv HTML-ului le puteti gasi la [1]. Una din ideile acestui document este ca browserele trebuie sa ignore marcajele si atributele pe care nu le inteleg. Aceasta idee a avut ca efect din cate cred eu aparitia unui limbaj mai simplu care sa poata fi parsat rapid de catre browsere. Si astfel s-a dezvoltat HTML-ul.

Succesul avut de Web a cauzat o dezvoltare rapida a browserelor care analizau marcajele HTML. Astfel au aparut o gama larga de marcaje si atribute care puteau fi scrise fara prea multe constrangeri. Browserele au trebuit sa tina cont de aceste constrangeri si au devenit foarte complexe. Pe de alta parte s-a sesizat utilizarea HTML-ului pentru adnotarea documentelor si o slabiciune de-a sa – faptul ca nu se pot adauga marcaje noi.

In acest context in decembrie 1997 W3C (World Wide Web Consortium [2]) a lansat ca recomandare propusa versiunea 1.0 de XML [3]. In acest document se afirma ca XML este un subset al SGML care pastreaza caracteristici cum ar fi posibilitatea crearii de marcaje, posibilitatea de validare, posibilitatea de a fi citit si inteles de catre oameni. Totodata se afirma ca XML este creat pentru a fi folosit pentru adnotari, schimb de date, publicare documente si prelucreare automata de catre clienti (agenti) inteligenti. Pastrand

un subset al SGML impreuna cu un set de constrangeri documentele XML vor putea fi prelucrate foarte rapid de catre parsere neavand limitele date de complexitatea SGML. Vom vedea care sunt cazurile de utilizare actuale pentru XML dupa ce ii vom intelege sintaxa – dupa care ii vom intelege mai clar si avantajele si slabiciunile.

2. Sintaxa

Am vorbit in prima parte a acestei prezentari despre marcaje si atribute. Acum putem sa incercam sa le si definim. Pentru a ilustra sintaxa vom folosi exemplul din Anexa 1. Un marcaj este un text care descrie sensul sau structura unor date. Acest text este inserat intre semnele „<" si „>". Marcajele pot avea inceput si sfarsit ca de exemplu:

<nume>Popescu</nume> (a)

sau pot fi vide (fara text in interiorul marcajului)

<casatorit />

Marcajele sunt case-sensitive. La intalnirea unui marcaj de tipul <nume>Popescu</NUME> parserul va da o eroare.

Un element este tot ceea ce se afla intre marcajul de inceput si marcajul de sfarsit . (a) este un element. Numele elementului este „nume". Aceste nume trebuie sa indeplineasca cateva conditii [4].

- Pot sa contina litere, cifre si alte caractere
- Nu pot incepe cu cifre si cu semne de punctuatie
- Nu pot incepe cu „xml"
- Nu pot contine spatii

Un element poate contine mai multe elemente. Acest lucru determina aparitia unor relatii de rudenie intre elemente si a unor constrangeri suplimentare. Cea mai importanta dintre aceste constrangeri este aceea ca intr-un fisier XML trebuie sa existe un element radacina care sa le contina pe toate celelalte. Elementul radacina in Anexa 1 este <persoana>. O a doua constrangere este ca elementele nu pot fi incrucisate. Astfel urmatoarea constructie este incorecta <tranzactie><data>2005-11-05</tranzactie></data>. Cu alte cuvinte marcajele trebuie inchise invers fata de cum sunt deschise.

Relatiile de rudenie pot fi parinte-copil sau frate-frate. Un exemplu de relatie parinte-copil in cadrul unui element este urmatoarea.

<tranzactie>
<data>20-07-2005</data>
<ora>13:00:00</ora>
</tranzactie>

In acest exemplu elementul *<tranzactie>* este parinte al elementelor *<data>* si *<ora>* care sunt "frati".

Un element poate sa contina unul sau mai multe atribute. Atributele se specifica in cadrul marcajului de inceput al elementului si sunt de tip nume_atribut=”valoare”. Un exemplu in anexa 1 este in cadrul marcajului urmator: *<suma moneda=”RON”> 1000 </suma>*. Atributul in acest caz este moneda iar valoarea atributului este RON. Valorile atributelor se pun intotdeauna intre ghilimele.

Ar mai fi de mentionat ca intotdeauna un document XML incepe cu un rand in care se mentioneaza versiunea limbajului si codificarea folosita. In exemplul din anexa 1 se specifica faptul ca se foloseste versiunea 1.0 si ca se foloseste codificarea ISO-8859-2 care este codificarea folosita pentru limba romana.

Respectand aceste simple reguli de sintaxa vom putea obtine ceea ce se numeste un document XML bine format. Acest document se valideaza la randul lui cu ajutorul unui alt fisier in care se definesc tagurile folosite. Acest alt fisier poate fi de tip DTD sau XSchema si despre ele vom vorbi in laboratoarele viitoare.

3. Cazuri de utilizare

Din introducere si din scurta trecere in revista a sintaxei am putut observa ca XML-ul este un limbaj care permite structurarea si adnotarea datelor intr-un mod lizibil atat de catre oameni cat si de catre calculatoare. Avand aceste calitati putem oarecum deduce unde se poate folosi in mod practic si unde este mai putin recomandat.

Poate cel mai evident caz de utilizare este cel al fisierelor de configurare. In momentul de fata foarte multe aplicatii isi pastreaza fisierele de configurare in XML. Motivele sunt cele enuntate mai sus – parsare rapida pentru calculatoare, usor de citit de catre oameni, structura logica a datelor. Anexa 2 prezinta un exemplu de fisier de configurare pentru o aplicatie web scrisa in Java – fisier folosit de serverul Tomcat.

Un alt caz in care se foloseste XML si in care se pare ca va deveni un standard este cel al stocarii informatiilor din fisiere tip office. Dezvoltatorii de solutii office open-source au standardizat un format numit Open Document[5][6]. Acest format presupune salvarea oricarui document de tip office – document, foaie de calcul (spreadsheet), prezentare – intr-un format XML. Informatia este stocata in mai multe fisiere care sunt arhivate si astfel utilizatorul poate vedea numai un singur fisier de tip *.sxw sau *.odt, etc. Fisierul *content.xml* (in care apare informatia efectiva dintr-un fisier editat in Open Office) pentru un fisier in care este scris textul „**Hello World**” este prezentat in anexa 3. De ce s-a alex XML pentru un astfel de format? Pentru ca ofera avantajele structurarii informatiei si mai ales pentru ca s-au dezvoltat deja foarte multe alte limbaje si standarde pe baza lui. Aceste limbaje (XHTML, SVG, XSL, SMIL, XLink, XForms, MathML, Dublin Core) sunt folosite si ele in acest format si deoarece toate au la baza XML interoperabilitatea este garantata. XML are si un dezavantaj care este dat de faptul ca introduce informatie redundanta (overhead). Informatia redundanta este reprezentata de numele marcajelor care se repeta in document. Aceasta problema a fost rezolvata de cei de la Open Office prin arhivarea fisierelor XML intr-o arhiva zip. Astfel s-a rezolvat

problema spatiului suplimentar aparut prin introducerea unei mici pierderi in viteza. Mai recent (incepand cu Microsoft Office 2007) si Microsoft a adoptat formatul OOXML care dupa cum ii spune și numele este tot bazat pe XML și care este în acest moment în curs de standardizare.

Un alt caz de utilizare este in stocarea efectiva a datelor – mai precis baze de date. In momentul de fata XML poate fi folosit cu succes pentru baze de date mici. In momentul in care bazele de date cresc ca dimensiune overheadul devine inacceptabil. Din cauza spatiului mare ocupat si a necesitatii accesului foarte rapid la date arhivarea fisierelor XML cu date nu poate fi o solutie si in acest caz. Oricum, multe interfete pentru baze de date permit exportarea bazei de date in format XML – atat definitia tabelelor si a relatiilor intre ele cat si datele propriu zise. Doua exemple de aplicatii care au aceasta facilitate sunt MS Access si PHPMyAdmin[7]. Un alt exemplu de caz in care XML-ul nu a avut succes este legat tot de stocarea unor cantitati mari de date si de acces rapid la acestea – mai precis pastrarea fisierelor jurnal ale serverelor. In acest domeniu este folosit printre altele un standard numit CLF – Common Logfile Format – care are avantajul ca nu introduce informatie suplimentara [8].

XML sta la baza multor tehnologii si limbaje noi. Astfel pentru prezentare de continut pe web exista XHTML si WML (pentru dispozitive mobile), pentru descrierea unor fisiere XML exista XSchema, pentru transformarea fisierelor XML pentru a fi reprezentate exista XSL, pentru realizarea unor prezentari exista SMIL, pentru descrierea obiectelor grafice exista SVG, pentru reprezentarea semanticii unor domenii exista RDF si/sau OWL, pentru schimb de informatii intre aplicatii exista SOAP si asa mai departe, lista aceasta fiind departe de a cuprinde toate tehnologiile/limbajele existente bazate pe XML. De ce au aparut atatea limbaje bazate pe XML? Pentru ca are o structura simpla si totusi stricta, pentru ca este foarte usor de analizat si foarte usor de definit si dezvoltat.

4. Instrumente

Există o largă varietate de instrumente pentru editarea fişierelor XML. Din seria celor comerciale cele mai cunoscute sunt XMLSpy şi StylusStudio. Ambele oferă versiuni de „trial” pentru a vedea ce facilităţi oferă.
Datorită simplităţii sintaxei o gamă largă de editoare au facilităţi pentru crearea fişierelor XML: Notepad++ (http://notepad-plus.sourceforge.net), Context (http://www.context.cx/) din gama editoarelor mai simple şi Eclipse sau Netbeans din gama editoarelor mai complexe

5. Şabloane de proiectare a fişierelor XML

Site-ul XMLPatterns.com prezintă un mare număr de şabloane de construcţie a fişierelor XML. Folosirea acestor şabloane permite crearea unor documente mai uşor de citit, folosit şi prelucrat, permite gestionarea mai facilă a informaţiilor oferite.
În această secţiune vom rezuma şabloanele cele mai importante.

- Şabloane pentru folosirea Metadatelor (date despre date)

Când există metadate într-un document pot fi folosite mai multe soluţii pentru integrarea lor:

- Head+Body – varianta folosita şi în HTML – datele despre date se pun într-o secţiune specială a documentului numită head şi datele efective se vor plasa în secţiunea body.
- Metadatele se pun înaintea datelor în document. Motivul este destul de simplu şi anume în condiţiile în care procesarea XML-ului se face secvenţial s-ar putea ca un proces, după ce interpretează meta-datele să nu mai dorească sa prelucreze şi restul documentului şi implicit să nu-l mai încarce sau parseze
- În cazul în care există un mare număr de metadate, poate fi util ca acestea să fie plasate într-un fişier separat. Acest lucru este recomandabil mai ales în cazul în care este vorba de metadate comune mai multor fişiere (vezi ontologii şi RDF)

- Abstractizare

Cum se pot alege şi respectiv grupa elementele astfel încât structura XML-ului format să fie cât mai inteligibilă.

- Containere – în containere se pot pune mai multe elemente diferite dar care caracterizează un „părinte" comun dintr-un anumit punct de vedere. De exemplu pentru un om putem pune într-un container „date de identificare" elementele „nume", „prenume", „CNP"
- Colecţii – când în cadrul unui element pot apărea ca şi copii mai multe elemente de acelaşi tip acestea este recomandabil să se grupeze ca şi copii al unui element separat. Exemplu un om are mai mulţi copii ceea ce putem exprima în mai multe feluri:

```
<om>
    <date_identificare>…</date_identificare>
    <copil></copil>
    <copil></copil>...
..
</om>
```

Sau

```
<om>
    <date_identificare>…</date_identificare>
    <copii>
            <copil></copil>
            <copil></copil>...
    </copii>
..
</om>
```

A doua variantă presupune adăugarea elementului „copii" care reprezintă o colecţie de elemente de tip „copil"

- Elementele este bine să aibă o corespondenţă în cadrul domeniului descris (v. Descrierea claselor şi a proprietăţilor la programarea orientată obiect)

- Organizarea datelor

    - Declararea elementelor înainte de a fi referite. Cu ajutorul atributelor de tip IDREF elemente pot fi referite în cadrul altor elemente. Este recomandabil ca întâi să apară în document elementul şi apoi referinţa (tot pentru uşurinţa procesării)
    - Referirea datelor şi nu repetarea lor. Datele pot fi declarate folosind declaraţii de tip Entity sau pot fi identificate în mod unic folosind atribute de tip ID şi referite prin atribute de tip IDREF
    - Clasificare folosind atribute – când pot clasifica anumite elemente în mai multe feluri se pot introduce atribute pentru a nu complica ierarhia. În exemplul de mai sus elementului copil i se poate introduce atributul „sex" care practic introduce o clasificare suplimentară fără a adăuga elemente suplimentare.
- Altele
    - Reutilizare atribute – dacă mai multe elemente au aceleaşi atribute nu trebuie definite atributele de mai multe ori
    - Nume scurte dar inteligibile – „loc_de_munca" este preferat elementului "serv".

Pentru mai multe detalii şi pentru mai multe şabloane vizitaţi http://xmlpattern.com

6. Concluzii

XML-ul este un limbaj cu o sintaxa simpla si care permite doar structurarea datelor intr-o maniera proprie prin definirea propriilor taguri. Aceasta facilitate de structurare a datelor a permis folosirea sa pentru a dezvolta limbaje noi precum si pentru a fi folosit in noi standarde de stocare a datelor. Nu in ultimul rand XML-ul poate fi folosit pentru a schimba date intre aplicatiile care au nevoie de a comunica intr-un limbaj comun.

Chiar si in ciuda dezavantajelor pe care le are (overhead) calitatile XML mai sus mentionate ii permit sa fie folosit cu succes in multe domenii si sa fie piatra de temelie pentru si mai multe domenii.

7. Bibliografie

1. „The first version of HTML" - http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/MarkUp.html
2. World Wide Web Consortium – http://www.w3c.org
3. Press Release XML 1.0 - http://www.w3.org/Press/XML-PR
4. Tutorial XML – http://www.w3schools.com/xml/xml_elements.asp
5. Standardizarea Open Document – siteul web OASIS - http://www.oasis-open.org/news/oasis_news_05_23_05.php
6. Oasis OpenDocument Essentials - http://books.evc-cit.info/odbook/book.html
7. PHPMyAdmin – web site - http://www.phpmyadmin.net/home_page/index.php
8. CLF – formatul de inregistrare al jurnalelor - http://www.w3.org/Daemon/User/Config/Logging.html#common_logfile_format
9. Şabloane XML – www.XMLPattern.com

XHTML
Prezentare

1. Despre XHTML – scurta istorie

XML – (eXtensible Markup Language) este un limbaj de adnotare/structurare a datelor. Istoria aparitiei sale porneste de la SGML (Standard Generalized Markup Language) aparut in anii 80. Acest limbaj isi propunea la aparitie sa creeze documente ce pot fi analizate de catre masini prin introducerea de marcaje (sau taguri). SGML nu a fost un succes deoarece era foarte complex si astfel crearea de programe care sa-l analizeze.

Sa ne amintim cate ceva din laboratorul trecut, in care am discutat despre XML...

XML – (eXtensible Markup Language) este un limbaj de adnotare/structurare a datelor. Istoria aparitiei sale porneste de la SGML (Standard Generalized Markup Language) aparut in anii 80.

In anii 90 la CERN a aparut HTML-ul (HyperText Markup Language).

Succesul avut de Web a cauzat o dezvoltare rapida a browserelor care analizau marcajele HTML. Astfel au aparut o gama larga de marcaje si atribute care puteau fi scrise fara prea multe constrangeri. Browserele au trebuit sa tina cont de aceste constrangeri si au devenit foarte complexe. Pe de alta parte s-a sesizat utilizarea HTML-ului pentru adnotarea documentelor si o slabiciune de-a sa – faptul ca nu se pot adauga marcaje noi.

De-a lungul timpului limbajul HTML s-a dezvoltat, browser-ele au evoluat foarte mult, dar au aparut si foarte multe situri care nu respectau constrangerile si regulile descrise in HTML sau nu aveau sintaxa corecta. Pentru ce se doreste pe viitor, ca siturile sa poate fi intelese atat de oameni, dar si de computere, HTML nu este de ajuns. Atunci cand sintaxa este gresita sau siturile nu furnizeaza informatiile „semantice” dorite, cautarile pe motoare de cautare, gen Google, Yahoo! devin greoaie, apare informatie in exces si nedorita! De asemenea urmatorul pas catre dezvoltarea WWW-ului este ca siturile si companiile sa furnizeze informatie „semantica” (sub forma de servicii web, rdf, etc) pentru diversi agenti, crowler-e sau alte programe software. Cel mai concludent exemplu este cel dat de Tim Berners-Lee in legatura cu urmatorul pas, web-ul semantic

in Scientific American http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

Astfel a aparut XHTML, adica HTML scris cu reguli XML. Este unul dintre cei mai importanti pasi facuti in dezvoltarea web-ului. XHTML sau „E**X**tensible **H**yper**T**ext **M**arkup **L**anguage" are ca scop inlocuirea limbajul clasic HTML fiind o recomandare a organizatiei W3C. XHTML 1.0 a devenit o recomandare W3C in 26 ianuarie 2000 [w3schools]. Versiunea actuala de XHTML se bazeaza pe versiunea 4.01 a HTML-ului, fiind o versiune cu mai multe constrangeri dar mai clara si mai curata a acestuia.

2. Sintaxa

In acest capitol vom prezenta o parte din regulile pe care trebuie sa le respecte orice cod HTML pentru a putea fi validat ca fiind XHTML. De asemenea vom prezenta si exemple de cod de genul „asa nu" si varianta corecta si recomandata. Inca o data precizam ca multe browser-e afiseaza corect informatia chiar si atunci cand nu se respecta constrangerile de XML sau XHTML, dar atunci cand vrem ca informatia noastra sa poate fi corect interpretata de agenti software, crowlere, motoare de cautare sau browsere pentru dispozitive mobile (care sunt mult mai simple) atunci trebuie sa respectam recomandarile propuse de W3C.

In primul rand vom reaminti cele 4 conditii de la XML pe care si fisierele XHTML trebuie sa le respecte:

- imbricarea tagurilor
- tagurile trebuie sa fie inchise corect
- numele tagurile trebuie sa fie scrise cu litere mici
- orice document XHTML trebuie sa aiba un nod radacina
- numele atributelor trebuie sa fie scrise cu litere mici, de asemenea
- valorile atributelor trebuie scrise intre ghilimele („")
- folosirea referintei catre fisierul DTD care valideaza fisierul XHTML.

- imbricarea tagurilor:
  exemplu:
  <a>Hello world! INCORECT
  <a>Hello world! CORECT

  - tagurile trebuie inchise corect:
    exemplu:
    -cele mai multe greseli se fac cu tagurile <p> sau
    INCORECT:
    <p> Hello world!
    <p> How are you today?
    e incorect desi in multe browser-e continutul e afisat corect

CORECT:

* Hello world! *
* How are you today? *

Un alt exemplu este cel cu
 sau <hr>. Si tag-urile (elemente de marcaj) goale trebuiesc inchise:

*INCORECT: Hello, how are you?
*
*CORECT: Hello, how are you?
*

- toate elementele de marcaj trebuie scrise folosind litere mici:

CORECT:

```
<table>
    <tr>
        <td>
            Hello world
        </td>
    <tr>
</table>
```

INCORECT:

```
<Table>
    <TR>
        <TD>
            Hello world
        </TD>
    <TR>
</Table>
```

- si in cazul acesta browserele vor ignora literele mari si le vor considera lowercase. Sunt multe generatoare de cod HTML, mai vechi, care genereaza cod folosind litere mari, ceea ce este gresit!

- orice document HTML trebuie sa aiba un singur nod radacina si acela trebuie sa fie <html>

structura ar fi:

```
<html>
        <head> Head of the page </head>
        <body> This is the body of the page...
</body>
        </html>
```

- Atributele joaca un rol important ca si elementele si au parte de un tratament la fel ca si elementele in privinta literelor cu care sunt scrise.

Pe langa acest lucru mai sunt 2 constrangeri importante:

- valorile atributelor se scriu intre ghilimele:

<table background=#ffffff> INCORECT

<table background=”#ffffff”> CORECT

- o alta greseala este atunci cand in loc de un atribut se prescurteaza valoarea acestuia

exemplu:
<option selected> INCORECT
<option selected=”selected”> INCORECT

Lista cu toate atributele care trebuiesc scrise in acest fel o puteti gasi la [http://w3schools.com/xhtml/xhtml_syntax.asp]

De asemenea trebuie retinut ca in cazul tag-urilor pentru imagini atributul ***name*** a fost inlocuit cu ***id***

Orice document XHTML contine o referinta catre fisierul DTD care il valideaza si are urmatoarea structura:

```
<!DOCTYPE link_catre_fisierul_dtd>  //calea catre fisierul .dtd
<html xmlns="http://www.w3.org/1999/xhtml"> //trebuie pus namespace-ul
        //pentru a specifica locatia specificatiei pentru xhtml
<head>                  //tag folosit pentru diverse informatii, pentru motoarele de
        //cautare si nu numai
<title>Titlul pagini</title>
</head>
<body>                  //locul unde se pune efectiv informatia utila din pagina
.....
</body>
</html>         //se inchide nodul “root” al documentului
```

exemplu de <!DOCTYPE ...> :

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

Dupa cum bine stim, continutul siturile si structurarea informatiei sunt doua trasaturi importante ale siturilor, dar acum se pune accent si pe prezentarea acestora, pe stil, pe culori si pe imbinarea lor. Toate aceste lucruri trebuiesc facute simplu si eficient. In prezent aceste lucruri sunt realizate cu ajutorul limbajului CSS despre care vom vorbi in partea a doua a laboratorului.

3. Cazuri de utilizare

Dupa cum am spus si la inceputul acestui document, XHTML va permite o mai buna structurare si intelegere a continutului informatiei de diverse programe software dar si dispozitive. Mai ales acum cand sectorul dispozitivelor mobile a cunoscut o dezvoltare fantastica si unde browserele se dezvolta, dar beneficiind de resurse mai putine si limitate

in comparatie cu un sistem desktop, sunt mult mai simple si multe situri web nu sunt afisate corespunzator si datorita faptului ca nu respecta recomandarile XHTML 1.0.

4. Instrumente

Exista multe unelte pentru editarea de fisiere XHTML, de la simplele editoare text cu „sintaxa colorata” si pana la posibilitatea de a valida online un intreg sit sau pagina web [http://validator.w3.org/]
Iata cateva instrumente care va pot ajuta la scrierea de pagini HTML si XHTML:

- Macromedia Dreamweaver
- softul de la [http://savannah.nongnu.org/projects/xhtmltools/]
- XMLBuddy [http://www.download.com/XMLBuddy/3000-7241_4-10422909.html?tag=lst-0-5]
- HTML Tidy [http://tidy.sourceforge.net/#docs]
- si multe altele...
-

5. Concluzii

XHTML este limbajul binecunoscut HTML, dar scris cu reguli XML care se doreste a fi pasul urmator in dezvoltarea WWW-ului. Se doreste ca atat utilizatorii de pe calculatoare normale cat si cei care folosesc dispozitive mobile sa poata accesa orice sit web fara probleme. De asemenea folosind XHTML multe produse software sau agenti pot folosi informatiile de pe site-uri intr-un mod corect.

6. Bibliografie

1. http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21
2. http://www.w3schools.com
3. XHTML sintaxa http://w3schools.com/xhtml/xhtml_syntax.asp
4. Validator - http://validator.w3.org/
5. XHTML tool - http://savannah.nongnu.org/projects/xhtmltools/
6. XMLBuddy http://www.download.com/XMLBuddy/3000-7241_4-10422909.html?tag=lst-0-5
7. HTML Tidy http://tidy.sourceforge.net/#docs

DTD
Prezentare

1. Introducere

DTD (Document Type Definition) este primul limbaj apărut (în ordine cronologică) pentru a specifica structura documentelor XML. Definirea structurii fişierelor XML face posibilă validarea ulterioară a acestora. Motivele pentru care avem nevoie de definirea structurii fişierelor XML cu care lucrăm ar fi următoarele:

- Posibilitatea definirii unui standard pentru anumite tipuri de fişiere XML – exemplu SVG (Scalable Vector Graphics) – limbaj bazat pe XML şi caracterizat de un fişier DTD care permite definirea graficelor bidimensionale. Toate aplicaţiile care crează fişiere SVG se bazează pe acelaşi fişier DTD şi implicit vor crea fişiere XML ce au aceeaşi structură
- Posibilitatea validării fişierelor XML primite - DTD-ul permite efectuarea validării automate fişierului XML primit. Se poate verifica uşor (fără a fi necesar a fi scris cod de validare din partea programatorului) dacă fişierul respectă structura dată şi în cazul în care nu o respectă nu trebuie să pierdem timp încercând să-l prelucrăm.
- Posibilitatea ca fiecare fişier XML să aibă referinţă către (sau să conţină) fişierul ce-i defineşte structura.

Fişierele DTD permit definirea elementelor XML, a atributelor, a unor restricţii simple privitoare la cardinalitatea (de câte ori apar) elementelor şi la tipul lor. Ulterior a fost dezvoltată XML Schema – care permite declararea unor constrângeri mai avansate şi care, spre deosebire de DTD, este descris tot într-un fişier XML.

2. Sintaxa

Un fişier DTD poate fi conţinut în fişierul XML sau într-un fişier extern
În primul caz DTD-ul va fi descris în felul următor:
<!DOCTYPE element_rădăcină [conţinut propriu-zis]>
În al doilea caz în fişierul XML va fi referit fişierul DTD în felul următor:
<!DOCTYPE element_rădăcină [SYSTEM sau PUBLIC FPI] url>

“element_rădăcină” este în ambele cazuri elementul rădăcină al fişierului XML. Cuvântul cheie SYSTEM este folosit daca fişierul .dtd se află pe acelaşi calculator cu fişierul XML în care există referinţa. PUBLIC se foloseşte când fişierul .dtd se află la o adresă externă şi este urmat de FPI (Formal Public Identifier) – un string care identifică în mod unic un DTD ce poate fi folosit pentru un număr mare de documente (Exemplu de FPI: *"-//W3C//DTD HTML 4.01 Transitional//EN"* – identifică un DTD pentru HTML). Ultimul element al referinţei este URL-ul local sau extern unde poate fi găsit fişierul .dtd.

După cum spuneam în introducere fişierele DTD permit definirea structurii unui element. Sintaxa pentru a defini un element este următoarea:
<!ELEMENT nume tip sau (structura + descriptori)>
Tipul poate fi EMPTY în cazul elementelor vide sau ANY în cazul elementelor ce pot avea orice structură.
Dacă elementul nu este dintr-unul din tipurile de mai sus atunci el poate conţine text (#PCDATA – parsed character data) şi poate avea alte elemente copii. Elementele copii ce pot fi întâlnite în structura unui element sunt separate prin virgulă şi pot fi urmate de următorii descriptori

- * - elementul poate să apară de 0 sau mai multe ori
- + - elementul poate să apară de 1 sau mai multe ori
- ? – elementul poate să apară de 0 sau 1 ori
- Element1 | Element2 – poate să apară ori Element1 ori Element2

De exemplu un element „Om” ar putea avea următoarea structură:
<!ELEMENT Om (nume, prenume, cnp, copil*, casatorit_cu?, (M|F), parinte+)>
Elementul de mai sus este propus pentru a exemplifica descriptorii de mai sus. Vom vedea mai jos că structura acestui element se poate descrie mai bine folosind atribute.

Sintaxa definirii atributelor este următoarea:
<!ATTLIST element atribut tip valoare_implicită>
Element este numele elementului al cărui atribut îl definim. Atribut reprezintă numele atributului pe care îl definim.
Tipul poate avea mai multe valori dintre care cele mai frecvent folosite sunt următoarele

- CDATA – character data – text ce nu va fi parsat de parserul XML
- ID – valoarea atributului reprezintă un ID unic
- IDREF – valoarea atributului referă un ID unic
- IDREFS – atributul are ca valoare o lista de ID-uri
- (val1|val2|..|valn) – atributul are valorile cuprinse în lista de valori val1...valn

Valoarea implicită poate avea una din următoarele valori:

- Valoare – valoarea implicită a atributului
- #FIXED valoare – atributul are întotdeauna valoarea “valoare”
- #IMPLIED – atributul poate lipsi
- #REQUIRED – atributul trebuie să se găsească întotdeauna în documentul XML

În fişiere DTD se pot defini entităţi. Entităţile sunt prescurtări ale unor structuri de text ce apar mai des în fişierul XML.
Un exemplu ar fi următorul:

<!ENTITY CS “Computer Science”> - se defineşte entitatea cu numele “CS” ce prescurtează denumirea “Computer Science”.
În fişierul XML se va folosi în felul următor
<profil>&CS;</profil>. Înaintea entităţii se foloseşte „&” şi după aceasta se foloseşte „;”.

3. Instrumente

Instrumentele prezentate la capitolul dedicat XML oferă în general şi următoarele facilităţi pentru DTD:

- Validare XML conform cu DTD-ul dat (această facilitate este oferită şi de cea mai mare parte a browserelor precum şi de multe api-uri de prelucrare a XML-urilor
- Generare automată a fişierului DTD plecând de la un XML existent. Deşi este o modalitate deseori folosită pentru crearea fişierelor DTD, nu este frecvent şi corectă deoarece generatorul nu poate anticipa în general toate constrângerile pe care dorim să le impunem doar dintr-o variantă simplă de XML.

4. Concluzii

DTD este un limbaj foarte simplu pentru definirea structurii unui fisier XML. Necesitatea folosirii acestui limbaj provine din facilitatile de validare automata a structurii unui fisier XML ce refera un DTD.

5. Bibliografie

http://www.w3schools.com/dtd/default.asp

XML SCHEMA
Prezentare

1. Despre XML Schema – scurta istorie

XML Schema a inceput ca o initiativa a companiei Microsoft. Intre timp insa, consortiul W3C a preluat aceasta initiativa si a dezvoltat un set mai larg de cerinte si caracteristici pentru documente ce descriu tipuri de documente. Acestea sunt reunite sub numele de **Document Content Description** si informatii despre acest proiect sunt disponibile la http://w3.org/TR/NOTE-dcd. XML Schema a devenit o recomandare W3C pe data de 02 Mai 2001.

Spre deosebire de (sau in plus fata de) DTD-uri, XML Schema permite definirea regulilor si relatiilor intre elemente si atribute folosind un fisier XML (cu alte cuvinte, o schema XML este descrisa intr-un fisier XML). In plus, se adauga support pentru *spatii de nume* (namespaces), tipuri de date si caracteristici mai avansate cum ar fi *constrangerile* (constraints).

XML Schema este o alternativa la DTD care este bazata pe standardul XML. Ca si DTD, XML Schema este folosita pentru a descrie structura documentelor XML. Acest standard mai este cunoscut si sub numele de XML Schema Definition (XSD).

2. XML Schema vs. DTD

Inca de la primele utilizari ale XML-ului in aplicatii a devenit evident ca adeseori definirea tipurilor de documente folosind fisiere DTD are cateva neajunsuri, dintre care enumeram:

- Limbajul DTD foloseste o alta sintaxa nu cea de XML Schema ceea ce presupune ca:
    - Utilizatorii sa invete doua “limbaje” (desi nici unul nu este dificil)

    - Aplicatiile care folosesc validarea de DTD sa contina componente separate de procesare a DTD-urilor
- Sintaxa DTD este relativ restransa. In cadrul DTD ar fi dificil, de exemplu, de declarat un tip de document XML care contine rapoarte organizate pe luni si zile. Astfel, o regula evidenta ar fi ca un element *luna* ar trebui sa contina intre 28 si 31 de elemente de tip *zi*, lucru care se poate specifica in DTD insiruind de 28 do ori *zi* si apoi definindu-le pe restul ca fiind optionale (?). Un alt exemplu este acela de verificare a corectitudinii structurii CNP-urilor (numere alcatuite din 13 cifre). XML Schema face ca astfel de structuri sa poata fi definite mult mai usor.
- DTD-urile au o capacitate foarte limitata de tipuri de date (suporta doar 10 tipuri de date). Pentru a exemplifica, in DTD nu putem spune ca un atribut este de tip intreg pozitiv, ci doar il putem defini ca sir de caractere, identificator sau fragment de nume.
- DTD-urile suporta attribute identificatori si referinte catre acestia (ID si IDREF). Nu se pot insa trasa relatii clare cum ar fi: atributul *zi_ref* este o referinta catre un atribut de tip identificator din elementul *zi*.
- DTD-urile nu suporta spatii de nume (namespaces).

Aceste neajunsuri au condus la aparitia XML Schema care prezinta urmatoarele avantaje:

- Sunt scrise folosind aceeasi sintaxa ca si fisierele pe care le descrie, ceea ce presupune mai putina sintaxa de memorat.
- Suporta peste 44 de tipuri de date si permite, de asemenea, definirea propriilor tipuri de date.
- Sunt orientate pe obiecte (se poate restrictiona sau extinde un anumit tip, derivand un nou tip pe baza unuia vechi).
- Se pot defini mai multe elemente care sa aiba acelasi nume dar continut diferit.
- Suporta spatii de nume (namespaces).
- Sunt extensibile ceea ce permite utilizarea unei Scheme in cadrul alteia, se pot referi mai multe scheme in cadrul aceluiasi document si se pot crea tipuri de date proprii derivate din tipurile standard

3. Sintaxa

Pentru o mai usoara intelegere a XML Schema, vom utiliza un document XML pe care vom arata cum se defineste si utilizeaza o Schema.

```
<?xml version="1.0"?>
<student>
        <nume>Ion</nume>
        <prenume>George</prenume>
        <adresa>undeva in Romania</adresa>
        <grupa>352C1</grupa>
</student>
```

REFERIREA unei XML Schema intr-un document XML

Pentru a putea referi o XML Schema intr-un document XML se adauga cateva atribute elementului radacina a documentului:

Ex:

```
<student
        xmlns="http://www.acs.ro/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
        xs:schemaLocation="http://www.acs.ro/ student.xsd">
```

Aceste atribute au urmatoarea semnificatie:

- xmlns="http://www.acs.ro/" – defineste namespace-ul default, specificand validatorului ca toate elementele folosite in documentul XML sunt declarate in namespaceul de la adresa respectiva
- xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" – instanta Schemei utilizate pentru respectivul namespace
- xs:schemaLocation="http://www.acs.ro/ student.xsd" – defineste namespace-ul utilizat (http://www.acs.ro/) si locatia XML Schemei ce va fi folosita pentru namespace-ul respectiv (student.xsd)

STRUCTURA unei XML Schema

Orice XML Schema are ca radacina elementul <schema> care poata sa contina niste atribute:

Ex:

```
<?xml version="1.0"?>
<xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.acs.ro/"
        xmlns="http://www.acs.ro/"
        elementFormDefault="qualified">
...
...
</xs:schema>
```

Aceste atribute au urmatoarea semnificatie:

- xmlns:xs="http://www.w3.org/2001/XMLSchema" – indica faptul ca elementele si tipurile de date utilizate in Schema provin din namespace-ul definit la adresa respectiva. Totodata, se mentioneaza si faptul ca pentru a folosi elementele si tipurile de date din acest namespace, ele trebuiesc prefixate cu xs (xs este un nume aleator pe care il dam noi)

- targetNamespace="http://www.acs.ro/" – indica faptul ca elementele definite de aceasta Schema (student, nume, prenume, adresa, grupa) provin din namespace-ul de la adresa respectiva
- xmlns="http://www.acs.ro/" – indica faptul ca namespace-ul default e cel de la adresa specificata
- elementFormDefault="qualified" – indica faptul ca fiecare element utilizat in documentul XML si care au fost definite in Schema trebuie sa fie calificate de namespace-ul respectiv

Observatie: Namespace-ul default este cel care nu are asociat nici un sufix.

In continuare vom prezenta intreaga schema a documentului XML definit mai sus si vom continua discutia pe baza acestei scheme:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.acs.ro/"
xmlns="http://www.acs.ro/"
elementFormDefault="qualified">
<xs:element name="student">
   <xs:complexType>
     <xs:sequence>
              <xs:element name="nume" type="xs:string"/>
              <xs:element name="prenume" type="xs:string"/>
              <xs:element name="adresa" type="xs:string"/>
              <xs:element name="grupa" type="xs:string"/>
     </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

Elementul student este definit ca avand un **tip complex** deoarece contine alte elemente. Celelalte elemente (nume, prenume, adresa, grupa) au **tipul simplu** pentru ca nu contin alte elemente in interiorul lor. Acestea pot contine numai text in interiorul lor, dar acest text poate fi de unul dintre tipurile existente in XML Schema sau poate fi de un tip definit de utilizator si i se pot impune anumite restrictii sau se poate specifica faptul ca datele trebuie sa respecte un anumit pattern.

DEFINIREA ELEMENTELOR SIMPLE se face folosind structura:

<element name="nume_element" type="tip_element"/>

In cazul in care namespace-ul default este “http://www.w3.org/2001/XMLSchema” sau <xs: element name="xxx" type="yyy"/> in cazul in care acest namespace este definit ca in schema de mai sus. Nume_element este numele elementului din documentul XML iar tip_element poate fi unul din tipurile definite in XML Schema (ex: string, decimal, integer, boolean, date, time).

Observatie: Daca "http://www.w3.org/2001/XMLSchema" nu este namespace-ul default, tipurile de date trebuiesc prefixate cu prefixul specific acestui namespace (ex: xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time).

Elementele pot avea valori default sau fixate:

```
<xs:element name="color" type="xs:string" default="red"/>
<xs:element name="color" type="xs:string" fixed="red"/>
```

In primul caz, daca lipseste continutul elementului, acesta va fi adaugat automat ca fiind "red". In cel de-al doilea caz, daca continutul exista si este diferit de "red", se va furniza o eroare, iar daca nu exista, se va adauga automat ca fiind "red".

Elementele simple **NU** pot sa contina elemente. Daca un element are atribut/e, atunci acesta trebuie sa fie definit ca un element complex. In schimb, atributele sunt tot timpul definite ca fiind atribute simple.

DEFINIREA ATRIBUTELOR se face similar cu definirea elementelor simple, avand aceleasi semnificatii si aceeasi observatie:

```
<attribute name="nume_atribut" type="tip_atribut"/>
```

De asemenea, se pot asocia valori default sau fixate:

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

In plus fata de elemente, se poate specifica si daca respectivul atribut poate sau nu sa lipseasca. In mod default, atributul este optional, adica atributul poate sa lipseasca. Pentru a specifica faptul ca atributul este obligatoriu, se adauga atributul use:

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Observatie: In momentul in care se specifica un anumit tip de date pentru un element sau pentru un atribut din XML, se impune o restrictie asupra continutului acestuia (de exemplu, daca tipul este „date", se va genera o eroare in momentul in care se gaseste informatia „Buna ziua"). De asemenea, se pot adauga restrictii suplimentare asupra elementelor sau atributelor, restrictii care se numesc fatete (facets).

Restrictiile sunt folosite pentru a defini acceptate pentru elementele si atributele XML. Restrictiile pot fi de maimulte feluri:

- **Restrictii asupra valorilor continute in interiorul tagurilor**

Folosind tipuri implicite	**Folosind tipuri explicite**	**Explicatie**
<xs:element name="varsta"> <xs:simpleType> <xs:restriction base="xs:integer"> <xs:minInclusive value="0"/> <xs:maxInclusive value="80"/> </xs:restriction> </xs:simpleType>	<xs:element name="varsta" type="tip_v"> <xs:simpleType name="tip_v"> <xs:restriction base="xs:integer"> <xs:minInclusive value="0"/> <xs:maxInclusive value="80"/>	Impunerea faptului ca elementul „varsta" sa aiba valori cuprinse intre 0 si 80

</xs:element>	</xs:restriction> </xs:simpleType> </xs:element>	

Observatie: Cele 2 expresii sunt echivalente din punct de vedere ar definirii elementului de tip varsta. In cazul celei de-a doua forme, tipul „tip_v" poate fi refolosit in cadrul altor elemente sau extins/restrictionat, deoarece nu este o parte a elementului varsta ca in primul caz.

- **Restrictii asupra seturilor de valori posibile in interiorul tagurilor**

Folosind tipuri implicite	Folosind tipuri explicite	Explicatie
<xs:element name="varsta"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="20"/> <xs:enumeration value="30"/> <xs:enumeration value="40"/> </xs:restriction> </xs:simpleType> </xs:element>	<xs:element name="varsta" type="tip_v"> <xs:simpleType name="tip_v"> <xs:restriction base="xs:string"> <xs:enumeration value="20"/> <xs:enumeration value="30"/> <xs:enumeration value="40"/> </xs:restriction> </xs:simpleType> </xs:element>	Impunerea faptului ca elementul „varsta" sa aiba una din valorile 20, 30 sau 40

- **Restrictii asupra seriilor de valori – folosind patterne**

Folosind tipuri implicite	Folosind tipuri explicite	Explicatie
<xs:element name="cifra"> <xs:simpleType> <xs:restriction base="xs:integer"> <xs:pattern value="[0-9]"/> </xs:restriction> </xs:simpleType> </xs:element>	<xs:element name="cifra" type="tip_c"> <xs:simpleType name="tip_c"> <xs:restriction base="xs:integer"> <xs:pattern value="[0-9]"/> </xs:restriction> </xs:simpleType> </xs:element>	Descrierea unei cifre ca fiind orice intreg cuprins intre 0 si 9
<xs:element name="numar"> <xs:simpleType> <xs:restriction base="xs:integer"> <xs:pattern value="0|[1-9]([0-9])*"/> </xs:restriction> </xs:simpleType> </xs:element>	<xs:element name="numar" type="tip_n"> <xs:simpleType name="tip_n"> <xs:restriction base="xs:integer"> <xs:pattern value="[1-9]([0-9])*"/> </xs:restriction> </xs:simpleType> </xs:element>	Descrierea unui numar ca fiind numarul 0 sau o insiruire de oricate cifre cu conditia ca prima cifra sa fie cuprinsa intre 1 si 9
<xs:element name="CNP"> <xs:simpleType> <xs:restriction base="xs:integer"> <xs:pattern value="(1|2)[0-9]{12}"/> </xs:restriction> </xs:simpleType> </xs:element>	<xs:element name="CNP" type="tip_cnp"> <xs:simpleType name="tip_cnp"> <xs:restriction base="xs:integer"> <xs:pattern value="(1|2)[0-9]{12}"/> </xs:restriction> </xs:simpleType> </xs:element>	Descrierea unui CNP ca fiind un numar ce incepe cu cifra 1 sau 2 si care are inca 12 cifre dupa aceasta prima cifra

- **Restrictii asupra caracterelor „whitespace”** – pentru a specifica modul in care sunt interpretate caracterele “whitespace” se foloseste restrictia whiteSpace. Aceasta poate lua 3 valori posibile: preserve, replace si collapse. In primul caz, caracterele whitespace (line feed, tab, space, si carriage return) sunt afisate asa cum sunt intatlnite. In cazul replace, acestea sunt inlocuite cu spatii, iar in ultimul caz caracterele whitespace sunt inlocuite cu un singur caracter spatiu.

preserve	**replace**	**collapse**
<xs:element name="address"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:whiteSpace value="preserve"/> </xs:restriction> </xs:simpleType> </xs:element>	<xs:element name="address"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:whiteSpace value="replace"/> </xs:restriction> </xs:simpleType> </xs:element>	<xs:element name="address"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:whiteSpace value="collapse"/> </xs:restriction> </xs:simpleType> </xs:element>

- **Restrictii supra lungimii** – pentru a limita lungimea valorii continutului unui element, se vor folosi restrictiile: length, maxLength, si minLength. In cazul restrictiei length, lungimea continutului este exact cea specificata de valoarea data lui length. In cazul maxLength/minLength, lungimea continutului textului trebuie sa fie mai mare/mica decat valoarea specificata**.**

Sintaxa	**Explicatie**
<xs:element name="password"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:length value="8"/> </xs:restriction> </xs:simpleType> </xs:element>	Campul “password” trebuie sa aiba exact 8 caractere
<xs:element name="password"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:minLength value="5"/> <xs:maxLength value="8"/> </xs:restriction> </xs:simpleType> </xs:element>	Campul “password” trebuie sa aiba intre 5 si 8 caractere

- **Restrictii pentru tipurile de date**

Constrangere	**Descriere**
enumeration	Defineste o lista de valori ce pot fi acceptate.
fractionDigits	Specifica numarul maxim de decimale permise. Trebuie sa fie mai mare sau egal cu zero.
length	Specifica numarul exact de caractere sau de itemuri dintr-o lista. Trebuie sa fie

	mai mare sau egal cu zero.
maxExclusive	Specifica limita superioara pentru valorile numerice. Valoarea trebuie sa fie strict mai mica decat valoarea specifica astfel.
maxInclusive	Specifica limita superioara pentru valorile numerice. Valoarea trebuie sa fie mai mica sau egala cu valoarea specifica astfel.
maxLength	Specifica numarul maxim de caractere sau de itemuri dintr-o lista. Trebuie sa fie mai mare sau egal cu zero.
minExclusive	Specifica limita inferioara pentru valorile numerice. Valoarea trebuie sa fie strict mai mare decat valoarea specifica astfel.
minInclusive	Specifica limita inferioara pentru valorile numerice. Valoarea trebuie sa fie mai mare sau egala cu valoarea specifica astfel.
minLength	Specifica numarul minim de caractere sau de itemuri dintr-o lista. Trebuie sa fie mai mare sau egal cu zero.
pattern	Specifica un patter pe care trebuiesa il respecte datele respective
totalDigits	Specifica numarul exact de digiti permisi. Trebuie sa fie mai mare sau egal cu zero
whiteSpace	Specifica cum sunt tratate whitespaceurile

Inainte de a arata cum se definesc **elemente complexe** trebuie mai intai sa specificam ce sunt acestea. Un element complex este un element complex care contine alte elemente sau/si atribute. Sunt 4 tipuri de elemente complexe si oricare din aceste tipuri poate sa aiba si atribute:

a) elemente ce contin numai alte elemente in interiorul lor – au numai copii, fara text (ex: <employee><firstname>John</firstname><lastname>Smith</lastname></employee>)
b) elemente vide – ce nu contin nimic in interiorul lor (ex: <product pid="1345"/>)
c) elemente care contin in interiorul lor numai text (ex: <mancare tip="desert">tort</mancare>)
d) elemente mixte – care contin in interiorul lor atat elemente cat si text (ex: <scrisoare>Stimate Domnule <nume>Ion</nume>, Scrisoarea dumneavoastra <sid>102</sid> va ajunge pe data <data>15-11-2008</data>.</scrisoare>)

DEFINIREA ELEMENTELOR COMPLEXE:

a) elemente ce au doar copii in componenta lor

Folosind tipuri implicite	**Folosind tipuri explicite**	**Explicatie**
<xs:element name="employee"> <xs:complexType> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element>	<xs:element name="employee" type="personinfo"/> <xs:complexType name="personinfo"> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </xs:sequence> </xs:complexType>	Se specifica faptul ca elementul employee are doar doi copii (firstname si lastname) care trebuie sa apara strict in aceasta ordine datorita indicatorului <xs:sequence>. Folosirea tipului implicit il face inutilizat in alta parte spre deosebire de folosirea tipurilor explicite care permite

		refolosirea acestuia precum si crearea de noi elemente complexe pe baza acestuia

Exemplu de refolosire si extindere a tipului explicit „personinfo”:

Sintaxa	**Explicatie**
<xs:element name="employee" type="personinfo"/> <xs:element name="student" type="personinfo"/> <xs:element name="member" type="personinfo"/>	Tipul personinfo este utilizat atat la definirea elementului employee cat si la definirea elementelor student si member
<xs:complexType name="fullpersoninfo"> <xs:complexContent> <xs:extension base="personinfo"> <xs:sequence> <xs:element name="address" type="xs:string"/> <xs:element name="city" type="xs:string"/> <xs:element name="country" type="xs:string"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>	In acest fel s-a definit tipul fullpersoninfo care are un continut complex (<xs:complexContent>) si porneste de la tipul personinfo extinzandu-l (<xs:extension base="personinfo">) cu o secventa de alte elemente (<xs:sequence>… </xs:sequence>)

b) elemente vide (ex: <product pid="1345"/>) – pentru a defini un tip vid, trebuie sa definim un tip care sa contina numai elemente in interiorul sau, dar caruia sa nu ii adaugam, de fapt, nici un element. Astfel, el este definit ca fiind un tip complex, cu un continut complex, ceea ce arata faptul ca intentionam sa extindem sau sa restrangem continutul unui element de tip complex.

Sintaxa	**Sintaxa simplificata**	**Sintaxa simplificata folosind tipuri explicite**
<xs:element name="product"> <xs:complexType> <xs:complexContent> <xs:restriction base="xs:integer"> <xs:attribute name="prodid" type="xs:positiveInteger"/> </xs:restriction> </xs:complexContent> </xs:complexType> </xs:element>	<xs:element name="product"> <xs:complexType> <xs:attribute name="prodid" type="xs:positiveInteger"/> </xs:complexType> </xs:element>	<xs:element name="product" type="prodtype"/> <xs:complexType name="prodtype"> <xs:attribute name="prodid" type="xs:positiveInteger"/> </xs:complexType>

c) elemente care contin in interiorul lor numai text (ex: <mancare tip="desert">tort</mancare>) – acest tip contine numai continut simplu (text sau/si atribute). Pentru a defini un astfel de tip, se foloseste „simpleContent” si trebuie utilizata o extensie sau o restrictie.

Restrictie	**Extensie**
<xs:element name="nume"> <xs:complexType> <xs:simpleContent> <xs:restriction base="tip"> .... </xs:restriction> </xs:simpleContent> </xs:complexType> </xs:element>	<xs:element name="nume"> <xs:complexType> <xs:simpleContent> <xs:extension base="tip"> .... </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>

Observatie: Extensia/restrictia se foloseste pentru a extinde sau limita tipul de baza.

Exemplu:

Sintaxa	**Sintaxa cu tip explicit**
<xs:element name="mancare"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="tip" type="xs:string" /> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>	<xs:element name="mancare" type="tip_mancare"> <xs:complexType name="tip_mancare"> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="tip" type="xs:string" /> </xs:extension> </xs:simpleContent> </xs:complexType>

e) elemente mixte (ex: <scrisoare>Stimate Domnule <nume>Ion</nume>, Scrisoarea dumneavoastra <sid>102</sid> va ajunge pe data <data>15-11-2008</data>.</scrisoare>)

Folosind tipuri implicite	**Folosind tipuri explicite**	**Explicatie**
<xs:element name="scrisoare"> <xs:complexType mixed="true"> <xs:sequence> <xs:element name="nume" type="xs:string"/> <xs:element name="sid" type="xs:positiveInteger"/> <xs:element name="data" type="xs:date"/> </xs:sequence> </xs:complexType> </xs:element>	<xs:element name="scrisoare" type="tip_s"/> <xs:complexType name="tip_s" mixed="true"> <xs:sequence> <xs:element name="nume" type="xs:string"/> <xs:element name="sid" type="xs:positiveInteger"/> <xs:element name="data" type="xs:date"/> </xs:sequence> </xs:complexType>	Pentru a permite sa apara texte intre copiii elementului scrisoare, trebuie ca atributul „mixed” sa aiba valoarea „true”. Tagul <xs:sequence> specifica faptul ca elementele definite in cadrul ei (nume, sid, data) trebuie sa apara in ordinea specificata in elementul scrisoare.

INDICATORI – sunt sapte indicatori grupati in 3 categorii:

- **indicatori de ordine (Order indicators)**: all, choice, sequence – sunt utilizati pentru a specifica ordinea elementelor.
  - **All** – toti copii trebuie sa apara o singura data in cadrul elementului parinte, dar ca pot aparea in orice ordine.
  - **Choice** – orice copil poate sa apara in cadrul elementului parinte
  - **Sequence** – toti copiii trebuie sa apara in cadrul elementului parinte in ordinea specificata

All	**Choice**	**Sequence**
<xs:element name="person"> <xs:complexType> <**xs:all**> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </**xs:all**> </xs:complexType> </xs:element>	<xs:element name="person"> <xs:complexType> <**xs:choice**> <xs:element name="employee" type="employee"/> <xs:element name="member" type="member"/> </**xs:choice**> </xs:complexType> </xs:element>	<xs:element name="person"> <xs:complexType> <**xs:sequence**> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </**xs:sequence**> </xs:complexType> </xs:element>

- **indicatori de aparitie (Occurrence indicators)**: maxOccurs, minOccurs – sunt utilizati pentru a specifica de cate ori poate sa apara un element copil in cadrul parintelui sau
  - **maxOccurs** – specifica numarul maxim de aparitii a respectivului element
  - **minOccurs** – specifica numarul minim de aparitii a respectivului element

Sintaxa	**Explicatie**
<xs:element name="persoana"> <xs:complexType> <xs:sequence> <xs:element name="liceu" type="xs:string"/> <xs:element name="facultate" type="xs:string" maxOccurs="5" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>	Sintaxa defineste un element „persoana” care are 2 tipuri de copii: un copil „liceu” si un copil „facultate” care poate sa apara de un numar de ori cuprins intre 0 si 5

Observatii:

1) Pentru toti ceilalti indicatori (de ordine sau de grup), valorile implicite pentru maxOccurs si minOccurs sunt 1.
2) Pentru a permite unui element sa apara de oricate ori, se foloseste maxOccurs=”unbounded”.

- **indicatori de grup (Group indicators)**: Group name, attributeGroup name – folosite pentru a defini grupuri de elemente/atribute inrudite
    - **Group name** – se defineste folosind <group name="groupname">...</group>. In interiorul grupului trebuie definit unul din indicatorii de ordine. Dupa definirea grupului, acesta poate fi referit in alta structura:

Definirea unui grup de elemente	**Referirea acestui grup in alt element**
<xs:group name="persongroup"> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> <xs:element name="birthday" type="xs:date"/> </xs:sequence> </xs:group>	<xs:element name="person" type="personinfo"/> <xs:complexType name="personinfo"> <xs:sequence> <xs:group ref="persongroup"/> <xs:element name="country" type="xs:string"/> </xs:sequence> </xs:complexType>

    - attributeGroup name – se defineste folosind <attributeGroup name="groupname">...</attributeGroup>. Dupa definirea grupului, acesta poate fi referit in alta structura:

Definirea unui grup de atribute	**Referirea acestui grup in alta structura**
<xs:attributeGroup name="personattrgroup"> <xs:attribute name="firstname" type="xs:string"/> <xs:attribute name="lastname" type="xs:string"/> <xs:attribute name="birthday" type="xs:date"/> </xs:attributeGroup>	<xs:element name="person"> <xs:complexType> <xs:attributeGroup ref="personattrgroup"/> </xs:complexType> </xs:element>

Pentru a permite **extinderea Schemelor XML**, sintaxa permite folosirea unor elemente de tip whildcard (**<any> si <anyAttribute>**) ce permit imbogatirea Schemelor cu elemente/atribute a caror definitie nu este data in Schema respectiva.

Sintaxa	Explicatie
<xs:element name="person"> <xs:complexType> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> <xs:any minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>	Prin folosirea elementului any, continutul elementului „person” poate fi extins cu un alt element, care poate sa nu fie declarat in Schema curenta. Astfel, daca se declara un element „child” intr-o alta schema si apoi se specifica in documentul XML ca se vor folosi ambele scheme, atunci elementul persoana poata ca contina ca si copil si elementul „child” si astfel documentul XML sa fie valid.

Pentru o mai buna intelegere a elementelor <any> si <anyAttribute>, se pot urmari exemplele de la adresele:

http://www.w3schools.com/schema/schema_complex_any.asp respectiv
http://www.w3schools.com/schema/schema_complex_anyattribute.asp

In unele cazuri, se doreste ca in documentul XML sa existe alternative. De exemplu, se poate defini un element adresa in care unul dintre campuri sa fie judetul, pentru locuitorii din provincie sau sectorul pentru locuitorii din capitala. Pentru astfel de situatii, XML Schema pune la dispozitie atributul **substitutionGroup**. Astfel se pot defini doua elemente in care unul sa fie un substitut pentru celalalt:

```
<xs:element name="judet" type="xs:string"/>
<xs:element name="sector" substitutionGroup="judet"/>
```

Astfel, daca avem o XML Schema care defineste o adresa in felul urmator:

```
<xs:element name="judet" type="xs:string"/>
<xs:element name="sector" substitutionGroup="judet"/>
<xs:complexType name="adresa">
  <xs:sequence>
    <xs:element ref="judet"/>
  </xs:sequence>
</xs:complexType>
```

Atunci, urmatorul document XML este valide:

```
<adress book>
    <adresa><judet>Calarasi</judet></adresa>
    <adresa><sector>trei</sector></adresa>
</adress book>
```

Observatii:

1) Tipul elementelor care se doresc a substitui trebuie sa fie acelasi sau, in cel mai rau caz, derivate din tipul de baza (care se doreste a putea fi substituit). Daca tipul este acelasi, atunci nu mai este nevoie sa fie definit tipul elementelor care vor substitui elementul de baza
2) Pentru a putea folosi substituirea este necesar ca elementele implicate in aceasta sa fie declarate global (adica sa fie copii directi ai elementului „schema”)
3) Este posibila impiedicarea incercarii de substitutie, prin folosirea atributului block care trebuie sa aiba valoarea "substitution". Astfel, daca in exemplul de mai sus, elementului judet i se asociaza block="substitution", atunci documentul XML nu mai e valid:

```
<xs:element name="judet" type="xs:string" block="substitution"/>
```

Tipuri de date:

- String – folosit pentru date ce au valori ce contin siruri de caractere. Din acest tip sunt derivate urmatoarele tipuride date: normalizedString, token, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, Qname

- Date – folosit pentru date ce contin informatii legate de timp sau de date. Din acest tip sunt derivate: date, time, dateTime, duration, gDay, gMonth, gMonthDay, gYear, gYearMonth
- Decimal - folosit pentru date ce au valori de tip numeric. Din acest tip sunt derivate: byte, decimal, int, integer, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, short, unsignedLong, unsignedInt, unsignedShort, unsignedByte
- Boolean
- Binare:
  - base64Binary (Base64-encoded binary data)
  - hexBinary (hexadecimal-encoded binary data)
- Altele: anyURI, double, float, NOTATION, Qname

4. Cazuri de utilizare

XML Schema poate fi utilizat oriunde se folosesc documente XML pentru a specifica structura acestora.

5. Instrumente

In prezent, exista o serie de unelte dezvoltate pentru XML Schema: XMLSpy/Altova, Stylus Studio 2007, oXygen XML, Xerces-C++ 2.8.0, xnsdoc 1.2 - XML Schema documentation generator samd. O lista mai completa se gaseste la adresa: http://www.w3.org/XML/Schema. In general, aceste instrumente ofera posibilitatea validarii Schemei XML, validarea documentului XML cu Schema asociata, generarea automata a XML Schema pe baza unui XML, precum si conversia DTD-urilor in XML Schema.

6. Concluzii

XML Schema este un standard utilizat pentru definirea structurii fisierelor XML la fel ca si DTD-ul. Din motive prezentate in capitolul al doilea, credem ca XML Schema tinde sa inlocuiasca in viitor DTD-ul, reprezentand principala metoda de a defini structura acestuia. De asemenea, s-a observat si faptul ca XML Schema este un limbaj foarte bogat si mai ales preocuparea dezvoltatorilor acestui limbaj pentru reutilizare si extensie.

7. Bibliografie

1. Validator XML Schema http://www.w3.org/2001/03/webdata/xsv
2. Validator XML Schema http://www.validome.org/grammar/
3. http://www.w3.org/TR/xmlschema11-1/

4. http://www.w3.org/XML/Schema
5. Tutorial XML Schema http://www.w3schools.com/schema/default.asp
6. http://www.w3.org/TR/xmlschema-0/
7. http://www.w3.org/TR/xmlschema-1/
8. http://www.w3.org/TR/xmlschema-2/
9. Tutorial XML Schema http://www.xfront.com/xml-schema.html
10. Tutorial XLM Schema www.utdallas.edu/~lkhan/Fall2003/xml-schema_2.pdf

Anexa 1. – Exemplu de document XML

```
<?xml version=”1.0” encoding=”ISO-8859-2”?>
<persoana>
<nume>Popescu</nume>
<prenume>Ion</prenume>
<informatii_personale>
        <oras>Bucuresti</oras>
        <strada>Calea Mosilor</strada>
        <nr>100</nr>
        <tel_fix>0213003000</tel_fix>
        <tel_mobil></tel_mobil>
        <e-mail>popescu.ion@gmail.com</e-mail>
        <casatorit />
</informatii_personale>
<tranzactii>
        <tranzactie>
                <data>20-07-2005</data>
                <ora>13:00:00</ora>
                <suma moneda=”RON”>1000</suma>
                <directia>venit</directia>
                <descriere>salariu</descriere>
        </tranzactie>
        <tranzactie>
                <data>20-07-2005</data>
                <ora>13:05:00</ora>
                <suma moneda=”RON”>35</suma>
                <directia>plata</directia>
                <descriere>telefon</descriere>
        </tranzactie>
</tranzactii>
</persoana>
```

Anexa 2. Fisier de configurare Tomcat – web.xml pentru o aplicatie jsp

```
  <?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
    <context-param>
        <param-name>language_def</param-name>
        <param-value>/WEB-INF/languagedef.xml</param-value>
        <description>File name of language definition file.</description>
    </context-param>
    <servlet>
        <servlet-name>Dispatcher</servlet-name>
        <servlet-class>Util.Dispatcher</servlet-class>
    </servlet>
```

```
 <servlet>
        <servlet-name>Logout</servlet-name>
        <servlet-class>Util.Logout</servlet-class>
  </servlet>
 <servlet>
        <servlet-name>SetSearchProjects</servlet-name>
        <servlet-class>Util.SetSearchProjects</servlet-class>
  </servlet>
  ...
 <servlet-mapping>
        <servlet-name>Dispatcher</servlet-name>
        <url-pattern>/Dispatcher</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
        <servlet-name>Manager</servlet-name>
        <url-pattern>/Manager</url-pattern>
  </servlet-mapping>
  ...
 <session-config>
        <session-timeout>30</session-timeout>
  </session-config>
 <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
  </welcome-file-list>
</web-app>
```

Anexa 3. Fisierul content.xml pentru un document Open Office in care apare textul „Hello World”

```
<?xml version="1.0" encoding="UTF-8" ?>
<office:document-content
    xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
    xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"
    xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
    xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
    xmlns:draw="urn:oasis:names:tc:opendocument:xmlns:drawing:1.0"
    xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
    xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
    xmlns:svg="urn:oasis:names:tc:opendocument:xmlns:svg-compatible:1.0"
    xmlns:chart="urn:oasis:names:tc:opendocument:xmlns:chart:1.0"
    xmlns:dr3d="urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0"
    xmlns:math="http://www.w3.org/1998/Math/MathML"
    xmlns:form="urn:oasis:names:tc:opendocument:xmlns:form:1.0"
    xmlns:script="urn:oasis:names:tc:opendocument:xmlns:script:1.0"
    xmlns:ooo="http://openoffice.org/2004/office"
    xmlns:ooow="http://openoffice.org/2004/writer"
    xmlns:oooc="http://openoffice.org/2004/calc"
    xmlns:dom="http://www.w3.org/2001/xml-events"
    xmlns:xforms="http://www.w3.org/2002/xforms"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    office:version="1.0">
    <office:scripts />
    <office:font-face-decls>
        <style:font-face style:name="Tahoma1" svg:font-family="Tahoma" />
        <style:font-face style:name="Arial Unicode MS" svg:font-family="'Arial
        Unicode MS'" style:font-pitch="variable" />
        <style:font-face style:name="Tahoma" svg:font-family="Tahoma"
        style:font-pitch="variable" />
        <style:font-face style:name="Times New Roman" svg:font-family="'Times
        New Roman'" style:font-family-generic="roman" style:font-
        pitch="variable" />
    </office:font-face-decls>
    <office:automatic-styles>
        <style:style style:name="P1" style:family="paragraph" style:parent-style-
        name="Standard">
            <style:text-properties fo:color="#800000" fo:font-weight="bold"
            style:font-weight-asian="bold" style:font-weight-complex="bold" />
        </style:style>
    </office:automatic-styles>
    <office:body>
        <office:text>
            <office:forms form:automatic-focus="false" form:apply-design-
            mode="false" />
            <text:sequence-decls>
```

```
                    <text:sequence-decl text:display-outline-level="0"
                    text:name="Illustration" />
                    <text:sequence-decl text:display-outline-level="0"
                    text:name="Table" />
                    <text:sequence-decl text:display-outline-level="0"
                    text:name="Text" />
                    <text:sequence-decl text:display-outline-level="0"
                    text:name="Drawing" />
                </text:sequence-decls>
                <text:p text:style-name="P1">Hello World!</text:p>
            </office:text>
        </office:body>
    </office:document-content>
```