Az XML (2. rész)

Folytatjuk gondolatfüzérünket a hordozható adatokról, vagyis a mindenki által könnyen használható dokumentumok készítéséről.

Sorozatunk második részében befejezzük az XML-adatfolyamok formai és nyelvtani érvényességének rövid ismertetését, majd áttekintjük azokat a fontosabb módszereket, amelyek az XML formát nagyon hatékony és széleskörűen használható eszközzé teszik.

A DTD tulajdonságlista (attributum list) megadása

Az eddigiekből kiderült, hogy az elemtulajdonságok az egyes XML-elemek jellemzésére szolgálnak, ezért a DTD-re alapuló nyelvtani érvényességvizsgálat során ezek elemzése is fontos követelmény. Természetesen nem mindegyik XML-elemnek van tulajdonságlistája, de ha létezik, akkor olyan DTD-beli nyelvtani szabályokat kell megfogalmaznunk, amelyek leírják, hogy az egyes elemek milyen jellemzőket tartalmazhatnak, illetve milyen értékeket vehetnek fel.

A VKONYV XML-adatfolyam <VENDEG>-elemének egyetlen tulajdonsága a sorszám. A tulajdonságlisták megadásának általános formája a következő:

```
<!ATTLIST egy_xml_elem_neve
          tulajdonsÆg_nØv1
                 tulajdonsÆg_jellege1 jelzı
           ...
          tulajdonsÆg_nØv_n
                 tulajdonsÆg_jellege_n jelzı
>
```

A <VENDEG>-elem példájánál maradva így adhatjuk meg a sorszam tulajdonság nyelvtanát:

```
<!ATTLIST VENDEG sorszam CDATA #REQUIRED>
```

A fenti sor mondja meg az elemző program számára, hogy a VENDEG elemnek létezik egy sorszam nevű tulajdonsága, amely CDATA-jellegű és az elem minden előfordulásánál kötelezően szerepelnie kell.

Az egyes tulajdonságok jellege a következők valamelyike lehet:

- CDATA – ezzel csak annyit állítunk, hogy a tulajdonság egy tetszőleges karaktersorozat
- ID – az ID-tulajdonság értékének egy névnek kell lennie,
- IDREF vagy IDREFS,
- ENTITY vagy ENTITIES,
- NMTOKEN vagy NMTOKENS,
- a névlista egyszerű felsorolástípus, például alma, körte, szilva.

Jelzőből a következő négy fajta ismeretes:

- #REQUIRED: az elem minden dokumentumbeli előfordulásánál egyértelműen meg kell adni a tulajdonság értékét.
- #IMPLIED: a tulajdonság megadása nem kötelező és nincs alapértelmezett értéke. Ha nincs érték megadva, az XML-feldolgozónak anélkül kell továbbhaladnia.
- Egyetlen érték: ekkor ez az egyetlen alapértelmezett értéke lesz a tulajdonságnak, például: "3,14"
- #FIXED „érték”: ilyenkor nem kötelező a tulajdonság megadása, de ha megtesszük, akkor ennek az értéknek kell lennie.

Egyedbevezetés (Entity declaration) a DTD-ben

Az XML-adatfolyam jól formált felépítését ismertető szakaszban már szó esett az egyedekről – most azt is leírjuk, hogyan és hol hozhatjuk létre őket.

A DTD-leírásban a következő három fajta egyed lehetséges: ***belső***, ***külső*** és ***paraméteregyedek***.

A ***belső*** egyedek makróhelyettesítő módszert valósítanak meg. Az

```
<!ENTITY MOL "Magyar Olaj Øs GÆzipari Rt.">
```

sor egy MOL nevű egyedet hoz létre, amelyet az XML-szövegben "&MOL;" alakban hívhatunk meg, és ennek hatására történik meg a szöveghelyettesítés.

A külső egyedek lehetővé teszik, hogy az XML-dokumentumból egy másik fájl tartalmára hivatkozzunk. Így szöveges és futtatható adatot is tartalmazhatnak. Ha a külső fájl szöveges adatot tartalmaz, az az eredeti dokumentumban a hivatkozási pont helyére illesztődik, és úgy kerül feldolgozásra, mintha a hivatkozó dokumentum része lenne. A futtatható adat értelemszerűen nem kerül elemzésre.

Paraméteregyedek csak a DTD-leírásban fordulhatnak elő.

1. lista

```
<!ELEMENT VKONYV (VENDEG)* >
<!ELEMENT VENDEG (NEV, EMAIL?, DATUM, SZOVEG)>
<!ATTLIST VENDEG sorszam CDATA #REQUIRED>
<!ELEMENT NEV (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT DATUM (#PCDATA)>
<!ELEMENT SZOVEG (#PCDATA)>
```

2. lista

```
<?xml version="1.0" encoding="WINDOWS-1250" ?>
<?xml-stylesheet type="text/css"
                 href="vkonyv.css" ?>
<VKONYV>
  <VENDEG sorszam="1">
...
</VKONYV>
```

3. lista

```
NEV { display: block; font-size: 20pt;
                 font-weight: bold; }
EMAIL { display: block; font-size: 15pt;
                 font-weight: bold; }
DATUM { display: block; font-size: 12pt;
                 font-weight: bold; }
SZOVEG { display: block; font-size: 10pt;
                 font-weight: bold; }
```

A paraméteregyedek megadása onnan ismerhető fel, hogy a nevük előtt a „%” (százalékjel) található. A paraméteregyed-hivatkozások a DTD-ben minden hivatkozás során feloldásra kerülnek, így a helyettesített szöveg is része a megadásnak. Nézzünk egy példát – eredetileg ezt írnánk:

```
<!ELEMENT KONYV (ELOLAP, LAPOK+, ZAROLAP)>
<!ELEMENT FUZET (ELOLAP, LAPOK+, ZAROLAP)>
```

A hasonlóság kihasználásával hozzunk létre egy *TARTALOM* nevű paraméteregyedet:

```
<!ENTITY %TARTALOM "ELOLAP, LAPOK+, ZAROLAP">
```

Az így kapott egyeddel a `KONYV` és `FUZET` elemek nyelvtani szabályai egységesebben fogalmazhatók meg:

```
<!ELEMENT KONYV (%TARTALOM;)>
<!ELEMENT FUZET (%TARTALOM;)>
```

A VKONYV XML-adatfolyam nyelvtani szabályai

A DTD-vel való ismerkedésünk befejezéseként alkossuk meg a VKONYV XML-adatfolyam nyelvtani szabályait (***1. lista***), amit a vkonyv.dtd fájlban tárolhatunk.

Az XML-adatfolyam megjelenítése

Az XML-fájlok semmilyen adatot nem tartalmaznak arra nézve, hogyan kell őket megjeleníteni. Azt láttuk, hogy az IE5 vagy a Netscape 6 alaphelyzetben faformában jeleníti meg az XML-leírásokat, kihasználva, hogy egy-egy fával lehet jellemezni őket. Az XML-dokumentumok megjelenítésére jelenleg két szabványos megoldás létezik:

- a CSS (Cascade Stylesheet) és
- az XSL (Extensible Stylesheet Language).

A CSS

A CSS részletes leírása a W3C webhelyről tölthető le (körülbelül 350 oldal), itt csak a használatát mutatjuk be. A CSS-leírások jellemzően ****.css*** fájlokban találhatók. Legyen adott egy ***vkonyv.css*** látványtervleírásunk. Ekkor az első változatú VKONYV XML-karakterfolyam megjelenítésére a CSS használatát a ***2. listán*** látható utasítással írhatjuk elő a megjelenítő alkalmazás számára. A szemléletesség kedvéért egy nagyon egyszerű CSS-t (lásd ***3. lista***) készítettem, amelyet a vkonyv.css fájlban tároltam.

A ***vkonyv.xml*** fájlba beszúrt vkonyv.css stíluslap-hivatkozás eredményeként az IE5 az XML-adatfolyamunkat az alapértelmezett fa helyett szintenként jeleníti meg. Érdekességképpen: ezt a formát a szabvány azért nevezi cascade-nak, mert a megjelenítő alkalmazás a megjelenítési beállítások különféle szintjeit egymás után vizsgálja meg. (Ezek a következő szintek: a böngésző alapértelmezései, egy külső stíluslap meghatározásai, a `<style>...</style>` tag által helyileg megadott lehetőség; majd a közvetlen stílusjegyek: `<B>`, `<FONT>`).

Az XSL

Az XSL több, mint egyszerű stíluslap-meghatározó nyelv, inkább olyan szövegfeldolgozónak tekinthető, mint a Perl vagy az AWK nyelvek. A feldolgozás eredmény-karaktersorozata természetesen egy HTML-dokumentum is lehet, amit a böngészők meg tudnak jeleníteni.

Az XSL két önálló nyelvi részből áll: az átalakító nyelvből (XSL Transformation) és a formázó nyelvből (XSL Formatting Objects, XSL-FO).

Az Apache Software Foundation több projektje is foglalkozik olyan ingyenes, a GNU/GPL felhasználási szerződésének hatálya alá eső és a Java nyelvre épülő rendszerek előállításával, amelyek XML-dokumentumok XSL-alapú megjelenítésével foglalkoznak. Ezen programok legfontosabb képviselői a következők:

- Cocoon – XML-alapú webes kiadványkészítő csomag,
- Xalan, mely az XSLT nyelvet képes értelmezni és futtatni,
- FOP (Format Objects Processor) – XSL formázóobjektumokat valósít meg (ezeket elsősorban a nem szövegalapú eredménydokumentumoknál használjuk).

5. lista A vkonyv1.xsl fájl tartalma

```
<?xml version="1.0" encoding="WINDOWS-1250" ?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
        <html><body> <H1>VEND GK NYV</H1>
      <xsl:apply-templates/>
      </body></html>
  </xsl:template>
  <xsl:template match="VKONYV"> <br/>
  <xsl:apply-templates/> <hr/>
  </xsl:template>
  <xsl:template match="VKONYV/VENDEG">
  <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="VKONYV/VENDEG/NEV">
     <hr/>Neve: <b>
     <xsl:value-of select="."/> </b>
  </xsl:template>
  <xsl:template match="VKONYV/VENDEG/EMAIL">
     Email c me: <b>
     <xsl:value-of select="."/> </b>
  </xsl:template>
</xsl:stylesheet>
```

Az XSL átalakító nyelv elemeivel olyan szabályokat határozhatunk meg, amelyek segítségével egy XML-dokumentumból egy másik dokumentumot, illetve általánosabban fogalmazva bármilyen bájtsorozatot (html-, pdf-, rtf-, ps-dokumentumokat) hozhatunk létre. Ennek következtében a létrehozott dokumentum nem szükségszerűen lesz XML-megfelelő. Az XSL Formatting Objects (XSL-FO) az XSL-ajánlás része. Ez a CSS-hez hasonlóan meghatározza az XML-dokumentumok kinézetének leírási lehetőségeit. A módszer a különböző tipográfiai, nyomdai kiadványoknál megszokott oldal- és szövegformázási elemek (fejléc, lábléc, tartalomjegyzék stb.) szabályainak leírásához is eszközöket szolgáltat. A szemléletesség érdekében elkészítettük a ***vkonyv1.xsl*** fájlleírását, amelyet a 4. (20. CD Magazin/XML) és ***5. listán*** látható módon – a CSS-hez hasonlóan – építhetünk be egy XML-dokumentumba.

Az ***ábra*** mutatja a VKONYV XML adatfolyam kinézetét akkor, amikor a vkonyv1.xsl stíluslapot használtuk. Észrevehető, hogy ez a módszer a megjelenítendő adatok körét is könnyen kezelni tudja (itt most csak a ***Név*** és az ***E-mail*** lett megjelenítve). Az XSL-lapok az egyes elemek megjelenítéséhez a mintaillesztés módszerét használják. A tagnevek `"xsl:"`-tal kezdődnek, utalva rá, hogy itt olyan XML-elemek találhatók, amelyeket az XSL-alkalmazás

VENDÉGKÖNYV

Neve: Nyíri Imre Email címe: irinyi@mol.hu

Neve: Koller József Email címe: jkoller@mailbox.hu

7. lista Egy SAX-elemző és -feldolgozó

```
//
import java.net.*;
import org.xml.sax.*;
import oracle.xml.parser.v2.*
//
//
// Az osztÆly leszÆrmazottja az
// alapØrtelmezett
// SAX-esemØnyeket kezelı osztÆlynak
//
public class TVendegListaSAX
             extends HandlerBase
{
    public static int vendeg_cnt = 0;
// Itt szÆmoljuk a vendØgeket

    //--------------------------------------
    // Ez egy esemØnykezelı eljÆrÆs, amely
    // az elemre lØpØskor mindig  nmßk dıen
    // megh v dik
    //--------------------------------------
    public void startElement( String name,
                AttribiteList attrlist )
    {
       if ( name.equals("VENDEG") )
       {
            vendeg_cnt++;
            szÆmoljuk a ➥vendØgeket
       }
    } // end startElement

   //-------
   // main
   //-------
   public static void main(String[] args)
   {

      if ( arg.length != 1 )
      {
        System.out.println
("HasznÆlat: java TVendegListaSAX xml_fÆjl");
      }

       // Az esemØnykezelı osztÆly
       // lØtrehozÆsa
       TVendegListaSAX esemenykezelo = new
                    TVendegListaSAX();

       // Egy SAX parser objektum
       // lØtrehozÆsa
       SAXParser parser = new SAXParser();

      // Az esemØnykezelı bejegyzØse a
      // most lØtrej tt parser objektumnÆl
      parser.setDocumentHandler
                       ( esemenykezelo );

      // Kezdj k az elemzØst, ek zben minden
      // œj VENDEG elemre lØpØskor  nmßk dıen
      // megh v dik a startElement tagf ggvØny,
      // azaz szÆmoljuk a vendØgeket
      try
      {
           parser.parse( UrlFromFÆjl
           ➥.createURL(args[0]));
      }
      catch (Exception)
      {
           System.out.println
            ("HIBA az elemzØs sorÆn!");
      }

        //--- BefejezØs l ki rjuk az
        // eredmØnyt a konzolra
        System.out.println("VendØgek szÆma:"
           + Integer(vendeg_cnt).toString() );

    } // end main

} // end TVendegListaSAX class
```

keretében akarunk használni. Ez a hivatkozási forma egy XML-névtér (namespace) szabványra épül, melynek célja, hogy az egyes XML-alkalmazások tagjai külön-külön névtérben legyenek (hasonlóan a C++ névtereihez). Ez esetben az `"xsl:"`-ben az `"xsl"` szó a névtér neve.

Röviden nézzük át, hogyan határozza meg a megjelenést a vkonyv1.xsl fájl tartalma. A fájl első sora arra utal, hogy XML-adatfolyam fog következni. A második sor az `"xsl"` névteret vezeti be. A 3–7 sor azt mondja, hogy az eredmény-karaktersorozatba az egyes elemek megjelenítése előtt írjuk ki a `"<html><body><H1>VEND GK NYV</H1>"` füzért, majd a mintaleírásokat (`apply-templates`) alkalmazva kezdjük el feldolgozni az XML-fa egyes elemeit. Az elemfeldolgozások után a befejező lépés, hogy a készítendő HTML-dokumentumot is befejezzük, azaz kiírjuk a `"</body></html>"` karakterfüzért. A következő sablonszakaszok arról rendelkeznek, hogy mi a teendője a fa bejárása során, illetve akkor, amikor adott XML-elemnél tart a feldolgozás. A `<xsl:template match="VKONYV/VENDEG/NEV">` sorban a `"VKONYV/VENDEG/NEV"` kifejezést XPathnak hívjuk. Az XSL-vezérlő működését jobban átgondolva világossá válik, hogy az `"apply-templates"` kulcsszóval egy rekurzív meghatározást (definíciót) tudunk a feldolgozásra megadni. Az `<xsl:value-of select="."/>` mindig a pillanatnyi faelem (node) értékét adja vissza.

Az IE5 azért tudja megjeleníteni XSL-stíluslappal ellátott XML-fájlunkat, mert ismeri az XSL átalakító nyelvet. Érdekességképpen megemlítjük, hogy az Apache Xalan használatával közvetlenül is előállíthatunk egy (XML, XSL) pár által meghatározott eredményfájlt. Nézzük meg a Xalan közvetlen használatát a fenti példára!

```
//A Xalan hasznÆlata (Java-alkalmazÆs, melyet
//a Process-osztÆly ind t)
java org.apache.xalan.xslt.Process -IN
➥vkonyv1.xml -XSL vkonyv1.xsl -OUT e.html
```

A parancs végrehajtása után egy ***e.html*** eredményfájl kelet-

kezik, amely a megadott (XML, XSL) pár alapján készült el. Ez a folyamat azért fontos, mert így a HTML-fájlok dinamikus előállítása kiszolgálóoldalon történhet.

Az XML-adatfolyam elemzése (Parse)

Az XML-dokumentumokhoz megadott nyelvtanok egyrészt egy új dokumentum létrehozásakor, másrészt egy létező dokumentum érvényességének ellenőrzéséhez szükségesek. Az XML-dokumentumok elemzésére három szabványos API terjedt el (a harmadik csak Javából használható):

- DOM-alapú (Document Object Modell alapú API) elemzők,
- SAX-alapú (Simple API for XML) elemzők,
- JAPX – Java API for XML Parsing.

A DOM felépíti az XML-adatfolyamnak megfelelő objektumokból álló fát, melynek elemeit ezután közvetlenül el lehet érni. Ez azt jelenti, hogy a memóriában létrejött objektumok tagfüggvényei és adattagjai elérhetők. A DOM fogalma ismerős lehet azok számára, akik a böngészőkben már írtak JavaScript, Java vagy Visual Basic Script programot, hiszen az ott használt Window, Document, Link, Form objektumok szintén egy DOM-fa alkotóelemei. Ez fa a dokumentum betöltődése során épül fel.
A SAX nem épít DOM-szerkezetet, ugyanis itt az elemzés során mindig csak a pillanatnyi elemet látjuk. A SAX emiatt az XML-elemek közvetlen elérését nem teszi lehetővé, csak sorosan olvassa fel őket. Ez a módszer viszont biztosít egy másik dokumentumelemzési és feldolgozási lehetőséget, ugyanis az XML-adatfolyam elemzése közben különféle események jönnek létre, amelyekre eseménykezelő függvényeket írhatunk.
A DOM és SAX API-k megadása a szabványos IDL nyelven van megfogalmazva, ezért azokat C++, Java, Pascal nyelveken kell megvalósítani. Az IBM, Oracle, Sun, Apache rendelkeznek Javában megvalósított változatokkal. Az Apache XML-elemzője az „Xerces”, amelyet mindenkinek a figyelmébe ajánlunk.
A példákat most az „Oracle XML Developer's Kit for Java” csomag használatával készítettük el, amely a
➲ http://www.otn.oracle.com címről tölthető le (itt C/C++ csomagok is találhatók).

Egy DOM-elemző működése

Egy DOM-elemző a VKONYV XML adatfolyamból hozzátartozó XML-fa felépítésének megfelelő szerkezetet épít fel a memóriában.
A Java oldaláról vizsgálva ez azt jelenti, hogy XMLElement, XMLAttr, XMLNode típusú osztályok objektumai vesznek részt a DOM-fa felépítésében. Ahelyett a felépített DOM-fa elemei XML-elemek, tulajdonságokat és adatokat megvalósító objektumok. Magát az XML-dokumentumot egy XMLDocument osztálybeli objektum képviseli.
A szemléletesség érdekében tekintsünk egy egyszerű Java programot (6. lista, lásd 20. CD Magazin/XML), ami elemzi a VKONYV XML-adatfolyamot, illetve feldolgozási tevékenység gyanánt a Java konzolon egymás alatt megjeleníti a vendégek neveit. A programban megjegyzések magyarázzák el a feldolgozó algoritmus működését.
A `java TVendegListaDOM vkonyv.xml` parancsra a program ezt jeleníti meg a konzolon:

```
Nyiri Imre
Koller J zsef
```

Egy SAX-elemző működése

A SAX-elemzők sorosan végigolvassák a bemenő XML-adatfolyamot, miközben különféle eseményeket hoznak létre.
Az elemző, feldolgozó alkalmazások írása során a fő feladat a megfelelő eseménykezelő eljárások megírása és bejegyzése.
A könnyebb érthetőség érdekében írjunk olyan Java SAX-elemzőt (*7. lista*), ami a konzolra kiírja, hogy hány vendég van leírva az XML-fájlban.
A `java TVendegListaSAX vkonyv.xml` parancsra a program a konzolon a következőt jeleníti:

```
A vendØgek szÆma: 2
```

A DOM és a SAX API az irodalomjegyzékben megadott összes webhelyről letölthető, tanulmányozásra ajánljuk őket.

További XML-módszerek röviden

Az XLL protokoll

A XLink és az XPointer feladata felváltani a HTML-dokumentumokból megismert egyszerű dokumentum-összekapcsoló és -hivatkozó (link) szolgáltatásokat. Az XLL-protokoll jóval összetettebb dolgokra képes, mint HTML-beli testvére:

- Közvetett hivatkozási lehetőség. Ez azt jelenti, hogy egy hivatkozás egy másik hivatkozásra hivatkozva érheti el a céldokumentumot. Ezzel a módszerrel valósítható meg a különféle telefonkönyvek könnyű létrehozása.
- Egy kapcsolat több dokumentumra is mutathat.
- Az XLink különböző források címzését teszi lehetővé (xml, html, pdf).

Az XLL lényeges része az XPointer, amely az XML-dokumentumok egy-egy részét címezi meg. Ennek során ismét felhasználja az XSL-nél már említett XPath nyelvet. Az XLL-protokoll azonban még fejlesztés alatt áll.

Az XSchema formai követelményeket leíró nyelv

A schema fogalom az adatbázis-kezelő rendszerek fogalomtárából érkezett az XML-módszerek világába, ahol az egyes adatelemek leírása sokkal szabatosabb, azaz az adattípus-fogalomnak megfelelő pontosságú. A cél a DTD nyelvnek egy olyan nyelvre történő cseréje, amelyben a nyelvtani szabályok sokkal többet árulnak el az egyes adatelemekről, mint az egyes adatokra kijelentett `(#PCDATA)` állítás. Más módszereknél már megszokhattuk az `integer`, `double`, `string`, `boolean` adattípusok használatát, ezt az XML-nél is megtartjuk.
A DTD felépítése nem XML-megfelelő, ellentétben az XSchema XML-alapú szerkezetével, ami további érv az XSchema használata mellett.
Mindenki saját XML-alapú schema nyelvet határozhat meg, és ha ellenőrzőt is készít hozzá, máris használatba lehet venni. Az XSchema nyelv szintén fejlesztési szakaszban van.

XML-RPC

Ez a módszer a távoli eljáráshívást az XML eszközeinek felhasználásával valósítja meg.

Nyíri Imre
(inyiri@mol.hu) jelenleg a MOL Rt.-nél dolgozik. Informatikai vállalkozásában az Internet, a Linux, valamint a Java programozás gyakorlati hasznosításával foglalkozik, örök szerelme a C++ maradt.

Kapcsolódó címek

➲ http://www.w3c.org
➲ http://www.xml.org
➲ http://www.xml.apache.org