XmlGessünk 13. rész - Az XML Schema II.

Az előző részben láthattuk, hogyan kell közvetlen egymásba ágyazással, referenciákkal és típusok definiálásával egyszerűbb sémákat szerkeszteni. Részletesen megnéztük hogyan lehet egyszerű típusokat létrehozni a már meglévő típusokból. Ebben a részben a komplex típusokat tekintjük át. Megnézzük, hogy összetettebb típusok létrehozására milyen fejlettebb nyelvi elemek állnak rendelkezésre.

Összetett típusok

A korábbi példákban a complexType sémaelemet mindig egy-egy elem deklarációján *belül* használtuk fel, mint például:

```
<xsd:element name="konyv">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="cim" type="xsd:string"/>
...
```

Ebben az esetben összetett típusunknak nincs neve, ezért nem is használható fel másutt, csak a konyv elemen belül. Az ilyen típusdeklarációt anonymousnak hívjuk. Ennek hátránya, hogy nem lehet újra és újra felhasználni, azaz csak azon a helyen hasznos, ahol definiáltuk.
A típusdeklarációkat a schema elem alá is kiemelhetjük. Ha nevet is adunk nekik, bármely elem definiálásánál felhasználhatjuk őket:

```
<!-- konyvsema3.xsd -->

<!-- Az egyszerű típusok deklarációja -->
<xsd:simpleType name="nevTipus">
 <xsd:restriction base="xsd:string">
  <xsd:maxLength value="32" />
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="szuletettTipus">
 <xsd:restriction base="xsd:date"/>
</xsd:simpleType>
<xsd:simpleType name="jellemzesTipus">
 <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<!-- Az összetett típusok definíciója -->
<xsd:complexType name="szereploTipus">
 <xsd:sequence>
  <xsd:element name="nev" type="nevTipus"/>
  <xsd:element name="baratja" type="nevTipus"/>
  <xsd:element name="szuletetett"
   type="szuletettTipus"/>
  <xsd:element name="jellemzes"
   type="jellemzesTipus"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="konyvTipus">
 <xsd:sequence>
  <xsd:element name="cim" type="nevTipus"/>
  <xsd:element name="szerzo" type="nevTipus"/>
  <xsd:element name="szereplo"
   type="szereploTipus"/>
 </xsd:sequence>
 <xsd:attribute name="isbn" type="isbnTipus"
  use="required"/>
</xsd:complexType>

<xsd:element name="book" type="konyvTipus"/>
```

A példában két összetett típust (a szereploTipust és a konyvTipust) deklaráltuk. Látható, hogy a típusokba foglalt elemek hivatkozhatnak további típusokra, így alakul ki a tervezett elemhierarchia.
A típusok a kiemeléssel és megnevezéssel újrahasznosíthatóvá váltak. Nyilvánvaló, hogy ha egy megnevezett típust bármely elem típusaként felhasználhatunk, ezzel munkát spórolunk meg a séma megírása és karbantartása során. Hamarosan kiderül, hogy ennél jóval többet is tudnak a kiemelt és elnevezett elemek. Ehhez azonban még át kell tekintenünk néhány fogalmat.

Tartalomtípusok

Mi lehet egy elem tartalma?

- Lehet üres, azaz nincs se gyermekeleme, se közvetlen tartalma.
- Csak gyermekelemei vannak.
- Csak szöveges tartalma van.
- Gyermekelemek és szöveges tartalom vegyesen található benne.

Emellett még mind a négy esetben attribútumokat is tartalmazhat az adott elem. Hogyan lehet e különböző tartalmakat csoportosítani és formálisan leírni?
Az eddig példákban már láttuk, hogyan kell olyan összetett típusokat létrehozni, amelyek csak gyermekelemeket és/vagy attribútumokat tartalmaztak, ilyen volt például a konyvTipus.
Tisztán tartalmat hordozott például a cim elem. Hogyan lehet olyan típust definiálni, amely közvetlen tartalmat hordoz, *és* vannak attribútumai is? Ehhez be kell vetnünk egy új sémaelemet, a simpleContentet:

```
<!-- konyvsema4.xsd részlet-->
<xsd:complexType name="osszetettNevTipus">
  <xsd:simpleContent>
    <xsd:extension base="nevTipus">
      <xsd:attribute name="nem" type="xsd:string">
      </xsd:attribute>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Azért egyszerű tartalom (simpleContent), mert a komplex típusunknak nincsenek gyermekelemei, csak közvetlen szöveges tartalma és attribútumai.
Az xsd:extension a simpleContenten belül azt jelzi, hogy a base attribútumban megadott ***egyszerű típust*** (simple type) vagy egyszerű tartalmat hordozó ***összetett típust leszármaztatás*** segítségével egészítjük ki. Ez azt jelenti, hogy az így előállt egyszerű tartalmú összetett típusunk minden jellemzőjében a nevTipussal lesz azonos, csak kiegészítjük egy „nem” nevű attribútummal. Ha a nevTipusnak lennének attribútumai, akkor azokat automatikusan örökölné az osszetettNevTipus összetett típusunk.
Próbáljuk ki mire jutottunk! Ha megnézzük az előző (konyvsema3.xsd) forrást, akkor láthatjuk, hogy a szereplő neve egy nevTipus típussal van leírva. Ez nem enged meg semmilyen attribútumot a névben, így a következő részlet nem érvényes az eredeti séma alapján:

```
<nev nem="fiú">Snoopy</nev>
```

Ha azonban a sémában a

```
<xsd:element name="nev" type="nevTipus"/>
```

sort kicseréljük az alábbira:

```
<xsd:element name="nev"
type="osszetettNevTipus" />
```

akkor mindjárt elfogadja a „nem” attribútumot is. Ebből látszik, hogy működik ugyan az új attribútum deklarációja, de még nem látszik a leszármaztatás hatása. Ezt is könnyen ellenőrizhetjük, hisz a nevTipusunk maximum 32 karakter hosszú neveket engedett meg, így ha működik az öröklés, akkor ez igaz lesz az osszetettNevTipusra is.

```
<nev
 nem="fiú">Snoopyiiiiiiiiiiiiiiiiiiiiiiiiii</nev>

Validation Error: The 'nev' element has an invalid value according to its data type.
```

Remek, megy a leszármaztatás, a 32 karakternél hosszabb szövegeket az osszetettNevTipus sem fogadja el.
Most tehát egy egyszerű típusból származtattuk le az egyszerű tartalmat hordozó összetett típusunkat. Most, ha kell egy olyan típus, ami mindenben azonos az osszetettNevTipussal, csak még kell hozzá egy plusz attribútum, akkor egyszerűen le kell származtatnunk, és ki kell egészítenünk az új típust a plusz attribútummal. Például lehet mindenkinek megszólítása:

```
<nev2 nem="fiú" megszolitas="Mr.">Snoopy</nev2>
```

Egy ilyen elemet a következő komplex típus ír le:

```
<!-- konyvsema5.xsd részlet -->
<xsd:complexType name="mrNevTipus">
 <xsd:simpleContent>
  <xsd:extension base="osszetettNevTipus">
   <xsd:attribute name="megszolitas"
    type="xsd:string">
   </xsd:attribute>
```

```
  </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>
```

Ebben a példában egy egyszerű tartalmat hordozó összetett típusból örököltünk.
Azaz (egyszerű vagy összetett típusból örököltetve) most már attribútummal, és anélkül is létre tudunk hozni egyszerű tartalommal rendelkező elemeket.
Ha egy összetett típusban vegyesen, *gyermekelemeket és közvetlen tartalmat* is szeretnénk használni, akkor ezt a mixed attribútum segítségével jelezhetjük. Például ahhoz, hogy a szereploTipusnak lehessen tartalma is a gyermekelemeken felül:

```
<szereplo>Itt ugyan semmi értelme
  <nev nem="fiú">Snoopy</nev>
  szövegnek, de miért ne?
...
```

Az ezt megengedő komplex típus deklarációja a mixed attribútummal jelzi szándékunkat:

```
<!-- konyvsema6.xsd részlet -->
<xsd:complexType name="szereploTipus"
 mixed="true">
```

A gyermekelemek között megbújó szöveges tartalom még abból az időből lehet ismerős, amikor az xml elődjét, az SGML-t, szövegek kiegészítésére használták (pl. a HTML-ben is ezt tesszük). A mai világban, amikor az xml-t főleg információk átvitelére használjuk, a mixed modell használata nem nagyon elterjedt. Az információt attribútumok és egyszerű tartalommal rendelkező gyermekelemek hordozzák.
Hogyan lehet üres elemet definiálni? Egyszerűen nem írunk gyermekelem-definíciókat a complexType belsejébe, és nem engedjük meg a mixed modellt sem.
Már csak egy dolog maradt hátra. Ha van simpleContent, akkor várhatóan lesz complexContent is. Ezzel hozhatunk létre leszármaztatott összetett tartalmat (gyermekelemeket, attribútumokat, esetleg közvetlen tartalmat) hordozó típusokat.
Például minden szereplőről le szeretnénk írni annak korát is (illékony adat, de miért ne?):

```
...
  <kor>13</kor>
 </szereplo>
```

Mivel már van egy összetett típusunk a szereplők leírására, kár lenne veszni hagyni, inkább származtassunk abból egy újabb típust, és egészítsük ki egy „kor” elemmel:

```
<!-- konyvsema7.xsd részlet -->
<xsd:complexType name="bovitettSzereploTipus">
 <xsd:complexContent>
   <xsd:extension base="szereploTipus">
    <xsd:sequence>
     <xsd:element name="kor" type="xsd:int" />
    </xsd:sequence>
   </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Ha belegondolunk, ez a klasszikus, OOP-s értelemben vett öröklés. A sématípusok nagyon hasonlóak a programozott típusokhoz, például C# (C++) nyelven így nézne ki (egyszerűsítve) a szereploTipus és a bovitettSzereploTipus:

```
class szereploTipus {
  string nev;
  string nev2;
  string baratja;
  datetime szuletett;
  string jellemzes;
}
class bovitettSzereploTipus: szereploTipus {
  int kor;
}
```

A séma típusszármaztatási lehetősége jól kihasználható lenne az xml séma és a programozott osztályok közötti átjárást biztosító sémafordítókban (schema compilers, .NET-ben a wsdl.exe), azonban ezek az eszközök jelen pillanatban nem nagyon élnek ezzel a lehetőséggel. Mi magunk viszont egyszerűen használhatjuk őket. Nem túl bonyolult, a névterek bevezetése után azonban eléggé kacifántos tud lenni. Egyesek úgy érvelnek, hogy ne használjuk a séma öröklési lehetőségeit. Viszont a típusok újrafelhasználásra szükségünk van, és erre van is lehetőségünk: a csoportosítás.

Csoportosítások

Gyakori, hogy több elemet vagy attribútumot több összetett típusban is fel szeretnénk használni. Összeszedhetjük őket egy-egy alaptípusba, és leszármaztatással bárhol felhasználhatjuk őket. Másfelől össze tudjuk terelni őket egy-egy csoportba, és típusainkból a csoportokra hivatkozhatunk!

```
<!-- konyvsema8.xsd részlet -->
<!-- elemcsoport létrehozása -->

<xsd:group name="konyvFoElemek">
 <xsd:sequence>
  <xsd:element name="cim" type="nevTipus"/>
  <xsd:element name="szerzo" type="nevTipus"/>
 </xsd:sequence>
</xsd:group>

<!-- attribútumcsoport létrehozása -->

<xsd:attributeGroup name="konyvAttributumok">
 <xsd:attribute name="isbn" type="isbnTipus"
  use="required"/>
 <xsd:attribute name="rendelheto"
  type="xsd:string"/>
</xsd:attributeGroup>
```

A group sémaelemen belül hasonló tartalmat láthatunk, mint amit az összetett típusok deklarálásánál elemeztünk. Az attribútumok összefogására külön sémaelem van, az attributeGroup. Mivel az attribútumok sorrendje (definíció szerint) érdektelen, ezért ebben az esetben nem kell a sequence vagy bármely más sorrendet és/vagy számosságot befolyásoló compositor.
Az elkészült csoportokra (hivatalos nevük Model Group) hasonlóan hivatkozhatunk, mint az egyedi elemekre és attribútumokra: a ref attribútumon keresztül. Lássuk a csoportokat alkalmazó konyvTipust:

```
<xsd:complexType name="konyvTipus">
  <xsd:sequence>
    <xsd:group ref="konyvFoElemek" />
    <xsd:element name="szereplo" ... />
  </xsd:sequence>
  <xsd:attributeGroup ref="konyvAttributumok" />
</xsd:complexType>
```

Pont olyan, mint az előző számban látott globális elem- és attribútumhivatkozás ref-fel, csak itt csoportokra hivatkozunk.
A Compositorok közül eddig csak a sequence compositort láttuk, ami azt írja elő, hogy a benne felsorolt particle-öknek a megadott sorrendben kell szerepelni a példánydokumentumokban. A particle elemek, elemcsoportok és a bármilyen elemet reprezentáló any elemek összessége. A célnévtérbe elemeket generálni képes séma komponenseket összefoglalóan particle-öknek hívjuk (szép magyar szó!).
További két compositor is van, a *choice* és az *all*. A choice, ahogyan a neve is utal rá, a benne definiált elemek vagy elemcsoportok között biztosít választási lehetőséget. Például hozzunk létre egy olyan csoportot, amely egy ember nevét ábrázoló elemeket tartalmaz. A szabály legyen az, hogy vagy meg kell adni az ember teljes nevét egyben, vagy szétbontva családi és kereszt-, valamint egy opcionális második keresztnévre:

```
<xsd:group name="nevek">
  <xsd:choice>
    <xsd:element name="nev" type="xsd:string" />
      <xsd:sequence>
      <xsd:element name="csaladinev"
       type="xsd:string" />
      <xsd:element name="keresztnev"
       type="xsd:string" />
      <xsd:element name="masodikKeresztnev"
       type="xsd:string" minOccurs="0" />
    </xsd:sequence>
```

```
  </xsd:choice>
</xsd:group>
```

Látható, hogy a choice-on belül nemcsak egyszerűen elemeket vagy csoporthivatkozásokat, hanem minden további nélkül további compositorokat is használhatunk.
Az all compositor azt jelenti, hogy a benne felsorolt elemek (és semmi más, még csoport sem) bármilyen sorrendben szerepelhetnek a példánydokumentumban. Például a konyvTipus esetén nem biztos, hogy cim, szerzo és szereplo elemek sorrendje fontos. Eddig ott sequence compositor volt, cseréljük azt ki all-ra. Ebből problémáink származnak, mert szereplo elemekből több mint egy is megengedett, viszont az egyértelműség miatt az all compositor ezt nem engedi meg: minden elem maxOccurs értéke (kardinalitása, számossága) legfeljebb 1 lehet. Emiatt azt gondolnánk, hogy csak a cim és a szerzo elemek lesznek az all-ban, a szereplo elemdeklarációt pedig belerakjuk egy sequence-be. Azonban egy összetett típuson belül nem lehet csak egy compositor, azaz vagy all, vagy sequence. Mit lehet tenni? Sajnos ezt a feladvány nem lehet megoldani az all compositorral, azaz le kell nyelnünk, hogy meg lesz kötve az adatok sorrendje, marad a sequence.
Ennek ellenére számos helyen jól bevethető az all compositor. Például gyakori, hogy adatbázistáblák tartalmát generáltatjuk le xml-ként. A táblák sorait egy-egy elem reprezentálja:

```
<cuccok>
  <tabla1><oszlop1/><oszlop2/></tabla1>
  <tabla1><oszlop2/><oszlop1/></tabla1>
</cuccok>
```

Ilyenkor a feldolgozás szempontjából általában teljesen érdektelen a táblák oszlopait ábrázoló elemek sorrendje, ezért például a következő séma írhatja le az adatokat:

```
<xsd:element name="cuccok">
  <xsd:complexType>
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="tabla1"
       type="tabla1Tipus" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="tabla1Tipus">
  <xsd:all>
    <xsd:element name="oszlop1" />
    <xsd:element name="oszlop2" />
  </xsd:all>
</xsd:complexType>
```

Megszorítások

Gyakran logikai kapcsolat van az xml dokumentum által szállított információk között. Ezen kapcsolatokat, és az adatokban rejlő szabályszerűségeket általában constraintek, megszorítások írják elő a különböző adattároló és szállító rendszerekben. Közismert például, hogy az adatbázistáblákban (relációkban) a sorok, azaz a modellezett entitiáspéldányok egyediségét az elsődleges kulcs hivatott ellenőrizni, „megszorítani". Ha a táblák között logikai kapcsolat van, akkor az idegen kulcsok lehetséges értékeinek szerepelnie kell a kapcsolódó tábla elsődleges kulcshalmazában (tartomány épségi szabály).
Egy xml dokumentumban gyakran többféle információtartalmat találunk. Az adatok között a sémadokumentumban definiálhatunk kötöttségeket. A sémakötöttségek gyakorlatilag megegyeznek az adatbázisokban megszokott megszorításokkal, hisz a gyakorlatban a legtöbb xml dokumentum forrása egy-egy adatbázis.
Az egyik leggyakoribb feladat az egyediség biztosítása valamely elem vagy attribútum értékkészletén. Erre való a unique sémaelem:

```
<!-- konyvsema10.xsd részlet -->
<xsd:element name="konyv" type="konyvTipus">
 <xsd:unique name="EgyediSzereploNev">
  <xsd:selector xpath="szereplo" />
  <xsd:field xpath="nev" />
 </xsd:unique>
</xsd:element>
```

A selector elem xpath attribútumában megadott XPath kifejezés jelöli ki azt az elemet, amelyen értelmezni kell a field elem xpath attribútumában megadott érték egyediségét. Adatbázis-hasonlattal élve a selector választja ki a táblát, a field az egyedi értékeket tartalmazó oszlopot. A példánkban nem lehet két azonos *nev*ű szereplő *ugyanabban* a *könyv*ben. De különböző könyvekben minden további nélkül lehetnek azonos nevű szereplőink.

Hasonló módon szeretnénk biztosítani a könyvek isbn számainak egyediségét. Ehhez egy kicsit átalakítjuk a korábbi példánkat, hogy több könyvet is tudjon tárolni, és megadjuk, hogy az isbn attribútum egyedi legyen:

```
<xsd:element name="konyvek">
  <xsd:complexType>
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="konyv" type="konyvTipus">
      <!-- Az előző szereplőnév megszorítás -->
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:unique name="EgyediISBN">
    <xsd:selector xpath="konyv" />
    <xsd:field xpath="@isbn" />
  </xsd:unique>
</xsd:element>
```

Elég valószínű, hogy az isbn attribútumot a könyvek egyedi azonosítására használjuk, erre a célra azonban nem a unique elem az igazi, mert az megengedi azt is, hogy az egyedi érték (field) ne szerepeljen a céldokumentumban. Adatbázis-hasonlattal élve: megengedi a null (xml-ben nil) értéket is.
Valódi elsődleges kulcs jellegű adatok megkötésére inkább használjuk a key sémaelemet. Ez egy olyan unique megkötés, ami nem engedi meg a null értékeket. A használata teljesen azonos a unique-éval.
Egymástól függő adatok esetén hasznos lehet idegen kulcsok definiálása. Erre a keyref sémaelem használható, mellyel hivatkozást (referenciát) hozhatunk létre egy key-jel vagy unique-kal azonosított kulcsra. Például kössük meg, hogy egy szereplő barátja csakis a könyvben definiált szereplők közül kerülhet ki.

```
<!-- konyvsema11.xsd részlet -->
<xsd:keyref refer="EgyediSzereploNev"
 name="FKSzereplo">
  <xsd:selector xpath="szereplo" />
  <xsd:field xpath="baratja" />
</xsd:keyref>
```

Példánkban a keyrefet is a konyv elem belsejébe kell helyezni, azonban legtöbbször a hierarchia különböző pontjai között szoktunk hivatkozásokat definiálni.
Láthatjuk, hogy ezeknek a funkcióknak semmi közük a korábbi részekben tárgyalt struktúradefiniáló elemekhez. Azok olyan típusok leírására valók, amelyeket sokszor programozott típusok (class-ok) és xml dokumentumok közötti átjárásra használunk. Nem nehéz kitalálni, hogy a fő motiválóerő a Webszolgáltatás technológia háttértámogatása volt. Ezzel szemben a megszorítások, kulcsok egyértelműen adatok értékeinek megregulázására alkalmasak, ez pedig adatok átvitele, üzleti adatok cseréje közben lehet hasznos. De a kettőt lehet ötvözni is, mert például egy .NET-es Webszolgáltatásban egy paraméterként átadott típusos DataSet táblái között lehetnek kapcsolatok, és ezt a DataSet key-keyref elemekkel modellezi le. A táblák adatai xml-ként mennek át, és az adatok épségét a kapcsolatok is elősegítik.

Zárszó

Szinte megismertük a séma összes elemét, így a következő -záró- részben áttekintjük a gyakorlati felhasználás részleteit COM és .NET világban is.

A cikkben szereplő URL-ek:
[1]: A cikkben szereplő példák http://technet.netacademia.net/download/xml

Soczó Zsolt
Zsolt.Soczo@netacademia.net