XMLgessünk – 12. rész: Az XML Schema

Számtalanszor hivatkoztunk már XML sorozatunkban az XML sémára, így hát itt az ideje, hogy közelebbről megismerjük. Ebben a részben áttekintjük az alapismereteket, és megnézzük az egyszerű típusok képzésének fortélyait. A következő részben áttekintjük az összetett típusok kialakításának részleteit, a harmadik záró részben pedig tervezési mintákat nézünk át újrafelhasználható és jól olvasható sémák írásához, valamint megnézzük milyen eszközökkel lehet tényelegesen kihasználni a sémák adta előnyöket.

Kezdjük az alapokkal. Mi az a séma?

Az xml sémaleíró nyelvek célja, hogy formális leírást adjanak xml dokumentumosztályok definiálására. Másképpen megfogalmazva nyelvtani leírások, amelynek segítségével xml struktúrák által ábrázolni kívánt üzleti szabályokat definiálhatjuk.
Egy-egy xml dokumentumminta nem elég az egyértelmű leíráshoz. Nézzük például az alábbi xml töredéket:

```
<helyzet>
  <szelesseg>47.974262</szelesseg>
  <hosszusag>24.290234</hosszusag>
  <bizonytalansag mertekegyseg="meter">10
  </bizonytalansag>
</helyzet>
```

Ránézésre sejthetjük, hogy egy hely földrajzi koordinátáit írja le ez a töredék. De milyen pontosságú a két koordinátát ábrázoló szám? Milyen határok között mozoghatnak az értékek? Kötelező-e megadni a bizonytalanság értékét? Milyen mértékegységeket használhatunk? Ezek mind nagyon fontosak az adatok értelmezéséhez, és ezeket kellene valahogyan formálisan leírni. Erre való a séma.

Mit ír le egy séma?

- Definiálja milyen elemek és attribútumok lehetnek egy dokumentumban
- Definiálja az elemek egymásba ágyazását
- Definiálja a gyermekelemek sorrendjét
- Definiálja a gyermekelemek számát
- Definiálja, hogy egy elem üres, szöveget, vagy további gyermekelemeket tartalmaz
- Definiálja az elemek és attribútumok adattípusát
- Alapértelmezett értéket definiál elemekhez és attribútumokhoz

Egyszerűbben megfogalmazva a dokumentum struktúráját, valamint a benne található adok típusát és értelmezési tartományait definiálja.

Mit jelent az a szó, hogy validálás?

A sémák egyik fő célja, hogy miután formális leírást adtak egy dokumentumosztályról, ezek után programozottan meg lehessen mondani, hogy egy konkrét xml dokumentumpéldány megfelel-e a sémának? Ezt a folyamatot nevezzük validálásnak, és az olyan xml doksit, ami egy sémának megfelelő validnak.

Mire jó a séma?

Számos helyen használjuk a sémákat, Don Box szerint „a séma a villanyáram és a melegvíz az xml technológiákban”, azaz e nélkül nem lehet kulturáltan, komolyan xml-ben fejleszteni.
Az első felhasználás a struktúradefiniálás. Hogyan magyarázom el az üzleti partnereimnek, hogy milyen formátumú xml dokumentumokat várok például a megrendelések leírására? Ahelyett, hogy körbemagyaráznám a formátumot, egyszerűen adok neki egy sémát, és azt mondom, hogy így néz ki a megrendelőlap. Ebből a precíz leírásból ő már képes a sémának megfelelő xml dokumentumokat generálni. Egyes eszközök (pl. SQL Server 2000) a séma minimális kiegészítésével azonnal képesek xml kimenetet generálni a táblák adataiból.
Miután bejött hozzám egy megrendeléstétel, le kell ellenőriznem, hogy az tényleg a sémának megfelelő formátumú, azaz validálásra használom a sémát. Validálás nélkül letárolni a kapott információkat (minimum erkölcsi) öngyilkosság.
A SOAP által javasolt kommunikációs modellben a kommunikáló programok között xml formátumban utaznak az információk, például a távoli eljáráshívások paraméterei. Mivel mindkét oldal objektumokban gondolkodik, szükség van egy olyan formális leírásra, amely segítségével a programozott típusokat (osztályok, interfészek, primitív típusok) *automatikusan* át tudjuk fordítani xml struktúrákká és vica versa. A programozók objektumokban és tagváltozókban gondolkodnak, az xml pedig elemekben és attribútumokban. A két világ áthidalására a SOAP szabvány is az XML sémára bízza az összes olyan típus leírását, amelyet eleve ismer a séma (amit nem, azt kiegészítették a SOAP specifikációban).

Milyen sémákat ismerünk?

A legrégebbi sémaleíró nyelv a Document Type Definition, a DTD. Ő az SGML világból jött át, ám az XML világban nem igazán hatékony. A fő problémák vele:

- Nem xml formátumú
- Nem ismeri a névtereket
- Nem bővíthető
- Csak egy teljes dokumentumra alkalmazható
- Fő probléma: nem ismeri a programozott világban megszokott adattípusokat

Világos volt, hogy le kell cserélni egy jobban „xmlesített” sémaleírásra. Ez azonban nem volt egyszerű meccs. Többféle javaslat is napvilágot látott, név szerint a legfontosabbak: RELAX, TREX, Schematron, SOX, DSD, XDR és XML Schema.
A sémák egy része szabályalapú, más részük nyelvtani elemekből építkezik. A szabályalapúak általában fejlettebbek az xml dokumentumpéldányok szabályellenőrzésében (validálás), a nyelvtanalapúak pedig jobbak struktúra definiálásra. Az előbbire az egyik legfejlettebb nyelv a Schematron, az utóbbira az XML Schema. A szabályalapúakkal le lehet írni olyan

környezetfüggő kötöttségeket is, amelyeket nem lehet nyelvtannal megfogalmazni. Például ha egy elemnek van x nevű attribútuma, akkor kell lenni y-nak is. Vagy ha egy elem szülője z, akkor kell legyen j nevű attribútuma.
Ezzel szemben a nyelvtanalapú sémákból remek programozott objektumokat lehet generálni: osztályokat, interfészeket. Ez a Webszolgáltatások egyre erősödő piacán nagyon fontos feladat, és talán ez is oka annak, hogy a World Wide Web konzorciumnál az XML Schema Definition (XSD) győzött, mint xml séma ajánlás. Ez nem jelenti azt, hogy a többi kihal, valószínűleg hosszabb távon a szabályalapúak közül is ki fog fejlődni egy jól használható leírás.

Az XML Schema

2001. május óta az XML Schema (XSD) az aktuális, a w3c által javasolt sémaleírás. A formátum gyökerei a Microsoft XML Schema Reduced (XDR) sémához kötődnek, melyet (micsoda meglepetés) a Microsoft fejlesztett ki. Azért kellett az MS-nek az XDR-el bajlódni, mert a DTD kevés volt, végleges séma még nem volt a látóhatáron, így a korai xml támogatottságú termékekhez kellett egy fejlettebb sémaleíró nyelv. Az SQL Server 2000, az Internet Explorer 5.x, a Biztalk 2000, az Exchange 2000 mind-mind XDR-t használnak.
Tavaly május óta van szabványos sémánk, így az SQLXML2-től már használhatjuk az XSD-t az SQL Server programozásához, az MSXML4 már XSD kompatibilis, valamint a .NET framework osztályait már eleve úgy tervezték, hogy ismerjék az XSD-t is. Azaz mai fejlesztésekhez mindenképpen ez a megfelelő sémanyelv.
Nézzünk most meg egy xml dokumentum példányt, majd írjunk hozzá sémát!

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<konyv isbn="0836217462">
 <cim>Kutyavilág</cim>
 <szerzo>Charles M. Schulz</szerzo>
 <szereplo>
  <nev>Snoopy</nev>
  <baratja>Peppermint Patty</baratja>
  <szuletett>1950-10-04</szuletett>
  <jellemzes>extrovertált beagle</jellemzes>
 </szereplo>
 <szereplo>
  <nev>Peppermint Patty</nev>
  <szuletett>1966-08-22</szuletett>
  <jellemzes>kövér, vakmerő leányzó</jellemzes>
 </szereplo>
</konyv>
```

A séma megírásához egyszerűen végigmegyünk az összes elemen, és definiáljuk őket. Előtte azonban az xsd:schema elemmel kell indítanunk:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

A schema elem a séma dokumentum gyökéreleme, amelyben az xsd prefixhez kötjük a séma aktuális névterét. Ennek az elemnek lesz még számos attribútuma, amely a teljes séma értelmezését befolyásolja, de erről egy kicsit később.
A konyv elemünkhöz definiálnunk kell egy elemet, amelynek neve konyv. Ennek az elemnek vannak gyermekelemei és attribútumai, ezért őt az xml sémában *komplex típus*nak tekintjük (complex type). A gyermekelemeket a *sequence* elemen belül definiálhatjuk:

```
<xsd:element name="konyv">
 <xsd:complexType>
  <xsd:sequence>
```

A sequence elem egy *compositor*, és azt írja elő, hogy a benne felsorolt gyermekelemeknek pontosan a megadott sorrendben kell szerepelni az xml dokumentumpéldányokban. További két compositor is van, a *choice* és az *all*. Ezekre még visszatérünk.
Jöhet a cím és a szerző, ezeknek nincs gyermekeleme vagy attribútuma, így ők egyszerű típusok, simple type-ok.

```
<xsd:element name="cim" type="xsd:string"/>
<xsd:element name="szerzo" type="xsd:string"/>
```

A type attribútumban írjuk elő az egyszerű típusok adattípusát. Az xsd prefix azt jelzi, hogy a séma szabványban definiált string típust szeretnénk használni.
A szereplo elem egy kicsit huncutabb, mert abból minimum egy, maximum akárhány darab is lehet. Mivel vannak gyermekelemei egyértelmű, hogy komplex típus lesz, a számosságot pedig a elemdefinícióban tudjuk jelezni:

```
<xsd:element name="szereplo"
 minOccurs="1" maxOccurs="unbounded">
 <xsd:complexType>
  <xsd:sequence>
```

A minOccurs írja elő legalább hány példány kell az elemből, a maxOccurs pedig hogy legfeljebb mennyi. Az unbounded a végtelent jelöli. Mindkettő alapértelmezett értéke 1, azaz ha nem írjuk ki őket pontosan egy elemet kell a jelzett helyen találnunk.
Ezek után jöhetnek a gyermekelemek:

```
<xsd:element name="nev" type="xsd:string"/>
<xsd:element name="baratja" type="xsd:string"
 minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="szuletett" type="xsd:date"/>
<xsd:element name="jellemzes"
 type="xsd:string"/>
```

Látható, hogy egy szereplőnek több (vagy akár semennyi) barátja is lehet, amit nem lehetett kiolvasni az xml mintánkból, ezt csak az tudhatja, aki ismeri az adatok logikáját, és ezért kell a séma egy mintapélda helyett.
A szuletett elem típusa xsd:date, azaz ebben az elemben csak érvényes dátumokat reprezentáló sztringeket lehet megadni.
A szereplő leírás kész, zárjuk le a megkezdett elemeket:

```
  </xsd:sequence>
 </xsd:complexType>
</xsd:element> <!-- szereplo -->
```

A könyvben található gyermekelemek leírása is kész, zárhatjuk a sort:

```
</xsd:sequence>
```

Most jön az isbn attribútum. Az attribútumdeklarációknak kötelezően a gyermekelemek után kell szerepelni.

```
<xsd:attribute name="isbn" type="xsd:string"/>
```

Majd zárjuk a konyv elemet leíró komplex típust, magát az elemdeklarációt és a teljes sémát is:

```
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Készen is vagyunk! Ez volt az „orosz Matrjoska baba” tervezés, amelyben a séma írása közben pontosan követtük a leírandó dokumentum szerkezetét.
Az áttekintés kedvéért íme a teljes séma:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<-- konyvsema1.xsd -->
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="konyv">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="cim" type="xsd:string"/>
    <xsd:element name="szerzo"
     type="xsd:string"/>
    <xsd:element name="szereplo"
     minOccurs="1" maxOccurs="unbounded">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="nev"
        type="xsd:string"/>
       <xsd:element name="baratja"
        type="xsd:string" minOccurs="0"
        maxOccurs="unbounded"/>
       <xsd:element name="szuletett"
        type="xsd:date"/>
       <xsd:element name="jellemzes"
        type="xsd:string"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element> <!-- szereplo -->
   </xsd:sequence>
   <xsd:attribute name="isbn"
    type="xsd:string"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Második nekifutás - modularizálunk

Az előző „nekirugaszkodunk és egyszuszra megírjuk” módszer bonyolultabb struktúrák esetén gyorsan áttekinthetetlenné válhat, gyorsan túlnyúlik a jobb oldal a képernyőn . Ennél laposabb sémát hozhatunk létre, ha kihasználjuk, hogy az előre definiált elemekre hivatkozhatunk a struktúra kívánatos pontján.
Ebben a tervezési módszerben globálisan, a schema elem alatt felsorolunk minden elemet, attribútumot, és az összetett típusokban csak *hivatkozunk* (reference) a már deklarált tagokra:

```
<-- konyvsema2.xsd -->
<xsd:schema ...>
 <!-- egyszerű típusok definíciója -->
 <xsd:element name="cim" type="xsd:string"/>
 <xsd:element name="szerzo" type="xsd:string"/>
 <xsd:element name="nev" type="xsd:string"/>
 <xsd:element name="baratja" type="xsd:string"/>
 <xsd:element name="szuletett" type="xsd:date"/>
 <xsd:element name="jellemzes"
  type="xsd:string"/>
 <xsd:attribute name="isbn" type="xsd:string"/>
 <!-- komplex típusok -->
 <xsd:element name="szereplo">
  <xsd:complexType>
   <xsd:sequence>
   <!-- az egyszerű típusokra a ref attribútumon
        keresztül hivatkozunk  -->
    <xsd:element ref="nev"/>
    <!-- a számosságot itt jelöljük ki, nem az
         elem definíciójánál -->
    <xsd:element ref="baratja"
     minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="szuletett"/>
    <xsd:element ref="jellemzes"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="konyv">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="cim"/>
    <xsd:element ref="szerzo"/>
    <xsd:element ref="szereplo"
     minOccurs="1" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute ref="isbn"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Ez az objektumorientált programozási elvekre emlékeztet, amelyben kis építőkockákból (alaptípusok - egyszerű típusok) építünk fel nagyobb blokkokat (osztályok, struktúrák - összetett típusok).
Felfoghatjuk úgy is, hogy felül létrehozzuk az elemek, attribútumok absztrakt leírását, a komplex típusokban pedig példányosítjuk őket, konkrét elfordulásokat hozunk létre belőlük.

Típusos megközelítés

A harmadik sémaépítési módszerünkben összetett és egyszerű adattípusokat definiálunk, és ezek felhasználásával építjük fel az elemeket és attribútumokat.
A módszer hasonló lesz az előzőhöz, csak most nem kész elemeket és attribútumokat hozunk létre, hanem a típusdefiníciókat megnevezzük, és ezekre hivatkozunk az elemekben és attribútumokban.
Eddig csak komplex típusokat használtunk, azonban az XSD lehetőséget biztosít egyszerű típusok definiálására is. Az egyszerű típusok őseit a sémában definiált alaptípusok adják. Összesen 44-en vannak, az ábrán az anySimpleType leszármazottjai (ld. következő kép). A beépített típusokból származtatjuk le saját egyszerű típusainkat listagenerálással, megszorítással vagy unióképzéssel.
A listagenerálással képzett típus az alaptípus példányok whitespace-ekkel elválasztott listája.

```
<xsd:simpleType name="nevnapok">
  <xsd:list itemType="xs:date">
</xsd:simpleType>
```

Egyféle ennek megfelelő tartalom (a szóköz és az újsor is whitespace):

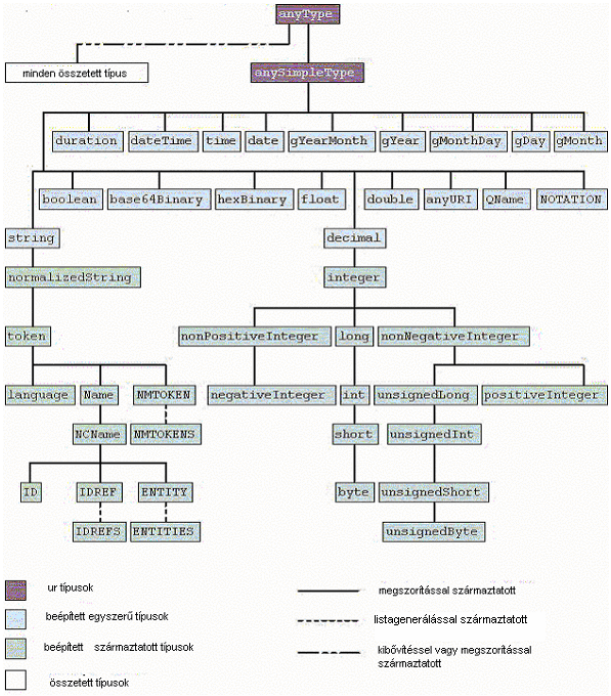
```
2002-03-22 2002-04-10
```

```
2001-01-05
```

Jóval gyakoribb a megszorítással képzett típus, amelyben az alaptípus érvényes értékeit korlátozzuk le.

```
<xsd:simpleType name="nevTipus">
 <xsd:restriction base="xsd:string">
  <xsd:maxLength value="32"/>
 </xsd:restriction>
</xsd:simpleType>
```

Ebben a példában a netTipus névvel ellátott egyszerű típus a string séma alaptípusból származik megszorítással (restriction), ahol a szöveg hosszát legfeljebb 32 karakterre korlátoztuk le.



❑ Az XML Schema típusdefiníció hierarchiája

A megszorításokat előre definiált facetekkel írhatjuk le, ilyen volt a maxLength is. Az alábbi táblázatban összefoglaltam a többi facetet is.

Facet	Jelentés
length	Az adott adattípuson értelmezhető alapegységek száma. Például egy string esetén a karakterek száma.
minLength	Minimális hossz.
maxLength	Maximális hossz.
pattern	Reguláris kifejezéssel megfogalmazott kötöttség, amellyel az adott típus szöveges reprezentációját tudjuk megfogni.
enumeration	Az alaptípus értékeit korlátozza le egy véges halmazra.
maxInclusive	Felső határ, beleértve a megadott értéket is. .
maxExclusive	Felső határ, a megadott értéket már kizárva.
minInclusive	Alsó határ, beleértve a megadott értéket is.
minExclusive	Alsó határ, a megadott értéket már kizárva.

totalDigits	Egy decimális típusból leszármazott típus számjegyeinek maximális száma (egész + törtrész).
fractionDigits	Egy decimális típusból leszármazott típus *törtrészében* a számjegyek maximális száma.

Az könyvek isbn számai pontosan tíz decimális számjegyből állnak, ezt az alábbi módon lehet megfogalmazni a pattern facet és benne reguláris kifejezések használatával:

```
<xsd:simpleType name="isbnTipus">
 <xsd:restriction base="xsd:string">
  <xsd:pattern value="[0-9]{10}"/>
 </xsd:restriction>
</xsd:simpleType>
```

Az enumeration típusra egy példa:

```
<xsd:simpleType name="szin">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="barna"/>
  <xsd:enumeration value="fekete"/>
  <xsd:enumeration value="szoke"/>
 </xsd:restriction>
</xsd:simpleType>
```

Az unióképzéssel létrehozott típusok alaptípusok és/vagy általunk definiált egyszerű típusok összessége:

```
<xs:simpleType name="variant">
 <xs:union memberTypes="xsd:float xsd:integer xsd:string xsd:dateTime xsd:boolean xsd:long "/>
</xs:simpleType>
```

Az így létrehozott típusunk erősen emlékeztet a Visual Basic variant típusára, ami sokféle adattípust képes leírni.
Végül lássuk hogyan áll össze a kép. A korábban deklarált egyszerű típusok helyhiány miatt nem szerepelnek a listában.

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

<!-- Az egyszerű típusok deklarációja -->
<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"/>
```

folytatjuk...

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

Soczó Zsolt
Zsolt.Soczo@netacademia.net