SZAKDOLGOZAT

Bak Dániel Krisztián

Debrecen
2008

Debreceni Egyetem
Informatikai Kar

XML ALAPÚ FELÜLETEK TERVEZÉSE ADATCENTRIKUS WEBES ALKALMAZÁSOKHOZ

Témavezető:
Adamkó Attila
egyetemi tanársegéd

Készítette:
Bak Dániel Krisztián
programtervező informatikus

Debrecen
2008

TARTALOMJEGYZÉK

Tartalom

Bevezetés

Eddigi programozói pályafutásom alatt többször kerestek meg azzal, hogy honlapot készítsek. Az igények felmérése során világossá vált, hogy a megrendelő céljait valójában nem egy statikus honlap, hanem egy kis- vagy közepes méretű webalkalmazás fedné le. Miközben ezen munkákon dolgoztam szembesültem rengeteg olyan problémával, melyek mind a tervezés hiányából, mind pedig az implementáció során felmerülő hibalehetőségekből eredtek. Ezért keltette fel érdeklődésemet amikor Adamkó Atilla tanár úr megkeresett egy az ezeket a problémákat kiküszöbölő és a munkát mind gyorsabbá mind egyszerűbbé tevő megoldást kínáló projekt ötletével. A szakdolgozatom ezen projektben való részvételem eredményeinek bemutatására irányul.

A projekt célja Webes Információs Rendszerek fejlesztése, a tervezéskor létrejött UML diagramokból történő kódgenerálással. Ehhez természetesen a tervezés során ki kell egészítenünk az UML diagramokat az implementációhoz szükséges információkkal is.

Az alkalmazásunk alapját egy XML Schema (továbbiakban: XSD) jelenti, mely magában hordozza az adatlogikát. Ezen sémából kiindulva pedig ki tudjuk generálni mind az adatbázis sémát mind pedig a felhasználói felület megjelenítéséhez és kezeléséhez szükséges fájlokat.(Lásd: Függelék: 1. ábra).

A kódgenerálás alapja az UML osztálydiagramból az UML-szerkesztő program által kigenerált XML Metadata Interchange (továbbiakban: XMI) fájl. Jóllehet az XMI egy, az OMG által kidolgozott, szabvány, mégis felléphetnek kompatibilitási problémák, amikor egy bizonyos UML szerkesztő által kigenerált XMI kódot egy másik szerkesztő, importálás után nem feltétlenül helyesen jelenít meg. Ezért felhívnám a figyelmet arra, hogy az általam írt kód az „Altova UModel” program 2006-os verziójának második kiadásával generált XMI fájlokra működik helyesen.

A következő lépés ezen XMI fájlból az XSD fájl előállítása. Ez egy eXtensible Stylesheet Language Transformations (továbbiakban: XSLT) transzformáció eredményeként áll elő. Ezt hajtja végre a uml2xsd.xsl fájl, melyet Rákos Dániel hallgatóval készítettünk közösen. Ennek a transzformációnak az eredménye a schema.xsd fájl.

Ezután következik a projekt teljes egészében rám eső része, a schema.xsd-ből az xforms alapú felületek kigenerálása és a navigáció megoldása néhány általam írt php-illetve xsl fájl segítségével. (Lásd: 2.2-es pont)

A projekt következő lépései fejlesztés alatt állnak, ezek a következők: az XSD alapján az adatbázis-séma előállítása, ebből az adatbázis létrehozása, illetve az adatbázis és az xforms alapú felületek kommunikációja eXtensible Markup Language (továbbiakban: XML) alapokon. Az adatbázissal történő kommunikációt jelenleg XML fájlok reprezentálják.

A projekt önmagában újításnak számít, hisz a tervezés után olyan kész kódokat kapunk, melyekkel az implementálás minimálisra csökkenthető, vagy akár teljesen elhagyható.

Ahhoz, hogy az általam jelenleg bemutatott kódokat működésre bírjuk, szükség van egy szerverprogramra, mely képes php szkripteket futtatni, illetve kezeli a HTTP kéréseket. Mivel dinamikusan generálódnak a felületek, ezért szükség van egy XSLT processzorra is, melyet egy php szkript hív meg, az aktuális adatoknak megfelelően felparaméterezve, hogy hajtsa végre a transzformációt, majd megjeleníti az eredményt. Az általam jelenleg használt XSLT processzor a Saxon 9a, melynek a bináris(.exe) változatát építettem be a projektbe, így jelenleg csak Windows operációs rendszer alatt működik. Megtalálható belőle Java nyelven írt változat is, ennek használatához csak kis módosítást kell beírnunk a használt php fájlokba, de ez a verzió lassabb, és nehéz működésre bírni. A transzformációkat egy php nyelven írt interfészen keresztül hajtom végre, melyet a sourceforge oldalán találtam (Irodalomjegyzék [IN.1]). Erre azért volt szükség, mert jóllehet létezik olyan beépített php kiegészítő, mellyel végre tudunk hajtani XSLT transzformációkat, de ez sajnos csak az XSL stylesheet version 1.0-ra működik, és itt a transzformációk bonyolultsága megkövetelte a 2.0-ás verzió használatát. Ahhoz hogy az xforms alapú felületek megjelenjenek és működjenek „Mozilla Firefox” böngésző programra és az ehhez készült „Mozilla XForms” kiegészítőre van szükségünk, mivel jelenleg a böngészők nem támogatják natívan az xforms-t.

A fejlesztési konfigurációm tehát:

- Windows Vista operációs rendszer
- Szerverprogram: WampServer Version 2.0
- Böngészőprogram: Mozilla Firefox 3.0.4 + Mozilla XForms 0.8.6ff3 add-on

- XML szerkesztő (beépített XSLT processzorral): Oxygen XML Editor 9.1

A témaválasztásom azért esett erre a projektre, mert fejlesztésében már korábban is részt vettem, és ily módon a szakdolgozatom keretei közt tudom befejezni.

Másrészről többször fejlesztettem már webes alkalmazásokat, elsősorban adminisztrációs felületeket, és tudom, hogy mennyi mechanikus munkát kíván az implementáció. Ezzel együtt rengeteg hibalehetőséget is rejt magában ez a munka. Amennyiben viszont automatizáljuk ezeket a lépéseket, nem csak a mechanikus munkát takarítjuk meg, de nagyon sok hibalehetőséget is kizárunk.

Mindezek mellett érdekelnek az itt használt technológiák, mindegyikkel dolgoztam korábban más projekteken is. Tapasztalatom szerint viszonylag kevesen értenek mélyebben ezekhez az egyébként hatékony technológiákhoz, így témaválasztásommal némileg ezen technológiák megismertetése is célom. Úgy gondolom, hogy a modern informatikában egyre nagyobb teret nyerő XML-ben rejlő lehetőségek még messze nincsenek kihasználva, eme projekt egésze példát mutat rá, hogy hogyan tudjuk hatékonyan, és jól használni őket.

1. A használt technológiák

1.1 Az XML

Az XML (eXtensible Markup Language) a W3C által ajánlott általános célú leíró nyelv, speciális célú leíró nyelvek létrehozására. Az elsődleges célja strukturált adatszerkezetek tárolása, de használható más nyelvek definiálására is. Mint látni fogjuk, a felhasznált technológiák közül egyedül a PHP nem XML alapú nyelv, a többi mind az.

Az XML dokumentum önmagában egy fa adatstruktúrát reprezentál. Az adatok reprezentálása az ún. 'tag'-ek közt történik, melyekhez plusz információ hordozására attribútumokat is csatolhatunk. Egy elem a következőképpen néz ki az XML-ben:

```
<elem_neve attribútum_neve="attribútum értéke">Elem tartalma</elem_neve>
```

Attribútumból akármennyit csatolhatunk egy elemhez. Az elem tartalma lehet szöveges (karakteres) adat, vagy másik elem(ek), ily módon valósítva meg a fa-struktúrát. Egy dokumentum csak akkor tekinthető XML dokumentumnak, ha jól strukturált (well-formed), azaz megfelel az XML specifikáció [IN.2] által leírt szintaxisnak. A szintaxis legfontosabb szabályai:

- Egyetlen gyökérelem lehet a dokumentumban, ezt csak XML deklaráció, feldolgozó utasítások és megjegyzések előzhetik meg.
- A nem üres elemeket mind nyitó, mind záró tag-eknek kell határolni.
- Az üres elemek megjelölhetők üres elem (önlezáró) tag-gel, pl.: `<elem_neve/>`
- Minden attribútum érték idézőjelek között van, vagy szimpla (') vagy dupla (") idézőjelek között. Szimpla idézőjel szimpla idézőjelet, dupla idézőjel dupla idézőjelet zár.
- Az elemek egymásba ágyazhatók, de nem lehetnek átfedők. Így minden elemet egy másik elem kell hogy magában foglaljon, kivéve természetesen a gyökérelemet.
- Az elem nevek kisbetű-nagybetű érzékenyek.
  helyes: `<elemNeve></elemNeve>`
  helytelen: `<elemNeve></elemneve>`
- A dokumentum karakterkészletének meg kell felelnie a benne leírt karakterkódolásnak. A karakterkódolás általában az XML deklarációban van

meghatározva, de a szállító protokoll (pl. HTTP) is meghatározhatja. Ha nem adjuk meg az XML deklarációt, akkor alapértelmezés az UTF-8 és az UTF-16.

- Mivel a többek közt a ‘<’, ‘>’ karaktereknek speciális jelentésük van, ezért ha a szövegben használni kívánjuk őket, ún. ‘entity’-ket kell helyette használnunk. Pl.: < = `<`

Ahhoz viszont, hogy az XML dokumentum érvényes (valid) legyen, meg kell hogy feleljen egy adott sémának, ahol az alapvető XML szintaxishoz képest még egyéb megszorításokat tehetünk a dokumentum struktúrájára, illetve az adattagok és attribútum értékek típusára vonatkozóan. Ennek ellenőrzését hívjuk az XML dokumentum validálásának. Erről bővebben az XSD bemutatásánál írok.

Fontos még szót ejteni az XML névterekről. Ezek lehetővé teszik, hogy egy dokumentum több szótárból tartalmazzon XML elemeket és attribútumokat névütközések nélkül. Ez azért nagyon fontos, mert ha egy XML dokumentum több XML dokumentum elemeinek az összekeverésével készül el, névterek használata nélkül gyakran lépnének fel névütközések. A különböző XML feldolgozók a névtérből tudják, hogy mely elemekkel dolgozzanak. A névterek definiálása az ‘xmlns’ attribútum megadásával történik, melynek értéke minden esetben egy URI (Uniform Resource Identifier). Az URI egy sztring, mely egy internetes erőforrást azonosít. Amennyiben xmlns=”URI” formában adjuk meg, úgy az alapértelmezett névteret (default namespace), adjuk meg, mely arra az elemre, illetve annak összes névtérrel nem ellátott gyermekelemeire vonatkozik, amelyiknél ezt az attribútumot megadtuk. Amennyiben viszont nevesítjük névterünket: xmlns:névtér=”URI” formában, csak akkor vonatkozik adott elemre és gyermekelemeire a névtér, ha az elem nevek előtt megadjuk az adott névteret is névtér:elemnév formában. Például az xforms alapú felületek XHTML dokumentumokba vannak ágyazva. Itt a HTML és az Xforms elemek keverednek. Pl.:

```
<html
   xmlns=”http://www.w3.org/1999/xhtml”
   xmlns:xforms= "http://www.w3.org/2002/xforms">
<body>
   <label>Ez a teszt:</label>
   <table><tr><td>
      <xforms:input ref=”Test/elem”>
            <xforms:label>tesztelem</xforms:label>
      </xforms:input>
      </td></tr>
   </table>
</body></html>
```

Ebben az esetben két névteret definiáltunk: az alapértelmezett névtér (default namespace) az xhtml elemekhez tartozik, míg egy nevesített névtér ‘xforms’ néven az xforms elemekhez tartozik. Ily módon nincs ütközés az xhtml label és az xforms label között, amely elem mindkét nyelv része. Az xforms feldolgozó tudja, hogy neki mely elemekkel kell foglalkoznia, szintúgy az (x)html értelmező.

1.2 XMI-UML

Az XML Metadata Interchange szabvány [IN.3] MOF metaadatok XML formában való tárolására, illetve programok, alkalmazások közti átvitelére lett kidolgozva. Az XMI-t az Object Management Group (OMG) dolgozta ki, mely eredetileg az objektum-orientált rendszerek szabványainak lefektetésére jött létre, de mostanra programok, rendszerek és üzleti folyamatok modellezésére fókuszál. Az OMG hozta létre a Meta Object Facility (MOF)[IN.4] architektúrát, melynek része a Unified Modeling Language (UML)[IN.5] is. Ez a négyszintű architektúra négy absztrakciós szinten határozza meg a modellek felépítését (M3: meta-meta modell, M2: meta modell, M1: modell, és M0: adat modell). A számunkra jelenleg fontos réteg az M2-es, ebbe tartozik az UML grafikus nyelv, mely az objektumorientált programozás szabványos specifikációs nyelve.

Az XMI leggyakoribb felhasználási területei:

- A metaadatok cseréjének egyszerűsítése UML alapú modellezési eszközök és MOF metaadat-raktárak között
- A modellek eljuttatása a modellező eszközöktől a kódgeneráló szoftverekhez - erre használjuk mi is.

1.3 Az XSLT

Az XSLT (eXtensible Stylesheet Language Transformations) úgyszintén egy W3C ajánlás [IN.6], egy XML-alapú fájlformátumot illetve a hozzá tartozó feldolgozó rendszert jelöli. Az XSLT feldolgozót XML dokumentumok más formátummá, vagy más sémának megfelelő XML dokumentumokká alakításához használják. A konverzió során az eredeti file megmarad, s annak tartalma alapján létrejön egy új fájl a célformátumban. A keletkező dokumentum formátuma lehet többek között XML, HTML, XHTML, vagy sima szöveg például. A XSLT eredeti, és máig legelterjedtebb felhasználása, mint azt a neve is mutatja (stylesheet=stíluslap), hogy XML dokumentumokat a böngésző által megjeleníthető formátumba (HTML, XHTML) alakítson, ám fejlődése alkalmassá tette ennél jóval

komplikáltabb feladatok elvégzésére is. Jelen projektben kulcsszerepet játszik, mivel ezen transzformációk során jön létre az alkalmazás alapját jelentő XSD dokumentum, illetve az adatbevitelt és módosítást lehetővé tevő Xforms alapú felületek is. Ezt bővebben a projekt megvalósításáról szóló részben ismertetem.

Az XSLT egy deklaratív nyelv, XML szintaxist használ. Szabály-alapú nyelv, mint azt az XSLT feldolgozásánál ismertetem, szabályok alkalmazásával állítja elő ez eredményt. Az XSLT ily módon nem más, mint egymásba ágyazott template-ek összessége. Ezen template-ek gyökéreleme a

```
<xsl:template match=”/”>
      [utasítások]
</xsl:template>
```

(A ‘/’ helyett használható az XML dokumentum gyökérelemének neve is.)

Ezen template feldolgozása indul meg először, majd az

```
<xsl:call-teplate name=”template_név”/>
Vagy az
<xsl:apply-templates/>
```

Utasításokkal tudunk más template-eket meghívni. Ezen template-ek paraméterezhetők, beleértve magát a stíluslapot is. Természetesen létezik rekurzió és iteráció is. Az XSLT 2.0-ban definiálhatunk függvényeket is, melyek sajátossága, hogy itt nincsen context-node (aktuális csomópont). Ezen függvények nagyon hasznosak, ha egy utasítás végrehajtásához olyan információra van szükségünk, melyet egy másik részfa tartalmaz, de nem akarunk kilépni az aktuálisan feldolgozandó csomópontból.

Az XSLT feldolgozáshoz szükség van egy XSLT feldolgozó szoftverre, a feldolgozandó XML dokumentumra, de hozzá tud férni más XML dokumentumokhoz is, az XSLT stíluslapra, mely tartalmazza a feldolgozás szabályait, illetve céldokumentum(ok)ra. A feldolgozás a stíluslap szabályainak beolvasásával és a transzformálandó XML dokumentumból a forrásfa létrehozásával kezdődik. Majd a forrásfa gyökérelemét kezdi feldolgozni a gyökérelemre vonatkozó template utasításainak alkalmazásával. Ezt a template-et minden xslt-nek tartalmaznia kell, különben nem indul el a feldolgozás. Ezután minden csomópontra alkalmazza a rá illeszkedő template vagy utasításait. Az utasítások vagy csomópontot hoznak létre, vagy tovább olvasnak a forrásfában. A végeredmény, miután nincs több feldolgozandó csomópont, a célfából előálló kimenet lesz.

Az XSLT erősen támaszkodik az XPath nyelvre [IN.7], mely csomópontok helyének megadásához szükséges az XML dokumentumon belül. Az XML dokumentumokat fa adatstruktúraként kezeli, és így az XPath kifejezésekkel az XML fa részfáit jelölhetjük ki feldolgozásra. Nagyon sok függvényt tartalmaz melyek közt akár az eljárás-és objektumorientált nyelvekből jól ismert sztring-függvényeket is megtalálhatjuk.

1.4 Az XSD

Az XML Schema Definition W3C ajánlás [IN.8] egy XML séma nyelv, melynek segítségével megkötéseket tehetünk az XML dokumentumunk struktúrájára, és adattagjainak illetve attribútum értékeinek típusára vonatkozóan. Ehhez rendelkezésünkre állnak beépített típusok is, mint például az ’xsd:date’ illetve létrehozhatunk saját típusokat is. Az XSD úgyszintén XML szintaxist használ, ily módon sokkal hatékonyabban tudjuk megszabni a strukturális és egyéb megszorításainkat, mint elődjével a DTD (Document Type Definition)-vel. Az XSD dokumentum alapvetően elemek, azok attribútumai és típusának definiálásából áll. A projektben használt alapvető elemek ismertetése:

- <xsd:element> : az XML dokumentum egy elemének definiálására szolgál. A ’name’ attribútum adja meg az elem nevét, vagy a ’ref’ attribútummal hivatkozhatunk, egy a sémában máshol definiált elemre. A ’type’ attribútummal a típusát (lásd lent), a ’maxOccurs’ illetve ’minOccurs’ attribútumokkal pedig az elem előfordulások számát adhatjuk meg.
- <xsd:attribute> : egy XML elem egy attribútumának definícióját adhatjuk meg vele. Úgyszintén rendelkezik a ’name’ és a ’type’ attribútumokkal, melyek ugyanazon funkciókat látják el, mint korábban.
- <xsd:simpleType> : egyszerű típus definiálására szolgáló elem, általában valamely beépített típus megszorításával hozzák létre. Amennyiben egy <xsd:element> vagy <xsd attribute> elemben szerepel, nem lehet nevesíteni, egyébként igen, és így nevét használhatjuk a ’type’ attribútum értékeként.
- <xsd:complexType> : komplex típus definiálására szolgál, megadhatunk <xsd:element> elemeket melyek az adott komplex típusú elem gyermekelemeiként jelenhetnek meg

- <xsd:extension> : típus kiterjesztés csak <xsd:complexContent> elemben szerepelhet és kötelezően meg kell adni a ’base’ attribútumát, mely megadja, hogy mely elemből származtatjuk.

Nagy vonalakban ezen elemekből épül fel a sémánk, oly módon hogy deklaráljuk a gyökérelemet, annak típusát és gyermekelemeit és azok típusát. Az XSD-ben alapvetően háromféle típus van. A beépített típusok, a fent említett <xsd:simpleType>, és az <xsd:complexType>. Egy <xsd:element> típusát kétféle módon adhatjuk meg. Használhatjuk a ’type’ attribútumot, melynek értéke lehet beépített típus, illetve nevesített egyszerű vagy összetett típus. Illetve az <xsd:element> gyermekelemeként közvetlenül is definiálhatjuk típusát az <xsd:complexType> vagy az <xsd:simpleType> elemek segítségével.

1.5 Az XForms

Az XForms szintén a W3C ajánlásával [IN.9] jött létre, ez a nyelv jelenti az eddig megszokott HTML-formok új generációját. A fentiekhez hasonlóan az XForms is az XML szintaxist használja. Nagy előnye a HTML űrlapokhoz képest, hogy elválasztja az adatot a reprezentációtól ily módon biztosítja az újrafelhasználhatóságot. Az XForms használatával sok szkriptet illetve fölösleges kliens-szerver kommunikációt megtakarítunk, mivel a modell alapján valós időben ellenőrzi a beírt szöveget, illetve csak helyes adatokat enged elküldeni, így nem kell a szerveren küldés után ellenőrizni, és hibát visszadobni, amennyiben helytelen.

Az XForms kitöltött adatokat az példány-adatokban (instance-data) helyezi el. Ez a példány jelenti az elküldendő XML dokumentum csontvázát. Amennyiben ezen példányban már vannak adatok, ezeket automatikusan betölti a megfelelő beviteli mezőbe. Ha típussal is felruházzuk a példány-elemeket, akkor a megfelelő típusú beviteli mező fog megjelenni, például ’xsd:date’ esetén egy kalendárium, melyből kiválaszthatjuk az aktuális dátumot (lásd Függelék: 3. ábra), így egyszerű típusok esetén elég a megjelenítéshez az <xforms:input> -elem használata.

Az XForms-ban három fő típusú elemet használhatunk. A modell leírásához, a felhasználói felület leírásához, és az interakciók és események kezeléséhez használható elemeket.

Az <xforms:model> elemben adhatjuk meg a példány-adatokat: <xforms:instance>, az esetleges további megszorításokhoz és adat-típus kötésekhez használható <xforms:bind> elemeket, illetve az adatok elküldéséhez szükséges adatokat: <xforms:submission>. Az

XForms Submit Protocol egy adatcsatornát definiál, ami az adat-példány és az XForms feldolgozó között létesít kapcsolatot. Ez leírja, hogy hogyan küld és fogad adatot az XForms, beleértve, hogy hogyan kell az űrlapnak a felfüggesztését, folytatását és befejeződését kezelni. A példányt megadhatjuk egy külső XML-file formájában is az ‘src’ attribútummal. Modellből többet is használhatunk, ilyenkor a felhasználói felületen meg kell adnunk, hogy az adott mező, mely modellbeli elemhez van kötve.

A felhasználói felületen adhatjuk meg a beviteli mezőket, illetve az interakciók és események kezelését szolgáló elemeket. Ezek elég nagy tárháza áll rendelkezésünkre, néhány a projektben használt elem bemutatása:

- <xforms:input> : a legalapvetőbb beviteli mező, kötelezően meg kell adni egy ‘bind’ vagy ‘ref’ attribútumot, mellyel a tartalmát kötjük az egyik példány-adathoz. Ezen adat típusától függ a megjelenése is ‘boolean’ típus esetén checkbox, ‘date’ típus esetén gombnyomásra egy naptárat látunk, stb.
- <xforms:select> : Egy listából történő egy vagy több elem kiválasztására szolgáló mező, értékét az <xforms:value> gyermekelemében tárolt adat jelenti. A ‘ref’ vagy ’bind’ attribútum itt is kötelező.
- <xforms:trigger> : Egy gomb jelenik meg a felhasználói felületen, mellyel interakció esetén különböző aktivitásokat hajthatunk végre. Pl.:

```
<xforms:trigger>
      <xforms:insert nodeset=”root/child”ev:event=”DOMActivate”>
      <xforms:label>elem hozzáadása</xforms:label>
</xforms:trigger>
```

A fenti példában egy új elemet (vagy elem-csoportot) szúrunk be a form-ba lenyomás hatására.

- <xforms:action> : Egy esemény hatására aktiválódó elem, melynek segítségével különböző eseményeket válthatunk ki, vagy utasításokat hajthatunk végre. Az ev:event attribútum megadása kötelező, ez adja meg, hogy mely esemény kiváltódása esetén aktivizálódjon az

elem. Pl:

```
<xforms:input ref=”root/child”>
     <xforms:label>gyermekelem:</xforms:label>
      <xforms:action ev:event=”xforms:invalid”>
            <xforms:message>Hibás adat!</xforms:message>
      </xforms:action>
</xforms:input>
```

A fenti példában az ’xforms-invalid’ (hibás adat) esemény hatására egy üzenetet jelenítünk meg a felhasználónak, melyben közöljük, hogy nem helyes adatot próbál beírni.

- <xforms:label> : statikus vagy dinamikus szövegek megjelenítésére használjuk.
- <xforms:repeat> : adat, illetve adat-csoport ismétléséhez használható, ilyenkor a példány adatokban az elem vagy elem csoport úgyszintén ismétlődik.

pl.:
```
<xforms:repeat nodeset=”root/child”>
      <xforms:input ref=”.”>
      </xforms:input>
</xforms:repeat>
```

Ekkor a root elem child gyermekeleme és a hozzátartozó mező ismétlődhet. Új elem beszúrását a fenti xforms:trigger példáján szemléltettem.

- <xforms:submit> : az elküldéshez használható elem, kötelező attribútuma a ’submission’, mely megadja, hogy a modellben definiált elküldési lehetőségek közül melyiket válassza.

Ezek az XForms általam leginkább használt elemei, de ezen kívül rengeteg elem és lehetőség áll még rendelkezésünkre a form és az adatok kezelésére.

Az XForms igazi ereje a fent említett MVC kialakításában rejlik, könnyen karbantartható, és magasabb rendű adatstruktúrákat képes hatékonyan kezelni. Nagy hátránya viszont, hogy nincsen natív támogatása jelenleg egyetlen elterjedt böngészőben sem, mely nagy akadálya az XForms térnyerésének.

1.6 Az XHTML

Az XHTML (eXtensible Hipertext Markup Language) W3C ajánlás a korábbi HTML nyelv újrafogalmazása az XML szintaxis szabályainak betartása mellett (well-formed). Az XHTML 1.1 modulárissá teszi a nyelvet ily módon névterekkel ellátva, XML adatokat is elhelyezhetünk az XHTML dokumentumunkban. Ez azért fontos, mert így lehet a projektben XForms ’szigeteket’ elhelyezni az XHTML-en belül. Az XHTML elemeket a HTML értelmező dolgozza fel, míg az XForms elemeket az XForms processzor. Fontos még megjegyezni, hogy

A még a böngészők által nem támogatott XHTML 2.0 ajánlás teljesen szakít a múlttal, lefelé nem kompatibilis, és a nyelv részévé tenné az XForms-modul.

1.7 A PHP

A PHP Hipertext Preprocessor egy nyílt forráskódú szerveroldali szkriptnyelv, mely máig nem rendelkezik hivatalos specifikációval. Bátran állíthatom, hogy mára a legnépszerűbb szkriptnyelvvé nőtte ki magát. Legfőképp dinamikus weboldalak készítésére használják, de az objektumorientált technológia beépülésével felhasználási területe egyre nő. Nagyon rugalmas nyelv, nem szükséges a változókat deklarálni, szintaktikája egyszerű.

Amikor egy HTTP kérés érkezik a kiszolgálóhoz, először lefuttatja a PHP szkriptet egy külső modul segítségével, majd a kimenetet küldi válaszul a kérésre. Pl.:

```
$date=date("Y-m-d");
echo "a mai dátum: ".$date;
```

Tehát jelen esetben a válasz a `"a mai dátum: 2008.11.21"` szöveg lesz. A PHP-t a projekten belül a navigáció megoldására és a transzformációk meghívásának végrehajtására használom.

2. A projekt megvalósítása

Ahhoz, hogy a rendszer működését megértsük, előbb néhány szó a projektről. A cél egy olyan rendszer fejlesztése, mellyel web alapú információs rendszerek implementációját lehet viszonylag egyszerűen, kódgenerálással létrehozni. Ahhoz, hogy szemléltessük a kódgenerálás működését, egy, a doktoranduszokat nyilvántartó rendszer modelljét adtuk meg, melyet témavezetőm, Adamkó Attila tanár úr tervezett. Látni fogjuk, hogy a rendelkezésre álló UML class diagramból [2. ábra] készült XMI dokumentumból hogyan készül el az alkalmazás központi adatstruktúráját meghatározó XSD dokumentum, és hogy ebből a sémából hogyan generálódnak ki a felhasználóval való kapcsolattartáshoz szükséges XForms oldalak.

2.1. XMI dokumentumból az XSD séma előállítása

Mint a bevezetőben említettem, a projekt ezen részét Rákos Dániel hallgatóval közösen készítettük. A központi kérdés ennél a lépésnél, hogy az objektumorientált paradigmában megszokott eszközöket (osztály, generalizáció, asszociáció) milyen, az XML Schema-ban használható elemekké alakítsuk.

Maga az egész csomag (UML:package), amely a class diagramot tartalmazza, lesz a gyökéreleme a séma által elfogadott XML dokumentumoknak, mint az az alábbi kódrészletből kitűnik:

```
<xsd:schema xmlns:uml="http://schema.omg.org/spec/UML/2.1"
            xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="XPhD">
```

Esetünkben ’XPhD’ a csomag neve.

Az osztályok (UML:Class), nevesített típusként jelennek meg a sémában:

```
<xsd:complexType name="SzemelyType">
      <xsd:sequence>
         <xsd:element name="nev"/>
         <xsd:element name="szuletesi_datum" type="xsd:date"/>
         <xsd:element name="allampolgarsaga"/>
         <xsd:element name="email"/>
      </xsd:sequence>
      <xsd:attribute name="id"/>
</xsd:complexType>
```

A fenti példa a ’Személy’ osztály reprezentálása. Mint látható, rendelkezik egy ’id’ attribútummal, maradva az analógiánál: ezt használjuk OID (Object ID)-ként, ezzel fogjuk beazonosítani az egyes példányokat. Az osztály attribútumai XML elemekként fognak

megjelenni az XML példányban. Ezek rendelkezhetnek típussal is, mint az fent a születési dátumnál látható. Ezen nevesített típusokkal adjuk meg a gyökérelem gyermekelemeinek típusát.

A generalizáció (UML:generalization) egy ’xsd:extension’ elemként jelenik meg a sémában, melynek a ’base’ attribútuma adja meg a szülőosztály típusát.

```
<xsd:complexContent>
   <xsd:extension base="SzemelyType">
      <xsd:sequence>
         <xsd:element name="neptun_kod"/>
         <xsd:element name="aktiv" type="xsd:boolean"/>
         <xsd:element name="egyetemi_vegzettseg"/>
      </xsd:sequence>
   </xsd:extension>
</xsd:complexContent>
```

A fenti kódok Rákos Dániel hallgató munkájának eredményeképp állnak elő. Az én feladatom volt megoldani az asszociációk kérdését.

Az egyirányú asszociációk megoldása úgy történt, hogy az adott elem gyermekelemeként jelenik meg az UML-ben használt attribútum gyanánt. Természetesen, hogy ne lépjen fel redundancia, ezek az elemek, csak azonosítókat tárolnak.

```
<xsd:element name="biralo" maxOccurs="3" minOccurs="1"
type="id" default="Oktato"/>
```

Itt látható, hogy egy trükkhöz voltam kénytelen fordulni a tárolást illetően. Az elem típusa ’id’, de ebből még nem derül ki, hogy milyen típusú elemre hivatkozik. Az ’id’-k ugyan egyediek, de a felhasználói felület kialakításánál tudnom kell, hogy milyen elemre hivatkozik. Ezért a ’default’ attribútumban van letárolva a hivatkozott elem típusa. Ezt a következő XSLT kód hivatott megoldani:

```
<xsl:for-each select="$node">
            <xsl:if test="@xmi:id=$type">
                  <xsl:attribute name="type">id</xsl:attribute>
                  <xsl:attribute name="default">
                        <xsl:value-of select="@name"/>
                  </xsl:attribute>
            </xsl:if>
</xsl:for-each>
```

Itt a paraméterként kapott, csomópontokat tartalmazó ’node’ változó értékeit végigjárva keressük azt az elemet, amelynek az ’xmi:id’ attribútumának értéke megegyezik a paraméterként kapott keresett típusazonosítóval.

A kétirányú asszociációk már nagyobb gondot okoztak. Ha mindkét elemben eltárolom a másik azonosítóját nehezen kezelhető, hisz ugyanaz a kapcsolat kétszer szerepel. Erre azt a megoldást találtam, hogy az asszociációkat tároló elemeket nevesített xsd:complexType elemekben tároltam el, melyek egy adott asszociáció fajtát reprezentálnak. Ezekből akármennyi lehet, ily módon minden egyes asszociációhoz egy-egy elem példány áll elő. Ezen elemek csak azonosítókat tárolnak, azt pedig hogy milyen típusú elemre hivatkoznak vagy az elem nevében, vagy ha nevesített asszociációról van szó, akkor a fentiekhez hasonlóan a ’default’ nevű attribútumban tároltam el.

```
<xsd:complexType name="AssociationU40bc8fab-9cc6-4d9f-8068-dbf8db7838cd">
      <xsd:sequence>
         <xsd:element name="titkar" type="id" default="Oktato"/>
         <xsd:element name="DoktoriIskola"/>
      </xsd:sequence>
</xsd:complexType>
```

A komplex típus neve az ’Assotiation’ szóból és az osztály xmi:id-jából tevődik össze, ezzel garantálva az egyediséget. Mint itt látható, egy oktató lehet egy doktori iskola titkára. Ezért fontos, hogy az UML diagramunkban nevesítsük az asszociációkat (lehetőleg mindkét végét), mert az így nevesített kapcsolatok érthetőbbek lesznek a felhasználók számára.

Az asszociációs osztályok (UML:AssociationClass) szintén a fentiekhez hasonló adatszerkezetekké alakulnak, azzal a különbséggel, hogy ezek tárolják az asszociációs osztályban meghatározott kapcsolatspecifikus adattagokat is.

```
<xsd:complexType name="AssociationClassUd9b5a1ec-8d0b-4231-a24f-
da04b2d24c44">
      <xsd:sequence>
         <xsd:element name="vezeto" type="id" default="Oktato"/>
         <xsd:element name="Program"/>
         <xsd:element name="kezdesClassVariable"  type="xsd:date"/>
         <xsd:element name="befejezesClassVariable"  type="xsd:date"/>
      </xsd:sequence>
   </xsd:complexType>
```

Ahhoz, hogy az asszociációs osztály adattagjait egyszerű legyen megkülönböztetni a kapcsolat két végét leíró elemektől, a nevük után hozzáfűztem a ’ClassVariable’ kifejezést. Az asszociációs osztályt előállító kód részlete:

```
<xsd:complexType name="AssociationClass{@xmi:id}">
  <xsd:sequence>
   <xsl:call-template name="CreateAssociation"/>
    <xsl:for-each select="ownedAttribute">
     <xsl:element name="xsd:element">
      <xsl:attribute name="name" select="concat(@name, 'ClassVariable')"/>
       <xsl:if test="@type">
```

```
        <xsl:call-template name="type">
          <xsl:with-param name="type" select="@type"/>
          <xsl:with-param name="node" select="../../packagedElement"/>
        </xsl:call-template>
      </xsl:if>
    </xsl:element>
  </xsl:for-each>
 </xsl:sequence>
</xsl:complexType>
```

A kód a következőképp működik: miután előállt az 1. 2. sorban megadott utasítások alapján a komplex típust leíró elem nyitó tagja, illetve az elemeket tároló ’sequence’ elem nyitó tagja, meghívódik a ’CreateAssociation’ template, mely beazonosítja az xmi:id alapján az asszociációban résztvevő osztályokat, és létrehozza a megfelelő elemeket. Ezután kiválasztjuk az adott csomópont alatt található összes ’ownedAttribute’ nevű csomópontot (ezek tárolják az asszociációs osztály adattagjaira vonatkozó információkat) és mindhez előállítunk egy XML csomópontot meghatározó ’<xsd:element>’ elemet. A ’name’ attribútum értéke adja meg, hogy mi legyen a csomópont neve, ennek előállításához összefűzzük a csomópont XMI-szerinti nevét és a ’ClassVariable sztringet’. Miután ez megtörtént, leellenőrizzük, hogy van-e típus megadva az adattaghoz. Amennyiben igen, meghívjuk a ’type’ nevű template-et melyből kódrészletet már közöltem fent. Paraméterként átadjuk a típus azonosítóját, illetve az osztályokat tartalmazó csomópontot, majd lezárjuk a tag-eket, és készen van az asszociációs osztályt tartalmazó csomópontunk.

Az uml2xsd.xsl fájl tehát előállítja az XSD sémát mellyel meghatároztuk az alkalmazásunk alapját képző adatstruktúrákat. Itt találhatók meg az egyes osztályokat illetve az ezek közti kapcsolatokat reprezentáló elemtípusok. Ezen sémát felhasználva fog előállni az adatbázis séma, illetve ezt felhasználva jutunk el a végfelhasználói felületekhez is.

2.2 Az XSD sémából az XForms alapú felületek előállítása

Ezen formok jelentik a közvetlen kapcsolatot a felhasználóval. Itt tud új adatokat bevinni a rendszerbe, illetve már meglévőket módosítani. A formok generálása mindig dinamikusan történik, részint a sémától, részint pedig a felhasználó által választott aktivitástól függően.

2.2.1 Navigáció

A felhasználó először, egy a séma elemtípusok neveiből generált oldalt lát maga előtt. Itt tudja kiválasztani, hogy milyen típusú elemet szeretne módosítani, vagy bevinni a rendszerbe. Ez egy php-szkript által meghívott igen egyszerű XSLT transzformáció végeredményeként áll elő. A menu.xslt transzformáció a feldolgozást a Schema által meghatározott gyökérelemmel kezdi, majd bejárja gyermekelemeit és mindegyiknek kiírja a nevét illetve a „módosít” szöveget és mindkettőhöz létrehoz egy linket. Amennyiben új elemet kíván létrehozni az createForm.php szkriptre navigálunk, megfelelően felparaméterezve. Ha viszont meglevőt szeretne módosítani, akkor a showAll.php szkripre navigálunk, mely egy xslt transzformáció segítségével kiírja az oldalra az adott típusú elem minden előfordulását egy linkkel, mely szintén az createForm.php szkripre mutat.

A createForm.php szkript:

```
<?php
  include_once("../transform/XML_XSLT2Processor-0.5.0/XSLT2Processor.php");
  $date=date("Y-m-d");
  $schema=new DOMDocument;
  $schema->load("../uml2xsd/schema.xsd");
  $xsl=new DOMDocument;
  $xsl->load("xsd2xform.xsl");
  $transformer=new
XML_XSLT2Processor('SAXON',"../transform/saxon/Transform.exe", "CLI");
  $transformer->importStylesheet($xsl);
  $transformer->setParameter("", "class", $_GET["class"]);
  $transformer->setParameter("","date",$date);
  if($_GET["modify"]){
    $transformer->setParameter("","modify","true");
    $transformer->setParameter("","id",$_GET["id"]);
  }
  else{
    $transformer->setParameter("","id", md5($_GET["class"].time()));
  }
  $transformer->transformToURI($schema, "xform.xhtml");
  header("location:xform.xhtml");
?>
```

A szkript működése a következő: beállítjuk a date, schema, xsl és transformer nevű változóinkat. A date az aktuális dátumot, a schema a sémánk relatív elérési útját, az xsl a transzformációs file relatív elérési útját, a transformer pedig egy új XML_XSLT2Processzor típusú objektum lesz, melynek átadjuk az XSLT processzor relatív elérési útját. Ezután megadjuk a processzornak az xsl fájlt, melyet az xsl változónkban tároltunk el, majd felparaméterezzük a stíluslapunkat. A class paraméter adja meg, hogy mely osztály példányát

szeretnénk előállítani, így mely osztály adattagjaihoz tartozó mezőket kívánunk megjeleníteni. Ezt az előző oldalról a 'GET' HTTP request-tel továbbítottuk. Ezt a PHP a $_GET nevű asszociatív tömbben tárolja ezután, így fogok rá hivatkozni. Majd megvizsgáljuk, hogy kaptunk-e a $_GET tömbben 'modify' azonosítójú elemet. Ha igen, a felhasználó az előző oldalon megadta, hogy mely adatokat szeretné módosítani, így kaptunk egy 'id' azonosítójú elemet is. Ebben az esetben átadjuk a stíluslapnak a 'modify' és az 'id' értékét. Amennyiben nem módosításról van szó, generálunk egy új egyedi id-t (az osztály nevéből és az aktuális időpontból), és ezt adjuk át a stíluslapnak. Ezután utasítjuk a processzort, hogy hajtsa végre a transzformációt és a végeredményként előálló xform.xhtml oldalra navigálunk.

Miután a felhasználó kitöltötte az űrlapot, és a mentés gombra kattintott, a show.php oldalra kerül. Ez a szkript gondoskodik a kitöltött űrlapadatok mentéséről, bár mivel a projekt adatbázis modulját még nem kezdték el fejleszteni, így ez csak szimulálva van XML fájlok által. Az űrlapadatok részint az adott osztály nevét hordozó XML fájlba, részint pedig az Associations.xml fájlba kerülnek egy-egy XSLT transzformáció eredményeként. Amennyiben új adatokat kell menteni, ezek egyszerűen hozzáfűzésre kerülnek a már meglévőkhöz, ha pedig módosításra került sor, akkor a megfelelő elemek felülíródnak.

2.2.2 A formok előállítása - áttekintés

A formok előállítását az xsd2xforms.xsl transzformáció végzi. Lényegében ez maga a projektből rám szabott feladat. A navigációval a tesztelhetőség miatt, a XSD generálással pedig az asszociációk megoldatlansága miatt kellett foglalkoznom.

Az XForms technológiával már a technológiákat bemutató részben megismerkedtünk, ebben a fejezetben mélyebben belemegyünk az XForms működésének konkrétumaiba. Majd hivatkozni fogok, mind a fent leírt sémára, mind pedig az UML diagramra.

Már láttunk XForms 'kódszigeteket', de az előttünk álló feladat felméréséhez tekintsük az alábbi egyszerű formot XHTML-be ágyazva:

```
<?xml version="1.0" encoding="UTF-8"?>
<html 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”
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns="http://www.w3.org/1999/xhtml">
   <head>
    <xforms:model schema=”http://kiszolgalo/schema.xsd”>
      <xforms:instance>
        <person>
         <fname xsi:type=”xsd:string”/>
```

```
          <lname xsi:type=”xsd:string”/>
        </person>
      </xforms:instance>
      <xforms:submission id="form1" action="submit.php" method="post">
            <xforms:action ev:event=”xforms-submit-error”>
               <xforms:message level=”modal”>
                  Hiba a küldéskor!
               </xforms:message>
            </xforms:action>
    </xforms:model>

   </head>
   <body>
     <h1>XForms Példa</h1>
     <xforms:input ref="fname">
      <xforms:label>Keresztnév</xforms:label>
     </xforms:input>
     <xforms:input ref="lname">
      <xforms:label>Család név</xforms:label>
      <xforms:message ev:event=”xforms-invalid” level=”modal”>
            Hibás adat!
      </xforms:message>
     </xforms:input>
     <xforms:submit submission="form1">
      <xforms:label>Submit</xforms:label>
     </xforms:submit>
  </body>
</html>
```

A fenti példában látható, hogy az XForms modellünket a ’head’ elemben helyezhetjük el. A modellhez a ’schema’ attribútum értékével rendelhetünk XML Schema-t, mellyel megkötéseket tehetünk a modellelemekre. Az ’xforms:instance’ elembe ágyazva helyezhetjük el adat-példányunkat, mely az elkészítendő XML dokumentum vázát adja. Itt az elemekhez rendelhetünk séma típusokat az xsi:type attribútum értékével. A modellhez még meg kell adni a ’xforms:submission’ elemet is, mely definiálja, hogy hova, hogyan küldjük el az adatokat. Itt kötelező az ’id’ attribútum megadása, hiszen submission elemből többet is definiálhatunk egy modellen belül, így az id attribútummal tudjuk beazonosítani, hogy melyik küldési mód szerint kívánjuk az adatokat elküldeni. Ezzel a módszerrel egyébként az XForms sokkal átláthatóbbá tette a küldést, mint amit a HTML űrlapoknál megszokhattunk. A fenti példából az is kitűnik, hogy hiba esetén ’xforms-submit-error’ üzenetet ’xforms:message’ küldhetünk a felhasználónak. A ’body’ elemben helyezhetjük el a formunk felületét illetve viselkedését meghatározó elemeket. Itt az ’xforms:input’ mezőt használjuk, melyet mint a technológia ismertetésénél írtam kötelezően hozzá kell kötnünk egy példány-adattaghoz. A ’ref’ attribútum használata esetén értéke egy XPath kifejezés lesz, mely megadja, hogy a példány gyökérelemétől hogyan tudunk eljutni a kívánt adattagig. Jelen esetben az adott adattag

közvetlenül a gyökérelem alatt helyezkedik el, így elég egyszerűen csak a nevével hivatkozni rá. Minden más ismerős lehet az 1.5-ös pontból.

A fenti példát elemezve tehát a következő feladatok állnak előttünk a transzformáció megvalósításánál. Létre kell hozni a sémából modellt és a beviteli mezőket illetve az ezek kezelését szolgáló elemeket. Létre kell hozni az XForms-t tartalmazó XHTML-t. Mivel ezen elemek jól keverednek, ezért a következő megoldáshoz folyamodtam. Az XSLT fő template-je a séma gyökérelemét kezdi feldolgozni. Létrejön a befogadó XHTML kerete, majd a megfelelő helyen meghívom az XForms elemeket előállító template-eket. Íme a kódrészlet:

```
<xsl:template match="xsd:schema">
    <html xmlns="http://www.w3.org/1999/xhtml"
        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"
        xmlns:ev="http://www.w3.org/2001/xml-events">
       <head>
        <xforms:model schema="http://127.0.0.1/project/uml2xsd/schema.xsd">
            <xsl:attribute name="id">
                 <xsl:value-of select="$class"/>
            </xsl:attribute>
            <xforms:instance>
                <xsl:choose>
                   <xsl:when test="$modify">
                      <xsl:call-template name="referenceModel">
                         <xsl:with-param name="type" select="$class"/>
                      </xsl:call-template>
                   </xsl:when>
                   <xsl:otherwise>
                    <xsl:element name="{$root/@name}">
                       <xsl:apply-templates mode="model">
                         <xsl:with-param
                                    name="type"
                                    select="fn:getType(xsd:element)"/>
                       </xsl:apply-templates>
                    </xsl:element>
                   </xsl:otherwise>
                </xsl:choose>
            </xforms:instance>
            <xsl:choose>
             <xsl:when test="$modify">
              <xforms:submission id="submit"
      action="http://127.0.0.1/project/navigation/show.php?type={$class}&amp;modify=true" method="post">
                <xforms:action ev:event="xforms-submit-error">
                  <xforms:message level="modal">
                        Hiba a küldéskor!
                  </xforms:message>
                </xforms:action>
              </xforms:submission>
             </xsl:when>
             <xsl:otherwise>
```

```
              <xforms:submission id="submit"
action="http://127.0.0.1/project/navigation/show.php?type={$class}"
method="post">
                 <xforms:action ev:event="xforms-submit-error">
                   <xforms:message level="modal">
                        Hiba a küldéskor!
                   </xforms:message>
                 </xforms:action>
              </xforms:submission>
             </xsl:otherwise>
           </xsl:choose>
         </xforms:model>
        </head>
        <body>
          <h1>
            <xsl:value-of select="$class"/>
          </h1>
       <xsl:apply-templates mode="form">
          <xsl:with-param name="type" select="fn:getType(xsd:element)"/>
       </xsl:apply-templates>
       <table>
          <tr>
            <td align="center">
               <xforms:submit submission="submit">
                     <xforms:label>Mentés</xforms:label>
               </xforms:submit>
            </td>
          </tr>
       </table>
     </body>
    </html>
   </xsl:template>
```

A template az xsd:schema elemet dolgozza fel. Először létrehozza a ’html’ elemet a dokumentumunk gyökérelemét a megfelelő névtér attribútumokkal. Itt látható ezen névterek hasznossága, hisz jelen esetben öt különböző szótárból használunk elemeket dokumentumunk felépítéséhez. Ezen névterek nélkül elkerülhetetlenek lennének névütközések, illetve nem lenne meg a különböző feldolgozók alkalmazásának a lehetősége. Ezután létrehozzuk a ’head’ elemet, mely egyéb információk mellett a modellünket is tartalmazni fogja. Ezen elemen belül létrehozzuk az ’xforms:model’ elemet, melynek ’id’ attribútumát a createForm.php által beállított ’class’ paraméter értékére állítjuk. Ezen paraméter adja meg, hogy a sémán belül mely osztályt reprezentáló elemet kívánjuk feldolgozni, mint azt a navigáció ismertetésénél írtam. Következik az ’xforms:instance’ elem előállítása, mely tartalmazza a példány-adat információkat. Ekkor döntési helyzetbe kerülünk, mivel ha már meglévő adatokat szeretnénk módosítani, akkor ezen adatokat be kell töltenünk a példány-adatokba, egyébként pedig egy üres XML-vázat kell létrehoznunk. Hogy módosításról van-e

szó, azt a ’modify’ paraméter adja meg, melyet szintén a createForm.php állít be a korábbi felhasználói interaktivitástól függően. Az ’xsl:choose’ elem egy többirányú elágaztatást valósít meg. Az ’xsl:when’ elem ’test’ attribútumának az értéke határozza meg, hogy belépünk-e az adott végrehajtási ágba. Itt elég pusztán a paraméterre hivatkozni, hisz ha nem módosításról van szó, nem kapunk ilyen paramétert. Ha módosításról van szó, akkor a ’referenceModel’ template-et hívjuk meg, mely megkeresi a szintén paraméterül kapott ’id’ alapján az adott elemet illetve az elem kapcsolatait az adatbázis-kommunikációt szimuláló XML fájlokból, és előállítja a példány-adatokat. Ha viszont új adatokat szeretnénk bevinni, akkor az ’xsl:apply-templates’ utasítással parancsot adunk a nem nevesített template-ek végrehajtására ’model’-módban. Egy ilyen template van, mely a séma alapján előállítja a példányt. Az ’xforms:submission’ elem létrehozásánál is vizsgálnunk kell, hogy módosításról van-e szó, hisz ettől függően felparaméterezett php szkriptnek kell elküldenünk a példány-adatokat. A paraméterezéshez a korábban már említett HTTP GET kérést használjuk, melyet az URL címben a http://domainév/elérési_út-hoz hozzáfűzött ’?’ karakter után kulcs=érték párokat adhatunk meg a ’&’ karakterrel tagolva.

Pl.: `http://localhost/project/xsd2xforms/createForm.php?class=Oktato&modify=true`

Itt a protokoll mindig a HTTP, a domain név a helyi gépen futó szervert azonosító ’localhost’, az elérési út a ’project/xsd2xforms/createForm.php’ az átadott paraméterek pedig a ’class’ és a ’modify’ értékeik pedig rendre ’Oktato’ és ’true’. Az ily módon felparaméterezett php szkript tudni fogja, hogy mely osztályhoz tartozó adatokat kapott, illetve hogy módosítás történt-e, így helyesen tudja majd paraméterezni a ’save.xsl’ illetve az ’associations.xsl’ transzformációkat. Miután létrejött az ’xforms:submission’ elem, létrehozunk alá egy gyermekelemet, mely figyelmezteti a felhasználót, ha nem sikerült az adatok elküldése. Ezzel létrejött a modell, lezárjuk a ’head’ elemet, és megnyitjuk a ’body’ elemet, mely az XHTML dokumentumunk törzsét jelenti. Az oldal tetejére kiírjuk az adott osztály nevét, hogy a felhasználó tudja hol van éppen. Mivel a felhasználói felület független attól, hogy új adatok kerülnek-e bevitelre, vagy meglévők módosulnak-e, így elég utasítást adni a template-ek ’form’ módban történő futtatására. Végezetül pedig létrehozzuk az adatok elküldéséhez szükséges ’xforms:submit’ elemet, mely egy gombot fog jelenteni a felületen.

Nagy vonalakban így áll elő a kívánt form, most nézzük az egyes részeket részletesebben!

2.2.3 A modell előállítása

Mint azt a 2.2.2 pontban láthattuk, a modellt vagy a ’referenceModel’ vagy pedig a ’model’-módban futtatott template végzi el. Nézzük előbb, hogyan áll elő modellünk, mikor új adatokat szeretnénk bevinni. A kódrészlet:

```
<xsl:template match="xsd:complexType" mode="model">
   <xsl:param name="type"/>
    <xsl:if test="@name=$type">
       <xsl:choose>
         <xsl:when test="xsd:complexContent/xsd:extension">
            <xsl:element name="{$class}">
               <xsl:attribute name="id">
                <xsl:value-of select="$id"/>
               </xsl:attribute>
               <xsl:call-template name="createModelNodes">
                 <xsl:with-param name="node"
select="xsd:complexContent/xsd:extension/xsd:sequence/xsd:element"/>
               </xsl:call-template>
               <xsl:call-template name="model">
                  <xsl:with-param name="type"
select="xsd:complexContent/xsd:extension/@base"/>
               </xsl:call-template>
            </xsl:element>
            <xsl:call-template name="associationForModel">
              <xsl:with-param name="node" select="../xsd:complexType"/>
              <xsl:with-param name="type" select="substring-before(@name,
'Type')"/>
          </xsl:call-template>
          <xsl:call-template name="associationForModel">
             <xsl:with-param name="node" select="../xsd:complexType"/>
             <xsl:with-param name="type" select="substring-
before(xsd:complexContent/xsd:extension/@base, 'Type')"/>
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
           <xsl:element name="{$class}">
               <xsl:attribute name="id" select="$id"/>
               <xsl:call-template name="createModelNodes">
                <xsl:with-param name="node"
select="xsd:sequence/xsd:element"/>
              </xsl:call-template>
           </xsl:element>
           <xsl:call-template name="associationForModel">
              <xsl:with-param name="node" select="../xsd:complexType"/>
              <xsl:with-param name="type" select="substring-before(@name,
'Type')"/>
           </xsl:call-template>
      </xsl:otherwise>
     </xsl:choose>
  </xsl:if>
</xsl:template>
```

A template minden a gyökérelem alatt található ’xsd:complexType’ nevű elemre lefut. Ha az adott csomópont neve megegyezik a ’type’ paraméter értékével, akkor elkezdődik a csomópont feldolgozása. Itt egy döntési helyzet áll elő. Amennyiben egy generalizációval

rendelkező osztályt reprezentáló elemről van szó, akkor mind az adott osztály mind pedig a szülőosztályának attribútumaihoz elő kell állítani az őket reprezentáló elemeket. Ennek ellenőrzése látható az ’xsl:when’ elem ’test’ attribútumának értékében, azaz ha létezik az ’xsd:complexContent’ elem (és persze alatta az ’xsd:extension’), akkor generalizált osztályt reprezentáló típusról van szó. Ekkor először az adott osztály attribútumait reprezentáló elemeket hozzuk létre. Először előállítjuk az osztály példányát reprezentáló XML részfa gyökérelemét, melynek neve megegyezik az osztály nevével, majd beállítjuk az ’id’ attribútumát a createForm.php-től kapott ’id’ paraméter értékére. Ezzel azonosítottuk a majdan előálló példányunkat. Ezután meghívjuk a ’createModelNodes’ nevű template-et, mely paraméterként egy csomópont-halmazt (node-set) vár, és előállítja az adott csomópontok mindegyikére a megfelelő példány-elemet. Paraméterként átadjuk neki az xsd:complexContent/xsd:extension/xsd:sequence által tárolt xsd:element nevű csomópontokat. A sémában ugyanis ezek reprezentálják az osztály saját attribútumait. Ha kész vagyunk és előálltak az adott osztály attribútumait tároló elemek, meghívjuk a ’model’ nevű template-et, mely egy típust vár paraméterként, amely típushoz tartozó elemeket kívánjuk létrehozni. Paraméterkény átadjuk neki a szülőosztályt reprezentáló típus nevét. Ez az egyszerű template a következőképp néz ki:

```
<xsl:template name="model">
    <xsl:param name="type"/>
    <xsl:for-each select="../xsd:complexType">
       <xsl:if test="@name=$type">
          <xsl:call-template name="createModelNodes">
            <xsl:with-param name="node" select="xsd:sequence/xsd:element"/>
            <xsl:with-param name="type" select="$type"/>
          </xsl:call-template>
       </xsl:if>
    </xsl:for-each>
</xsl:template>
```

Azaz csak annyit csinál, hogy végigkeresi a komplex típusokat, és ha megtalálja a szülőosztályt reprezentáló típust, meghívja a fent említett ’createModelNodes’ template-et paraméterül adva a típushoz tartozó elemek csomópont-halmazát. Ezzel előállt az adott osztály attribútumait reprezentáló XML-részfa a példányban, így lezárhatjuk az osztály nevével ellátott elemet.

Most jön az osztályhoz tartozó kétirányú asszociációkat, illetve asszociációs osztályokat reprezentáló elemek létrehozása. Ez a folyamat nagyban hasonlít a fent leírtakhoz,

csak a különbségeket emelném ki. Az associationForModel template-et hívjuk meg mindkétszer (egyszer a szülő, egyszer a gyerek osztályt reprezentáló típussal), máshogy paraméterezve. Először a gyermekosztály nevét, másodszor pedig a szülőosztály nevét kapja meg, és előállítja azon asszociáció típusokat tároló elemeket, melyek egyik résztvevője az adott osztály. Ezt névegyezőség alapján vizsgálja. Ha nincs az adott osztálynak szülőosztálya, akkor is a fenti folyamatok futnak le, azzal a különbséggel, hogy mivel nincs szülőosztály nem kell kétszer meghívni az elemeket előállító template-eket.

Amikor viszont már létező adatokat szeretnénk módosítani, a modellünkbe be kell töltenünk ezen adatokat. Hogy ezen adatokat hogyan kapjuk meg technikailag az még kérdéses, viszont biztos, hogy XML formátumban érkeznek. Ezért a kód akkor is működőképes lesz ezen adatokra, amennyiben az interface ki lesz cserélve alatta. Jelenleg megadott helyen levő XML fájlokkal dolgozik. Tehát mint azt korábban már láthattuk, módosítás esetén a ’referenceModel’ template hívódik meg:

```
<xsl:template name="referenceModel">
  <xsl:param name="type"/>
  <xsl:for-each select="document(concat('../data/',$type,'.xml'),
root()/child::node())">
    <xsl:for-each select="child::node()">
      <xsl:element name="{name()}">
        <xsl:for-each select="node()[@id=$id]">
          <xsl:if test="name()">
            <xsl:element name="{name()}">
              <xsl:copy-of select="@*"/>
              <xsl:for-each select="node()">
              <xsl:if test="name()">
               <xsl:element name="{name()}">
                 <xsl:copy-of select="@*"/>
                 <xsl:value-of select="text()"/>
               </xsl:element>
              </xsl:if>
             </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:for-each>
        <xsl:choose>
        <xsl:when test="$schema/xsd:complexType[@name=concat($type,
'Type')]/xsd:complexContent">
          <xsl:call-template name="associationsReference">
            <xsl:with-param name="type" select="$type"/>
            <xsl:with-param name="firstrun" select="true()"/>
          </xsl:call-template>
          <xsl:call-template name="associationsReference">
            <xsl:with-param name="type" select="substring-
before($schema/xsd:complexType[@name=concat($type,
'Type')]/xsd:complexContent/xsd:extension/@base, 'Type')"/>
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
```

```
          <xsl:call-template name="associationsReference">
             <xsl:with-param name="type" select="$type"/>
          </xsl:call-template>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:element>
   </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

Ez a template először megnyitja az adott típusú elemeket tároló fájlt. Ehhez egy egyszerű elnevezési konvenciót használtam, az adott osztály (melyet ezen típussal reprezentáltunk) neve megegyezik a file nevével. A megnyitás a document() XSLT függvénnyel történik, mely egy fájlnevet, és opcionálisan egy csomópont nevet vár paraméternek. A gyökérelem nevével létrehozunk egy új elemet, majd a gyermekelemeit vizsgálva keressük azt, amelyiknek az 'id' attribútuma megegyezik a createForm.php-tól kapott 'id' paraméterrel. Ha megvan, akkor lemásoljuk az elemet, és minden gyermekelemét illetve ezek attribútumait és értékeit. Ezután létre kell hoznunk az asszociációs adatokat hordozó elemeket. Itt viszont felmerül egy probléma. Ha átnézem az asszociációkat tartalmazó XML fájlt, és kiválasztok minden asszociációt, amiben részt vesz az adott objektumot reprezentáló elem, akkor lehetséges, hogy ha korábban nem vett részt egy bizonyos típusú asszociációban, akkor ehhez a modell elem nem jön létre, ily módon, mivel a form elem kötése üres lesz, ki se tudunk választani ilyen típusú asszociációt. Ezt a következőképpen oldottam meg: először megint meg kell vizsgálnunk, hogy nem származtatott osztályról van-e szó. Amennyiben igen kétszer kell majd meghívni az 'associationsReference' template-et először a gyermekosztály reprezentáló típussal, majd a szülőosztályt reprezentáló típussal. Mivel az, hogy ez az adott osztály származtatott-e, az XML fájlban már nincs letárolva, ezért a sémában ellenőrzöm, majd az eredménytől függően meghívom az 'associationsReference' template-et. Ez a template ellenőrzi, hogy milyen asszociációban vehet részt az adott típusú osztály és, hogy van-e olyan csomópont az Associations.xml fájlban, melynek típusa megegyezik az éppen vizsgált asszociáció típussal, és amely egyik gyermekelemének az értéke megegyezik a módosítandó elem id-jával. Erre írtam egy XSLT függvényt, mely egy csomópontot, egy id-t, és egy asszociáció nevet vár paraméterként, és attól függően, hogy talál-e a csomópont gyermekelemeit bejárva ilyen nevű asszociációt, melynek egyik gyermekelemének értéke az adott id, akkor igazat, ellenkező esetben hamisat ad vissza. Ha van ilyen csomópont, akkor

meghívja a ’searchAssociations’ template-et, mely az Associations.xml-ben megkeresi az adott csomópontot és bemásolja az értékeit. Ha nincs, akkor pedig a sémában keressük az adott típusú asszociációt reprezentáló elemet, és előállítja belőle a megfelelő példány-elemet. Ily módon biztosítottuk, hogy akár részt vett az adott elem egy bizonyos típusú asszociációban, akár nem, a példány-elem mindenképpen előáll.

Ezzel a modellünk készen is van, mind az új adatok beviteléhez, mind pedig meglevő adatok betöltéséhez működik. Nekikezdhetünk tehát a felhasználói felület és az események vezérlését szolgáló elemek előállításához.

2.2.4 A felhasználói felület előállítása

Mint a 2.2.2 –es pontban láthattuk a felhasználói felület elemeit a ’form’-módban futtatott template végzi. Mivel ez a template nagyban hasonlít a ’mode’-módban futtatott template-re, ezért ismertetésétől eltekintek, helyette inkább az eltérésekre fordítanék nagyobb figyelmet.

Míg a példányelemek előállítása viszonylag egyszerűen zajlik a séma által definiált elemnevekből (ezért is nem részleteztem ott), a felhasználói felület elemei nem ilyen egyszerűen állnak elő. Az osztály attribútumait reprezentáló elemekhez a felületelemek előállítását a ’createFormNodes’ template végzi:

```
<xsl:template name="createFormNodes">
  <xsl:param name="node"/>
  <xsl:param name="type"/>
  <xsl:element name="table" namespace="http://www.w3.org/1999/xhtml">
    <xsl:for-each select="$node">
      <xsl:for-each select="xsd:sequence/xsd:element">
        <xsl:element name="tr" namespace="http://www.w3.org/1999/xhtml">
         <xsl:element name="td" namespace="http://www.w3.org/1999/xhtml">
            <xsl:element name="xforms:label">
              <xsl:value-of select="@name"/>
            </xsl:element>
         </xsl:element>
         <xsl:choose>
         <xsl:when test="fn:isComplex(concat(@default,'Type'))=true()">
              <xsl:element name="xforms:select">
                <xsl:attribute name="ref">
                  <xsl:value-of select="$class"/>/<xsl:value-of
select="@name"/>
                </xsl:attribute>
                <xsl:attribute name="model">
                  <xsl:value-of select="$class"/>
                </xsl:attribute>
                <xsl:if test="@type">
                 <xforms:action ev:event="xforms-invalid">
                   <xforms:message level="modal">
                        hibás típus<br/> várt:
```

```
                        <xsl:value-of select="@type"/>
                      </xforms:message>
                    </xforms:action>
                  </xsl:if>
                  <xsl:call-template name="reference">
                    <xsl:with-param name="type">
                      <xsl:value-of select="@default"/>
                    </xsl:with-param>
                  </xsl:call-template>
              </xsl:element>
              </xsl:when>
              <xsl:otherwise>
               <xsl:element name="xforms:input">
                 <xsl:attribute name="model">
                   <xsl:value-of select="$class"/>
                 </xsl:attribute>
                 <xsl:attribute name="ref">
                  <xsl:value-of select="$class"/>/<xsl:value-of
select="@name"/>
                 </xsl:attribute>
                 <xsl:if test="@type">
                  <xforms:action ev:event="xforms-invalid">
                   <xforms:message level="modal">
                     hibás típus<br/> várt:
                     <xsl:value-of select="@type"/>
                   </xforms:message>
                  </xforms:action>
                </xsl:if>
              </xsl:element>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:element>
       </xsl:for-each>
      </xsl:for-each>
     </xsl:element>
    </xsl:template>
```

A template futása úgy indul, hogy létrehoz egy HTML táblázatot a felületelemek tárolására. Minden összetartozó felirat és beviteli mező egy-egy sorban, úgy hogy a feliratok és a beviteli mezők külön sorban vannak. Ez csak a megjelenés miatt fontos, illetve hogy esetlegesen megkönnyítsük a designer dolgát. Végigjárjuk a paraméterül kapott csomópont (mely a sémában az adott osztályt leíró típus csomópontja) minden gyermekelemét és első lépésben létrehozunk egy címkét (xforms:label) mely az adott elem nevét tárolja, ezt fogja a felhasználó is látni az oldalon. Ezután megvizsgáljuk, hogy az adott elem típusa egyszerű vagy összetett. Ha egyszerű, akkor létrehozunk egy ’xforms:input’ elemet, beállítjuk a kötést (a ref attribútum értékét), mely egy XPath kifejezéssel megadja, hogy a mezőbe beírt értéket mely példány-elemmel kötjük össze. Mint már említettem ez a kötés kötelező, mivel a küldésnél ennek a példány elemnek az értéke lesz a begépelt adat, illetve a példány-elem

típusából tudja az XForms processzor, hogy milyen mezőt rajzoljon ki, illetve mi alapján ellenőrizze a begépelt adat helyességét. Amennyiben nem helyes adatot visz be a felhasználó, kiváltódik az xforms-invalid esemény. Erre az eseményre figyel a következőként létrejövő elem az xforms:action. Ez az 'action' aktiválódik az 'xforms-invalid' esemény hatására és feldob egy 'xforms:message' ablakot, mely közli a felhasználóval, hogy hibás típusú adatot vitt be, illetve, hogy mi az elvárt.

Ha viszont komplex típusú az adott elem, akkor egy 'xforms:select' elem (vagy xforms:select1 de ennek generálásának bemutatását kihagyom, mivel csak a nevükben van eltérés) jön létre. Ez az elem egy adott listából történő választás lehetőségét adja meg. A fentiekhez hasonlóan beállítódik a kötés és a listaelemek létrehozásához meghívjuk a 'reference' template-et:

```
 <xsl:template name="reference">
        <xsl:param name="type"/>
        <xsl:element name="xforms:item">
            <xsl:element name="xforms:label">
                   <xsl:value-of select="'nincs'"/>
            </xsl:element>
            <xsl:element name="xforms:value"/>
       </xsl:element>
       <xsl:for-each select="document(concat('../data/',$type,'.xml'),
root()/child::node())">
          <xsl:for-each select="child::node()/child::node()[/@id]">
                 <xsl:element name="xforms:item">
                     <xsl:element name="xforms:label">
                         <xsl:choose>
                             <xsl:when test="exists(./nev) ">
                                 <xsl:value-of select="./nev"/>
                             </xsl:when>
                             <xsl:otherwise>
                                 <xsl:value-of select="@id"/>
                             </xsl:otherwise>
                         </xsl:choose>
                     </xsl:element>
                     <xsl:element name="xforms:value">
                          <xsl:value-of select="@id"/>
                     </xsl:element>
                 </xsl:element>
          </xsl:for-each>
       </xsl:for-each>
</xsl:template>
```

Mint a későbbiekben látni fogjuk ez a template ki van egészítve az asszociációk bizonyos funkcióinak kezeléséhez szükséges utasításokkal, ezeket most kihagytam, majd ott ismertetem. A template azzal kezdi működését, hogy létrehoz egy üres listaelemet, „nincs" – felirattal, hogy módosítás esetén üresre legyen állítható az értéke. Majd megnyitja a típus nevével ellátott XML fájlt és a benne talált elemekhez mindhez létrehoz egy 'xforms:item'

elemet, melynek gyermekelemei egy ’xforms:label’ és egy ’xforms:value’. A címke értéke, ha van ’név’ nevű elem, akkor annak értékére, más esetben az ’id’ attribútum értékére állítódik be. A ’value’ elem értéke minden esetben az ’id’ attribútum értéke lesz. Ezzel létrejöttek az osztály attribútumaihoz rendelt felületelemek.

Az asszociációkat reprezentáló elemek beállításához szükséges elemeket létrehozó template az ’associationForForm’ nevet viseli. Mivel ez a template igen hosszú (majdnem 300 sor), ezért magát a kódot nem illesztem be, csak működését ismertetem. A template bejárja a séma komplex típust leíró elemeit, és ha az típus nevében megtalálja az ’Association’ sztringet, illetve a típus által definiált gyermekelemek nevei közt az egyik a paraméterként kapott nevet viseli, akkor tudjuk, hogy az adott típusú elem részt vehet ebben a típusú asszociációban, így lehetőséget kell biztosítanunk a felhasználónak, hogy beállíthassa ezt a kapcsolatot. Ahhoz, hogy kezelhető legyen az asszociációban résztvevő elemek számossága létrejön egy ’xforms:repeat’ nevű elem. Ez némi magyarázatra szorul. Mint azt a séma generálását leíró részben (2.1) írtam, úgy oldottam meg az asszociációk kérdését, hogy asszociáció típusokat definiáltam, melyek példányai tartalmaznak egy-egy asszociációt és a benne résztvevő elemeket leíró adatokat. Ez azt is jelenti, hogy amennyiben egy asszociációban az egyik típusú elemből egy, a másik típusú elemből pedig például három szerepelhet, akkor három asszociáció példány áll elő melyeknek egyik eleme (amelyik egyszer szerepelhet) ugyanazt az értéket hordozza, míg a másik különböző a három asszociáció példányban. Az ’xforms:repeat’ elem, mint azt az 1.5.-ös pontban leírtam, arra való, hogy adatcsoportot ismételjen. Ily módon, amennyiben a felhasználó több elemet kapcsolatba szeretne hozni az éppen szerkesztettel ehhez beilleszthet egy újabb beviteli mezőt.

A címke úgy áll elő, hogy ha van ’property’ az asszociáció azon végéhez mely az adott osztályba csatlakozik, akkor ezen ’property’ neve lesz a címke értéke, egyébként pedig az asszociációban résztvevő másik osztály neve. Ezután elkezdődhet a beviteli mező előállítása, mely minden esetben egy ’xforms:select1’ elem lesz, mely az egy elem kiválasztását teszi lehetővé. Ehhez meghívom a fent ismertetett ’reference’ template-et. A template futásában a következő kiegészítések találhatók meg: mivel az ’xforms:select1’ elem az asszociációban résztvevő másik elemet tartalmazó példány elemhez van kötve, ezért szükséges, hogy az aktuálisan szerkesztett objektumot reprezentáló elem ’id’-ját beállítsuk az asszociációban. Ehhez az ’xforms:setvalue’ elemet használom, mely az ’xforms-select’ esemény hatására

aktivizálódik. A ’reference’ template paraméterben megkapja az asszociációt tároló elem nevét, és ennek segítségével beállítja az ’xforms:setvalue’ ref attribútumának értékét, mely megadja, hogy a példányban melyik elemnek szeretnénk értéket adni. A ’value’ érték, ami megadja, hogy mit szeretnénk értékül adni, pedig az ’id’ paraméter értéke lesz. Ily módon, ha a felhasználó kiválasztja az egyik listaelemet, az asszociációban résztvevő mindkét fél beállítódik. A másik kiegészítés pedig az asszociációs osztály típusú elemeket érinti. Ha ugyanis egy asszociációs osztályt reprezentáló típusú elemet szeretnénk megjeleníteni, akkor az asszociáció kiválasztásának lehetőségének megadásán túl meg kell jelenítenünk az asszociációs osztály attribútumait képviselő elemekhez a beviteli mezőket. Ezt oly módon oldottam meg, hogy ezen elemekhez tartozó beviteli mezők csak akkor jelennek meg, ha a felhasználó kiválasztja a kapcsolatban résztvevő másik elemet. Tehát ezen adatok megadására csak akkor van lehetőség, ha maga az asszociáció létrejött. Ahhoz hogy amennyiben lehetőség van rá, több ilyen asszociációt is létrehozzunk szükségünk lesz két gombra, melyekkel hozzáadni, illetve elvenni lehet ilyen típusú elemeket. Ezért létrehozunk két ’xforms:trigger’-t egyiket a +, másikat a – jellel feliratozzuk. A hozzáadást az ’xforms:insert’ elemmel végezhetjük. Meg kell adni a nodeset attribútum értéket, mely megadja, hogy melyik csomópontot szeretnénk beszúrni, itt beállítódik a korábban változóba mentett, az aktuálisan feldolgozásra kerülő asszociáció neve. Beállíthatunk egy ’if’ attribútumot is, az xforms processzor csak ezen attribútum értékétől függően végzi el a beillesztést. Ezzel tudjuk az asszociáció számosságát kezelni. Ha például az olyan típusú elem, melyhez a most előálló formot készítjük egyszer, míg az asszociációban résztvevő másik típusú elem pedig háromszor szerepelhet az adott asszociációban, akkor az ilyen típusú asszociációt reprezentáló elem legfeljebb háromszor állhat elő. Ezt oly módon lehet kezelni, hogy az ’xforms:trigger’ if attribútumának értékét ’count(asszociációt reprezentáló példányelem < 3)’ –ra állítjuk. Ily módon, ha már van három ilyen típusú asszociációt reprezentáló elemünk, akkor a gomb lenyomására nem enged többet hozzáadni. A törlést az ’xforms:delete’ elemmel lehet megoldani. Ez az elem törli a legutolsó asszociációt reprezentáló elemet. Ennek is megadtam egy ’if’ attribútumot, mivel az XForms processzor akkor is engedné törölni, ha csak egy ilyen elem van, így ha mindet törölte a felhasználó, de közben rájön, hogy egyet mégiscsak szeretne beállítani, akkor hiába kattintana a hozzáadás gombra, nincs mit hozzáadni. Ezért a törlést oly módon korlátoztam, hogy csak addig lehet törölni, míg legalább két elem van, ha már csak egy maradt, azt nem lehet törölni.

Ezzel előállt az XForms alapú felületünk összes eleme, egy önmagában működő kód jött létre, melyet ha szükséges természetesen lehet módosítani. Ezzel a kódgenerálás magyarázatának végéhez értem, most lássunk egy konkrét esetben előálló XForms kódot!

2.3 Példa a végeredményre

Ebben a fejezetben egy konkrét példán keresztül szeretném bemutatni az előálló XHTML-XForms kódot mely a böngészőben a 4. ábrán ábrázolt módon jelenik meg. A transzformációs fájlt a következő paraméterekkel hívtam meg:

- type : Szemely
- id : 2cb54c91885faa7a436a55a0b1a60b0a (melyet a createForm.php generált)

Az előálló kód pedig:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns:xforms="http://www.w3.org/2002/xforms"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
   <xforms:model schema="http://127.0.0.1/project/uml2xsd/schema.xsd"
id="Szemely">
    <xforms:instance>
     <XPhD>
       <Szemely id="2cb54c91885faa7a436a55a0b1a60b0a">
           <nev></nev>
           <szuletesi_datum xsi:type="xsd:date"></szuletesi_datum>
           <allampolgarsaga></allampolgarsaga>
           <email></email>
       </Szemely>
       <AssociationU007054c1-b4ee-4733-9436-3c3538d948c2 xmlns="">
          <Szemely></Szemely>
          <Publikacio></Publikacio>
       </AssociationU007054c1-b4ee-4733-9436-3c3538d948c2>
       <AssociationClassU4e641710-76ad-4836-861d-6253b3d4d0a1 xmlns="">
          <Property1 default="TudomanyosFokozat" xsi:type="id"></Property1>
          <Szemely></Szemely>
          <megszerzes_datumaClassVariable xsi:type="xsd:date"/>
       </AssociationClassU4e641710-76ad-4836-861d-6253b3d4d0a1>
      </XPhD>
     </xforms:instance>
     <xforms:submission id="submit"
action="http://127.0.0.1/project/navigation/show.php?type=Szemely"
                        method="post">
        <xforms:action ev:event="xforms-submit-error">
          <xforms:message level="modal">Hiba a küldéskor!</xforms:message>
        </xforms:action>
     </xforms:submission>
   </xforms:model>
 </head>
 <body>
```

```
   <h1>Szemely</h1>
   <xforms:group>
       <xforms:label>nev</xforms:label>
       <xforms:input model="Szemely" ref="Szemely/nev"/>
       <xforms:label>szuletesi_datum</xforms:label>
       <xforms:input model="Szemely" ref="Szemely/szuletesi_datum">
         <xforms:action ev:event="xforms-invalid">
        <xforms:message level="modal">hibás típus<br/>várt:xsd:date
          </xforms:message>
         </xforms:action>
       </xforms:input>
       <xforms:label>allampolgarsaga</xforms:label>
       <xforms:input model="Szemely" ref="Szemely/allampolgarsaga"/>
       <xforms:label>email</xforms:label>
       <xforms:input model="Szemely" ref="Szemely/email"/>
  </xforms:group>
  <xforms:repeat nodeset="AssociationU007054c1-b4ee-4733-9436-3c3538d948c2"
                    id="AssociationU007054c1-b4ee-4733-9436-3c3538d948c2">
       <xforms:label>Publikacio</xforms:label>
       <xforms:select1 ref="Publikacio" model="Szemely">
            <xforms:item>
               <xforms:label>nincs</xforms:label>
               <xforms:value></xforms:value>
            </xforms:item>
            <xforms:item>
               <xforms:label>publikacio1</xforms:label>
               <xforms:value>publikacio1</xforms:value>
               <xforms:action ev:event="xforms-select">
                 <xforms:setvalue model="Szemely"
ref="../../AssociationU007054c1-b4ee-4733-9436-3c3538d948c2/Szemely"
value="'2cb54c91885faa7a436a55a0b1a60b0a'"/>
             </xforms:action>
            </xforms:item>
      </xforms:select1>
  </xforms:repeat>
  <xforms:trigger>
     <xforms:label>+</xforms:label>
     <xforms:action ev:event="DOMActivate">
        <xforms:insert nodeset="AssociationU007054c1-b4ee-4733-9436-
3c3538d948c2" position="after"/>
     </xforms:action>
  </xforms:trigger>
  <xforms:trigger>
     <xforms:label>-</xforms:label>
       <xforms:action ev:event="DOMActivate">
         <xforms:delete nodeset="AssociationU007054c1-b4ee-4733-9436-
3c3538d948c2" at="last()" if="count(//AssociationU007054c1-b4ee-4733-9436-
3c3538d948c2) &gt; 1"/>
       </xforms:action>
  </xforms:trigger>
<xforms:repeat nodeset="AssociationClassU4e641710-76ad-4836-861d-
6253b3d4d0a1" id="AssociationClassU4e641710-76ad-4836-861d-6253b3d4d0a1">
   <xforms:label>Property1</xforms:label>
     <xforms:select1 ref="Property1" model="Szemely">
        <xforms:item>
          <xforms:label>nincs</xforms:label>
          <xforms:value></xforms:value>
          <xforms:action ev:event="xforms-select">
```

```
            <xforms:toggle case="AssociationClassU4e641710-76ad-4836-861d-
6253b3d4d0a1notSelected"/>
        </xforms:action>
      </xforms:item>
      <xforms:item>
        <xforms:label>Doktor</xforms:label>
      <xforms:value>7023c6dd3d08094bee4f2850d443a3aa</xforms:value>
        <xforms:action ev:event="xforms-select">
          <xforms:setvalue model="Szemely"
ref="../../AssociationClassU4e641710-76ad-4836-861d-6253b3d4d0a1/Szemely"
value="'2cb54c91885faa7a436a55a0b1a60b0a'"/>
      </xforms:action>
        <xforms:action ev:event="xforms-select">
         <xforms:toggle case="AssociationClassU4e641710-76ad-4836-861d-
6253b3d4d0a1selected"/>
        </xforms:action>
     </xforms:item>
   </xforms:select1>
   <xforms:switch>
     <xforms:case id="AssociationClassU4e641710-76ad-4836-861d-
6253b3d4d0a1notSelected"/>
     <xforms:case selected="true" id="AssociationClassU4e641710-76ad-4836-
861d-6253b3d4d0a1selected">
        <xforms:input ref="megszerzes_datumaClassVariable">
           <xforms:label>megszerzes_datuma</xforms:label>
        </xforms:input>
     </xforms:case>
   </xforms:switch>
  </xforms:repeat>
  <xforms:trigger>
    <xforms:label>+</xforms:label>
    <xforms:action ev:event="DOMActivate">
     <xforms:insert nodeset="AssociationClassU4e641710-76ad-4836-861d-
6253b3d4d0a1" position="after"/>
    </xforms:action>
  </xforms:trigger>
  <xforms:trigger>
   <xforms:label>-</xforms:label>
   <xforms:action ev:event="DOMActivate">
     <xforms:delete nodeset="AssociationClassU4e641710-76ad-4836-861d-
6253b3d4d0a1" at="last()"if="count(//AssociationClassU4e641710-76ad-4836-
861d-6253b3d4d0a1) &gt; 1"/>
   </xforms:action>
  </xforms:trigger>
  <xforms:submit submission="submit">
  <xforms:label>Mentés</xforms:label></xforms:submit>
 </body></html>
```

A fenti kód egy személy típusú elem előállításához alkalmas. Sok, a szemléltetéshez fölösleges kódot kihagytam, így csak a típus által definiált elemek, egy asszociációt reprezentáló elem, és egy asszociációs osztály példányát reprezentáló elem előállítására alkalmas kódot szemléltetem.

A kezdő ’html’ tag attribútumai határozzák meg az oldalon használt névtereket. Ezek fontosságáról, szerepéről már írtam az 1.1-es pontban. Itt látható az is, hogy az alapértelmezett (default) névtér az XHTML névtér. A ’head’ elem megnyitása és az oldalon

található tartalom típusának megadása (’meta’ tag) után láthatjuk az ’xforms:model’ elem nyitó tag-jét. Attribútumai, az ’id’ a modell azonosítására, a ’schema’ pedig a sémainformációk betöltésének a segítésére szolgál. A modell elemben közvetlenül láthatjuk a példány elemek, az ’xforms:instance’ megadását. Az ’XPhD’ nevezetű elem lesz a gyökérelem, az alatta a ’Szemely’ nevű tag-ek által közbezárt elemek pedig a ’SzemelyType’ típus által definiált elemek egy-egy példánya lesz. A ’Szemely’ elem lezárása után látható egy asszociáció típus egy példányát, illetve egy asszociációs osztály típus egy példányát jelentő elemek. Ezek a következők:

- személy – publikáció: ki lehet választani a most szerkesztett személy által írt publikáció(ka)t
- személy-tudományos fokozat: ki lehet választani az adott személy tudományos fokozatát, kiegészítő információként a kapcsolathoz pedig megadható a megszerzés dátuma

Ezzel be is fejeztük példányunk áttekintését, következik az ’xforms:submission’ elem, mely a küldés helyét (az ’action’ attribútumban) és módját (a ’method’ attribútumban) határozza meg, illetve tartalmaz minimális hibakezelést is, amennyiben tájékoztatja a felhasználót a küldés esetleges sikertelenségéről. Ezt valósítja meg az ’xforms:action’ elemben található ’xforms:message’ elem. Ezzel vége a modellnek lezárjuk az ’xforms:model’ és a ’head’ elemeket.

A ’body’ elemben találhatók a felületet leíró elemek. Először kiíratjuk a böngészővel, hogy milyen típusú elemet szerkeszt a felhasználó, ezt valósítja meg a ’h1’ tag-ekkel határolt ’Szemely’ szöveg. Ezután következnek a személy típusú elem gyermekelemeihez rendelt beviteli mezők. Látható, hogy minden gyemekelemhez egy ’xforms:label’, mely a mező megnevezését tartalmazza és egy ’xforms:input’, mely a bevitelt lehetővé tevő elem állt elő. Az ’xforms:input’ mezőbe beírt értékek fogják megadni a típushoz tartozó ’nev’, ’szuletesi_datum’, és ’allampolgarsaga’ gyermekelemek értékeit. Látható, hogy mivel a ’szuletesi_datum’ elemnek van típusa (’xsd:date’) ezért, amennyiben a felhasználó helytelen adatot visz be a már ismertetett ’xforms:message’ elem figyelmezteti erről.

Ezt követik az asszociációkat leíró elemekhez rendelt beviteli mezők. Mint írtam a 2.2.4 –es pontban ezek egy-egy az ismétlést lehetővé tevő ’xforms:repeat’ elemben találhatók. Az ’xforms:repeat’ elem nodeset értéke az asszociáció típusának modellbeli példányához van kötve. Az elemben mindkét esetben egy ’xforms:select1’ elem található, mely a kiválasztást

teszi lehetővé. Az ’xforms:item’ tag-ek által közrezárt elemek, az ’xforms:label’ és az ’xforms:value’, rendre a megjelenítendő nevet és a kiválasztani kívánt elem értékét tartalmazzák. Mindkét esetben minden listaelemhez tartozik egy ’xforms:setvalue’ elem is mely, mint azt a 2.2.4-es pontban tárgyaltam, az asszociáció másik végét állítja az aktuálisan szerkesztett elem id-jára. A második esetben, asszociációs osztály típusú elemről lévén szó található egy ’xforms:toggle’ nevű elem is, mely az adott listaelem kiválasztására reagálva aktiválja a ’case’ attribútumban meghatározott ’xforms:case’ megfelelő id-val rendelkező elemét. Ez, mint az lentebb látható egy ’xforms:switch’ nevű elemben található, és a tartalma pedig egy beviteli mező, amely az asszociációhoz rendelt plusz információ megadását teszi lehetővé. Jelen esetben ez a ’megszerzes_datuma’ elemhez rendelt beviteli mező. Mindkét ’xforms:repeat’ elem után megtalálható a hozzájuk rendelt két-két ’xforms:trigger’ elem, melyek a hozzáadást és a törlést valósítják meg a 2.2.4-es pontban leírtak szerint.

Végezetül megtalálhatjuk a küldést lehetővé tevő ’Mentés’ feliratú gombot kirajzoló ’xforms:submit’ elemet, melynek megadjuk, hogy mely ’xforms:submission’ elemet aktiválja. Mivel csak egy ilyen elemünk van, annak id-ját adjuk meg az erre szolgáló ’submission’ attribútum értékeként. Az ’Oktato’ típushoz előálló képernyőképet jelenítem meg a 5. ábrán.

Összegzés

A tárgyalt projekt célja olyan webes információs rendszer fejlesztése, melynek programkódjait a tervezéskor létrejövő modellekből generáljuk. Ezzel gyakorlatilag az MDA (Model Driven Architecture) alapelveit alkalmazzuk.

Az én feladatom volt a felületeket generáló kódok megírása, melyet úgy érzem sikerrel tejlesítettem, hiszen a generálás után működő felhasználói felületek állnak elő.

A használt technológiák egy része még ugyan újnak számít és nincs meg hozzá a kellő támogatás, de remélem idővel kialakul ezen technológiák teljes körű támogatása, ily módon még egyszerűbbé, és gyorsabbá téve a program működését. Sokat nyertem ezen technológiák megismerésével, hiszen ezeket más projektekben, melyeken dolgoztam is sikerrel tudtam alkalmazni. Például az egyik cégnél szükség volt arra, hogy programokhoz a ’Help and Manual’ program segítségével írt leírásokat, a ’JavaHelp’ program által megjeleníthető formátumba alakítsuk. Mivel mindkét program támogatta az XML formátumot, ezért írtam egy XSLT transzformációt, mely ezt az átalakítást elvégzi, így hosszú és fáradtságos JAVA kódolástól mentettem meg ezen dolgozó kollegáimat. Egy másik projekt, melyen jelenleg is dolgozom célja az ügyvédek számára olyan rendszer létrehozása, mellyel egyszerűen tudnak egyszerűsített társasági szerződéseket létrehozni elektronikus, nyomtatható formában. Mivel egy-egy ilyen szerződés adatainak belső struktúrája elég bonyolult, ezért az XForms technológiát használom az űrlapok létrehozásánál, sok kényelmi funkciót beépítve, hogy minél egyszerűbb és értelemszerűbb legyen a kitöltés. Mivel az XForms technológia még nincs támogatva, ezért egy külső ingyenes programot használok (AjaxForms), mely az XForms technológiát használva XHTML nyelven írt forráskódjaimat, HTML és javascript alapú kódokká konvertálja. Így tudom használni az XForms nyújtotta előnyöket, és a böngészők támogatását is élvezem.

Mindezekért szeretném megköszönni Adamkó Attila tanár úrnak, hogy figyelmemet ezen technológiák felé fordította, illetve mindazon segítségért, melyet ezen technológiák elsajátításáért nyújtott. Szeretném még megköszönni projektvezetőimnek is, hogy mikor ötleteimmel megkerestem őket, nyitottak voltak ezen technológiák használatára.

Remélem, hogy sikerült felkeltenem az olvasó érdeklődését és amennyiben még nem ismerné behatóbban ezen technológiákat, de szeretné őket megismerni, ajánlom figyelmébe a felhasznált irodalom jegyzékét.

Felhasznált irodalom

Erik T. Ray (2001): *Learning XML*. Sebastopol, O'Reilly Media, Inc.

Michael Kay (2004): *XSLT 2.0 Programmer's Reference, Third Edition*. Indianapolis, Wiley Publishing Inc.

Ken Holman (2004): *Practical Transformation Using XSLT and Xpath.* Cambrige, MIT Press

Doug Tiduell (2001): *Mastering XSLT Transformations.* Sebastopol, O'Reilly Media, Inc.

Micah Dubinko (2003): *XForms Essentials*. Sebastopol, O'Reilly Media, Inc.

T.V.Raman (2003): *XForms: XML Powered Web Forms.* Reading, Addison Wesley Professional

Eric van der Vlist (2002): *XML Schema.* Sebastopol, O'Reilly Media, Inc.

Peter Moulding (2002): *PHP haladóknak.* Budapest, Perfact-Pro Kft.

Soczó Zsolt (2001): *XMLgessünk! – cikksorozat*. Budapest, tech.net magazin

Adamkó Attila (2008): *Webes Információs Rendszerek Modellezése.*Debrecen, Informatika a felsőoktatásban 2008 (http://www.agr.unideb.hu/if2008/kiadvany/papers/C82.pdf)

Az internetről:

http://en.wikipedia.org/wiki/Xhtml

http://php.net/manual/en/

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

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

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

http://en.wikipedia.org/wiki/Xforms

http://www.mozilla.org/projects/xforms/

http://xformsinstitute.com/lesson1.php (itt csak az első oldalt adtam meg, a többi is elérhető)

Irodalomjegyzék

[IN.1] : http://sourceforge.net/projects/xslt2processor/

[IN.2] : http://www.w3.org/TR/xml/

[IN.3] : http://www.omg.org/docs/formal/07-12-01.pdf

[IN.4] : http://www.omg.org/docs/formal/06-01-01.pdf

[IN.5] : http://www.omg.org/docs/formal/07-11-04.pdf

[IN.6] : http://www.w3.org/TR/xslt/

[IN.7] : http://www.w3.org/TR/xpath

[IN.8] : http://www.w3.org/TR/xmlschema-0/ http://www.w3.org/TR/xmlschema-1/ http://www.w3.org/TR/xmlschema-2/

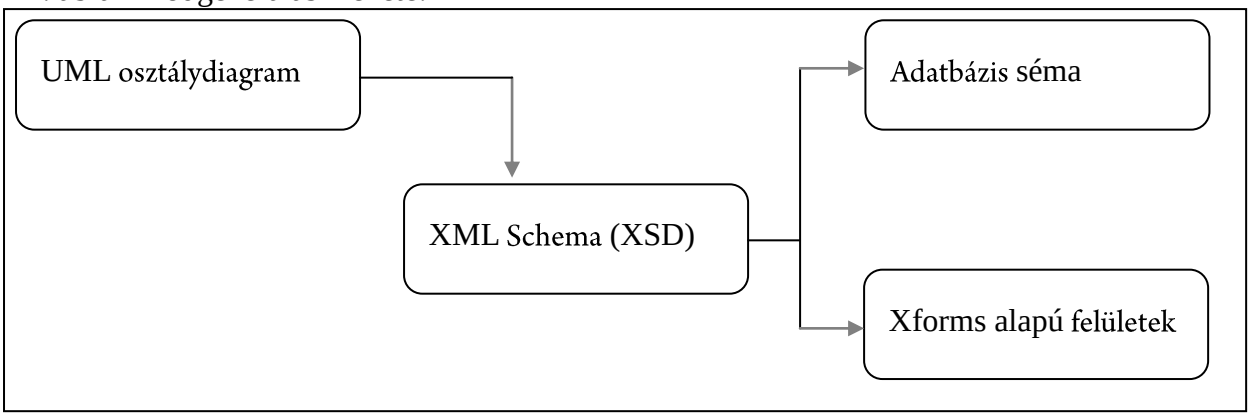
[IN.9] : http://www.w3.org/TR/xforms/

[IN.10]: http://www.w3.org/TR/xhtml1/

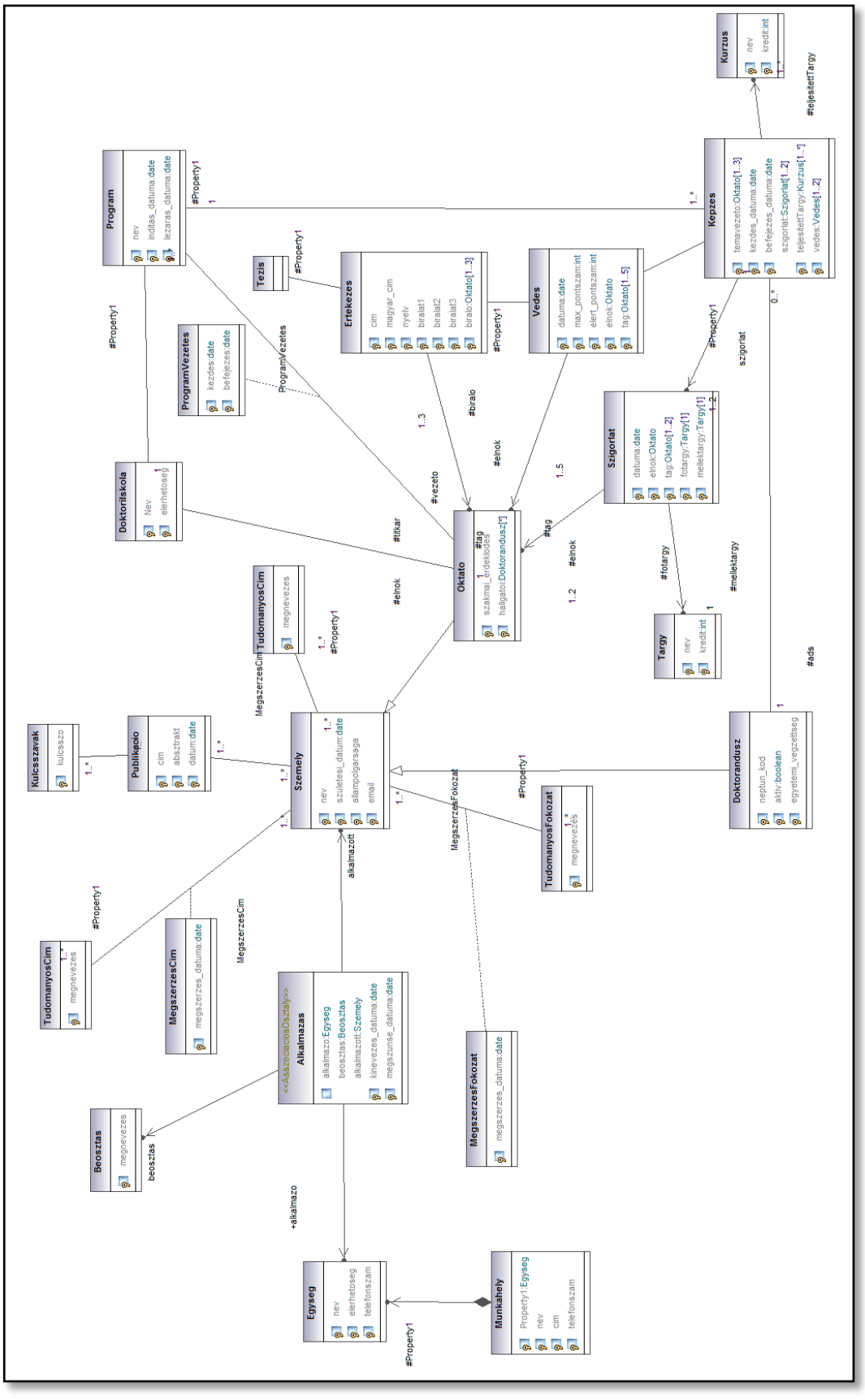
[IN.11]: http://www.w3.org/TR/xhtml2/

Függelék

1. ábra A kódgenerálás menete:

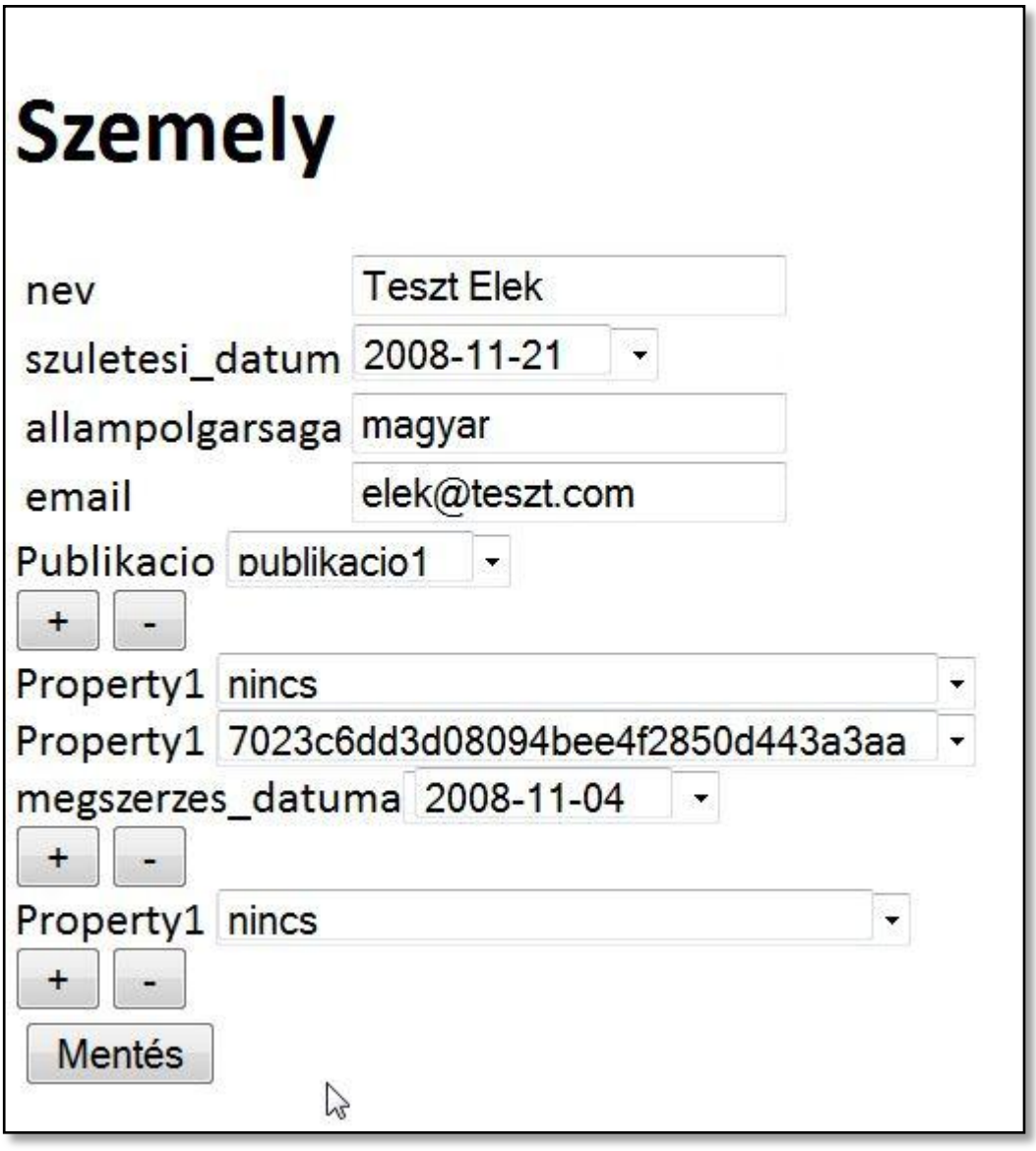


2. ábra Az UML diagram:



3. ábra A dátum kiválasztása:

4. ábra A2.3-as pontban bemutatott kód megjelenítése a böngészőben:



5. ábra A végeredményként előálló egyik form képernyőképe: