SZAKDOLGOZAT

Tóth Zoltán

Debrecen

2010

Debreceni Egyetem

Informatika Kar

WEBES INFORMÁCIÓS RENDSZEREK MODELLEZÉSE, FEJLESZTÉSE

Témavezető
Dr. Adamkó Attila
egyetemi adjunktus

Készítette
Tóth Zoltán
programtervező informatikus

Debrecen
2010

Webes információs rendszerek modellezése, fejlesztése

Tartalomjegyzék

1. Bevezetés....4

2. Általános programfejlesztés – életciklusok....5

2.1. A szoftverfolyamat....5
2.2. A szoftverfolyamat modelljei....5
2.2.1. A vízesés modell....6
2.2.2. Az evolúciós modell....6
2.2.3. Komponens alapú szoftvertervezés....7

3. Web Modeling Language....9

3.1. WebML dióhéjban....9
3.2. Adat intenzív webalkalmazások....10
3.3. Model, View, Controller....11

4. Adatmodell a WebML-ben....13

4.1. Az adatmodell....13
4.2. Entitás....13
4.3. Attribútum....14
4.4. Kapcsolat....14
4.5. Többértékű attribútum....16
4.6. Strukturált attribútum....16
4.7. Kapcsolat attribútumokkal és n-ed leges kapcsolat....17
4.8. IS-A hierarchia....17
4.9. Struktúra minták....18

5. Hypertext modell a WebML-ben....20

5.1. Hypertext modell....20
5.2. Alap tartalmi egységek....20
5.3. Hivatkozások....27
5.4. Lapok és tartományok....29
5.5. Oldal nézetek....31

6. A WebML egyéb részei....32

6.1. Műveleti egységek....32
6.2. Származtatás és személyre szabhatóság....32

7. WebML gyakorlatban: WebRatio....34

7.1. WebRatio....34

7.2. Használt technológiákról röviden....35
7.3. Az adatmodell....36
7.4. A logikai modell....42
7.5. Az oldal nézet alkalmazott eszközei....43
7.6. A modul nézet alkalmazott eszközei....52
7.7. A megjelenítési modell....57
8. Összefoglalás....58
9. Irodalomjegyzék....59

1. Bevezetés

A választott témám érdekesnek ígérkezett: Webes információs rendszerek modellezése és fejlesztése. Mindezt a világ legelső modell vezérelt fejlesztői környezetében, aminél a tervezésen van a hangsúly, a kódot pedig legenerálja. A feladatom egy doktori képzést nyilvántartó webalkalmazás írása volt, ennek követelményei az egyetem doktori szabályzatában vannak.

A webes alkalmazások összetett rendszerek, melyek különböző hardveres és szoftveres összetevőkre, protokollokra, nyelvekre, interfészekre és szabványokra épülnek. A Web Modeling Language (WebML) ezek modell vezérelt fejlesztése mellett kötelezte el magát. Célom a WebML világába betekintés nyerése és megismerése egy adat intenzív webalkalmazás fejlesztésén keresztül. Ehhez egy különleges fejlesztői környezetet a WebRatio-t használom, ami teljesen a WebML által lefektetett szabályokra épít.

Célom továbbá felmérni, hogy a fejlesztői környezet valóban segíti-e ezt a típusú fejlesztést, valóbaj gyorsítja-e, megfelelő opciókkal, eszközökkel rendelkezik-e, lefedi-e egy szoftver fejlesztésének teljes életciklusát, illetve az alkalmazás fejlesztése során használt eszközöket bemutatni. Megfelelő-e a WebML és a WebRatio egy valódi webalkalmazás létrehozására?

A dolgozat első fejezetében olvashatunk az általános szoftverfejlesztési gyakorlatokról az ismertebb életciklusokról. Az ezt követő fejezetben a WebML-ről egy bevezető leírást kapunk. Megtudjuk, hogyan és miért jött létre és hogy mik a fő tulajdonságai. A 4., 5. és 6. fejezetekben részletesen mutatom be a WebML elméleti alapjait. Megismerjük a web modellező nyelv felépítését és megtudjuk mi a strukturális, hypertext és megjelenítési modell az elkészített alkalmazásomból vett példák segítségével. A 7. fejezet a 4., 5. és 6. fejezet elméleti alapozása után teljesen a gyakorlatra helyezi a hangsúlyt. A Webratio-t mutatom be, azt, hogy hogyan és mivel végezhetjük a szoftverfejlesztést.

2. Általános programfejlesztés – életciklusok

2.1. A szoftverfolyamat

A szoftverfolyamat tevékenységek és kapcsolódó eredmények sora, amelyek egy szoftvertermék előállításához vezetnek. Ezek a szoftverfejlesztés kezdeteitől indulhatnak, de napjainkban gyakoribb eset, hogy az új szoftver fejlesztése meglévő rendszerek kiegészítése és módosítása, illetve kulcsrakész rendszerekkel történő integrálása és konfigurálása.

A szoftverfolyamatok összetettek és nagyban függenek emberi tényezőktől és döntésektől. Éppen ezért, mivel emberi beavatkozásokat és kreativitást igényelnek, a folyamatok automatizálására történő erőfeszítések csak korlátozottan arattak sikert.

Habár számos különböző szoftverfolyamat létezik, vannak alapvető tevékenységek, amelyek minden szoftverfolyamatban közösek:

1. Szoftverspecifikáció. A szoftver funkcióit, illetve annak megszorításait meg kell határozni.
2. Szoftvertervezés és implementáció. A specifikációnak megfelelő szoftvert elő kell állítani.
3. Szoftvervalidáció. A szoftvert validálni kell, hogy biztosítsuk, azt fejlesztettük ki, amit az ügyfél kívánt.
4. Szoftverevolúció. A szoftvert úgy kell alakítani, hogy megfeleljen a megrendelő kívánsága szerint történő változtatásoknak.

Ideális szoftverfolyamat nincs, de számos terület van, ahol a szervezeten belüli szoftverfolyamatokon javíthatunk.

2.2. A szoftverfolyamat modelljei

A szoftverfolyamat modellje a szoftverfolyamat absztrakt reprezentációja. Az általános modellek nem a szoftverfolyamat pontos, végleges leírásai, sokkalta inkább hasznos absztrakciók, amelyeket a szoftverfejlesztés különböző megközelítési

módjának megértéséhez használunk. Tekinthetjük őket akár folyamat-keretrendszereknek is, melyek kiegészíthetőek és adaptálhatók.

Jelen pillanatban a szoftvertervezői gyakorlatban három általános modell terjedt el széles körben. Nem kizárólagos a használatuk és gyakran keverednek is egymással, főként a nagy rendszerek fejlesztésekor.

2.2.1. A vízesés modell

Ez a folyamat alapvető tevékenységeit a folyamat különálló fázisaiként tekinti.

A szoftver életciklusa:

1. Követelmények elemzése és meghatározása.
2. Rendszer- és szoftvertervezés.
3. Implementáció és egységteszt.
4. Integráció és rendszerteszt.
5. Működtetés és karbantartás.

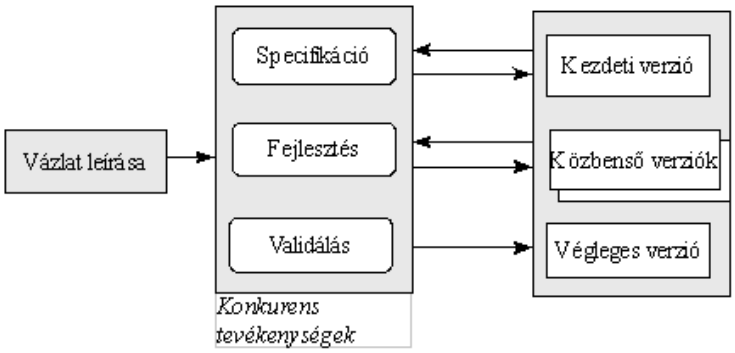
A fázisok eredményei egy vagy több dokumentum amit jóváhagytak. A következő fázis nem indulhat el addig, amíg az előző fázis be nem fejeződött, persze a gyakorlatban ezek a szakaszok átfedhetik egymást.

A vízesésmodell problémáját a projekt szakaszainak nem rugalmas részekké történő felosztása okozza. A folyamat korai szakaszaiban kell állást foglalnunk és elkötelezni magunkat. Az utólagos hibajavítás igen költséges lehet.

A vízesésmodell csak akkor használható jól, ha már előre, pontosan ismerjük a követelményeket.

2.2.2. Az evolúciós modell

Az evolúciós modell alapötlete az, hogy ki kell fejleszteni egy kezdeti implementációt, azt a felhasználókkal véleményeztetni, majd sok-sok verzión keresztül addig finomítani, amíg a megfelelő rendszert el nem értük. Érvényesíti a tevékenységek közti párhuzamosságot és a gyors visszacsatolásokat.



ábra 1: Evolúciós modell

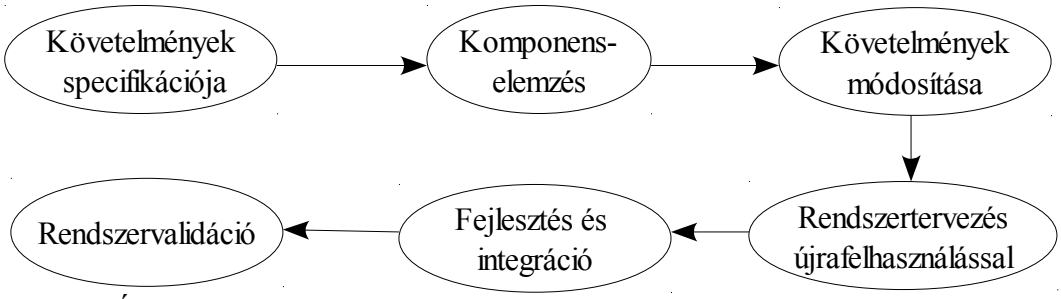
Két különböző típusa ismert:

1. Feltáró fejlesztés. A folyamat célja a megrendelővel közösen feltárni a követelményeket és kialakítani a végleges rendszert. A fejlesztés a rendszer már ismert részeivel kezdődik. A végleges rendszer úgy alakul ki, hogy egyre több, az ügyfél által kért új tulajdonságot társítunk a már meglévőkhöz.
2. Eldobható prototípus készítés. Célja az ügyfél céljainak lehető legjobb megértése, prototípusok segítségével jobban megismerjük azokat. A prototípusnak próbálgatós módszerrel az ügyfél által kevésbé meghatározott követelményekre kell összpontosítania.

Az ügyfél kívánságainak megfelelési szándékot tekintve hatékonyabb a vízesésmodellnél, továbbá nagy előnye, hogy a rendszer-specifikáció inkrementálisan fejleszthető.

2.2.3. Komponens alapú szoftvertervezés

A szoftverfolyamatok többségében megtalálható valamelyest a szoftverek újrafelhasználása. A legtöbb esetben akkor következik be, amikor a projekten dolgozók ismernek olyan tervezést vagy akár kódot, ami hasonlít a kívánthoz. A komponensalapú szoftvertervezés célja így nem más, mint hogy már elkészített és validált szoftverelemeket építsünk be. Az újrafelhasználás nagyon gyakran szükséges a gyors fejlesztéshez.



ábra 2: Újrafelhasználás-orientált fejlesztés

A komponensalapú modell nyilvánvaló előnye, hogy csökkenti a kifejlesztendő szoftverek számát, így csökkentve a költségeket és a kockázati tényezőket. Ennél fogva a legtöbb rendszer gyorsabban leszállítható.

3. Web Modeling Language

3.1. WebML dióhéjban

A gyakorlatban a web fejlesztő eszközök egyszerűsítik a nagy mennyiségű adattal operáló webalkalmazások létrehozását és telepítését lap generátorok segítségével, mint például Microsoft's Active Server Pages vagy JavaSoft's Java Server Pages, amik elsődleges célja dinamikusan kinyerni az adatforrás tartalmát és beleilleszteni a felhasználó által programozott lapmintába. Habár ezek a rendszerek igen produktív implementációs eszközök, riasztó támogatást nyújtanak a követelmény meghatározás és a fejlesztési folyamat későbbi fázisai közötti rés áthidalására.

A W3I3 projekt az adat intenzív webalkalmazások intelligens információs infrastruktúrájára fókuszál. A projektben, amit a két vezető webfejlesztő igényei alapján alakítottak ki (Otto-Versand Németországból, ami e-kereskedelemre specializálódott és a Holland PPT (KPN) ami web-hosting szolgáltatásokban érintett), kifejlesztésre került egy újszerű web modellező nyelv (web modeling language), amit WebML-nek hívnak és egy támogató CASE környezet, ami Toriisoft néven ismert. A WebML a magas szintű, platformfüggetlen tervezését célozza meg adat intenzív webalkalmazásoknak, illetve olyan weboldalakét, amik fejlett tulajdonságokat igényelnek, mint a tartalom személyre szabása és az információ különféle eszközökre szállítása, mint a PC-k, PDA-k, mobiltelefonok. A Toriisoft tervező eszközök programcsomagja, ami lefedi a webalkalmazások teljes életciklusát és követi a modell-vezérelt megközelítést a Webre tervezéshez, a WebML használatával a középpontban.

A WebML egy konceptuális nyelv webalkalmazások magas szintű tervezéséhez, amelyet 1998-ban definiáltak. Számos egyetemen tanítják, illetve üzletileg is adaptálták, amit a WebRatio nevű program testesít meg. A WebML lehetővé teszi a fejlesztőknek egy oldal fő jellemzőit magas szinten kifejezni, anélkül, hogy komoly architektúrális részletekkel kellene foglalkozniuk.

A WebML grafikus, mégis formális, részletes leírásokat nyújt, megtestesítve a teljese fejlesztési folyamatot, amely vizuális tervezői eszközzel támogatható.

A WebML tervezési folyamatának fő objektívái a következők:

- kifejezni a webalkalmazás szerkezetét egy magas szintű leírással, ami használható lekérdezéshez, evolúcióhoz és karbantartáshoz
- ugyanahhoz a tartalomhoz több nézet nyújtása
- információs tartalmak összességének szétválasztása lapokká, navigációvá és megjelenésé, amik függetlenül határozhatóak meg és fejleszthetőek
- a fejlesztési folyamat alatt gyűjtött meta-információk tárolása repositorykban, amik felhasználhatóak az alkalmazás teljes élettartama alatt weblapok dinamikus generálására
- lehetővé tenni az adatmanipuláló műveletek meghatározását az oldal tartalmának frissítésére vagy tetszőleges külső szolgáltatásokkal való kölcsönhatásra
- felhasználók és közösségek modellezése expliciten a repositoryba, lehetővé téve személyre szabott politikák és egy az egyhez alkalmazások meghatározását

3.2. Adat intenzív webalkalmazások

A WebML modell vezérelt fejlesztését teszi lehetővé adat intenzív webalkalmazásoknak. Ezen webalkalmazások webes rendszerek, melyek célja közzétenni és karbantartani nagy mennyiségű adatokat.

Számos válfaja létezik, amelyek a következők: kereskedelem, tartalom, szolgáltatás és közösség orientáltság. Kereskedelem orientált webalkalmazások az elektronikus katalógusok, webshopok, aukciós portálok stb. Tartalom orientáltak az online hírportálok, digitális könyvtárak. Szolgáltatás orientáltak a rendelés követő rendszerek, foglalási rendszerek, illetve a turista információs rendszerek. Közösség orientáltak az üzenőfalak, a nevükben is megjelenő közösségi portálok, mint az iwiw vagy a facebook. Mindezekből következik, hogy igen komplexek tudnak lenni.

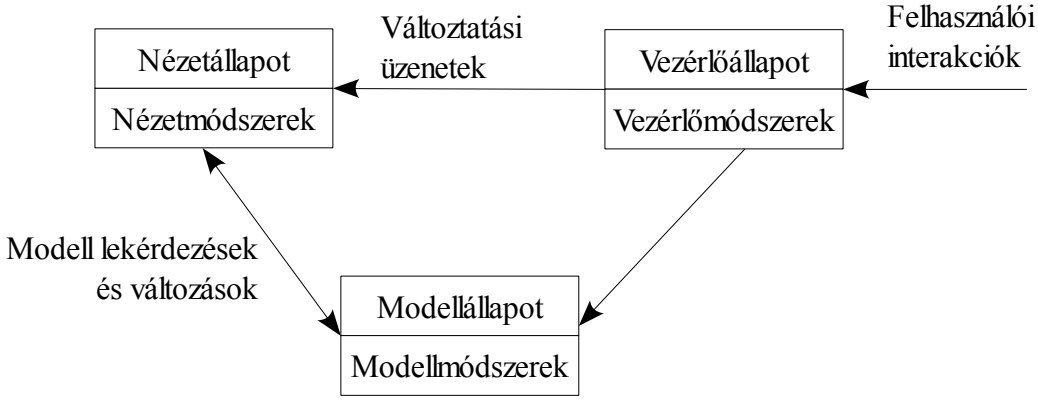
Ezen alkalmazások fejlesztésének gyakori hiányossága a modell vezérelt támogatás és a megalapozott szoftvertervezési módszerek. A hiányosságok eredménye rengeteg kézzel írott kód, teljesen adatközpontú módszerek, amelyek

nem fedik le a navigációs kezelői felületet, így elhanyagolva azt. A navigáció és megjelenés gyengén modellezetté válhat.

Az alkalmazásunk fejlesztését megnehezítheti továbbá, hogy a tervezés során több megjelenítő eszközre kell gondolnunk, hiszen az oldalt nem csak PC-ről tekinthetik meg, hanem PDA-ról vagy akár mobiltelefonról is. Érezhető, hogy mennyire fontos is szétválasztani a modellt, az irányítást és a megjelenést, így lehetővé téve, hogy bármilyen változás esetén, ne az egész alkalmazásunkat kelljen átszabni.

3.3. Model, View, Controller

Az MVC, azaz Model, View, Controller egy régóta létező (1979) és jól bevált architektúrális minta, amelyet a Xerox laborjaiban a Smalltalk fejlesztésekor dolgoztak ki. A MVC célja szétválasztani az üzleti logikát az adat megjelenítéstől azaz a felhasználói felülettől. Így a felhasználói felület és változása nem befolyásolja az adatkezelést, illetve az adatokat is átszervezhetjük anélkül, hogy a felhasználói felülethez nyúlnánk. Azért hogy ezeket megvalósítsa a minta a modell és a nézet közé bevezet egy vezérlőt, azaz elválasztja az üzleti logikát (model), a felhasználói interakciókat (controller) és a megjelenítést (view).



ábra 3: MVC architektúrális minta

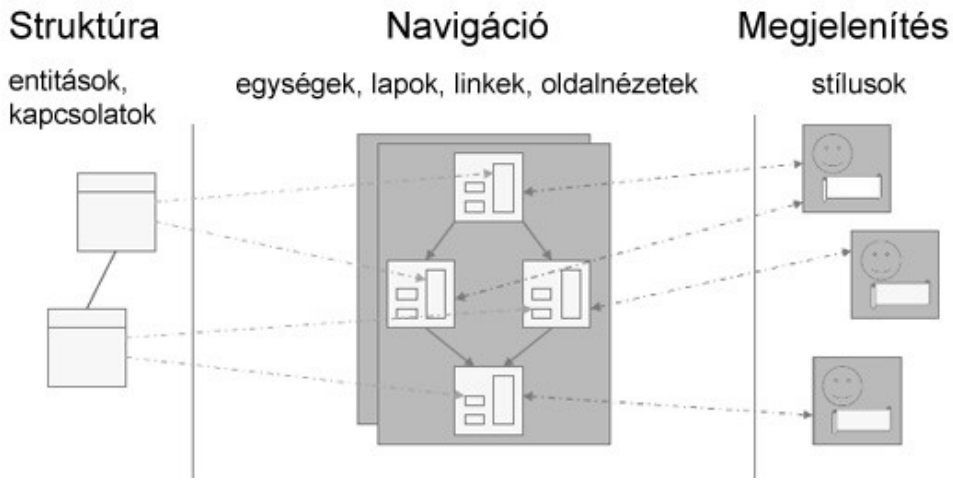
A modellben van jelen az üzleti logika, ez a réteg foglalkozik az adateléréssel és azok szerkezetével. Ezen réteg változása nem vonja maga után a nézet változását. Az adatbázis elérése is ebbe a rétegbe tartozik.

A nézet, a tartalom felhasználónak való megjelenítésével foglalkozik, amit a

modell példányain keresztül érünk el. Egy modellre, korlátlan számú nézetet hozhatunk létre.

A controller foglalkozik a felhasználó interakcióival, illetve összeköti a nézetet és a modellt. Ha a felületen kattintunk valamire, azt a kontroller kezeli le, ha kell szerkeszti a modellt, és ha vannak változások megjeleníti azt a nézeten.

Az MVC minta megoldást kínál arra, hogy kis változtatással és kevesebb időigénnyel több kimeneti eszközre fejlesszük ki alkalmazásunk. Továbbá, mint fentebb említettem, az adatok átszervezése sem befolyásolja egyik nézetet sem.



ábra 4: WebML architektúra

A WebML a 4. ábrán látható módon szintén három részre választja szét az architektúrális tervezést éppen az MVC mintának megfelelően.

4. Adatmodell a WebML-ben

4.1. Az adatmodell

A struktúra tervezése alatt az adatmodellt (data model) értjük. Az adatmodellben tárolunk minden az alkalmazás üzleti folyamataihoz szükséges adatot, entitások és ezek között lévő kapcsolat formájában ábrázolva. Az adatmodell természetesen az alkalmazás egyik alappillére, ezért nagyon fontos, hogy jól megtervezett legyen.

Az adatmodellezés eredménye egy koncepcionális séma, amely egyszerű és olvasható formában közvetíti a rendelkezésre álló tudást az alkalmazás adatairól. Egy ilyen séma tervezése előzi meg az adatokon működő üzleti függvények tervezését és a fizikai szerkezeteket támogató adattárház kialakítását, frissítését, kinyerését egyaránt.

Az adatmodellezés egyike a legrégebbi és állandó tudományágaknak az információtechnológiában, amelyhez jól kialakult modellező nyelv és útmutató létezik. A WebML az adatmodellezéshez az Entinty – Relationship (E-R) modellt használja.

Amikor egy alkalmazás adatmodelljét tervezzük, a következő két kérdést kell minden esetben szem előtt tartanunk:

- Milyen objektumokat jelenítünk meg az oldalon?
- Ezen objektumoknak milyen tulajdonságaik vannak és hogyan kapcsolódnak egymáshoz?

Célunk tehát a való világ objektumait és a köztük lévő kapcsolatokat lemodellezni, melyhez a WebML a következő eszközöket nyújtja: entitások, attribútumok, kapcsolatok, IS-A hierarchiák.

Az adatmodell megtervezésekor lényegében az adatbázisunkat is megterveztük, így arra már nem kell jelentős időt és számottevő energiát fordítani.

4.2. Entitás

Egy entitás egy sor elem „leírója”, amik megtalálhatók a való világban és amelyeknek hasonlóak a jellemzőik. Az objektum orientált világból ismerten itt is

absztrahálunk. Egy entitás lesz például a könyv, hiszen általános. Minden könyvnek van szerzője, címe, ISBN száma stb.

Éppen ezért nagyon fontos eleme a webalkalmazásoknak, hiszen a való világ egyéni elemeit az entitások példányaiban tároljuk, amiket gyakran objektumoknak is nevezünk. Egy entitás összes példánya adja meg a népességét az entitásnak.

Az entitások rendelkezhetnek leszármazottakkal és attribútumokkal.

ábra 5: Entitásra példa

Ábrázolása tehát egy téglalappal történik, aminek felső részében az entitás neve (az 5. ábrán "Book") található vonallal elválasztva az alsó részben található attribútumoktól (az 5. ábrán: "Title: string", "Price: float", stb).

4.3. Attribútum

Egy attribútum (nevezik még skaláris vagy mono-attribútumnak) egy közös tulajdonsága az entitások példányainak. Minden könyvnek van például szerzője, címe, megjelenési éve stb. és minden szerzőnek van neve, születési éve és egyebek. Előfordulhat azonban, hogy néhány objektumnál talán nincs értelmes vagy jelentételi értéke egy attribútumnak, például a szerző halálának a dátuma nem értelmezett, ha él még.

Minden attribútumnak a fejlesztő által definiált neve és típusa van. Az attribútumok típusai a máshonnan már jól ismert megszokottak lehetnek, pl.: String, Url, Image, Integer, Float, Date. Az attribútum értékkel rendelkezik, ami lehet NULL is és nincs belső alstruktúrája, tehát nem tagolódhat tovább.

Ábrázolása az entitásban név: típus formában történik.

4.4. Kapcsolat

Adatmodellezéskor egy kapcsolat alatt bináris kapcsolatot értünk. A kapcsolatok az entitás példányok közti különböző típusú kötelékek modellezésére szolgál: fizikai

belefoglaltság, logikai belefoglaltság, függőség, birtoklás, használati jog stb.

Egy bináris kapcsolat példánya két objektum között azt a tényt jelöli a való világban, amely magába foglalja mindkét objektumot. Ilyen kapcsolat példány egy könyv és az szerzője között fennálló birtoklási viszony. Tehát egy kapcsolat az objektumok közötti kapcsolat példányok halmazának leírója.

A kapcsolatnak egyéni nevet adhatunk. Ábrázolása két entitás vonallal való összekötéssel történik.

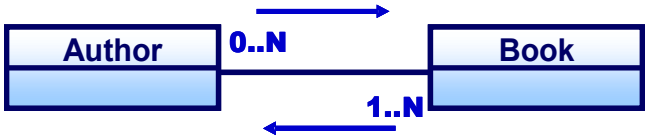
A kapcsolatoknak van kapcsolatszerepük. A kapcsolatszerep egyik a két iránynak, amelyből a kapcsolatot tekinthetjük. Tehát a szerző_könyv kapcsolat két kapcsolatszereppel rendelkezik: szerző_a_könyvhöz, könyv_a_szerzőhöz.

ábra 6: Kapcsolatszerepekre példa

szerep1: author2book

szerep2: book2author

Mindkét irányra a kapcsolatban meghatározhatóak a minimum és maximum számossági megszorítások. A szerző_könyv példánál maradva ez azt jelenti, hogy egy könyvnek lennie kell legalább egy szerzőjének, hiszen könyv nem létezhet szerző nélkül, míg egy szerzőhöz tartozhat akárhány könyv, vagy akár semennyi sem.



ábra 7: Kapcsolat számosságra példa

author2book minimum számossága: 0 maximum számossága: N

book2author minimum számossága: 1 maximum számossága: N

A számossági megszorítások alkalmazása fontos, mivel a struktúra sémáját még olvashatóbbá és kifejezőbbé teszik, továbbá befolyásolják az adatbázis fizikai

struktúrájának a tervezését is.

4.5. Többértékű attribútum

Egy többértékű attribútumnak nevezünk egy olyan attribútumot, amihez egy objektumnak több értéke lehet. Például egy személynek több lakcíme is lehet. Persze egy attribútum csak egy értéket tárolhat, így a többértékű attribútumot valójában egy kapcsolattal és entitással reprezentáljuk.

Ez a plusz entitás általában gyenge lesz, tehát nem létezhet önállóan a többértékű attribútumot tartalmazó entitás nélkül.

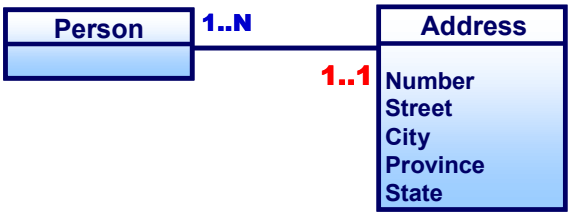
ábra 8: Többértékű attribútum példa

Tehát a 8. ábrán az "Address" entitás gyenge és nem létezhet a "Person" entitás nélkül. Ugyanakkor a "Person" entitás is megköveteli "person2address" kapcsolatszerepen keresztül, hogy legalább egy "Address" entitással kapcsolatban legyen, hiszen a "person2address" minimum számossága egy.

4.6. Strukturált attribútum

Strukturált attribútumról akkor beszélünk, ha az attribútum belső struktúrával rendelkezik. Szintén úgy reprezentáljuk, mint a többértékű attribútumot, tehát egy kapcsolat és egy entitás segítségével.

Példaként egy cím különböző mezőkből állhat, mint az irányítószám, város, utca, házszám és így tovább.



ábra 9: Strukturált attribútum példa

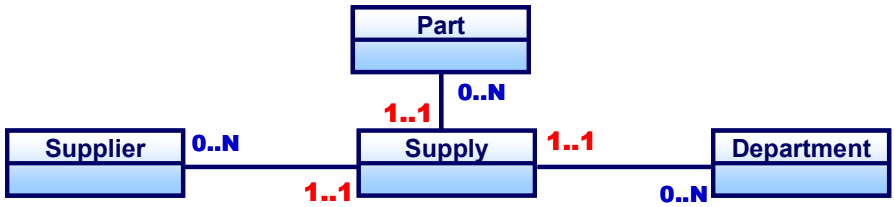
A 9.ábrán látható "Address" entitás itt is gyenge, és függ a "Person" entitástól.

4.7. Kapcsolat attribútumokkal és n-ed leges kapcsolat

Egy kapcsolat attribútumokkal, egy tulajdonság leírása ami objektum párokra (általánosabban egy sor objektumra) utal. Ilyen például a jegy amit a hallgató a kurzuson kap. A 10. ábrán látható "Grade" entitás gyenge.

ábra 10: Kapcsolat attribútumokkal példa

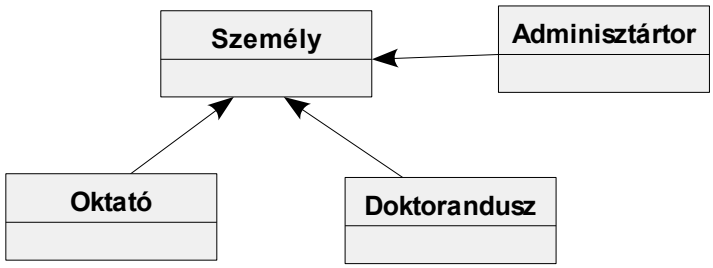
Egy n-ed leges kapcsolat annyiban különbözik az előbbitől, hogy több, mint két entitás vesz részt benne, tehát n > 2.



ábra 11: N-ed leges kapcsolat példa

4.8. IS-A hierarchia

Mint az objektum orientált nyelvekben, itt is létezik öröklődés. Az IS-A hierarchia, amit talán "ez egy"-nek fordíthatunk, egy speciális összeköttetés két entitás között, ami azt jelenti, hogy az egyik entitás, azaz az al-entitás egy speciális esete a másiknak, azaz a szuper-entitásnak. Az al-entitás magában foglalja a szuper-entitás összes tulajdonságát, illetve az al-entitás rendelkezhet ezen felül további, saját tulajdonságokkal is. A többszörös öröklődés nem megengedett, viszont számos szintje lehet.



ábra 12: IS-A hierarchia példa

Az elkészített webalkalmazásomban az általános személyből specializálom az adminisztrátort, a doktoranduszt és az oktatót. A doktorandusznak és az oktatónak még külön, saját tulajdonságaik vannak, míg az adminisztrátort csupán megkülönböztetem. Ám mind rendelkezik a személy tulajdonságaival. Ennek sematikus IS-A hierarchiája a 12. ábrán látható.

Ábrázolása entitásokat összekötő nyíllal történik, az al-entitás mutat a szuper-entitásra, amiből specializálódik.

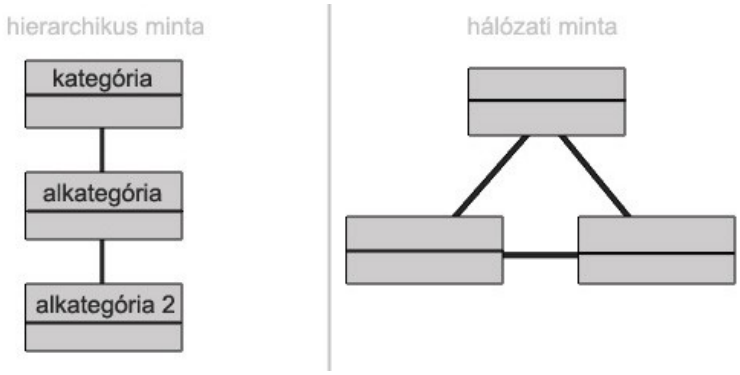
4.9. Struktúra minták

A struktúra minták tipikus struktúra tervezési megoldások, melyek kombinálhatóak egymással: csillag, hierarchikus, hálózati.

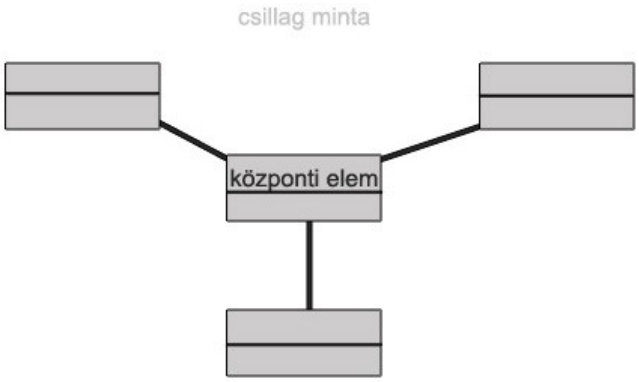
A hierarchikus minta kategorizálásra alkalmas. A felhasználó fentről lefelé haladva böngészheti azt. Gyakorta használt webshopoknál.

A hálózati mintánál nincs központi objektum, minden elem azonos fontosságú és részletes leírás tartozik mindegyikhez. A felhasználó böngészhet az egyik objektumtól a vele kapcsolatban álló másikig.

A csillag mintának van egy könnyen felismerhető központi eleme, amihez számos informatív elem kapcsolódik. Böngészéskor a felhasználó ezekre "ugorhat" és vissza a főelemre.



ábra 13: Hierarcihus és hálózati struktúra minta



ábra 14: Csillag struktúra minta

5. Hypertext modell a WebML-ben

5.1. Hypertext modell

A hypertext modell célja magas szinten modellezni egy dinamikus webalkalmazás felületét és a kölcsönhatásokat a háttérben lévő üzleti logikával és adatokkal. Másik célja lehetővé tenni oldalsablonoknak illetve adat hozzáférő és manipuláló lekérdezéseknek a dinamikus generálását.

A hypertext oldal nézetekből, lapokból, tartalmi egységekből és hivatkozásokból áll.

5.2. Alap tartalmi egységek

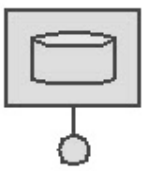
Tartalmuk a szelektoroknak megfelelő entitás példányok és ezen tartalom megjelenítésére illetve kezelésére szolgálnak.

A szelektorok a megfelelő objektumok kiválasztásához szükséges feltételeket definiálhatnak és a kapott paraméterek alapján döntenek. Szelektorral minden egység rendelkezhet.

Alap tartalmi egységből hét különböző létezik. Minden egység rendelkezik bemeneti és kimeneti paraméterekkel, a bemenetet pedig kötelező az egységnek magának kiszámolnia. Az egységek az információt egymás közt hivatkozásokon keresztül szállítják.

Bemeneti paramétereiket többnyire hivatkozásokon keresztül kapják, továbbá egymással szintén azokon keresztül cserélnek információt, tehát kimenő paramétereiket is hivatkozásokon keresztül továbbítják.

Azon egységeknél, amelyek objektum tartalmat képesek megjeleníteni, azoknál testreszabható, hogy mely attribútumokat jelenítse meg.



ábra 15: adat egység

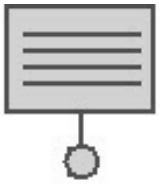
A adat egység (data unit) információt jelenít meg egy objektumról. Bemenete lehet az objektum OID-ja (objektum azonosító, angolul object identifier), amit megjeleníteni akarunk, vagy a szelektor számításaihoz szükséges paraméterek, mint a attribútum értékek

összehasonlításhoz, illetve egy kapcsolatban résztvevő objektum OID-ja. Például ha egy könyvet akarunk megjeleníteni, a cím attribútumát összehasonlíthatjuk egy értékkel. Ha pedig kapcsolat szerint akarunk "szűrni" akkor jó például az anya gyermek kapcsolat, hiszen minden gyermeknek csak egy anyja van, így a gyermek OID-jával megjeleníthetnénk az anyát.

Kimenete a megjelenített objektum OID-ja. Amennyiben nem tartozik hozzá sem bemeneti hivatkozás, sem szelektor, akkor az egység hibás.

ábra 16: Adat egység megjelenítés

A programomban számos helyen alkalmaztam ezt az egységet: szigorlat, tézis, védés, felhasználó és egyéb objektumok megjelenítésére. A 16. ábrán látható, amin az egység egy user objektumból jeleníti meg a felhasználónév és email tulajdonságokat, míg a password és oid attribútumok nem láthatóak.

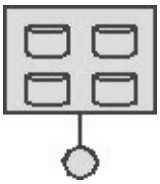


ábra 17: Mutató egység

A mutató egység (index unit) elemek egy listáját jeleníti meg. A szelektor számításaihoz szükséges bemenő paraméterei lehetnek értékek attribútumok összehasonlításokhoz és az entitáshoz kapcsolódó objektum OID-ok. Ha nincs bemenet, akkor az összes objektumot listázza. Kimeneti paramétere a felhasználó által kiválasztott objektum OID-ja.

Ez az egység volt talán a leggyakrabban használt alapegységem, mivel számos adatot jelenítek meg, amelyeket aztán ki lehet választani. A 18. ábrán látható az oktató tudományos címeit listázó mutatóegység megjelenítése, melyből még "Módosít" és "Töröl" hivatkozások is indulnak. Használtam még például az oktató és doktorandusz publikációi, tudományos fokozatai megjelenítésénél is.

ábra 18: Mutató egység megjelenítés



ábra 19: Multiadat egység

A multiadat egység (multi-data unit) egy entitás egyszerre több példányát képes megjeleníteni. A szelektor számításaihoz szükséges bemeneti paraméterek az értékek attribútum összehasonlításokhoz és kapcsolatokban résztvevők OID-jai. Kimeneti paramétere a megjelenített objektumok OID-jainak egy tömbje.

A multiadat egység nem kapott szerepet a programomban, nem tartottam szükségesnek. Példaként megemlíthetjük egy könyvet véleményező hozzászólásokat, amiket felhasználók írtak egy adott időpontban.

A mutató és a multiadat egység nagy hasonlóságokat mutat, és mégsem ugyanazok, hiszen más más cél vezérli őket. A mutató egység hozzáférési módszerként használható, hogy részletes információkat mutasson meg egy objektumról. A multiadat egység egyszerre sok objektum információit jeleníti meg, habár ezt a mutató egység is tudja. Viszont a multiadat egység úgy jeleníti meg azokat, mint egyedülálló adategységeket, tehát nem listaszerű külsőt kap.

Az igazi különbség a kimeneti paramétereknél van. Míg a mutató egység egy kiválasztott objektum OID-ját bocsájtja ki, addig a multiadat egység minden megjelenített objektum OID-ját kibocsájtja.

ábra 20: Beíró egység

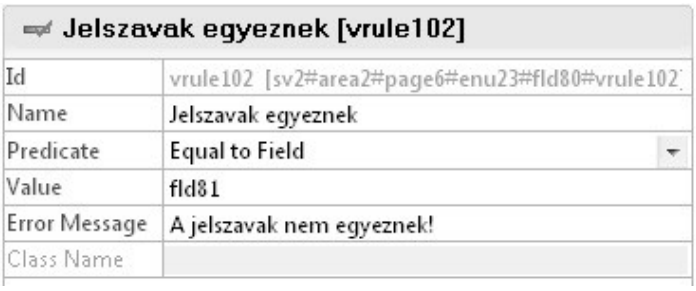
A beíró egység (entry unit) űrlapok leírásához szolgáltat támogatást. Segítségével a felhasználó információkat nyújthat be. A beírt tartalom kimenetként más egységeknek szállítódik kimenő hivatkozásokon keresztül, hivatkozási paraméterek segítségével. Tipikusan HTML-re fordítódik. Mint a 20. ábrán látható, kis jelöléséhez kis kör alulra nem kapcsolódik, azaz nem tartozik hozzá entitás.

A beíró egységek kétfajta minialkalmazást tartalmazhatnak adatok beviteléhez. Az egyik ilyen a mező (field) amellyel új értékeket vihetünk be, a másik a választó mező (selection field) amellyel egy listából választhatunk ki egy értéket.

A mezők különböző tulajdonságokkal rendelkeznek. Ilyen az előtölthetőség, azaz már egy létező érték betöltése a mezőbe. A mező előtölthető több érték konkatenációjával is. Az előtöltéshez az érték lehet adatbázisból vett dinamikus vagy a modellben definiált statikus.

A módosíthatóság meghatározza, hogy a mezőben szereplő érték változtatható-e, igen vagy nem értékekkel rendelkezhet. Rendelkezik még láthatósággal, aminek az értéke ha rejtett, a mező nem jelenik meg és lehet látható. Természetesen van a mezőnek adattípusa, illetve bemenet érvényesítési szabályokat is illeszthetünk rájuk.

A beíró egység tartalmát csak akkor küldhetjük el, ha a mezők és választó mezők tartalma minden egyes validációs szabálynak megfelelnek. Számos validációs szabály van (date, integer, not null, begins with stb.), melyeket a mezőkre és választó mezőkre alkalmazhatunk. Hibaüzenetük személyre szabhatóak külön-külön egyenként, de általánosan is beállítható. Azaz ha nem állítjuk be külön a validációs szabálynak a hibaüzenetet, akkor az általános üzenet jelenik meg.

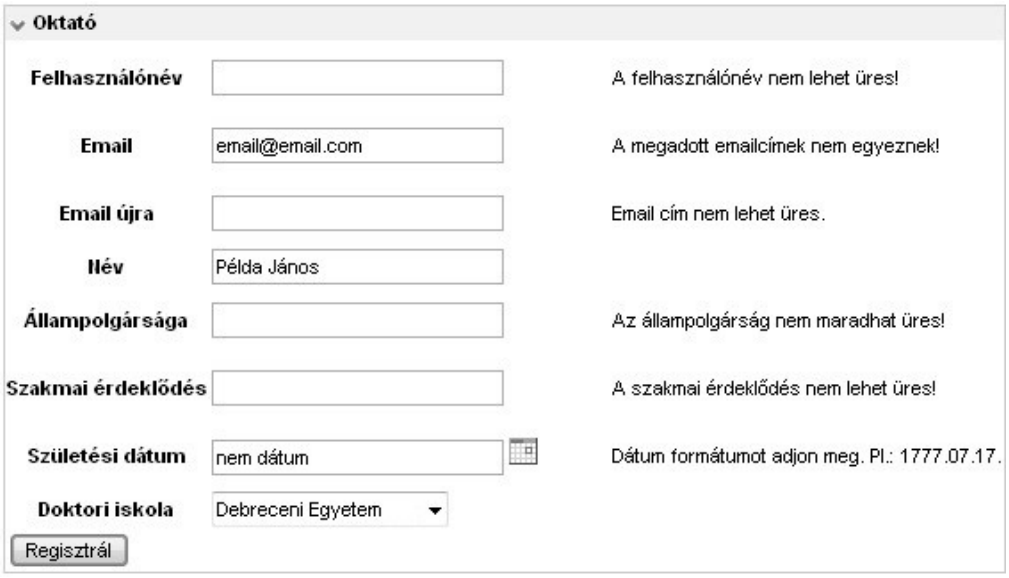


ábra 21: Validációs szabály tulajdonságai

A 21. ábrán a validációs szabály azt a mezőt hasonlítja az "fld81" azonosítójú mezőhöz, amelyhez hozzáadtuk. Ezzel a szabállyal két mezőt hasonlítok, így ellenőrzöm, például hogy a két beírt jelszó azonos-e.

A validációs szabályok rendelkezhetnek értékkel is, amelyeket a szabályok használhatnak fel. Például, mint a programomban, a telefonszám mező tartalmának +36-al kell kezdődnie, hogy az adatokat elküldhessük. Ehhez a begins with szabályt használtam az értéke pedig "+36" volt. Tehát azon adatok felelnek meg a szabálynak, amik +36-al kezdődnek.

A 22. ábrán látható a validáción át nem ment egység. Amikor megpróbáltuk elküldeni az adatokat, a mezők egy része nem felelt meg a validációs szabályoknak, így üzeneteket kaptunk a mezők jobb oldalán.



ábra 22: Validációs szabályoknak nem megfelelő form

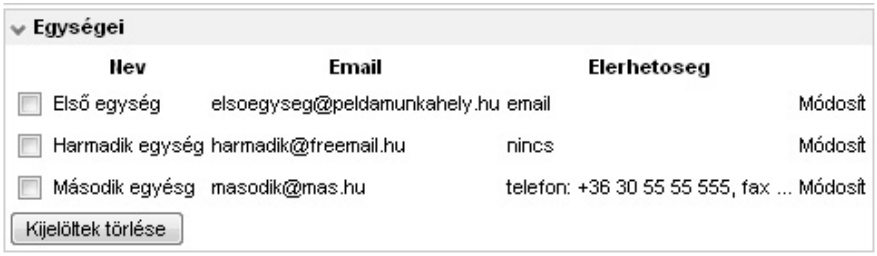
Az egyetlen mód adatok bevitelére ez az egység, így sokszor alkalmazott a programomban (bejelentkezés, doktorandusz adatai, oktató adatai, kurzus adatai stb.).

ábra 23: Többválasztású egység

A többválasztású egység (multichoice unit) elemek egy listáját jeleníti meg, amelyek közül a felhasználó egyet vagy többet is megjelölhet bejelölőnégyzetek segítségével. Tehát annyiban különbözik a mutató egységtől, hogy itt több elemet választhatunk ki.

Bemeneti paraméterei a szelektor számításaihoz szükséges értékek attribútum összehasonlításokhoz és a kapcsolatokban résztvevők OID-jai. Ha nincsenek bemeneti hivatkozások, akkor minden entitás példányt megjelenít. Kimenete a kiválasztott objektumok OID-jai. Előválasztóval vagy máshogy nevezve preszelektorral rendelkezik, ami lehető teszi, hogy a listázott elemek egy részét előre bejelöljük kiválasztottként.

A programomban csupán egyszer alkalmazott egység. Arra használom, hogy egy munkahelyhez tartozó egységek közül egyszerre többet jelölhessek ki törlésre.



ábra 24: Többválasztású egység megjelenítés

Mint a 24. ábrán látható, a listán szereplő elemek egyenként is kiválaszthatóak módosításra, mintha mutató egység lenne, de több egység is kiválasztható törlés céljából. Módosításra egyszerre nem választhatóak ki.

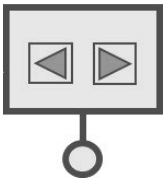
ábra 25: Hierarchikus egység

Hierarchikus egység (hierarchical unit) elemek egy listáját hierarchikusan rendezett bejegyzéssekkel jeleníti meg, egymással kapcsolatban álló entitások használatával. A felhasználó a hierarchia bármely szintjéről kiválaszthat egyetlen elemet.

A szelektor számításaihoz szükséges bemenő paraméterei lehetnek értékek attribútum összehasonlításokhoz és az entitáshoz kapcsolódó objektum OID-ok. Kimeneti paramétere a felhasználó által kiválasztott objektum OID-ja. Fontos, hogy a hierarchiát egymással kapcsolatban lévő entitások között állítjuk fel.

Erre az egységre egyszer sem volt szükségem a programomban. Példaként talán fel lehetne hozni egy bútorbolt termékeinek kategorizálását és listázását egészen egyed szintig. Legkülső kategóriák lehetnének a székek, asztalok, ágyak stb., majd ezen belül a megfelelő modellek: Linda asztal, Petra asztal stb. A Linda asztal, Petra asztal objektumok más entitáshoz tartoznak, mint az asztal objektum. A hierarchiát a kapcsolatuk állította fel.

ábra 26: Hierarchikus megjelenítés példa



ábra 27: Görgető egység

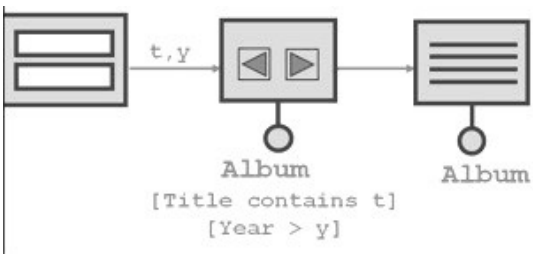
A görgető egység (scroller unit) böngésző tulajdonságokat ad objektumok egy halmazának. A blokkfaktorban (blockfactor) megadott számú objektummal rendelkező részhalmazokra bontja azt és hivatkozásokat jelenít meg a halmaz első, előző, következő és utolsó részhalmazához.

Bizonyos mennyiségű görgetett objektumot blokkfaktornak nevezünk. Azaz a hozzákapcsolt egység egyszerre annyi objektumot jelenít meg maximum, amennyi a blokkfatorban szerepel.

A szelektor számításaihoz szükséges bemenő paraméterei lehetnek értékek attribútum összehasonlításokhoz és az entitáshoz kapcsolódó objektum OID-ok. Kimenete az aktuális blokkba tartozó OID-ok készlete.

Az adat, az index és a multiadat egységekkel kapcsolatban használjuk és egy lapra helyezzük azokkal.

Görgető és adat egység összekapcsolásakor az entitások ugyanazok. Az adat egység automatikusan szinkronizál hozzá. Azaz az adatokat küldhetjük az adat egységnek és a görgető egység kezeli, hogy melyiket jelenítsük meg. Ilyenkor természetesen a blokkfaktor értéke 1 és a részhalmazok 1 objektumot tartalmaznak.

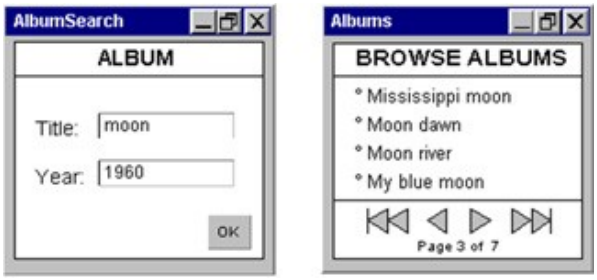


ábra 28: Beíró, görgető és mutató egység összekapcsolva

A görgető és mutató egység összekapcsolása kiválóan alkalmas keresési eredmények lapozására.

A 28. ábrán a beíró egységben megadott értékek a görgető egység bemenő paraméterei attribútum összehasonlításokhoz. A "t" értéket hasonlítja a "title"attribútumhoz illetve az "y" értéket a "year" attribútumhoz és ezen kritériumoknak megfelelő objektumokat fogja majd blokkokra bontani. Kimenete a megfelelő blokkba tartozó OID-ok, amit a mutató egység jelenít meg.

Ezt az egységet nem alkalmaztam a programomban, hiszen a témából adódóan nagy számú adattal nem dolgozom, amit ilyen módon tördelni kellene.



ábra 29: Valós példa keresési eredmények böngészésére a görgető egység segítségével

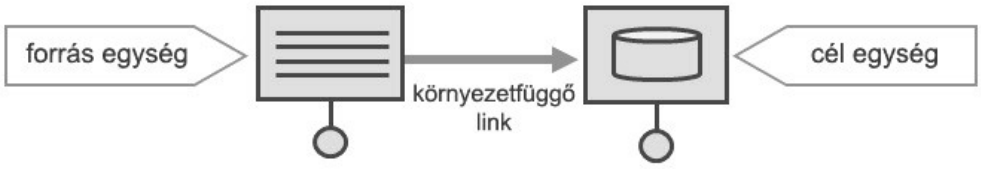
5.3. Hivatkozások

A hivatkozások lapokat és egységeket köthetnek össze és felelősek az adatok szállításért és a tartalmak közötti navigálásért. A hivatkozásoknak négy típusa van:

- környezetfüggetlen hivatkozások (non-contextual links)
- környezetfüggő hivatkozások (contextual links)
- szállító hivatkozások (transport links)
- automatikus hivatkozások (automatic links)

A környezetfüggetlen hivatkozások adatot nem szállítanak, pusztán a lapok közötti navigálásra alkalmasak. Akkor alkalmazom, amikor valamilyen új tartalomhoz akarok navigálni, anélkül, hogy bármilyen adatra szükségem lenne. Például amikor új doktoranduszt akarok létrehozni. Az általános doktorandusz lapról egy sima hivatkozás segítségével átnavigálok arra a lapra, ahol az új doktorandusz adatait felvihetem. Ugyanez a helyzet új oktató és új munkahely felvételekor is.

A környezetfüggő hivatkozás egy irányított kapcsolat két egység között. Ez a két egység a forrás és a cél egység. Webalkalmazásokban rendszerint egyszerű szöveges hivatkozásként vagy küldő gombként jelenik meg.



ábra 30: Környezetfüggő link

A környezetfüggő linkek célja a felhasználói navigáció és információk szállítása egyik helyről a másikra. Mellékhatásként egy számítást is indít. Ilyen hivatkozással indítunk el gyakran valamilyen műveletet amihez adatra van szükség.

Például a programomban a bejelentkezési adatokat is ez a hivatkozás szállítja a bejelentkező egységbe (login unit). Vagy ha az oktatók listájából kiválasztok egyet, akkor ilyen típusú hivatkozás navigál el az oktató szerkesztése lapra és adja meg az oktató OID-ját a lapon található egységeknek. Tehát a mentés, módosítás, létrehozás, törlés, kiválasztás hivatkozások mind ilyenek, mindig szállítanak valamilyen adatot ami szükséges ezen műveletekhez.

Az adatok hivatkozás paraméterek használatával a hivatkozásokon keresztül szállítódnak. Egy paramétert meghatároz a neve, célja és forrása. A forrása az egyike a forrás egység kimenő paramétereinek, míg a célja az egyike a célegység bemenő paramétereinek. Ezek összepárosítását a felhasználó határozza meg.

Például a bejelentkezésnél a felhasználónév mező és a jelszó mező értéke a bejelentkezés gombra kapcsolva elszállítódik, mint kimenő paraméter a bejelentkező egység Username és Password bemenő paramétereibe. A felhasználónevet logikusan a Username, míg a jelszavat a Password paraméterrel párosítottam.

Léteznek alapértelmezett hivatkozás paraméterek, ezeknél a cél és forrás automatikusan párosított. Olyankor lehetséges ez, ha egyértelműen következik. Azaz a forrás entitás és cél entitás megegyezik vagy a két entitás kapcsolatban áll.

Például az oktatókat listázó mutató egység hivatkozással össze van kötve egy adategységgel, ami részletesen megjeleníti az adatait. Mindkét egységhez az oktató entitás tartozik. Kapcsolat esete például, ha egy mutató egységgel az oktatóhoz kapcsolódó doktoranduszokat jelenítem meg és az oktató és doktorandusz között létezik egy oktató_doktorandusz kapcsolat.

ábra 31: Automatikus hivatkozás

Egy automatikus hivatkozás előre beállított, alapértelmezett adatokat szállít a cél egységnek azonnal, a forrás egység megjelenítése után, felhasználói közbeavatkozás nélkül. Utólag viszont a felhasználó megváltoztathatja az átadandó adatokat, gyakorlatilag újra küldheti azokat, egy hivatkozás segítségével, ami lehet gomb, vagy

szöveg formájú. A görgető egység is ilyen típusú hivatkozáson keresztül kapcsolódik más egységekhez.

ábra 32: Szállító hivatkozás

Egy szállító hivatkozás előre beállított, alapértelmezett adatokat szállít a cél egységnek, azonnal a forrás egység megjelenítése után, felhasználói közbeavatkozás nélkül. Utólag a felhasználó nem változtathatja meg az átadandó adatokat. Sem szöveges hivatkozás, sem gomb nem reprezentálja, tehát a felhasználói felületen nem jelenik meg.

A programomban ez a leggyakrabban használt hivatkozástípus. Egy művelethez több entitásból is származhat adat, viszont egy környezetfüggő hivatkozás csak egy entitásból tud szállítani egy entitásnak, így nélkülözhetetlenek a szállító hivatkozások. Továbbá az egy lapon megjelenő minden egységnek többnyire szüksége van bemenő paraméterekre, és azoknak egy egységből, ilyen típusú hivatkozásokkal adjuk át.

Például a programban kiválasztjuk az oktatót, akkor az OID-ja egy környezetfüggő hivatkozáson keresztül eljut egy kiválasztó egységbe (selector unit) ,amely számos egyéb egységbe küldi el a lekérdezett oktató bármely adatát. A publikációit, tudományos fokozatait és címeit, hallgatóit listázó mutató egységek mind megkapják az OID-ját, hogy egy kapcsolati szelekció alapján megjelenítsék az információkat.

5.4. Lapok és tartományok

ábra 33: Lap ábrázolása

Egy lap (page) egy vagy több olyan információt tárol, amelyeket egyszerre lát a felhasználó. Minden megjeleníteni kívánt egységet lapokra kell elhelyeznünk, így a lapok fontos alapkövei egy alkalmazásnak. A felhasználó ezekből a lapokból álló oldalon más néven oldal nézeten navigálhat. Az oldal és oldal nézet szavakat szinonimaként használom. Lapok egymásba ágyazása megengedett, egy lapnak lehetnek al-lapjai.

Egy lapnak különböző tulajdonságai lehetnek: főlap (homepage), referenciapont lap (landmark page), alapértelmezett lap (default page).

A főlap, mint a nevében szerepel a főlapja egy oldalnak, ez az amit a felhasználó először lát amikor belép egy oldalra. Minden oldal nézetnek tartalmaznia kell egy lapot, amit „home”-ként jelölünk, de csak egy főlapot tartalmazhat, többet nem. Használata kötelező, tehát a programomban is minden oldalon található egy ilyen lap, amit kezdőlapnak szántam.

A referenciapont lapok globálisan látható lapok. A felhasználó az oldalról bárhonnan odaugorhat, olyan mintha az oldal minden lapjától odavezetne egy környezetfüggetlen hivatkozás. Tehát ezen jelzéssel ellátott lapok egy főmenüt alkotnak, ami a weboldalon mindig látható. Ilyen lapokat az oktató és hallgató oldal nézeteinél alkalmaztam, hogy szabadon navigálhassanak az oldalon.

Fontos, hogy olyan lapot, ami egy másik lapról vár nélkülözhetetlen adatokat környezetfüggő hivatkozáson keresztül, ne adjunk meg referenciapont lapnak, hiszen akkor a hivatkozás átugrásával navigálhatnánk oda anélkül, hogy adatokat adtunk volna át az egységeknek.

Az alapértelmezett lap egy tartomány kezdőlapja, amivel minden tartománynak rendelkeznie kell.

Az előbbi tulajdonságokat a 34. ábrán látható jelekkel jelezzük egy lap belsejében.

ábra 34: Homepage, Default és Landmark jelzések

Egy tartomány (area) logikailag homogén lapok összessége. A tartományok egymásba ágyazhatóak, tehát egy tartományban al-tartományok hozhatóak létre. Referenciapont tulajdonsággal rendelkezhetnek és természetesen van nevük.

Minden tartományban kell lennie egy alapértelmezett lapnak vagy egy alapértelmezett al-tartománynak, azért hogy értelmet adjon a referenciapont tartományoknak és a tartományokra mutató környezetfüggetlen hivatkozásoknak. Ha az al-tartományok is rendelkeznek a referenciapont jellemzővel, akkor a főmenünek almenüje keletkezik és így tovább.

Ábrázolása úgy történik, mint egy lapé, csupán a háttérszíne más: világoskék. Ha referenciapont tartomány akkor azt a 34. ábrán látható landmark jelzéssel jelezzük.

A programban, az adminisztrátori oldalon szám szerint 7 tartományt hoztam létre

al-tartományok nélkül és ezek alkotják az adminisztrátor főmenüjét, hiszen megadtam nekik a referenciapont tulajdonságot. Például minden, az oktatóhoz szorosan kapcsolódó lap az oktató menedzsment tartományba került: oktató menedzsment home, oktató létrehozás, oktató szerkesztése, kulcsszó javaslat.

5.5. Oldal nézetek

Az oldal nézetek (site views) olyan lapoknak és vagy tartományoknak a halmaza amelyek egy összefüggő nézetét adják az oldalnak. Ugyanahhoz az adatmodellhez több oldal nézet is definiálható, így különböző oldal nézetek jeleníthetőek meg különféle felhasználóknak és a különféle kimeneti eszközökön.

Az oldal nézetek publikusak és védettek lehetnek. A publikus oldalakat bárki megnézheti, míg a védett oldalakhoz felhasználói hitelesítés tartozik. A felhasználói hitelesítés bejelentkezéssel történik. A bejelentkezett felhasználó, csak a felhasználói csoportjának megengedett oldalakat tekintheti meg. Nem csak egész oldal nézeteket, hanem lapokat és tartományokat is védetté tudunk tenni.

A programomban minden felhasználói csoportnak saját nézete van. Bejelentkezés, hallgató, oktató és adminisztrátor nézetekkel rendelkezem. Ezek közül csak a bejelentkezési oldal publikus, míg a többi védett. A hallgatói oldalnézetet a doktoranduszok láthatják bejelentkezés után, az oktatóit az oktatók, míg az adminisztrátorit az adminisztrátorok.

Csak egy oldal nézet rendelkezhet "home" jellemzővel, ez lesz az az oldal, ami a weblap címére bejön. Az alkalmazásomban ez az oldal nézet a bejelentkező.

6. A WebML egyéb részei

6.1. Műveleti egységek

A műveleti egységek (operation units) általános külső vagy beépített tartalom manipuláló műveleteket modelleznek. Tehát az alkalmazás adatait menedzseli. Ha az adataink adatbázisban tárolódnak, akkor ezek adatbázis műveleteket képviselnek.

Bemeneteik egy vagy több hivatkozás, melyekből legalább egy környezetfüggő hivatkozás a többi pedig szállító. Kétféle kimeneti hivatkozással rendelkeznek. Az OK hivatkozás jelzi, hogy a művelet sikeresen hajtódott végre, míg a KO link annak sikertelenségére utal.

A WebML előre definiál néhány gyakran használt beépített műveletet, amelyekkel az oldal tartalmát menedzselhetjük. Ezek tradicionális adatbázis műveletek: objektum létrehozás, objektum módosítás, objektum törlés, kapcsolat létrehozása, kapcsolat törlése. Ezeket az egységeket később, a WebML gyakorlatban: WebRatio fejezetben mutatom be részletesen.

A beépített műveletek viselkedése és implementációja rejtve van a felhasználó elől, felhasználásra készen a modellben definiáltak.

Nem beépített művelet például a levélküldés.

6.2. Származtatás és személyre szabhatóság

A származtatás (derivation) egy modellezési fázis, ami két dolgot tesz lehetővé. Növeljük egy entitás tartalmát, úgy hogy attribútumokat adunk hozzá más entitásból importálva vagy más kapcsolódó objektumokból kiszámítva. Meghatározhatjuk az entitások vagy kapcsolatok populációját, a magába foglalt objektumok valamely jellemzői alapján.

A származtatás származtató lekérdezésekből áll, amelyeket a WebML OCL -el fejezhetünk ki. Származtathatunk entitásokat, kapcsolatokat és attribútumokat is.

A programomban néhányszor alkalmaztam attribútumok származtatását. Például a programvezetésben az elnök nevét az oktató nevéből származtattam. Vagy az személy alkalmazásánál (munkaviszony) az alkalmazó attribútumot is származtattam

a munkahelyi egység nevéből.

A személyre szabás három részből áll: hozzáférés szabályozás, oldal nézet hozzárendelés és lap személyre szabás.

Hozzáférés szabályozás alatt a be-, kijelentkező műveleteket értjük, amivel a felhasználót azonosíthatjuk.

A felhasználók egy vagy több csoporthoz tartozhatnak és mindegyik felhasználónak tartoznia kell egy alapértelmezett csoportba. Az oldal nézeteket csoportokhoz rendelhetjük, így bizonyos oldal nézeteket csak bizonyos csoportok tagjai érhetik el. Egy csoporthoz több oldal nézetet is rendelhetünk. A programomban az oktató csoporthoz az oktató oldal nézetet, a hallgató csoporthoz a hallgató nézetet, míg az adminisztrátor csoporthoz az adminisztrátor nézetet rendeltem. Tehát egy bejelentkezett oktató az oktatói oldal nézetet látja, míg egy hallgató a hallgatóit.

Lapok személyre szabása alatt azt értjük, hogy a lapon felhasználó vagy csoport függő tartalom jelenik meg. Ilyenre kell gondolni például, mikor valahol a felhasználó fő adatait illetve a hozzákapcsolódó egyéb adatait jelenítjük meg. Például a doktorandusz neptun kódját, felhasználónevét, email címét, mint főadatokat és képzéseit, szigorlatait, védéseit stb., mint egyéb adatokat. jelenítjük meg, a felhasználótól függően.

Létezik egy előre definiált anonymus hozzáférés, ami a nem bejelentkezett felhasználhatókat jelenti. Ők a nem védett oldal nézetekhez férhetnek hozzá.

7. WebML gyakorlatban: WebRatio

7.1. WebRatio

Webratio az első modell vezérelt fejlesztés környezet, ami lehetővé teszi Webalkalmazások modellezését és automatikus generálását. Az alkalmazások standard Java programok és bármilyen működési környezetbe installálhatóak: Tomcat, JBoss, Resin, IBM WebSphere, stb. Az alkalmazások bármilyen információs rendszerrel és adatbázissal kölcsönhatásba léphetnek, például: Oracle, DB2, SQL Server, PostrgreSQL, MySQL stb. A WebRatio az alkalmazás fejlesztési életciklusát menedzseli.

Egy alkalmazás létrehozása három lépésből áll.

Először definiáljuk az alkalmazásmodellt, ami a webalkalmazás három fő szintjét tartalmazza: adat, logika (hypertext), megjelenítés. A három szint különálló és független. Lehetőség van például létrehozni egy alkalmazást úgy, hogy csak a megjelenítési szintet változtatjuk meg, megtartva ugyanazt a logika és adat szinteket.

Az adatmodellt egy E-R diagrammon keresztül definiáljuk. A logikai modellt a WEBML vizuális nyelven keresztül határozzuk meg. Ez a nyelv a webalkalmazás minden logikai és funkcionális követelményét definiálja. A prezentációs modellt olyan szabályok segítségével definiáljuk, amelyek meghatározzák a szerkezetet, bármilyen interpreter nyelvben (rendering language): html, css, javascript, flash, pdf stb.

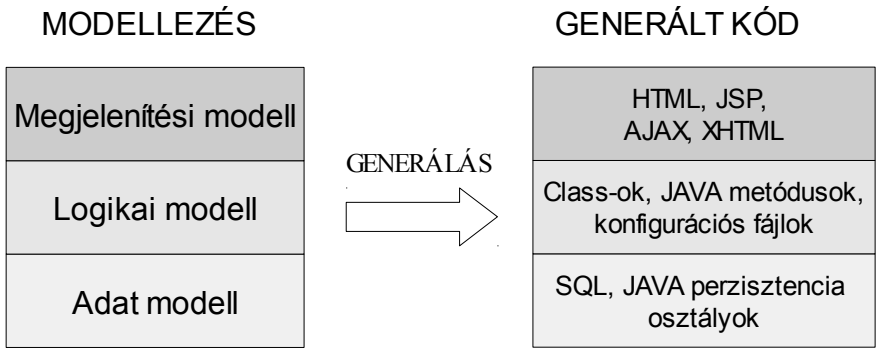
Az alkalmazásmodell meghatározása után a WebRatio-val legenerálhatjuk a Webalkalmazást. Ez a második lépés. A generált alkalmazás teljesen független a WebRatio-tól.

Az adatmodellt arra használjuk hogy legeneráljuk és fenntartsuk a fizikai adatbázis struktúráját. Az is lehetséges hogy a létező adatbázishoz kapcsolódva hozzuk létre automatikusan az aktuális adatmodellt. Azaz a fizikai adatbázist képezzük le logikaivá. Egy alkalmazáshoz egynél több fizikai adatbázis is tartozhat.

A logikai modellt arra használjuk, hogy Java osztályokat és konfigurációs fájlokat hozzunk létre, amelyek szükségesek az alkalmazás funkcióinak futtatásához: adatok olvasása és írása, tranzakciók, eljárások, figyelmeztetések, számítások stb.

A megjelenítési modellt dinamikus JSP oldalak létrehozására használjuk a kívánt

szerkezettel és interpreteres nyelvvel.



ábra 35: WebRatio: modellből kód

Harmadik lépés. A WebRatio által generált kód teljesen személyre szabható. A személyre szabhatóság generálási szabályokon nyugszik, nem a generált kódon. A WR egy nyílt fejlesztésű környezet, ami megengedi a felhasználónak saját komponensek és generálási szabályok létrehozását.

A WebRatio az Eclipse nyílt forráskódú, platformfüggetlen fejlesztői keretrendszeren nyugszik. Az Eclipse számos pluginnal bővíthető, a WebRatio is lényegében egy plugin készlet hozzá. A WebRatio 5.1 három operációs rendszerre telepíthető (MacOSX, Windows, Linux). A windowsos telepítőben a szoftveren kívül még egy Tomcat webszerver található.

A telepítéshez nem, de ahhoz, hogy elindíthassuk a WebRatio-t szükséges még a Java Development Kit 5.0-ás vagy magasabb verziója. JDK-ból én az 1.6.0 Update 18 x86-os verziót használtam. Ki tudja miért, ennek 64 bites változatával képtelen volt elindulni. Adatbázisként az Apache Derby-t használtam, mivel ez nem igényelt külön telepítést.

7.2. Használt technológiákról röviden

A JavaServer Pages (JSP) egyszerű és gyors megoldást kínál dinamikus webtartalmak létrehozására. Lehetővé teszi, hogy a Java és HTML kódot számos módon keverjük. Egy JSP fájl első meghívásakor önműködő servletté alakul, ez lefordítódik és bekerül a servlet tárolóba.

Ez a servlet tároló nálam az Apache Tomcat-ben található. Az Apache Tomcat az

Apache Software Foundation által fejlesztett nyílt forráskódú implementációja a Java Servlet és JavaServer Pages technológiáknak. Lényegében egy webszerver.

Az Apache Software Foundation fejleszti az Apache Derby-t is. Az Apache Derby egy nyílt forráskódú teljesen Java-ban implementált relációs adatbázis. Java, SQL és JDBC szabványokon alapul. A beépített JDBC driver segítségével bármilyen Java kódba beágyazható. Nagy előnye, hogy rendkívül kicsi, könnyen és gyorsan telepíthető és vehető használatba.

A Hibernate egy objektumrelációs perzisztencia és adatbázis lekérdező szolgáltatás. Elrejti a fizikai adatbázist, segítségével táblák helyett osztályokkal dolgozhatunk. A lekérdezésekben osztályneveket használunk.

Az AJAX (Asynchronous JavaScript and XML) egy meglévő szabványokból és technológiákból álló webfejlesztési technika, ami lehető teszi az adatcserét a szerverrel és a weboldal egy részének a frissítését anélkül, hogy újratöltenénk az egészet.

A Groovy egy gyors, Java platformra épülő objektumorientált dinamikus programozási nyelv. A Java erősségeire alapoz, illetve átveszi a Python, Ruby és Smalltalk erősségeit. Egy benne írt kód kinézetre nagyon hasonlít egy Java-kódra, ám nem teljesen ugyanaz. Dinamikus, tehát dinamikusan fordul Java bájtkóddá.

7.3. Az adatmodell

Miután összegyűjtöttük a követelményeket és tudjuk, hogy mit akarunk látni és készíteni, elkezdődhet az alkalmazás fejlesztése. Első dolgunk egy új projekt létrehozása után, az adatmodell létrehozása.

ábra 36:

Az adatmodell fő alkotóelemei az entitás, kapcsolat és specializáció (IS-A hierarchia). Ezen elemeket a 36. ábrán látható eszköztári ikonokkal helyezhetjük el a munkaterületen.

Az Entity ikonra kattintva majd a munkatérre hozhatunk létre entitást. Minden entitásban alapból szerepel egy OID attribútum kulcs tulajdonsággal. A létrehozott entitásoknak a properties fülön megadhatjuk a nevét, élettartamát és az attribútumaik

sorrendjét.

A neve UTF-8 karaktereket tartalmazhat, viszont az adatbázisban a generálás után az ékezetes karakterek "_"-lesznek helyettesítve, ami igen lerontja az olvashatóságát és használatát, ezért nem érdemes azokat használni. Viszont ekkor lokalizálni kell az alkalmazást. Itt is, mint minden programozási nyelvben beszédes neveket kell adni, hiszen minden entitással dolgozó egységnél ez a név mutatja majd, hogy melyik entitáson operál.

Az élettartam háromféle lehet: perzisztens (persistent), illékony session érvényű (volatile session scope) vagy illékony alkalmazás érvényű (volatile application scope). A session érvényű entitás példányok a session lejárta után törlődnek. Ilyet használunk például virtuális bevásárlókosárnál. Az alkalmazás érvényű entitás objektumok az alkalmazás leállításával tűnnek el. Az elkészített alkalmazásomban minden entitás perzisztens, nem volt szükséges ideiglenes entitások használata.

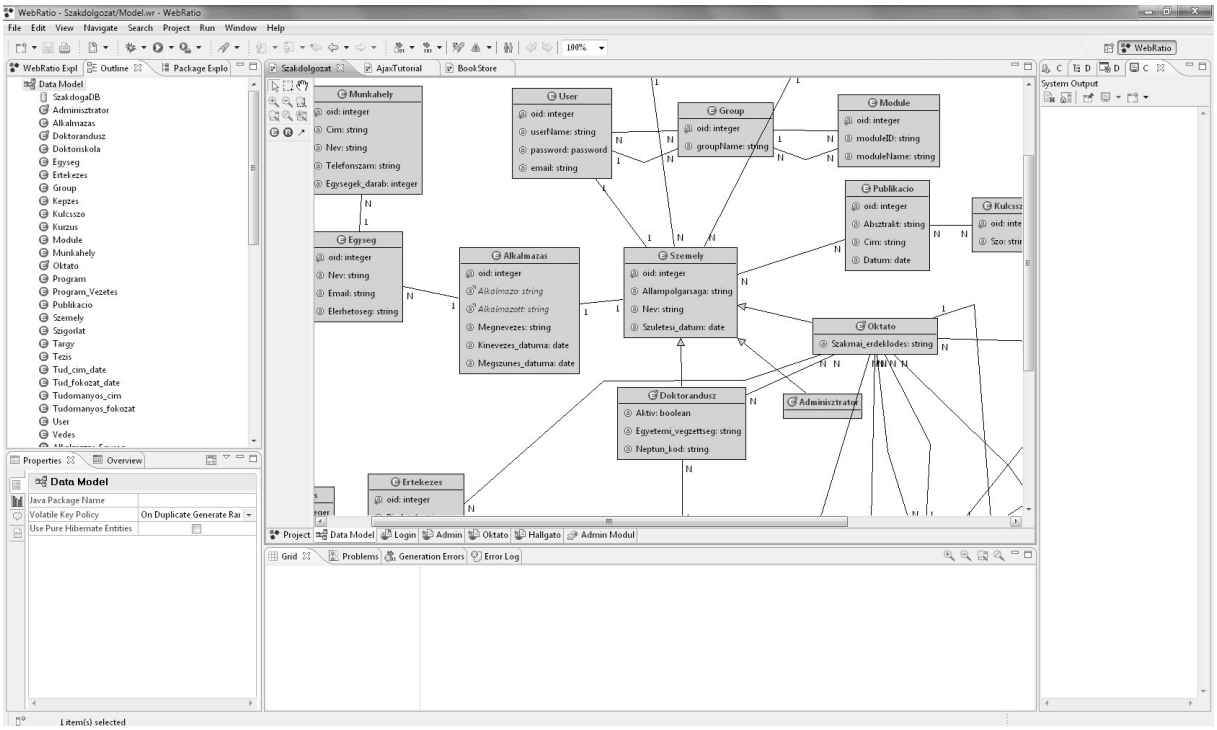
A properties fülön látható még az azonosító, szuper-entitás és származtatás tulajdonságok. Az entitás azonosítóját (Id) nem változtathatjuk meg, azt a WebRatio generálja egyszerű szabályt követve. Entitásnál "ent" plusz egy szám, kapcsolatnál "rel" plusz egy szám. Például "rel13" vagy "ent4". A számokat mindig növeli, amint elhelyezünk egy új entitást vagy kapcsolatot.

A szuper-entitás (super entity) értékét sem változtathatjuk meg. Akkor szerepel benne érték (egyébként üres), ha az entitás egy al-entitás, és ilyenkor a szuper-entitás neve szerepel benne. Mivel a Doktorandusz entitásom a Szemely-ből specializálom, ezért a Doktorandusz-nak itt a Szemely érték szerepel.

A származtatás (derivation) tulajdonságnak sincs értéke alapértelmezetten, és normális, nem al-entitásnál nem is változtatható meg. Amennyiben egy al-entitásról van szó entitást származtathatunk. Erre az tulajdonságra entitásoknál sosem volt szükségem.

Az entitás helyi menüjét előhozva adhatunk hozzá új attribútumot az "Add Attribute" menüpontot kiválasztva. Kiválasztva egy attribútumot azonosító, név, típus, származtatás, kulcs (key) és tartalom típus (content type) tulajdonságait láthatjuk a properties fülön.

A kulcs tulajdonság egy checkbox, amit ha kipipálunk, akkor az attribútum kulcsa lesz az entitásnak. Típusa lehet text, blob, string, time, integer és számtalan egyéb. A tartalom típusa tulajdonság csak a blob, text, string és url típusú attribútumoknál állítható. Ezzel határozhatjuk meg a MIME típusát ezeknek. Például, egy blob-ban mit fogunk tárolni: képet, videót, hangot, tömörített állományt.



ábra 37: Adatmodellezés WebRatioban

A származtatás mindig lehetséges. Ötféle származtatás létezik: egyszerű importált, konstans, komplex importált , kézzel származtatott és kiszámított attribútum. Hogy a származtatási tulajdonságokat beállíthassuk, új ablak nyílik meg a tulajdonság melletti szerkesztés ikonra kattintva. Az alkalmazásomban néhányszor alkalmazok egyszerű importált attribútumot. A megfelelő attribútumot a kapcsolatban álló entitásokon keresztül tallózva választhatjuk ki. Például az Alkalmazas és Egyseg entitásomat az Alkalmazas_Egyseg kapcsolat köti össze. Az Alkalmazas entitásban az Alkalmazo attribútumot az Egyseg Nev attribútumából származtatom. Ekkor a származtatás értéke ez lesz: "Self.AlkalmazasToEgyseg.Nev". A "Self" az aktuális attribútumra utal. A "Nev" az attribútum, ahonnan importáljuk az értéket. Míg a kettő között pontokkal elválasztva a kapcsolatszerep látható, ahogy eljutunk a "Self"-től a

"Nev"-ig. Tehát több kapcsolatszerepen keresztül, egymással közvetlenül nem kapcsolódó entitásokból is importálhatnék attribútumot.

Entitások közti kapcsolatot a Relationship ikont kiválasztva, majd először a forrás, majd a cél entitásra kattintva hozhatunk létre. Ezzel meghatároztuk a kapcsolat irányát is.

A properties fülön beállíthatjuk a kapcsolat nevét, a direkt kapcsolatszerep és inverz kapcsolatszerep nevét is. A két iránynak beállíthatjuk a maximum számosságát, ami lehet egy (one) vagy több (many). Az azonosító, forrás és cél entitás értékei nem változtathatóak. A User és Szemely entitások között mindkét számosság one - one, hiszen egy felhasználóhoz csak egy személy tartozhat és fordítva. Egy egységhez több alkalmazás, míg egy alkalmazáshoz csak egy egység tartozhat, így az AlkalmazasToEgyseg kapcsolatszerep számosság one, míg az EgysegToAlkalmazas-é many. Many-many értéket használok például az oktató és doktorandusz közti kapcsolatban, hiszen egy oktató több doktoranduszhoz kapcsolódhat és egy doktorandusz több oktatóhoz is.

A specializációhoz a Generalization nyilat kell kiválasztani az eszköztárból. Először kiválasztjuk, hogy melyik entitást specializáljuk, majd, hogy melyikből. Az Oktato, Doktorandusz és Adminisztrator entitásaim a Szemely entitásból specializálódnak, mivel mindegyikük rendelkezik névvel, állampolgársággal és születési dátummal, de sajátokkal is.

Egy új projektben az adatmodellben alapból szerepel három entitás: User, Group, Module. Ezek a 6.2.-es alfejezetben ismertetett személyre szabhatóság miatt szükségesek. A User a Group entitáshoz kettő (User_Group, User_DefaultGroup), míg a Group a Module entitáshoz szintén két relációval kapcsolódik (Group_Module, Groupe_DefaultModule).

Egy Module objektum moduleID attribútuma tartalmaz egy oldal nézet vagy lap vagy tartomány vagy műveleti egység azonosítót. Ennél fogva, egy oldal nézeten belül egy tartomány vagy lap megtekintését is, azon belül egy műveletet

végrehajtását is korlátozhatjuk. A Group entitáshoz Group_Module kapcsolaton keresztül több Module tartozhat, míg a Group_DefaultModule kapcsolaton keresztül csak egy. Azaz egy csoporthoz tartoznia kell egy alapértelmezett modulnak és számos modul tartozhat hozzá nem alapértelmezettként.

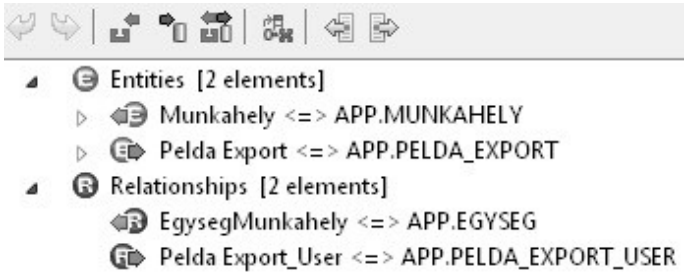
Egy felhasználó több csoporthoz tartozhat és lennie kell egy alapértelmezettnek. Ezt User_Group, User_DefaultGroup kapcsolatok teszik lehetővé.

Az egy csoportba tartozó bejelentkező felhasználó azokat az oldal nézeteket vagy lapokat láthatja (a publikusakon kívül), amelyek a Group_Module kapcsolaton keresztül meghatároztunk. Amikor a felhasználó bejelentkezik az oldalon az alapértelmezett csoportjának alapértelmezett moduljában eltárolt oldalra jut.

Mielőtt az entitásainkat az adatbázisba exportálnánk, be kell állítani annak az elérését. Kiválasztjuk az outline fülön az adatbázis ikonját majd áttérünk a properties fülre. Megadhatjuk a nevét. Nálam ez "SzakdogaDB". Kiválaszthatjuk a típusát egy listából, ami nálam az "Apache Derby". Az URL tulajdonság értékét a következőre állítottam: "jdbc:derby://localhost:1527/szakdoga;create=true". A "create=true" eredménye, hogy ha nem létezik ilyen nevű adatbázis, akkor létrehozza azt. Megadhatunk még felhasználónevet és jelszót hozzá. A jelszónál checkbox-al választhatjuk ki, hogy titkosítottan tárolja-e vagy sem. Kiválaszthatjuk még az alapértelmezett sémát egy listából, itt a WebRatio az "APP" sémát ajánlja, így ezt is használtam. Mindezek a mapping al-fülön találhatóak. További beállításokat végezhetünk a többi al-fülön.

Ha mindezek kész vannak exportálhatjuk az entitásainkat az adatbázisba, amit szinkronizáció segítségével tehetünk meg. Szinkronizálás során hasonlítja össze az adatbázist a mi adatmodellünkkel. Így ha valami plusz van az adatbázisban importálhatjuk az adatmodellbe, míg ha valami nincs benne akkor exportálhatjuk.

Szinkronizáláshoz az adatbázis ikon helyi menüjéből a Synchronize menüpontot kell kiválasztani amely egy új ablakot (Database Synchronizer) nyit. Az új ablakban tudjuk kiválasztani, hogy mit exportálunk illetve importálunk.



ábra 38: Database Synchronizer ablak

A 38. ábrán látható módon az ikonon elhelyezkedő balra nyíl jelképezi az adatmodellbe importálást, míg a jobbra nyíl az adatbázisba exportálást. Importálni és exportálni lehet mindent vagy egyenként is az egyes entitásokat, kapcsolatokat és entitások attribútumait.

A next gombra kattintva láthatjuk az exportálás SQL kódját, beállíthatjuk hogy tranzakcióként hajtsa-e végre illetve hogy a kódot elmentse és futtassa, csak futtassa vagy csak mentse. Megadhatjuk még a menteni kívánt SQL fájl nevét. Ha csak importálunk, akkor nincs exportálandó kód. A finish gombra kattintva hajthatjuk végre a műveleteket.

Az alkalmazásom adatmodellje 26 entitást, 86 attribútumot és 34 kapcsolatot tartalmaz.

Ezután olyan adatokat tölthetünk fel az adatbázisba, amelyek nincsenek kapcsolatban a személyre szabhatósággal. Ilyen adataim nekem a tudományos fokozatok és tudományos címek. Példa SQL kód a PhD fokozat feltöltésére:

```
INSERT INTO "APP"."TUDOMANYOS_FOKOZAT" ("OID", "MEGNEVEZES")
VALUES (1,'PhD');
```

Az adatbázissal kapcsolatban utolsó feladataink a logikai modell megalkotása közben/után lesz, ami alapján felvihetjük az adatbázisba a Group és Module objektumokat. Ez nincs automatizálva.

Például az alkalmazásomban létezik adminisztrátor csoport. Az adminisztrátori oldal nézet azonosítója "sv2". Ezt a következő SQL kóddal hozom létre:

```
INSERT INTO "APP"."MODULE" ("OID", "MODULEID", "MODULENAME")
VALUES (1, 'sv2', 'Adminisztráció');
INSERT INTO "APP"."GROUP" ("OID", "GROUPNAME","MODULE_OID")
VALUES (1, 'Admin', 1);
```

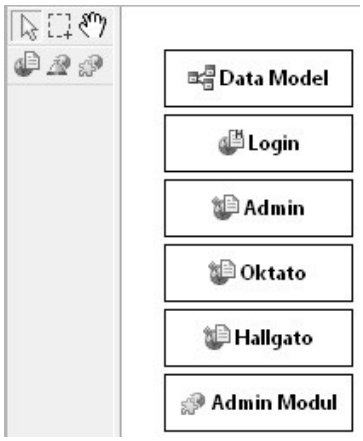
```
INSERT INTO "APP"."GROUP_MODULE" ("GROUP_OID", "MODULE_OID")
VALUES (1, 1);
```

7.4. A logikai modell

A munkatér alatt a project fület kiválasztva kezdhetünk neki a logikai modell létrehozásának. A properties fülön a projekt számos tulajdonságát állíthatjuk be vagy nézhetjük meg: általánosakat, a megjelenési szerkezetet (layout), biztonságot, statisztikát, stb. A statisztika szerint összesen 596 egységet használok az alkalmazásban, négy oldal és egy modul nézetben.

Háromféle nézettel rendelkezhetünk: oldal nézet (site view), szolgáltatás nézet (service view) és modul nézet (module view). Nézetek létrehozásához az eszköztáron lévő ikonokra kell kattintani, majd az új ablakban megadni a nevét. A szolgáltatás nézetre nem volt szükségem, ennek célja webszolgáltatás funkciók létrehozása webszolgáltatás műveletekkel. A modul nézetre szükségem volt, mivel minden összetettebb műveletet modullal valósítottam meg, így az oldal nézetet nem tesszük átláthatatlanná. Ilyen összetett művelet például az oktató vagy doktorandusz létrehozása és módosítása vagy törlése, tudományos fokozat hozzáadása és módosítása vagy törlése és még sok más. A modul nézetnek csak a neve változtatható, ezen kívül csak egy azonosítóval rendelkezik még.

Oldalnézetből négyet használok: Login, Oktato, Hallgato, Admin.



ábra 39: Project fül

Az oldal nézetnek is beállíthatjuk a tulajdonságait a properties fülön. Név, hivatkozás láthatósági politika, főoldal, védett, biztonságos, lokalizált és egyéni URL tulajdonságokat állíthatjuk be, továbbá megadhatjuk az oldalon található referenciapontok sorrendjét.

A Login oldal nézetem lett a főoldal, így a Home tulajdonságot kipipáltam. Az Admin, Oktato és Hallgato nézeteknél pipáltam ki a Protected tulajdonságot, mivel ezekre az oldalakra azt akarjuk, hogy csak az adott jogú felhasználók férjenek hozzá. A Secure tulajdonságot sehol sem jelöltem be, mivel a HTTPS-re nincs szükségem. A lokalizációt használom, kulcs érték párokat kell

megadni, a kulcs rész már ki van töltve. Egyéni URL-t sem állítottam be egyik oldal nézetnek sem. Ezek a tulajdonságok tartományoknál és lapoknál is megjelennek.

Követelmény, hogy minden védett oldalon ki lehessen jelentkezni, amit a Logout egységgel tehetünk meg.

A project fülön hozhatunk még létre context paramétereket. Öt már előre definiálva van. A context paraméterek értékét bárhonnan (hivatkozáson keresztül) elérhetjük, így ezek értékeit nem kell oldalról oldalra, hivatkozásokon keresztül küldenünk. A context paramétereket három egységgel kezelhetjük.

7.5. Az oldal nézet alkalmazott eszközei

ábra 40: Oldal nézet eszköztár

Ami minden egységre, tárolóra és hivatkozásra igaz az az, hogy mindnek adhatunk nevet. Hogy létrehozzunk egy elemet az eszköztárból kiválasztjuk azt, majd a munkatérre kattintva elhelyezzük.

Megkülönböztetünk műveleti (operation) és tartalmi (content) egységeket. A műveleti egységeket lapon kívülre lehet csak helyezni, míg a tartalmi egységeket csak lapra lehet helyezni.

A tartalmi egységekre, illetve a create, delete, modify, connect, disconnect, és reconnect műveleti egységekre feltételeket alkalmazhatunk: kulcs, attribútum és kapcsolatszerep. A connect, disconnect és reconnect egységekre ezen feltételek külön a forrás és külön a cél entitásra definiálhatók.

Kulcs alapján akkor szűrtem amikor egy konkrét objektumot akartam megjeleníteni. Például az oktató objektumát az OID alapján választottam ki és jelenítettem meg. Kapcsolatszerep alapján számtalanszor szűrtem. Így keresem meg az oktatóval annak a publikációit, fokozatait, alkalmazását stb.

Attribútum feltételt alkalmazok például felhasználó regisztrációnál a felhasználónevet illetően, nehogy azonos felhasználónévvel regisztráljunk be valakit.

A Login egységgel jelentkezhet be a felhasználó. Bemenő

hivatkozáson keresztül a felhasználónevet és jelszót adjuk át Kimenő hivatkozása KO és OK hivatkozás lehet. OK hivatkozásnak nincs értelme, hiszen sikeres bejelentkezésre a megfelelő oldalra ugrunk. Ha sikertelen a hivatkozás a KO hivatkozás céljához érkezünk. A Login oldalnézetben a bejelentkező egységet a KO hivatkozással egy Multi Message egységgel kötöttem össze. A KO hivatkozási paraméterének forrása egy konstans hibaszöveg, amit megadtam míg a célja a cél egység bemenő paramétere. Így ha rossz adatokkal próbálunk bejelentkezni egy hibaüzenetet kapunk majd.

A Logout egységgel jelentkeztethetjük ki a felhasználót. Érdemes referenciapontként megjelölni, így az alkalmazás menüjében mindig látható lesz majd, ahogy ezt mind a három védett oldalnézetemben tettem. Persze környezetfüggetlen hivatkozások segítségével is hivatkozhatunk rá, de értelmetlen. Beállításai között meg kell adni azt a publikus oldalnézetet, ahová a kijelentkezés után kerül a felhasználó.

Az Area tárolót tartományok létrehozására használjuk. Az adminisztrátori oldalnézeten előszeretettel használtam a koherens lapok összefogására. Minden tartományt referenciapont jelzővel illettem, hogy bárhonnan elérhetőek legyenek, azaz létrehozzák a menüt. Tartományt az oldalnézetre vagy egy másik tartományban hozhatunk létre. Környezetfüggetlen hivatkozások irányulhatnak rá.

A Page tárolóval hoztam létre a lapokat. A WebML-ből megismert jellemzőkkel bír. Minden tartományban van egy alapértelmezett. Környezetfüggetlen hivatkozások irányulhatnak rá és indulhatnak ki belőle. A lényegi tartalmat a felhasználóknak a lapokon jelenítjük meg. Azt, hogy a lapon hol jelenjenek meg az egységek a grid fülön állíthatjuk be a megjelenítési szerkezettől függően. Amennyiben Ajax technológiát akarunk használni a lapon, akkor azt a properties fül ajax al-fülén kell engedélyeznünk.

A Module egységet gyakran használtam, mivel az oldal nézetben a modul

nézetben létrehozott modulokat ezzel tudom felhasználni. A Module tulajdonságánál egyszerűen kiválaszthatjuk a kívánt modult a moduljaink listájából. Az automatikus hivatkozáson kívül bármilyen hivatkozás célja lehet. Forrásként szállító, KO és OK hivatkozásoknál szerepelhet. Ez az egység nem csak oldalnézeteknél, hanem szolgáltatás és modul nézeteknél is felhasználható.

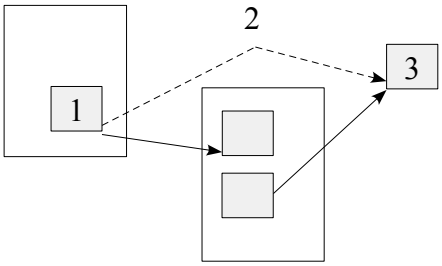
A Link azaz hivatkozás képviseli a környezetfüggetlen, környezetfüggő, szállító és automatikus hivatkozásokat. Akkor környezetfüggetlen, ha nem rendelkezik hivatkozási paraméterrel. A környezetfüggőt és környezetfüggetlent ennél fogva nem különbözteti meg a WebRatio, normál hivatkozásnak hívjuk. Hogy milyen típusú a hivatkozás, a beállításaiban adhatjuk meg. Lehet normal, transport vagy automatic.

Azt hogy a forrástól a cél egységig milyen paramétereket szállítson a Coupling tulajdonság beállításával adhatjuk meg. Létezik default coupling, de ezt többnyire ki kell kapcsolnunk és kézzel párosítani a forrás és cél paramétereket. A default coupling csak akkor használható, ha forrás és cél entitás megegyezik és csak OID-ot szállítunk a forrás egységtől a célig.

A normál hivatkozássokkal általában elindítunk valamit, amit számos szállító hivatkozással megtámogatunk. Például ha kiválasztunk egy doktoranduszt, akkor annak az OID-ja egy normál linken keresztül egy másik oldalra egy selector unit-ba jut, ahol lekérem a fő (Doktorandusz entitás) adatait. Ebből a selector unit-ból számos szállító hivatkozás megy további egységekben, hogy a doktoranduszhoz kapcsolódó adatokat megjeleníthessük. Ez egy folyamat számos hivatkozás segítségével végrehajtva.

Normál hivatkozásoknál engedélyezni lehet az Ajax technológiát. Az ajax al-fülön beállíthatjuk a tulajdonságokat. Például program vezetés létrehozásához és módosításához használom ezt. Az Auto-resizing, Open window és Closable tulajdonságok vannak bejelölve. Ha az új programvezetés vagy szerkesztés hivatkozásokra kattintok feljön egy ablak amiben kitölthetem az adatokat majd elküldhetem, mindezt az oldal újratöltése nélkül. Ennek az új ablaknak, ami ugye egy lap valójában a Page layout-ját WRDefault/Empty-re kell változtatni, hogy ne legyen az ablakon böngésző eszköztár, címsor és hasonlók.

Az adatok a hivatkozásokon keresztül a kiinduló forrás pontoktól lap szinten egyszerre szállítódnak. Ez azt jelenti, hogy egy szállító hivatkozással nem szállíthatok adatokat két normál link távolságú célhoz.



ábra 41: Paraméter szállítás

A 41. ábrán az 1-es egységtől a 2-es szállító hivatkozáson keresztül nem érkezik meg az adat a 3-as egységhez akkor, amikor a második lapon lévő normál hivatkozást használjuk, mert az már "elveszett".

Az OK linket követjük, ha egy művelet sikeresen hajtódott végre, míg a KO linket, ha sikertelenül. Paramétereket szállíthatnak, amelyeket coupling-al állíthatunk be. Fontos szerepet játszanak egy oldalnál. Szükségességük miatt zavaróan sok lehet belőlük lerontva így a terv kinézetét.

Az Admin és az Admin Modul nézetben rengeteg található belőlük.

A Set egységgel állíthatjuk be, a Get egységgel kérhetjük le, míg a Reset egységgel törölhetjük a context paraméterek értékeit. Azt hogy melyiket állítjuk be, kérdezzük le vagy töröljük az egységek Context Paramteres tulajdonságánál választhatjuk ki.

A Set és a Reset műveleti egység, míg a Get tartalmi. Set egységnek bemeneti hivatkozásai lehetnek, míg kimenetiek nem. A Get egységnél ez pont fordítva van. A Reset egységnek bármilyen bemeneti, kivéve automatikus hivatkozása lehet, míg a kimenete OK hivatkozás.

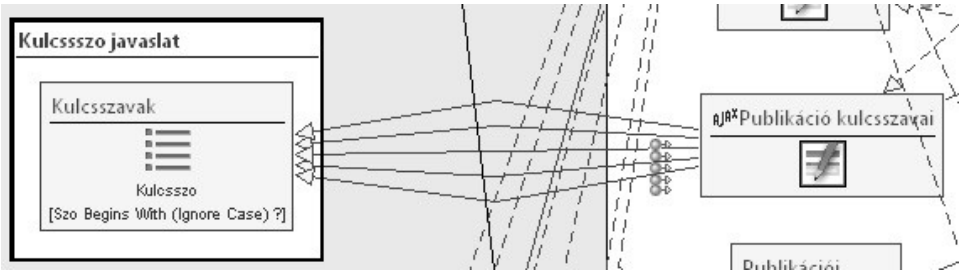
Az alkalmazásomban a Get és Set egységeket használtam, többnyire a doktorandusz és oktató oldalnézeteinél. Mindig amikor a felhasználó egy lapra navigál a Get egységgel szállító hivatkozásokon keresztül átadom UserCtxParam-ot, ami a bejelentkezett felhasználó User objektumának OID-ja. Így a felhasználóhoz kapcsolódó adatokat meg tudom jeleníteni.

Néha nehézségeket okoz, hogy több hivatkozásokon keresztül kell szállítani objektumazonosítót, ilyen esetekre is jól használhatóak a context paraméterek. Az alkalmazásomban egyetlen saját context paramétert definiáltam

KivalasztottDoktoranduszOID néven. Ennek a kiválasztott doktorandusz User OID-ját állítom be egy Set egységgel és a doktorandusz képzéseinél kérem vissza.

Az alap tartalmi egységek az 5.1. alfejezetben megismertetett módon működnek, így itt már nem részletezem azokat. Megadhatjuk hogy mely attribútumokat kívánjuk megjeleníteni, illetve ha több objektumot képes megjeleníteni akkor azok megjelenítési sorrendjét. Beállíthatjuk még, hogy az azonos tartalmakat szűrje a Distinct tulajdonság engedélyezésével, illetve beállíthatjuk, hogy maximum hány objektumot jelenítsen meg a Max Results tulajdonságnak értéket adva. Beállíthatjuk továbbá az egységből kiinduló normál és automatikus hivatkozások sorrendjét.

Az Ajax lehetőséget említeném még meg. Minden alap tartalmi egységet lehet "ajaxosítani". Én ezt a lehetőséget a beíró egység mezőinél használtam fel. Automatikus kiegészítést készítettem, hogy a publikációk kulcsszavaihoz javaslatokat adjon a már meglévők alapján.



ábra 42: Automatikus kiegészítés megvalósítása WebRatioban

Ehhez a beíró egység kiválasztott mezőjének tulajdonságainál, az ajax al-fülön engedélyezzük az Ajax Autocompletion-t illetve kiválasztunk egy normál hivatkozást, amin majd az adatok aszinkron módon szállítódnak. A javaslatok egy külön lapon lévő mutató egységből származnak (42.ábrán Kulcsszó javaslat lap és Kulcsszavak mutató egység). Ennek a lapnak módosítani kell a megjelenési szerkezetét, a Page layout és Unit layout értékének is WRDefault/AutoComplete-t kell kiválasztani. Azért hogy az automatikus kiegészítő helyesen működjön szűrni kell a mutató egység által megjelenített objektumokat, amit egy a mutató egységre alkalmazott begins with (ignore case) attribútum feltétellel teszünk meg. Csak azokat a kulcsszavakat listázza így, amelyek a begépelt tartalommal kezdődnek. A hivatkozásokon szállított paraméter azaz begépelt tartalom, az attribútum feltétel bemenő paramétere lesz.

A Multi Message egység hivatkozáson keresztül érkező adatok megjelenítésére szolgál. Képes tömböt is megjeleníteni. A felhasználót a végrehajtott műveletekről ezzel az egységgel informálom. Egy bemenő paramétere van, a Shown Messages és a bemenő KO és OK hivatkozásokban ez a hivatkozás paraméterek célja. Kimenő paramétere nincs.

A No Op Content egység semmit sem végez. Megjelenik az oldalon és kész. Én menünek használtam bizonyos helyeken. Például ilyen egységből indulnak az Új doktorandusz, Új oktató, Vissza és hasonló hivatkozások.

A Create, Delete és Modify egységek műveleti egységek. Bemeneti paraméterei az entitás attribútumai amin operál. Azt hogy melyik egységen operáljon a properties fülön az Entity tulajdonság beállításával adjuk meg. Automatikuson kívül, minden hivatkozás célja lehet. Kimenő hivatkozásai OK, KO és szállító típusúak lehet. A létrehozott műveleti egységeket védetté tehetjük, így a műveletet csak bizonyos csoportok felhasználói hajthatják végre.

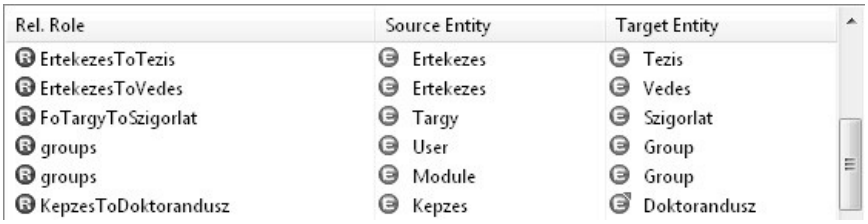
Egy objektum létrehozásakor az objektum azonosítót nem kell megadnunk, csupán a paraméterek szükségesek. Ha üresen hagyjuk a Create egység minden bemenő paraméterét, akkor egy objektum azonosítóval rendelkező "üres" objektum kerül az adatbázisba. A nem megadott attribútumoknál null érték szerepel. Ez az Avoid Blank Records tulajdonság engedélyezésével elkerülhetjük. Az egység kimenő paraméterei a létrehozott objektum OID-ja illetve attribútumai.

Objektum törléséhez a Delete egységnek csupán egy OID-ra van szüksége. Azt az entitás példányt törli, amelyhez az OID tartozik. Kimenő paramétere nincs.

A Modify egység egy objektum attribútumait módosítja. Ehhez meg kell adni a módosítani kívánt objektum OID-ját, illetve az új értékeket az bemeneti paraméterekhez párosítani. Ha valamelyik bemeneti paraméterhez nem tartozik érték, akkor azt nem módosítja. Kimenő paramétere a módosított objektum azonosítója.

Az adatmodellben lévő sok entitásból következik, hogy számos Create, Delete, Modify egységet kellett használnom.

A Connect, Disconnect, Reconnect is, mint a Create, Delete és Modify egységek műveletiek. Ezeknek nem entitásokat, hanem kapcsolatszerepet kell megadni (properties fül, Relationship tulajdonság) és majd az alapján határozza meg a forrás és cél entitást. A gyakorlatban lényegtelen, hogy melyik irányt választjuk, a lényeg hogy a kapcsolatszerepek a két entitás közti megfelelő kapcsolathoz tartozzanak.



ábra 43: Kapcsolatszerep kiválasztása ablak

Hivatkozásokkal kapcsolatban ugyanaz érvényes rájuk, mint a Create, Delete, Modify egységekre.

A Connect egység hozza létre a kapcsolatot két vagy több objektum között. Bemeneti paraméterei lehetnek a forrás és cél objektumok azonosítói, kapcsolat feltétel vagy attribútum feltétel. A kimenetiek a forrás és cél objektumok azonosítói.

Objektum azonosítókkal kapcsolom a felhasználót a csoportjához, az oktatót a publikációjához vagy a tudományos címhez vagy tudományos fokozathoz. Így kapcsolom a doktoranduszt a képzéséhez, a szigorlatához, védéséhez stb.

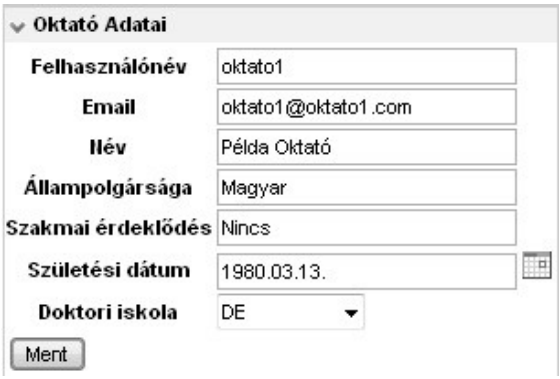
A Disconnect egységgel törölhetjük a fennálló kapcsolatot két objektum között. Bemenő és kimenő paraméterei, mint a Connect egységnek. Főként törlésekkor használom. Például amikor törlöm a doktorandusz védését, a védéshez kapcsolódó oktatókat el kell távolítani. Ugyanez a helyzet a szigorlattal vagy értekezéssel.

A Reconnect egység először törli, majd létrehozza a kapcsolatot. Olyan mintha először a Disconnect majd utána Connect egységet használnánk. Mivel a programomban minden merev szabályok szerint van a Reconnect egységre nem volt szükségem. Nem fordulhat elő például az, hogy egy védést, más doktoranduszhoz kapcsolok, se publikációt más oktatóhoz.

A Selector egységgel kérhetjük le objektumok tartalmát kulcs, kapcsolat feltétel vagy attribútum feltétel alapján. Bármilyen hivatkozás célja lehet. Ha lapon helyezkedik el, kimenő hivatkozása szállító típusú lehet, ha műveleti egységként

használjuk akkor KO és OK linkek is.

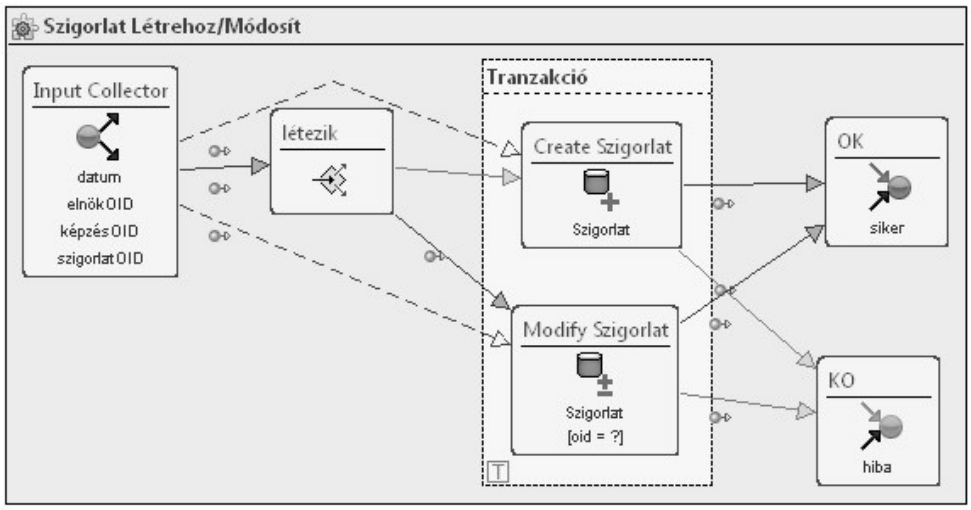
Adatok módosításánál mindig alkalmazott egység, hiszen a beíró egységben a mezőket először előtöltjük a régi adatokkal. Az adminisztrátor mindent tud módosítani, védések tartalmát, felhasználó adatokat, munkahelyet, egységet, programvezetést. Minden módosításhoz betöltöm a régi adatokat. De létrehozásokkor is alkalmazom, amikor valami választani való van. Például, hogy a doktorandusz képzése melyik programba tartozik. Az oktató és doktorandusz fokozatainak és címeinek a létrehozásához is ki kell választani egy listából a fokozat nevét vagy címét, amit szintén a Selector egységgel töltök elő.



ábra 44: Előtöltött oktató adatok az oktató szerkesztés lapon

Az Is Not Null egység egyszerű. Egy bemenő paraméter van az input, ami egyben a kimenő is. Eldönti, hogy a bemenet null-e vagy sem. A tulajdonságai között bekapcsolhatjuk azt, hogy az üres String-et is null-nak nézze. Be és kimenő hivatkozás bármi lehet. Ha a bemenet null, akkor a KO linken, míg nem null esetén az OK hivatkozáson haladunk tovább.

A moduljaimban használtam leggyakrabban, mivel a létrehozást és módosítást általában egybe vettem. Ha módosításról volt szó, ezzel az egységgel teszteltem, hogy átadtam-e létező objektum azonosítóját. Az OK ágon így a módosítás következett, míg a KO ágon a létrehozás.



ábra 45: Modul szigorlat létrehozására és módosítására

A 45. ábrán látható modul a következő módon működik. Az Input Collectorból egy OK link megy a létezik Not Null egységbe. Ezen az OK linken a szigorlatOID paramétert adom át. Ha van OID, akkor a Modify Szigorlat egységhez a OK linken keresztül jutunk tovább, amivel átadtam az egységnek a szigorlat OID-ját, míg a szállító linken keresztül a többi adatot (datum, elnökOID, képzésOID). Ha a módosítás sikeres az OK linken haladunk tovább amiben az OK Collector egységnek a siker paraméterébe a "Szigorlat modositasa sikeres." üzenetet adom át. Ha sikertelen akkor a KO Collector hiba paraméterébe a "Szigorlat modositasa sikertelen." üzenetet adom át.

Ha nem volt szigorlat OID, akkor a KO hivatkozás a Create Szigorlat egységhez vezet, ami szállító hivatkozáson keresztül szintén megkapta a többi adatot. Ha sikeres a létrehozás az OK, ha sikertelen a KO linken haladunk tovább és a "Szigorlat letrehozasa sikeres." vagy a "Szigorlat letrehozasa sikertelen." üzenetet adjuk tovább.

A Script egyég. Amit az előre definiált egységekkel nem tudok megoldani ezzel igen. Ez az egység groovy szkriptet futtat le. Bemenő és kimenő paramétereit érdemes a szkriptben meghatározni (WebRatio így ajánlja), de az egység helyi menüjével is megtehetjük. Háromszor alkalmaztam. Az input String kisbetűsítésére, a publikációhoz megadott kulcsszavak tömbbé alakításához, illetve a tömbből

visszaalakításhoz. Egy publikációhoz öt kulcsszó adható meg.

A tömbből visszaalakítás szkriptje:

```
#input String[] tomb
#output String ksz1, String ksz2, String ksz3, String ksz4, String ksz5
if( tomb.size() > 0){
      ksz1 = tomb[0]
} else{ ksz1 = null }
if( tomb.size() > 1){
      ksz2 = tomb[1]
} else{ ksz2 = null }
if( tomb.size() > 2){
      ksz3 = tomb[2]
} else{ ksz3 = null }
if( tomb.size() > 3){
      ksz4 = tomb[3]
} else{ ksz4 = null }
if( tomb.size() > 4){
      ksz5 = tomb[4]
} else{ ksz5 = null }
return ["ksz1":ksz1, "ksz2":ksz2, "ksz3":ksz3, "ksz4":ksz4, "ksz5":ksz5]
```

A Parameter Collector egység, mint a nevében szerepel paramétereket gyűjt. Bemeneti paraméterei a kézzel hozzáadott Collector paraméterek. Több forrásból származó paramétereket több célnak szállít el. A Hallgato nézetnél alkalmaztam, ahol a képzés OID-ját több Selector egységnek is tovább kellett adnom egyszerre.

7.6. A modul nézet alkalmazott eszközei

ábra 46: Plusz elemek az eszköz-tárban

Ennek a nézetnek az eszköztára tartalmazza az oldal nézet eszköztárában fellelhető minden elemet, de rendelkezik néhány újjal is.

A Operation Module újrafelhasználható műveleti egységek és műveleti csoportok (Operaton Groups) gyűjteménye. Tartalmi egységet nem tartalmazhat. Ilyen modulokat hoztam létre az összes kicsit is összetettebb művelethez: oktató hozzáadása/módosítása, doktorandusz hozzáadása/módosítása, doktorandusz törlése, oktató törlése, publikáció létrehozása/módosítása stb. Az ilyen modul műveleti egységnek számít az oldal nézetben, tehát lapra nem helyezhető. Modulba helyezhető modul egység. Input, OK és KO Collector-t tartalmazhatnak. Input Collector kötelező bele, mivel onnan indul el

a műveleti lánc.

Az Content Module újrafelhasználható tartalmi egységek gyűjteménye. Műveleti egységet nem tartalmazhatnak és az ilyen modulok csak lapra helyezhetőek el. Egyszer alkalmaztam, a publikáció lekérdezett kulcsszavainak, amit tömbben kapok meg, öt paraméterré szortírozásához. Input és Output Collector-t tartalmazhat

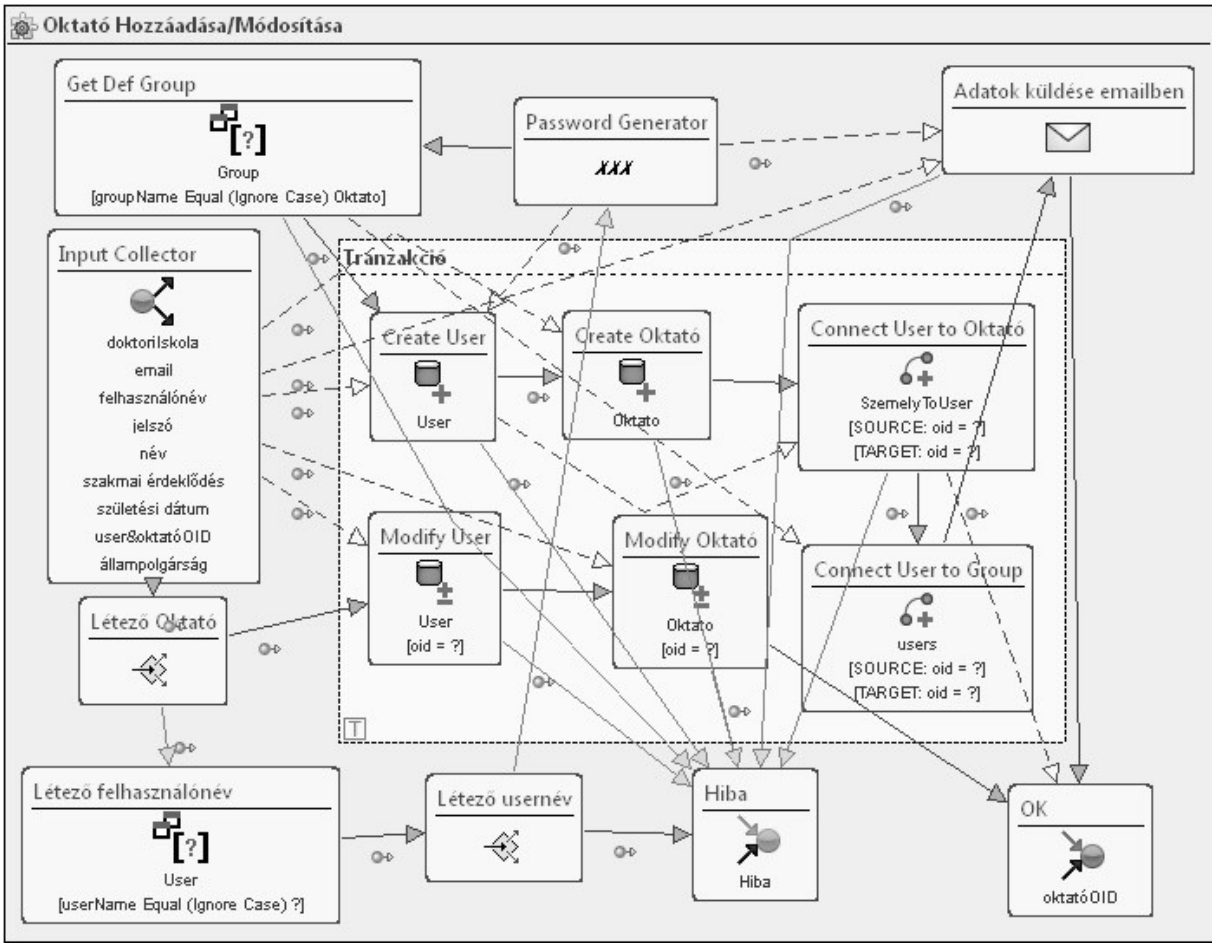
A Hybrid Module-t sosem használtam, az előző két modul keveréke, tehát minden szerepelhet benne.

A Content Module-nak, mint műveleti egységnek sikeres és sikertelen kimenetele lehet. Tehát ha a műveleti láncban a hiba hivatkozásokat a KO Collector-ba, a műveleti lánc végén lévő OK hivatkozást pedig az OK Collector-ba kötjük. Az OK Collector a sikeres műveleti lánc vége, míg a KO Collector a sikertelené. Adhatunk hozzájuk paramétereket, amik a modul kimenő paraméterei lesznek. A modulból induló OK link csak az OK Collector paramétereit míg a KO csak KO collector paramétereit látja, azaz szállíthatja.

Az Input Collector a modul bemenő paramétereit gyűjti össze. Paramétereket mi adhatunk hozzá. Mind a háromféle modulban szerepelhet.
Az Output Collector a modul kimenő paramétereit gyűjti, amiket kézzel adhatunk hozzá. Operation Module-ban nem szerepelhet.

Az Operation Group nem feloszthatóan és automatikusan végrehajtott műveletek sorozata. Tehát vagy az összes művelet végrehajtódik benne sikeresen vagy az összes a hibáig végrehajtott műveletet visszagörgeti. Tartalmazhat OK és KO Collector-t, én sohasem raktam bele, hiszen a modulban, amiben használtam már volt. Minden egynél több műveleti egységet használó folyamatban használtam, hiszen nem akartam, hogy az adatbázisba félkész adatok, mondhatni szemét kerüljön bele. Van egy Transaction tulajdonsága, ami alapértelmezetten be van kapcsolva, ekkor az Operation Group egy tranzakció. KO hivatkozás indulhat ki

belőle.



ábra 47: Oktató Hozzáadása/Módosítása modul

A Password egység jelszavak generálására alkalmas. Bemeneti paramétere nincs. Kimenete a generált jelszó. Tulajdonságainál a generált jelszó hosszát állíthatjuk be. Kis és nagy betűt és számokat tartalmazó jelszavat alkot. Ezzel generálom az oktató és doktorandusz regisztrációnál a felhasználók jelszavát illetve a doktorandusz neptun kódját.

A Mail egységgel email-t küldhetünk. Használatához be kell állítani az SMTP kiszolgálót a tulajdonságainál. Megadhatjuk még a formátumát, hogy sima szöveg vagy HTML legyen. A karakterkódolás, továbbá egy tárgy és body sablon is beállítható. A sablonban megadhatunk változókat $$valtozónév$$ formában,

amelyek bemeneti paraméterekként megjelennek az egységnél, illetve a levélben majd behelyettesítődnek a paraméterek értékeivel. Bemeneti paraméterei még a CC, To, From, Subject stb. paraméterek, tehát azok amik egy emailnél szokásosak és beállíthatóak. Hozzáadhatunk csatolmány paramétert is. Ezzel az egységgel küldöm el a regisztrációról a levelet az oktatónak és doktoranduszknak, benne a főbb adataikkal. A belőle induló KO linken keresztül a Failure Reasons paraméter adhat információt a levél elküldésének sikertelenségéről.

A Math egység egyszerűbb matematikai műveletekre használható. Hozzáadhatunk változót, ami a bemeneti paramétere lesz. A matematikai kifejezést a Default Expression tulajdonság értékeként adjuk meg. A kimenete boolean, integer , float és decimal típusú lehet. Kimeneti paraméterének a neve Result, és ebbe kerül a kiszámított érték.

Kétszer használom. Ha a munkahelyhez hozzáadunk egy egységet, akkor a munkahely, egység darabszám attribútumához hozzáadok egyet. Ha törlünk egy egységet csökkentem eggyel. Így tartom számon,hogy egy munkahelyhez hány egység kapcsolódik.

A Switch egység a bemeneti paraméter értékétől függően eldönti, hogy melyik hivatkozáson haladjon tovább. A Case tulajdonságánál adhatunk hozzá case értékeket. Az case értéket a kimenő OK hivatkozás Code tulajdonságánál kell kiválasztani. Amelyik hivatkozásnál code értéket nem adtunk, az lesz a default ág. Ha a bemenet értéke egyezik a case értékkel akkor azon a linken haladunk tovább, amelyik ezt a case értéket képviseli, ha nincs egyezés a default linken.

Egyszer, kétszer használtam. Például egy szigorlathoz két melléktárgy tartozhat csak. Így ha tárgyat akarok hozzáadni, meghatározom, hogy kettőnél több-van már-e vagy sem, majd egy Switch egységgel eldöntöm hogy az igaz vagy hamis hivatkozáson haladjak tovább.

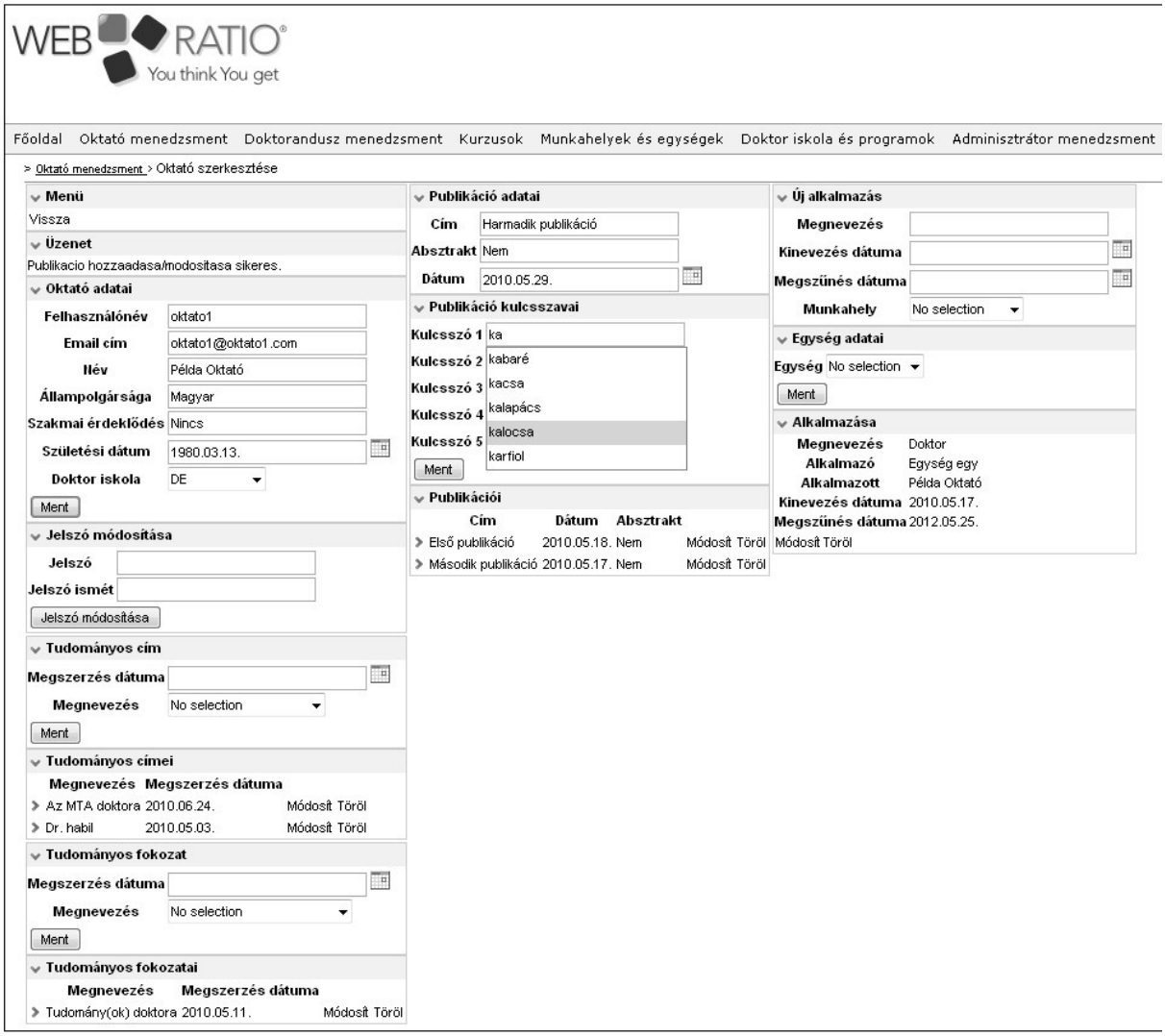
A Loop egység egy tömb elemeit iterálja. Bemenő paramétere tehát egy tömb. Kimenő paramétere a tömb következő eleme. Két OK hivatkozás indul ki belőle. Az

egyik a next a másik az end ág. Ezt a hivatkozások Code tulajdonságánál állíthatjuk be. A következő elemet mindig a next linken adja tovább, ha pedig nincs több elem a tömbben az end linken haladunk tovább. Természetesen a next ágon lévő műveleti láncnak a Loop egységben kell végződnie, hogy folytatódhasson az iteráció. Megadható neki második tömb is, amit mellékesen iterál addig, amíg a fő tömböt.

Tudományos címek törlésénél alkalmaztam például. Azok OID-jait, ami egy tömb, átadtam a Loop egységnek, így egy-egy OID-al tudtam foglalkozni.

7.7. A megjelenítési modell

Az alkalmazás külső megjelenésére nem fektettem hangsúlyt. Az alap WebRatio sítlust és WRDefault megjelenítési szerkezetet használtam. Ezeket a legapróbb szintig konfigurálni lehet. Tehát beállíthatom külön-külön projektre, oldalnézetre, tartományra, lapra és megjelenítő egységekre. Lehetőség van azonban saját stílusok és megjelenítési szerkezetek létrehozására.



ábra 48: Kép az alkalmazásból

8. Összefoglalás

Az előző fejezetekben leírtak alapján egy képet kaptunk, hogy mire képes és mikre lehet képes a WebRatio és WebML. Hatásosan segíti egy nagy webalkalmazás fejlesztését. Az is egyértelmű, ami a WebML céljában is szerepel, hogy nem kis honlapok készítésére alkották meg a modellező nyelvet és a fejlesztői környezetet, hanem komplex, adat intenzív webalkalmazások fejlesztéséhez..

A WebML széles skálán mozgó modelleket és eszközöket nyújt nekünk és biztosítja a paletta bővíthetőségét. Az adatmodellezést nagyon jól kezelhetővé és átláthatóvá tették, komplex adatbázist tudunk létrehozni SQL ismeretek nélkül, a lényegre, az adatokra koncentrálhatunk. Hypertext modellezéskor pontosan tudjuk, hogy mit fogunk kapni végeredményében, láthatjuk, hogy mely oldalakról hova lehet eljutni és hogy azok milyen elemeket fognak megjeleníteni. A modellezett webalkalmazásunknak bármilyen kinézetet könnyen és gyorsan elkészíthetünk.

A nyújtott eszközökkel teljesen a tervezésre, a feladatra és a követelményekre koncentrálhatunk az implementációt a beépített kódgenerátor elvégzi, így jelentősen gyorsítva a fejlesztést.

A fellelhető oktatási segédanyagok terén még csiszolandó a WebRatio, ez vélhetően abból fakad, hogy nem elterjedt. Tanulása eleinte nehéz, csakúgy, mint a vétett hibáink felderítése. Sok gyakorlás és a dolgok működésének felfogása után viszont könnyen megy a fejlesztés. Minden egyes eszközét nem használtam ki, ám így is látszik, hogy nagyon jól átgondolt és hatalmas lehetőségek vannak benne.

Számomra az alkalmazás fejlesztés a WebRatio-ban igen hamar monotonná vált, nem találtam benne új kihívást, amin gondolkodni kellett volna. Ilyen monotonitást más programozási nyelvben fejlesztéskor nem tapasztaltam. Egy másik negatív tulajdonsága a fejlesztési stílusa. Folyton váltani kell az egér és billentyűzet használat között.

A termék támogatása kiváló. Készséggel segítenek akármilyen kéréssel is fordulunk hozzájuk, mindezt igen rövid időn belül.

9. Irodalomjegyzék

Nyomtatott dokumentumok

Lerner, M. Reuven (2001): JavaServer Pages. In: Linuxvilág, II. évfolyam 5. szám, 70-73. p.

Sommerville, Ian (2007): Szoftverrendszerek fejlesztése. Második bővített kiadás. Bp., Panem.

Elektronikus dokumentumok

Ceri, Stefano - Fraternali, Piero - Bongio, Aldo: Web Modeling Language (WebML): a modeling language for designing Web sites.
http://www9.org/w9cdrom/177/177.html

Ceri, Stefano - Fraternali, Piero - Bongio, Aldo - Brambilla, Marco - Comai, Sara - Matera, Maristella (2003): Designing Data-Intensive Web Applications.
http://books.google.hu/books?id=yI31Mi738p8C&printsec=frontcover#v=onepage&q&f=false

WebML Courseware
http://www.webml.org/webml/page97.do

Internetes kutatás

Apache Derby
http://db.apache.org/derby/

Apache Tomcat
http://tomcat.apache.org/

Groovy
http://groovy.codehaus.org/

Hibernate
http://www.hibernate.org/

Model, View, Controller
http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
http://jankajanos.spaces.live.com/Blog/cns!C3E2695FC6F7B0A4!394.entry

WebRatio
http://www.webratio.com/

WebRatio Wiki
http://wiki.webratio.com/