Szakdolgozat

Imre Gábor

Debrecen
2009

Debreceni Egyetem

Informatikai Kar

CMS rendszer fejlesztése XML alapokon

Témavezető:
Dr. Adamkó Attila
egyetemi tanársegéd

Készítette:
Imre Gábor
Programtervező-informatikus

Debrecen
2009

1. Bevezetés 4
2. A CMS rendszer technológiai alapjai 6
2.1 XML részletesebben 6
2.2 A tervezett rendszer elvi alappillérei 6
3. A CMS rendszer architektúrája 8
3.1 Adatbázis 11
3.1.1 Konkrét adatbázisok 14
3.1.2 Összegzés 20
3.1.3 Adatbázis szerkezet 21
3.2 Szerver oldali program 27
3.2.1 Lekérések kliens oldalról és a generált dokumentum 27
3.2.2 Adatbázis kommunikáció Xquery-vel 33
3.2.3 XPath és XQuery 33
3.2.4 XQuery gyártása a kliens oldali lekérésekből 39
3.3 Kliens oldali követelmények 45
3.3.1 Az XSL és az XSLT 45
3.3.2 XML transzformáció böngészővel 49
3.3.3 XML transzformáció a szerver oldalon 50
4. Összefoglalás 51
5. Szakirodalom 53

1. Bevezetés

Napjainkban a rohamosan fejlődő informatikának egyik alap pillérévé vált a webes technológiák gyűjteménye. Korábbi időkben csak egy speciális területre kitalált szabványok és módszerek együttese mára minden alkalmazásba, fejlesztési eszközbe beférkőzött és a fejlődése minden korábbi várakozást is felülmúlt. Az internet most már nem csak otthonainkba, hanem minden elképzelhető helyre beférkőzött, így teremtve meg az igényt az egyre fejlettebb és szélesebb körben alkalmazható technológiák létrejöttére.

Nem fér kétség hozzá, hogy jelenleg a leginkább elfogadott szabvány, amelyet egyetlen nagyvállalat vagy szervezet sem kérdőjelezett meg, az XML. Ez a betűszó az informatikában már olyannyira elterjedt, hogy nincs az-az ember, aki ne találkozott vagy fejlesztett volna vele. Jelentőségét nem lehet eléggé kiemelni. Felhasználási területe szinte az egész informatika, a hozzá kapcsolódó szabványok és eszközök száma hatalmasra ugrott az elmúlt időkben. A technológia nem csak az adattárolásnak egy szabványosított módját jelentette, hanem a hozzá kapcsolódó lekérdező nyelvek, validálási módszereket is definiálta. Ezen dolgozat során elég sokat fogunk érinteni ezekből.

A webes rendszerek az információ könnyebb áromoltatása végett terjedtek el és egyik legfontosabb fogalmuk a tartalom. Ennek kezelésére jelent meg egy újabb betűszó a CMS (Content Management System), azaz a tartalomkezelő rendszer. Ezen rendszerek feladata a tárolt adataink kiszolgálása a felhasználó felé, ill. azok frissítése, bővítése, rendbetétele. Manapság igen jelentős számban juthatunk hozzá nyílt forráskódú tartalomkezelő rendszerekhez, melyek nagy része igen magas szintű szolgáltatások nyújtására képes. Alapvetően a legtöbb egy hagyományos vonalat követ, melyben egy adatbázisból a CMS rendszer adatokat szolgáltat megjeleníthető formában (általában HTML) a felhasználó felé, mindeközben jogosultságokat kezel, munkamenetet biztosít, azonosításra ad lehetőséget stb.

Ez a dolgozat egy kísérlet, egy olyan tartalomkezelő rendszer létrehozására, mely nagymértékben használja fel az XML-hez kapcsolódó szabványokat és eszközöket, mindemellett próbát tesz az eddig megszokott elrendezés egy újfajta szétosztására, ahol kissé felborítjuk az eddig megszokott sémát a tartalomkezelő rendszerek terén.

Elsősorban az utóbbi időben megjelent és előtérbe került technológiákra építünk és elsősorban arra, hogy már elég hatékony feldolgozó algoritmusokkal rendelkeznek az ezeket felhasználó

rendszerek (böngésző vagy adatbázis kezelő rendszer). Így lesz lehetőségünk arra, hogy a végén egy olyan architektúrát alkossunk, amely gyakorlatban is működhet.

2. A CMS rendszer technológiai alapjai

2.1 XML részletesebben

A már többször elhangzott betűszó, az XML azaz az eXtendible Markup Language egy olyan HTML-hez hasonló leíró nyelv, mely tartalom tárolásának szabványosítására jött létre. Építőköveit nevéből adódóan mi magunk adjuk meg és segítségével képesek vagyunk szabványosan elkülöníteni az adatot a megjelenéstől. Használata elsősorban az internetes vagyis webes területeken terjedt el először, de most már mindenhol megtalálható.

Alapvetően egymásba ágyazott „tag”-ekből, vagyis címkékből áll, melyeket megnyitunk, elemeket (más címkéket) helyezünk el, majd bezárunk. Így hoz létre egy hierarchikus strukturáltságot magán belül. Legegyszerűbben egy fával reprezentálható ez, ahol egy szinten tetszőleges számú és nevű elem lehet. Magukat a neveket mi magunk határozhatjuk meg, így tetszőleges adatot képesek vagyunk letárolni úgy, hogy az bizonyos mértékig akár emberi szem számára is tökéletesen olvasható. Ettől függetlenül elsősorban a gépnek szánjuk, hiszen nagyobb méretekben már a gyakorlott felhasználó számára is átláthatatlanná válik egy-egy részlet.

Az XML adatokat dokumentumokban tároljuk, melyeket egy fejléc és egy gyökér elem (tetszőleges névvel) tesz teljessé. Állományként általában .xml kiterjesztéssel találkozhatunk velük. Egy ilyen dokumentummal szemben két követelményünk lehet, az egyik a jól formáltság, mely esetében megfelel a szintaktikai szabályoknak, és a helyesség (valid), mely esetében megfelel egy előre definiált szemantikai szabályrendszernek. Ezeket a szabályrendszereket DTD vagy XML Schema segítségével tudjuk megadni és így kikötéseket tudunk definiálni egy adott dokumentumhoz.

Az XML-hez rengeteg kapcsolódó technológia tartozik, melyek nagy részét a W3C nevű szervezet specifikált (mint ahogy az XML-t magát is). Ezen dolgozat során ebből elég sokat fel fogunk használni, főleg azokat, melyek egy dokumentum bejárását segítik.

2.2 A tervezett rendszer elvi alappillérei

Ahhoz, hogy neki kezdjünk egy ilyen rendszer elkészítéséhez először is tisztázni kell az elvárásokat. Ebből jelen esetben nincsen túl sok, ugyanis a CMS rendszer definíciója szerint

tartalmat kell tudnunk kezelni, vagyis lehetőséget kell biztosítanunk arra, hogy a rendszerünk segítségével a végfelhasználók képesek legyenek tartalmat hozzáadni, azt módosítani és mindezt megnézni. Természetesen ezek a dolgok még kevesek egy modern tartalomkezelő rendszer tudásához, így szükségünk lesz munkamenet kezelésre, felhasználó azonosításra és még sok minden másra is, de a legfontosabb a tartalomkezelés.

Ennek ellenére szeretnénk elvárni a rendszerünktől, hogy képes legyen olyan alapvető funkciókat ellátni, melyek szerepelnek más modern tartalomkezelő rendszerekben. Így pl. el akarjuk érni azt, hogy lehessen egyszerűen híreket kihelyezni egy adott oldalra, fórum vagy legalább üzenő fal jelleggel tartalmat felvinni, esetleg wiki szerűen tartalmat közzétenni és szerkeszteni.

Jobban belegondolva, ezek mind ugyanazt a gondolkodást követik és könnyen lehet találni egy olyan megoldást, mely egy csapásra lehetőséget nyújt mindegyikre. Szimplán a tartalomkezelés megfelelő átgondolásával és a megjelenítés tartalomtól való leválasztásával elérhetjük mindezt.

Abból kiindulva, hogy nagyrészt csak szöveges adatokkal fogunk dolgozni (plusz dologként leimplementálhatunk dokumentumok, képek kezelését is), nagyjából kijelenthetjük, hogy a tárolt adatok függetlenül azok felhasználásától uniformizálhatók adatszerkezet szintjén. Ez azért fontos dolog számunkra, mert az XML dokumentumok jellegét kihasználva egy általános csomópontot fogunk definiálni, amely bármilyen jellegű tartalom tárolására alkalmas lesz, úgy, hogy mellette még tárolunk minimális meta információt, amely majd a megjelenítéséhez lesz szükséges.

Ezt annak reményében tesszük, hogy így tetszőleges információ tárolás és az azzal való kommunikáció megoldható. Mindemellett ahogy elhangzott, teljes mértékben leválasztjuk majd a megjelenítést az adathalmazzal való munkától, így nem is nagyon képzelhető el máshogy az adatok tárolása, mint univerzális adatszerkezetekben.

3. A CMS rendszer architektúrája

Egy webes alkalmazást általában három fizikailag különálló részre tudunk elkülöníteni a gyakorlatban, logikailag pedig tetszőlegesen sok komponensből állhat. Mint korábban említésre került, jelen rendszerünknél a hagyományostól eltérő szétosztásban fognak szerepelni a feladatok bizonyos mértékben.

Fizikailag megkülönböztethetünk egy kliens oldalt, ami a végfelhasználó számítógépét jelenti. Ez mind teljesítményben, mind tudásban változót jelent, így igen komoly feladatot jelent az itteni feladatok megfelelő implementálása, hiszen nem csak számítógépük konfigurációjában, hanem a böngészőben, operációs rendszerben és azoknak éppen aktuális beállításaiban is különbözhetnek. Ettől függetlenül jelen dolgozatban egy igen komoly feladat hárul majd erre a részre.

Rendelkezünk még egy fizikailag külön álló szerver oldali programmal is, mely egyfajta hidat képez a később említésre kerülő adatbázis és a kliens oldal közt. A szerver tudásával kapcsolatban lehetnek komolyabb elvárásaink, így nagyjából fix környezetben tudunk úgy dolgozni, hogy biztonsággal hordozható maradjon a programunk. Mivel ez főként egy átjátszó állomás a lekérésekhez, ezért nem árt ha gondolunk a teljesítmény tényezőkre is a későbbiekben, hiszen ennek a programnak több lekérést is képesnek kell lennie lekezelnie egy időben. Azonban nem egy ipari stabilitású rendszert tervezünk, ezért a kód optimalizálása nem képezi majd szerves részét a dolgozatnak.

A harmadik különálló részünk az adatbázisunk lesz, mely vélhetően ugyanazon a gépen fog tartózkodni mint a szerver oldali programunk (de később láthatunk arra is egy megoldást, hogy egy újabb komponens kapjon még helyet ugyanezen a gépen). Mivel elegáns és jól felhasználható rendszer létrehozása a célunk, ezért az adataink tárolását nem sima szöveges file-okkal kívánjuk megoldani, hanem választunk egyet a rengeteg jelen lévő ingyenesen használható adatbázisok közül. Hogy pontosan milyen is lesz ez, azt az Adatbázis fejezetben fogjuk részletesen kitárgyalni.

Mivel elsősorban XML technológiákkal dolgozunk és lehetőség szerint minél több X kezdőbetűs eszközt akarunk használni (ettől csak egy helyen fogunk valamennyire eltérni), így nyilvánvaló, hogy tárolt adatainkat XML formátumban fogjuk majd elhelyezni a szerveren, ennek legegyszerűbb megoldása a natív XML adatbázis. Kikötve, hogy ilyet

akarunk használni, kihasználhatjuk az XML dokumentumok hierarchikus strukturáltságát és így fogjuk felépíteni majd az adatbázisunk szerkezetét is. Ebből adódóan a szerver oldali programunk és az adatbázis között kizárólag XML részletek, dokumentumok lekérésére lesz lehetőség, melyet amennyiben lehetséges változtatás és módosítás nélkül akarunk majd továbbküldeni a kliens oldalra (már magában a lekérésben megpróbáljuk definiálni azokat a műveleteket, amik ilyenkor szükségesek lehetnek (pl. jogosultság kezelés stb.), így átadva ezt a munkát az adatbázis szoftvernek). Ezzel jelentős számítási kapacitást spórolnánk meg, ill. nem célunk újraimplementálni semmilyen már meglévő megoldást és végképp nem célunk több helyen elkezdeni különböző gyártmányú XML technológia implementációkkal játszadozni, ha nem muszáj.

Ezen gondolatmenet szerint a szerver oldali programunknak minél kevesebb munkát kellene végezni, ill. csak azokat a feladatokat kell ellátnia, amit máshova nem lehet pakolni. Így pl. a jelenlegi tartalomkezelő rendszerek szokásaitól eltérően a megjelenítési réteget teljes mértékben megpróbáljuk áthelyezni a kliens oldalra. Szerencsére erre pont létezik XML technológia, melyet felhasználhatunk (feltételezve, hogy a szerver oldalról már közvetlen XML kódrészletek jönnek a kliens felé), ez pedig nem más mint az XSLT (részletesebb leírást lásd később). Jelen dolgozatunknál ezt a technológiát amolyan template rendszerként fogjuk alkalmazni, vagyis eredeti céljának megfelelően XML kód részeket fogunk XHTML-be áttranszformálni. Bár a művelet meglehet számításigényes, de ez a terhelés a kliens oldalra fog jutni, ezzel mentesítve a szerverünket tőle, mindemellett legtöbb esetben olyan kis méretű adatokkal fogunk dolgozni, hogy egy XSLT-t támogató rendszer gond nélkül meg fog birkózni vele. Ha bizonyos esetben az adatok mégis nagy méretűek lennének (ez leginkább már a rendszer felhasználásától függ, nem pedig az implementációtól), akkor is az elvégzendő műveletek egyszerűek maradnak annyira, hogy gyakorlatban is használható legyen a megoldás.

Ilyen módon mentesítettük magát a szerver oldali részt a megjelenítés rétegétől. Emellett azért marad még feladata bőségesen. Bár nagyon sok vizsgált adatbázisrendszer megengedte, hogy a kliens oldal akár közvetlen hívjon rajta tárolt függvényeket, mi ezt most biztonsági és egyszerűsítési okokból nem alkalmazzuk. Későbbiekben ez elég jó optimalizálási lehetőséget nyújthat majd.

A szerver oldali programunknak elsődlegesen az adatbázisban tárolt adatok kiszolgálása lesz a

feladata. Emellett kénytelenek vagyunk itt elhelyezni a felhasználó kezelést is (munkamenet, autentikálás), hiszen ezt se kliens oldalra, se az adatbázisra nem bízhatjuk. Itt érkezünk el egy olyan ponthoz, ahol szándékosan kihagytuk egy jól specifikált XML alapú technológiát. Ugyanis, ha már mindenhol XML dokumentumokkal vagy dokumentum részekkel dolgozunk, szinte kínálja magát, hogy ezen részen SOAP protokoll segítségével folytassuk adatcserét. Ez az iparban már széles körben elterjedt webszolgáltatások alapja. Azonban ez egy web böngésző számára már talán túlságosan magas szintű és felesleges bonyolítás lenne, főleg ha azt vesszük, hogy jelen esetünkben is ezt egy egyszerű karakterlánccal helyettesítjük, nehezen és számításigényesen feldolgozható XML dokumentumok helyett. Persze ez csak akkor igaz, ha törekszünk arra, hogy ezen dolgozat végterméke a gyakorlatban is használható legyen, hiszen a könyvtárak és kódrészletek az előbb vázolt művelet egyszerűbbé tételére már léteznek, csupán a célunk nem az, hogy minden létező dolgot felhasználjunk. Így a szerver oldali programunknak még egy feladata lesz, hogy a bejövő lekéréseket (melyek már jellegükben is jobban tükrözik, hogy pontosan mit kérhetünk le kliens oldalról és hogyan) át kell fogalmaznia az adatbázis által érthető nyelvezetre. Ez a nyelvezet lesz az XQuery (lévén, hogy xml adatokkal kívánunk dolgozni). Ennek taglalását egy későbbi fejezetben folytatjuk.

3.1 Adatbázis

A fejlesztés megkezdése előtt az egyik legnagyobb feladat a megfelelő adattárolási módszer megválasztása volt. Lévén, hogy XML-el kívánunk mindenképpen dolgozni, ezért kézenfekvő volt, hogy olyan adatbázist vagy tárolási módot keresünk, melybe könnyen tudunk ilyen tartalmat feltölteni és visszakérni.

Az XML mint adattárolási forma már elég széles körben elterjedt az informatikában és szerencsére az adatbázis kezelő rendszerek is igen jelentős támogatást nyújtanak hozzá. Azokat a DBMS rendszereket, melyeken lehetséges XML-el dolgozni (általában itt csak a lekérdezésekre gondolunk) két fő csoportba oszthatjuk:

- XML felülettel rendelkező, más néven XML-Enabled adatbázisok, melyek képesek erre a formátumra transzformálni belső tartalmukat (pl. relációs adatbázisok, melyekből képesek vagyunk ilyen formában eredményt kapni egy lekérésre)
- Natív XML adatbázisok, melyek belső tárolási formája az XML-en alapszik, de nem feltétlenül szövegként tárolják azt

Általánosságban elmondható, hogy mindegyik felhasználható rendszer hasonló kommunikációs felülettel rendelkezik függetlenül attól, hogy milyen adattárolási formát használ. Számunkra a legfontosabb az, hogy egy XML dokumentumot tudjunk tárolni, melyet XQuery segítségével le tudjunk kérdezni és amennyiben lehetséges ugyanígy frissíteni is. Ez utóbbi természetesen nem lesz ilyen egyszerű, hiszen a W3C által specifikált XQuery frissítő nyelv az XQuery update még eléggé hiányos és az implementációk különböznek, így sajnálatos módon egy leimplementált rendszer alatt nem feltétlen lehet módosítás nélkül adatbázist cserélni.

Az elmondható a rendszerekről, hogy általában két módon tudunk velük kommunikálni. Mindegyik biztosít egy API-t, mely segítségével könnyedén tudunk lekéréseket intézni, sajnos programozási nyelvekre nézve az már eléggé változó, hogy melyikhez van ilyen. Szinte kivétel nélkül van persze egy C nyelven íródott API, melyet könnyedén tudunk felhasználni bármilyen nyelvből, viszont felé írni egy kezelő réteget sok esetben nehéz és főleg hosszú munka.

Mindemellett majdnem mindegyik rendszer biztosít egy (beállítástól függően) bárhonnan hívható felületet is, legtöbbször ezt webszolgáltatás alapokra helyezve, mely segítségével tárolt eljárásokat hívhatunk az adatbázisban. Ennek lehetőségei figyelemre méltóak, viszont a konzisztencia itt sem érhető tetten, hiszen ahány adatbázis kezelő rendszer, annyi kommunikációs felület. Mindemellett nagyon sok helyen érezzük, hogy a futási időben legyártott XQuery lekéréseink segítségével jobban ki tudjuk használni a lekérdező nyelv tudását, mintha csak tárolt eljárásokat vagy függvényeket használnánk. Ennek ellenére ezen funkció a későbbiekben igen hasznosnak bizonyulhat, amennyiben képesek leszünk általánosabb lekéréseket megfogalmazni ill. közvetlen kliens oldalról próbálunk adatot szerezni az adatbázisból. De ezek a gondolatok már nem képezik ezen mű tárgyát.

Természetesen a tárolt függvények erejét enélkül is ki fogjuk használni, hiszen a lekéréseinkben amennyiszer lehet, megpróbáljuk majd kiemelni az ismétlődő részeket. Főleg azért, mert a legtöbb XML-el dolgozó adatbázis kezelő rendszer komoly optimalizációkat ígér ezek használata mellett, így magát a tartalmazó utasításokat már jóval a lekérdezésünk előtt feldolgozza és akár az eredményeket gyorsítótárba is menti.

Elmondhatjuk, hogy az interneten fellelhető és ingyenesen felhasználható (a dolgozat megírása céljából csak ilyenekkel foglalkozunk) adatbázis kezelő rendszerek nagy része sajátos vagy pedig valamely már ismert adattárolási formát használ (relációs, OO stb.). Ezen felül nyújt felületet az XML lekérdezésekre és legtöbb esetben egy teljesen saját nyelvet az írási műveletekre. Általában rendelkeznek azzal a képességgel, hogy a feltöltött XML dokumentumainkat kollekcióba rendezzék, néhány pedig képes ezen kollekciókat hierarchikus rendben is tárolni. Ezeket a dokumentum csoportokkal szemben akár egyszerre is végezhetünk lekérdezést, ill. ilyen módon elegánsan szét is darabolhatjuk az „adatbázisunkat” több részre. Legtöbb esetben ezzel növeljük az alkalmazható optimalizációs megoldások számát is, ha nem mindent egy helyen tárolunk.

Lekérdezés szempontjából mindenhol megtaláljuk az Xpath kijelölő nyelvet, mely segítségével egyszerűen tudunk kijelölni csomópontokat egy xml, xhtml stb. Dokumentumban, amelyek megfelelnek egy bizonyos feltételnek. Akár önmagában ez is megfelelhetne számunkra, azonban ismerve az erre épített XQuery nyelv előnyeit mindenképpen megpróbálunk olyan adatbázis kezelő rendszerekkel foglalkozni, melyek támogatják az utóbbit.

Gyakran találkozunk olyan rendszerekkel, melyek közvetlen képesek számunkra XSLT transzformációkat végrehajtani. Ez szintén egy nagyon pozitív tulajdonság, bár a mi esetünkben ezt maximum egy helyen próbáljuk majd felhasználni, mivel ezen transzformációkat a kliens oldalra próbáljuk bízni. Ettől függetlenül lesz arról is szó, hogy mit kezdjünk azon felhasználókkal, kiknek nem adatott meg, hogy saját gépükön ilyen technológiákat használjanak (régi böngésző, le van tiltva stb.), számukra lehetőséget kell biztosítani arra, hogy a szerver végezze el ezeket a műveleteket a minél nagyobb elérhetőség jegyében. Persze ez már a gyakorlati alkalmazásnak egy szövevénye, így különösebb figyelmet nem szentelünk neki a különböző adatbázisrendszerek vizsgálatánál.

Számunkra a legfontosabb inkább az lesz, hogy az adott rendszer támogatja-e az XQuery lekérdező nyelvet. Ez a nyelv egy SQL-hez hasonló szintaktikával rendelkező lekérdezések megfogalmazására ad lehetőséget. Alapja az XPath (így nyilvánvalóan minden XQuery-t támogató rendszer támogatja az XPath-t is), azaz először kijelölünk csomópontokat és utána azokat esetlegesen tovább finomíthatjuk, sorba rendezhetjük és számunkra tetszőleges formában kaphatjuk vissza (azaz az XQuery lehetőséget nyújt a visszaküldött tartalom szerkezetének meghatározására is). Erről későbbiekben részletesebb tárgyalást is láthatunk.

Emellett elég sok adatbázis kezelő rendszerben találkozhatunk még Atomicity, Consistency, Isolation, Durability (ACID) támogatottsággal és elég sok kényelmi funkcióval, pl. grafikus felület, kifejezetten erős támogatás bizonyos programozási nyelvek iránt (ez általában Java-ra és Perl-re igaz a majdnem mindenhol támogatott C-n felül). Ezek a dolgok szintén nem kerültek különösebb figyelembevételre a meglévő rendszerek taglalásánál, hiszen közvetlen módon nem befolyásolják jelentősen a dolgozatot, későbbiekben viszont pozitív elbírálásban részesülhetnek.

3.1.1 Konkrét adatbázisok

Mint említésre került korábban, egy XML-el dolgozó rendszer számára sok lehetőséget találunk adataink eltárolására. Mivel valamelyest rendezett módon szeretnénk ezt megtenni és fontos valamennyire a teljesítmény is, ezért nem gondolkodunk sima szöveges állományokban, hanem az adatbázis kezelő rendszerek segítségét vesszük igénybe.

Ugyan nincs olyan jelentős választék jelen esetben, mintha egy SQL-t támogató rendszert keresnénk, de így is fellelhető néhány említésre méltó program, melyekből a következőkben néhányat kiemelünk, melyeket részletesebben körbe járunk.

BerkeleyDB XML

Az első adatbázis kezelő rendszer mellyel foglalkozunk egy igen érdekes alappal rendelkezik. Az eredetileg a berkeley-i egyetemen kifejlesztett szoftver egy beépülő adatbázis, vagyis nem önállóan működő, hanem saját programunk segítségével egy API-n keresztül kezelhető. A tartalmat egyszerű bináris állományokban tárolja a merevlemezen és mentesíti magát a kliens-szerver kommunikáció alól, így valamelyest teljesítményben előnyben van a többinél, viszont elveszít elég sok ebből adódó lehetőséget is (így pl. nincs lehetőség máshonnan lekérdezni az adatbázisból, vagy magát az adatbázist más gépre helyezni). Mindemellett az eredeti BerkeleyDB szoftver kulcs-érték párok tárolásával operál, tehát igen alacsony szintű is emellett. Bár kiváló un. lightweight alkalmazások készítésére, komplexebb adattárolásnál bizony elég sok minden leimplementálását a fejlesztőre hagyja. Használata ettől függetlenül elég elterjedt és találunk elég sok magas szintű adatbázis kezelő rendszert is, mely alapját ez jelenti.

Erre a rendszerre épült rá később egy XML adatbázis, mely ugyanúgy embedded (azaz beépülő) maradt, de maga az, hogy XML dokumentumokat kezelünk sokkal könnyebbé teszi a használatát is. Az időközben az Oracle szárnyai alá került termékcsalád leginkább a kifejezetten jó teljesítményével hívja fel magára a figyelmet, ill. azzal, hogy relatíve minimális munkával (csupán az API-t kell feltelepítenünk a futtató gépre) működésre bírható. A project továbbra is open-source maradt és egy igen jó alternatívát kínál a többi mellett.

Lekérdezésekhez XPath-t és XQuery-t támogat, ill. az W3C leendő szabványt az XQuery update-t használja frissítő nyelvnek. A gyorsabb lekéréseket kézileg beállítandó indexeléssel érjük el. Ez utóbbi igen nagy fegyvertény lehet, amennyiben nagyon jól megtervezett adatbázis sémával rendelkezünk és ehhez megfelelő módon állítjuk is be, azonban mint látni fogjuk, néhány másik adatbázis automatikusan is képes előállítani ezt számunkra és néhány ezred másodperc különbségért nem feltétlen éri meg kézzel próbálkoznunk. Bizonyos esetekben természetesen számíthat ez a differencia, de általában megpróbálunk majd az egyszerűségre törekedni.

A Berkeley DB XML szinte az összes nagyobb programozási nyelvhez rendelkezik API-val (talán egyedül a C#-ot lehet kiemelni, amely nem szerepel a listán), így mondhatni kifejezetten könnyen felhasználható bárhol. Nem rendelkezik viszont semmilyen beépített XSLT feldolgozóval, ill. nem találunk hozzá ingyenes grafikus felületet sem (ellenben több fizetős XML szerkesztő program szolgáltat ilyen tudást).

Összességében elmondható, hogy a Berkeley DB XML egy kifejezetten jól felhasználható adatbázis kezelő rendszer, mely megfelelő kezekben akkor is kiválóan teljesít, ha nem csak egy egyszerű alkalmazást fejlesztünk, hanem komolyabb rendszert. Legnagyobb előnye a gyorsasága, negatívumai közt viszont kiemelendő a dokumentációjának hiányosságai ill. a csekély támogatottság is (elég kevesen használják, így elég nehéz elakadás esetén segítséget kérni, amely nem a hivatalos support).

Sedna XML Database System

A Sedna XML Database System (mostantól csak röviden Sedna) egy igen impozáns tulajdonságokkal rendelkező nyílt forráskódú natív XML adatbázis. Kliens-szerver jellegű megközelítést alkalmaz, így a Berkeley DB XML-nél kiemelt pozitívumok és negatívumok itt tárgytalanok. Magát a szoftvert C++-ban írták meg és elég sok használatot megkönnyítő tudással rendelkezik.

Amit ki lehet emelni a pozitív tulajdonságai közül az leginkább az, hogy ACID mellett felhasználó kezeléssel is rendelkezik. Bár ez utóbbi talán elvárható egy ma sűrűn használt relációs adatbázisnál, de az XML adatbázisok körében ez kiemelendő tudás. Méghozzá a Sedna ezt kifejezetten magas fokon űzi, mindennel rendelkezik ami manapság elvárható ilyen téren. Természetesen ez még nem ad nekünk lehetőséget arra, hogy a CMS rendszer

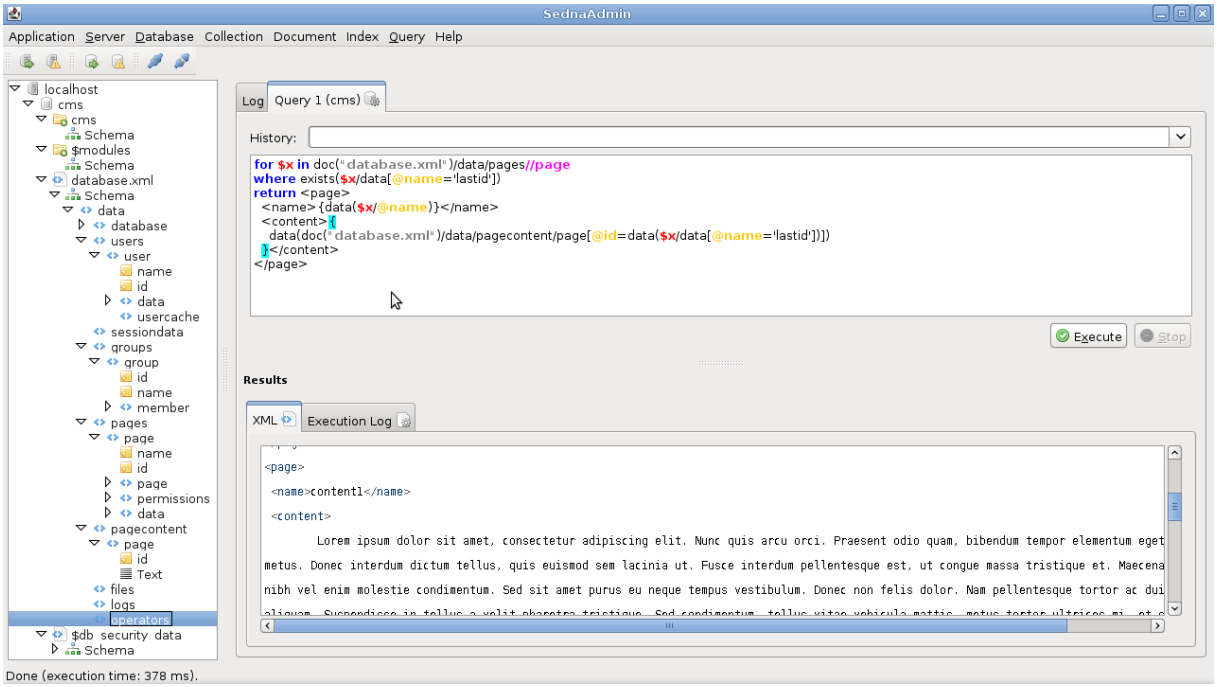
fejlesztése során szóba kerülő felhasználó kezelést átterheljük az adatbázisra (az már lassan azt jelentené, hogy teljesen felesleges a szerver oldali programunk).

A rendszer természetesen támogatja az XQuery-t, ellenben saját módszereivel tudunk csak módosításokat felvinni. Igencsak kiemelendő tulajdonsága, hogy lehetőségünk nyílik igaz XQuery „trigger”-eket létrehozni, hasonlóan mint a relációs adatbázisoknál. Megfelelően kihasználva ez elég nagy fegyvertény lehet a fejlesztő kezében. Érdekes dolog még, hogy beépített függvényekkel elérhetünk relációs adatbázisokat egy XQuery kifejezésen belül is, bár ennek a lehetőségnek már tényleg semmilyen felhasználása sem adott a fejlesztett rendszerünknél.

A Sedna-hoz már nem olyan bőkezűen osztogatták a letöltéssel együtt járó API-kat, ellenben elég sok elérhető az oldalukon, melyeket nem a készitők fejlesztettek. Nagyjából fedik azt a reális igényt, amit itt el lehet várni. Emellett a rendszer rendelkezik még egy elég jó és nagy tudású parancssori felülettel is, amely bátrabbak által akár mint kommunikációs felület is használható. Továbbá találhatunk hozzá egy elég jól használható és munkánkat jelentősen megkönnyítő grafikus felületet is, mely megint nem része az eredeti letölthető csomagnak.

Ami még kiemelendő, hogy a Sedna rendelkezik XML:DB felülettel is. Ez egy ODBC-hez vagy JDBC-hez hasonló szabványosított felület, mely segítségével az adatbázis kezelő rendszer teljes tudása elérhető és erős kerekítés után is csak a felére mondhatjuk a jelenlévő XML adatbázisoknak, hogy rendelkeznek ilyennel. Sajnálatos módon ennél a rendszernél csak Java számára létezik driver ehhez és az szintén nem az eredeti készítők munkája (bár egy nyílt forráskódú szoftvernél ez utóbbi ritkán jelenthet problémát).

A Sedna oldalán és a letölthető csomagban találhatunk egy viszonylag szűkszavú ellenben mindenre kiterjedő dokumentációt is, melyet hiányosnak egyáltalán nem nevezhetünk. Összességében az egyik legjobb natív XML adatbázissal van dolgunk, mely majdnem mindenben felveszi a versenyt társaival. Kicsit hiányolhatjuk az univerzális lekérdező felületeket, ill. néhány API-t (pl. Perl), de ennek ellenére mindenképpen jó alternatívának kiálthatjuk ki bármely mással szemben.



A Sedna XML database grafikus felülete

eXist-db

Az eXist-db (továbbiakban csak exist) egy Java-ban fejlesztett natív XML adatbázis kezelő rendszer, mely talán a legnépszerűbb mind közül. Ez főként annak köszönhető, hogy az alapját képező nyelv népes használói tábora van célközönségnek kitűzve. Ez egyben talán a legnagyobb negatívuma is a rendszernek, mely bár jó képességekkel rendelkezik, nem nyújt számunkra közvetlen alacsony szintű elérést a tartalomhoz. Vagy Java-t használunk és így kihasználjuk az XQueryGenerator nevezetű egységét az Exist-nek, vagy HTTP protokollon keresztül hívhatunk tárolt eljárásokat és függvényeket. Ez utóbbi bár elég hatékony, ahogy korábban is említve volt, sokkal jobban kihozhatjuk egy lekérés erejét, ha minden esetben megfogalmazhatunk saját lekéréseinket, ahelyett, hogy egy nagy általánost használnánk.

Ennek ellenére érdemes ránézni arra, hogy mit is tud valójában ez a program. Meglehetősen tömören összefoglalják a fejlesztők, hogy természetesen támogatja az XQuery-t, az XQuery update-t és még a szerver oldali XSLT transzformációt is. Ahogy igen sok helyen hallhatjuk, teljes weboldalakat összerakhatunk csupán ezek segítségével. Ennek érdekében elég sok webes elérési felületet is szolgáltat az adatbázis, közülük kiemelendő a SOAP és az XMLRPC interface (utóbbi igen elterjedt néhány natív XML adatbázis körében).

Emellett ahogy a Java-ban írt szoftvereknél már megszokott, egy elég jelentős támogatói bázissal is rendelkezik a rendszer, így az esetleges segítségkérés is sokkal kényelmesebb.

Természetesen a fejlesztők minden állítása ellenére mi most nem lennénk képesek szimplán az ő általuk kínált eszközökkel összerakni a tartalomkezelő rendszerünket a már korábban említett okok miatt, ennek ellenére bizonyos oldalakhoz kifejezetten passzolna az itt felvázolt felállás.

Emellett még meg kell említenünk egy negatívumot az Exist kapcsán. A korábban említettek mellett még hátrányként szól az adatbázis ellen a teljesítménye. A lekérések általában elég gyorsak, viszont ahogy növeljük a lekérdezett dokumentum méretét úgy a szükséges idő rohamosan nő. Már egy 10 MByte-os adatbázisnál (amely valljuk be, egy sűrűn használt rendszer esetében nem túlzottan nagy méret) igen komoly gondjai akadnak a feladat teljesítésével, néhány esetben irreálisan magas válaszidőket produkálva (10-15 perc).

Összességében elmondhatjuk, hogy bizonyos esetekben kifejezetten jó szolgálatot tud tenni számunkra az Exist, azonban ez a bizonyos eset nem mindenkinek felelhet meg. Számunkra a rendszer fejlesztése során kifejezett visszatartó erőt jelentenek a felsorolt negatívumok.

MonetDB/XQuery

Utolsóként érintünk még egy szintén nyílt forráskódú rendszert, az MonetDB-t. Ez egy alapvetően relációs adatbázis, melynek tetejére építettek egy XQuery felületet, a kettő akár párhuzamosan is használható, bár gyakorlatban nem sok értelme lenne.

Maga a rendszer elég egyszerű, ennek ellenére hihetetlenül erőteljes tulajdonságokat mutat. A fejlesztők leginkább a teljesítményét emelik ki, melyben a legimpozánsabb az a készség, hogy hatalmas dokumentumokat is képes emberi idő alatt lekérdezni. Tesztjeikben 11 GByte-ig mentek el, ahol már gondot okozott egy-egy XQuery lekérés. Ez a tulajdonság az XML adatbázisok körében igen egyedi.

Telepítése után azonnal rendelkezésünkre áll egy konzolos kliens program, amiben könnyedén kipróbálhatunk tetszőleges lekéréseket, adminisztrálhatjuk az adatbázisunkat és sok mást. Emellett rendelkezik egy webes adminisztrációs felülettel, mely meglehetősen primitív, de a kezdet kezdetén jó szolgálatot tud tenni, későbbiekben a használata feleslegessé

válhat (egyrészt több hibával is rendelkezik, másrészt a rajta felhasználható funkciók csak egy igen kicsiny része a konzolos kliens által kínáltaknak).

A MonetDB által kínált API-k is igen szegényesnek mondhatóak, ugyanis közvetlen alacsony szintű elérést biztosító un. Mapi felületen keresztül kezdésnek csak Java-ban vagy C/C++ -ban tudunk kommunikálni. Ismét előjön a probléma, hogy vagy megpróbáljuk más programozási nyelvből a C-s függvényeket használni vagy megmaradunk a tárolt eljárásoknál. Sajnos az interneten fellelhető könyvtárak ezen szoftverhez szinte kivétel nélkül csak a relációs adatbázis funkcionalitását fedik így az előbb vázolt probléma még ijesztőbb, ha nem az említett két nyelven próbálunk fejleszteni.

Természetesen a MonetDB támogatja az XQuery-t és az XQuery update-t is, emellett egy saját gyártmányú protokollt is használhatóvá tesz, az XRPC-t, mely bár névben kísértetiesen hasonlít az XML:RPC-re, valóságban nem ugyanaz. Ellenben ez is SOAP alapú, tehát használatával nem lehet probléma. A fejlesztők rendelkezésünkre bocsájtanak még egy JavaScript-es könyvtárt, amellyel azonnal elérhető az XRPC szolgáltatása a szervernek kliens oldalról is, amennyiben erre vetemednénk.

Összességében elmondható a MonetDB-ről, hogy ugyan egyszerű, de nagyszerű, ellenben nehézkesen megközelíthető. Nem rendelkezik hatalmas mennyiségű tudással, viszont teljesítménye kiugróan a legjobb a vizsgáltak közül és minden alap tudással rendelkezik, amire esetlegesen szükségünk van a CMS rendszerünk fejlesztése során (egyedül talán a már említett API hiányosságokat nem számolva).

A MonetDB/XQuery webes grafikus felülete

3.1.2 Összegzés

A fentieken kívül természetesen még rengeteg más rendszer létezik, amely hasonló tudást nyújt, de nagy részük vagy félkész vagy valamilyen más komolyabb hibával rendelkezik. Voltak olyan rendszerek, melyek a dokumentáció majdhogynem teljes hiányával küzdöttek, egyesek pedig a legalapvetőbb követelményeket sem érték el.

A felsoroltak viszont mind alkalmasak lennének valamilyen szinten arra, hogy felhasználjuk őket munkánk során. Mivel a fejlesztés alatt több is ki lett próbálva és természetesen mindegyik képes volt használható mintát produkálni, ezért bármelyiket alá lehetne rakni jelen lévő implementációnak, függően attól, hogy hol akarjuk alkalmazni azt.

3.1.3 Adatbázis szerkezet

Munkánk során nagyrészt XML dokumentumokkal fogunk dolgozni, így elkerülhetetlen, hogy előre megtervezzünk egyet, amely majd adataink forrásául szolgál. Jelen esetben eltekintünk attól, hogy ezt a dokumentumot szétszabdalva egy kollekció keretein belül használjuk, vagy egy dokumentumként kerül be adatbázisunkba. Inkább a részegységeit tekintjük át és azt, hogy mi lesz a feladatuk.

Ahhoz, hogy sorban tudjunk haladni és elkerüljük az előre hivatkozásokat, először is a felhasználók tárolását kell megoldanunk, ugyanis ez alapkövetelmény és későbbiekben sokat hivatkozunk majd rájuk. Elsősorban olyan adatokat akarunk elmenteni, amelyek minden felhasználónál jelen vannak (ilyenek pl. név, egy egyedi azonosító, jelszó (titkosítva)), ill. lehetnek még saját beállításai is, melyekkel majd a megjelenítéskor fogunk foglalkozni. Megpróbálunk majd minden elemhez az adatbázisban egy egyedi azonosítót rendelni annak érdekében, hogy elérésük mindenhol könnyű legyen. Ez az azonosító a lehető legegyszerűbb érték vagyis egy egész szám lesz. A legtöbb adatot az adatbázisban megpróbálunk szabványos módon leírni, hogy egyfajta konzisztenciát érjünk el a különböző részek közt. Így ahol lehet egy tulajdonságát az adott elemnek `<data name="név">érték</data>` formában adjuk meg. Lássunk egy példát arra, hogy hogyan tervezzük tárolni a felhasználókat

```
<users>
    <user name="admin" id="0"> <!—Egy adott felhasználó rendelkezik
egy névvel -->
        <data name="reg_date">2008.04.03:21:43</data>  <!—
Regisztrálási dátum -->
        <data name="admin">true</data>
         <!—Lehetőségünk lesz megadni amolyan root felhasználókat is
-->
        <data name="realname">Root admin</data> <!—esetlegesen más
felesleges információkat -->
        <data name="password">password</data>
        <usercache>
            <!—Minden más beállítást, mely nem feltétlen kötődik
minden felhasználóhoz egy külön csomópont alatt tárolunk -->
            <data name="név">érték</data>
            ...
        </usercache>
    </user>
        ...
 </users>
```

Mint látható megpróbáljuk egyszerűen és általánosan megfogni egy felhasználó tárolását. Mivel elsősorban tartalom centrikus lesz majd a rendszerünk, ezért a felhasználói jogosultságok a tartalommal rendelkező oldalakhoz kötődnek majd a felhasználók helyett.

Ahhoz, hogy a felhasználókat a későbbiekben könnyebben tudjuk kezelni, érdemes csoportokat létrehoznunk. A csoportok tekinthetőek úgy is, mint felhasználók, aki hivatkozik egyre, az hivatkozik az összes tartalmazott tagra. Főként majd a jogosultságkezelésnél lesz nagyobb haszna ennek.

```
<groups>
  <!-- felhasznói csoportok kerülnek ide -->
  <group id="0" name="Vendég">
      <member>admin</member> <!—egy tagját így írjuk le a csoportnak
-->
  </group>
  <group id="1" name="Regisztrált felhasználók">
      <member>admin</member>
  </group>
 ...
</groups>
```

Természetesen itt még különösebb jelentősége nincs, hogy egyedi azonosító vagy név szerint hivatkozunk egy felhasználóra (feltételezve, hogy a név ugyanúgy egyedi), maga az XML feldolgozó stringként kezeli mindkettőt és nem várhatunk teljesítménybeli javulást egyiktől sem.

Végül, hogy teljes legyen a felhasználókat érintő adatbázisrészek meghatározása, még egy csomópontot biztosítanunk kell, ahol a munkamenethez szükséges adatokat perzisztálja majd le a szerver oldali programunk.

```
<sessiondata>
     <session id="azonosító"> <!—minden munkamenet rendelkezik egy
generált egyedi azonosítóval -->
          <!-- a munkamenettel kapcsolatos adatokat a szerver
oldali program tölti fel és kérdezi vissza később -->
          <data name="user">admin</data>
          <data name="session_ends">2009. 04. 03:23:50</data>
          ...
     </session>
</sessiondata>
```

Innentől kezdve a legfontosabb dolgunk a tartalom kezelése lesz. Itt már kihasználjuk az XML dokumentumok hierarchikus rendezettségét, ugyanis szeretnénk adatainkat is ilyen módon tárolni. Mindemellett, ahogy korábban emlitettük, univerzális tárolási egységeket szeretnénk definiálni, melyek szinte bármiként felhasználhatóak. Megjelenésük a végfelhasználó felületén csupán attól függ majd, hogy milyen transzformációs szabályokat írunk le hozzájuk.

A tárolt tartalmat oldalak fogják magukba ölelni (page). Ezek valójában nem igazi hagyományos értelemben vett oldalak lesznek, csupán egy elnevezési konvenció. Egy adott lekérés majd több ilyet is magába foglalhat és egy adott weboldalon akárhányat megjeleníthetünk.

Mindemellett figyelembe kell vennünk azt is, hogy egy tipikus web2.0 –s rendszert fejlesztünk, melynek fontos tulajdonsága, hogy tartalommal a felhasználók töltik fel. Így akár szükségünk lehet olyan képességekre is, mint a tartalom verziókövetés alatt tartása (vagyis a korábbi változatok letárolása). Ennek szellemében külön választjuk egy adott oldal specifikációját és a konkrét tartalmát, melyet két különböző helyen tárolunk el. A tartalmakhoz hozzárendeljük a dátumot, amikor megszületett és az oldal azonosítóját, hogy vissza lehessen keresni, az oldalunkhoz pedig csak egy azonosítót, hogy azonnal meglehessen találni a legfrissebb hozzá tartozó tartalmat (mivel dátum alapján költséges lenne az adott csomópont megtalálása).

```
<pagecontent>
     <!-- a tartalmak azonosítójának nincs köze az oldalak
azonosítójához -->
      <page id="1" pageid="1" date="2009. 04.
30:20:00"><![CDATA[Tartalom... ]]></page>
     ...
</pagecontent>
```

Mivel egy XML dokumemfa tetszőleges mintát követhet, ezért az adott oldalak talán szemre kicsit kaotikusan, de a számítógép számára egyszerüen fogjuk tárolni. Ez azt jelenti, hogy minden ilyen elemhez az adott szintjén tároljuk az összes szükséges információt és gyerek elemei közt szerepelhetnek más ugyanilyen elemek is. Mivel rendszerünket úgy tervezzük, hogy legtöbbször egy adott szinten fogunk lekérdezni több ilyen oldalt, ezért ez teljesítmény szempontjából nem jelent hátrányt, mindaddig amíg esetlegesen el nem érünk egy olyan mélységet a fában, ami már gondot okozhat az adatbáziskezelő rendszerünknek. Ennek persze a valószínűsége normális felhasználási körülmények közt igen alacsony.

A jogosultság rendszer szintén megpróbálja majd kihasználni a hierarchikus felépítést. Azaz jogosultságokat műveletekre osztunk ki felhasználók vagy csoportok számára és ezek a jogosultságok öröklődnek a fában. Tehát egy megadott jog mindaddig érvényes lesz lefele, amíg valamelyik oldal felül nem írja azt a sajátjaival.

```
<pages>
    <page name="root" id="1">
        <page name="sub1" id="2">
            <!—Ez az oldal aloldal lesz a „root” nevű oldalnak -->
            <!—A jogosultságokat örökli majd a szülőtől -->
            <data name="lastid">2</data>
            <!—eltároljuk az éppen aktuális tartalom azonosítót -->
            <data name="author">admin</data>
            <!—eltároljuk mindenképpen a létrehozóját az oldalnak --
>
            ...
        </page>
        <page name="sub2" id="3">
```

```
            ...
        </page>

        <!-- a „root” nevű oldalhoz tartozó további információk -->

        <permissions> <!—A root oldalhoz tartozó jogosultságok ide
kerülnek -->
            <data name="művelet">jogosultak</permission>
            ...
            <!—Itt megengedünk, vagy letiltunk egy adott műveletet
(pl. olvasás, írás, jogosultság módosítása stb.) egy vagy több
felhasználó vagy csoport számára -->
        </permissions>

        <data name="creationdate">2008.04.03:21:58</data>
        <!—Letárolhatjuk a létrehozás dátumát -->
        <data name="lastid">1</data>
        <!—mindenképpen letároljuk a legutolsó ezen oldalhoz tartozó
tartalmat-->
        <data name="type">main_page</data>
        <!—Letárolunk minden oldalhoz egy típust is, mely majd a
kliens oldali megjelenítésnél lesz nagyon fontos -->
        <data name="history">false</data>
        <!—Jelezhetjük, hogy az adott oldalról kívánjuk-e megtartani
a régebbi verziókat, ha igen akkor minden esetben új tartalom
csomópontot hozunk majd létre minden egyes módosítással, egyébként
pedig mindig azt az egyet írjuk felül, amivel létrejött -->
        ...
    </page>
</pages>
```

Az adatbázis szerkezet természetesen megengedi, hogy a legfelső szinten több oldal is szerepeljen, de nem feltétlen szerencsés megoldás. Jobban járunk, ha az oldalunk belépési pontját (mely akár egy egészen komplex weboldal is lehet) a kliensnél definiáljuk, így megtartva egy egyértelmű felépítést.

Emellett adatbázisunk tartalmazhat még egyéb információkat, melyek nem feltétlen szükségesek és emiatt nem is említjük itt őket. Ilyen lehet pl. a naplóbejegyzések, esetleg állományok listája és még sorolni lehet a végtelenségig, hogy mik kerülhetnek bele, de mint már korábban is, megpróbálunk némileg az egyszerűségre törekedni.

3.2 Szerver oldali program

Eddig viszonylag kevés konkrétumról volt szó a szerver oldali programunkat érintően. Elsősorban ez annak köszönhető, hogy megpróbáltuk fenntartani azt a lehetőséget, hogy ez a program minél rugalmasabban bármilyen eszközökkel leimplementálható legyen és kommunikációja szabványos keretek mentén haladjon.

Ennek érdekében nem is kíván a szerző semmilyen technológiához vagy programozási nyelvhez ragaszkodni az implementáció során, így ez a komponens sokkal jobban alkalmazkodik a többi rész tulajdonságához, hogy tetszőlegesen cserélhető.

Amit eddig lekötöttünk a programunkkal kapcsolatban, az az volt, hogy minél kevesebb művelettel foglalkozzon és minél többet osszon szét a többi komponensnek. Így pl. el szeretnénk érni, hogy ennek a programnak konkrétan ne kelljen XML feldolgozással bajlódnia (ez ugyanis elég erőforrás igényes tud lenni, főleg ha nagyobb terhelés mellett vesszük figyelembe). Mivel a megjelenítési réteget már teljesen áthelyeztük a kliens oldalra, ezért már nem nagyon maradt feladat, amit lehetséges lenne máshol elvégeztetni.

Így összegezve a programunk elsődleges feladata, hogy a klienstől megkapott lekérdezést átkonvertálja egy XQuery kifejezéssé és az eredményt visszaszolgáltassa. Emellett természetesen egyszerű munkamenet kezelést is meg kell valósítani annak érdekében, hogy a felhasználók azonosítását hatékonyan tudjuk kezelni.

Mivel webes rendszerről lévén szó, ezért nyilvánvalóan ennek a résznek HTTP protokollon keresztül kell fogadnia hívásokat. Ezt megteheti egy web szerver (esetleges alkalmazásszerver) mögött is, de hatékonyabb tud lenni az, amikor magába az alkalmazásba írjuk bele a web szervert, így közvetlen módon tud válaszolni a kliensnek. Erre már minden nem egzotikus szóba jöhető programozási nyelv gyorsan megvalósítható lehetőséget nyújt. Bár nem akarjuk ezt se teljesen megkötni, a szerző implementációja e mentén fog haladni.

3.2.1 Lekérések kliens oldalról és a generált dokumentum

Korábban már sokszor volt szó a két fél közti kommunikáció mikéntjéről. Mint említve volt, itt van talán az egyetlen pont a tervezett rendszerben, ahol nem kívánunk XML-hez

kapcsolódó technológiát alkalmazni, egyszerűen hatékonysági okok miatt. A SOAP lekérésekhez szükséges JavaScript kód (márpedig webes rendszernél kliens oldalon csak arról beszélhetünk) többszöröse annak, mint ami egy egyszerű saját kitaláció lekérés formához szükséges, így az elvégzendő műveletek is. Természetesen itt is cserélhető lenne az implementáció és a szerver oldalon se jelenten túlzottan sok változást.

Először is tekintsük át azt, hogy mire lehet szüksége a kliensnek az adatbázisból. Elsősorban a tartalomra, azaz a korábban már tisztázott rendszerű oldalakra és az ehez kapcsolódó információkra. Szüksége lehet még felhasználói adatokra (főleg a megjelenítéshez kapcsolódó felhasználói beállításokra) és esetlegesen a kliens oldali adminisztráláshoz a csoportok, munkamenetek adataira is. Kikötve azt, hogy a munkamenet felhasználó számára is szükséges részeit a felhasználói adatok közt kimentjük, egy átlag felhasználónak semmiképpen nem lehet szüksége az előbbire.

Elsősorban azonban a tartalomlekérést taglalnánk. Vezessünk be egy általános formát a lekérésre. Tegyük fel, hogy egy lekérésben több kérés is található, melyeket külön, külön feldolgoz a programunk és válaszol mindegyikre egy válaszban. Ezzel megóvjuk magunkat attól, hogy egy összetettebb oldal adatainak beszerzéséhez többször kelljen HTTP hívást intéznünk.

Egy ilyen kérésen belül először is meg kell adnunk, hogy milyen típusú adatot akarunk lekérni. Ha oldalakat vagy csak egy oldalt szeretnénk lekérni, akkor legyen ez a kulcsszó a **page**. Maradva az oldalaknál, ekkor valamilyen módon meg kell adnunk azt, hogy mit akarunk pontosan lekérdezni. Korábban említésre került az adatbázis struktúrájánál, hogy célunk az, hogy egy kérésben az oldalak fájában egy szintet jelöljünk ki, ahol kiválasztunk tartalmakat. Ezért meg kell tudnunk adni egy ilyen szintet. Ezt legegyszerűbben egy URL jellegű útvonallal tudjuk majd megadni, ahol ’/’ karakterrel lesznek elválasztva azon oldalak nevei, ahol végighaladunk a fában. Itt is kénytelenek vagyunk kikötni, hogy egy szinten nem lehetséges, hogy több oldalnak ugyanaz a neve.

Természetesen megeshet, hogy egy alkalommal csak egyetlen oldalt, máskor pedig többet is le szeretnénk kérdezni és nem áll szándékunkban, hogy minden elemet lekérdezzünk az adott szintről. Ezért bevezetünk egy szűrési lehetőséget, ahol adott oldalak különböző tulajdonságaira lesz lehetőségünk majd szűrni. Ezt egy kritériumnak nevezzük el és az útvonal jelölés mögé rakjuk. Amennyiben nem szerepel, akkor feltételezzük, hogy csak

egyetlen oldalra van szüksége a kliens oldalnak, amennyiben szerepel (még akkor is ha üres), akkor az útvonal által kijelölt oldal alatti összes oldalt kijelöljük vele és ami megfelel a kritériumnak, azt visszaadjuk.

Mivel végig törekedtünk arra, hogy csak azokat az információkat kapja meg a kliens oldal, amelyeket ténylegesen kért is, ezért még mindig nem fejezhetjük be. Szükségünk van arra, hogy megadjuk, pontosan mikre van szüksége egy adott oldallal kapcsolatosan. Egyrészt nem akarjuk az oldal tartalmát egy az egyben visszaküldeni, másrészt vannak olyan információk, melyek nem közvetlenül az oldalhoz tartoznak, de kapcsolódnak hozzá (pl. a szerzőjének a teljes neve, vagy az azon a szinten elfoglalt sorszáma a többi oldal közt) és ezeket is elérhetővé kívánjuk tenni. Ezért létrehozunk még az egész lekérés végére egy paraméterlistát is, melyet majd minden oldalhoz kitöltve küld vissza a rendszerünk. Itt név – érték párokban határozzuk meg azt, hogy mi legyen a lekérdezett adat. A megadott néven fog majd a válaszban szerepelni, a lekért érték, azért, hogy az esetlegesen azonos nevű adattagok miatt ne legyen ütközés. A lekért érték pedig egység.név formában fog szerepelni, ahol az egység határozza meg azt, hogy mivel kapcsolatban akarunk lekérdezni valamit (pl. page, amikor maga az oldallal kapcsolatban valamit, author ha az oldal szerzőjével stb.) és a név pedig a konkrét neve a lekért információnak. Pl. ha az oldal tartalmát akarjuk megszerezni, akkor írhatjuk, hogy *content => page.content*, ha a szerző teljes nevét, akkor *full_name => author.realname*.

A szerver oldali programunkban természetesen szerepelnie kell az ilyen lekérésekhez tartozó szemantikai információnak (így tudni fogja, hogy ki az *author* vagy hogyan adja vissza a sorszámát az adott oldalnak). Lássunk egy konkrét példát az előbb leírtakra:

```
page:site/forum/Development/Test_topic[page.index<10](content =>
page.content, author => page.author, realname => author.realname)
```

Ilyen kérésből egy lekérésben természetesen több szerepelhet. A válasz erre pedig adódóan a lekért adatokból a következőféleképpen nézhet ki:

```
<result>
      <content>Tartalom1</content>
      <author>user1</author>
      <realname>valaki</realname>
</result>
<result>
      <content>Tartalom2</content>
      <author>user2</author>
      <realname>senki</realname>
</result>
...
```

A kapott választ pedig meglehetősen könnyű átalakítani HTML-be, főleg ha lekérdezzük mindig a minden oldalhoz kötelezően hozzátartozó típus értéket is, amely egyértelműen azonosítja azt a formát is, amilyenben meg kellene jelennie majd a tartalomnak.

Ezzel a módszerrel könnyedén tudunk konkrét oldalakat lekérni a rendszerből, melyek összesen három lépésben megválaszolásra is kerülnek. Később látni fogjuk, hogy ezeket a kéréseket könnyű szerrel át tudjuk alakítani XQuery kifejezéssé közvetlen módon. Mivel egyetlen kifejezésben próbálunk majd az adatbázissal is kommunikálni, így újabb lehetőséget nyitunk arra, hogy alkalmazhassa az optimalizációs megoldásait.

Természetesen lehetőséget adunk az oldalakon kívül más adatok lekérdezésére is. Így létezik még két lekérdezési lehetőség, az egyik a **user**, azaz a felhasználói adatok lekérdezése. Itt nyílván nem egy útvonalat fogunk megadni, hanem egy nevet vagy egy egyedi azonosítót és ez alapján kaphatunk jogosultságoktól függően információkat adott felhasználókról. Ezek a jogosultságok az egyszerűség kedvéért bele lesznek égetve a szerver oldali programunkba és csak nagyon alap ellenőrzéseket fogunk tenni. Így pl. azt mondhatjuk, hogy kliens oldalról a jelszót sosem szabad megkapnia a felhasználónak (ő beküldi a sajátját, de sem az övét, sem másét nem szolgáltatjuk vissza), ill. létezhetnek olyan adatok is, melyek a rendszer adminisztrátorain kívül senki másra nem tartoznak (jelenleg nem írtunk be az adatbázisba ilyet). Másik lekérdezési módunk lehet még az oldalak közti keresés is, ahol az útvonal helyett egy keresési kulcsszót vagy szavakat adunk meg és ezen oldalakról kaphatunk információkat. A hatékonyabb lekérések érdekében a szűrők közt megadhatunk kritériumot az

oldal típusára is természetesen, így szemantikai értelemben is leszűkíthetjük a keresést.

A tartalom lekérdezése után még egy fontos feladat hárul ezen szintaktikára, mégpedig a tartalom frissítése ill. újabb tartalom felvitele. Mivel itt is csak az oldalakkal fogunk foglalkozni, amik ugyebár bármit tartalmazhatnak, ezért mindösszesen három műveletbe próbáljuk majd belesűríteni ezt a részt.

Azon lekéréseknél, amelyek módosítást tartalmazhatnak, megpróbáljuk a beküldött adatokat szétválasztani magától a kérés fejlécétől. Mivel az előbb említett szintaktika vélhetően (a rendszer rendeltetésszerű használata esetén) vagy eleve URL elemek lesznek (amennyiben megkerüljük, vagy csak közvetetten érintünk web szervert) vagy pedig un. query string-ben fognak elhelyezkedni, ami szintén a http lekéréskor megadott URL része. Így nagy hiba lenne ebbe belehelyezni olyan adatokat, melyek vélhetően rengeteg speciális karaktert tartalmaznak és méretük sem ide való. Így alapkövetelménynek tekintve, hogy http protokollon keresztül kommunikálunk, elvárjuk, hogy ezen tartalmak POST típusú üzenet keretében a lekérés törzsében helyezkedjenek el (paraméterek).

Így tudunk egészében egy oldal létrehozására szolgáló lekérés fejlécének szintaktikáját definiálni, ami a kulcsszavon kívül tartalmaz még egy útvonalat, ahova az új tartalmat szeretnénk elhelyezni. Természetesen amennyiben már létezik ott meglévő oldal, akkor a műveletet nem tudjuk végrehajtani (az adatok felülírását a hibalehetőség csökkentése miatt szándékosan egy külön művelettel oldjuk meg). Ilyen esetben a visszakapott üzenet egy hiba lesz. Ahhoz, hogy megőrizzük az adatbázis konzisztenciáját, természetesen még egy pár szabályt be kell tartanunk, így egy szinten nem szerepelhet több oldal ugyanazzal a névvel. A létrehozáskor ezt is ellenőriznünk kell, persze lehetséges olyan helyzetek, amikor igazából nincs is neve az egy szinten lévő tartalmaknak (pl. fórum hozzászólások) és sok hasonló van belőle, ekkor egy tetszőleges egyedi nevet hozzárendelünk mindegyikhez (elvégre ez nem lesz mérvadó a megjelenés szempontjából), amelynek kitalálást lelki állapotunktól függően a kliensre vagy a szerver oldali program bízzuk. Ill. nem hozhatunk létre oldalakat olyan oldalak gyermekeként, melyeknek nem létezik szülője. Amennyiben ez a helyzet, akkor ismét nyugodtan dönthetünk két ekvivalens megoldás mellett, miszerint létrehozzuk a szülőket valamilyen üres anyaggal vagy pedig hibával térünk vissza (a hasonló hierarchikus elveket valló wiki szoftverek nagy általában az elsőt támogatják).

Nézzünk egy példát egy létrehozó műveletre:

```
create:site/Test/New page
```

Vegyük figyelembe azt, hogy igazából a lekérés maga tetszőleges karaktereket tartalmazhat, hiszen a web szerverrel való kommunikáció előtt URL kódolásba helyeződik át és a szerveren ugyanezt visszaalakítjuk majd, így a szóközök jelenléte nem probléma.

Ehhez a lekéréshez természetesen csatolni kell a http request törzsében néhány paramétert is. Mivel a munkamenet adataiból a legtöbb szükséges információ megszerezhető (jogosultságok, aktuális felhasználó neve stb.), ezért itt csak az oldal tartalmát kell definiálnunk és típusát, melyet két paraméterrel (**content**, **type**) meg is adhatunk.

A további szükséges információkat már a szerver oldali programunk fogja generálni a létrehozott oldalhoz (így a tartalom különválasztása, a módosítási dátumok megadása és így tovább). Szándékosan nem engedjük, hogy létrehozáskor egyből be lehessen állítani az adott oldal jogosultságait is. Erre nincs különösebb indok, csupán annyi, hogy más hasonló rendszerek is így működnek, ill. elviekben megpróbáljuk minél inkább minimálisra fogalmazni a műveleteinket.

Nagyon hasonló művelet lesz még az oldal tartalmának módosítása. Ennek a műveletnek az **update** nevet adjuk és a kulcsszó kivételével minden másban megegyezik majd a **create** műveleteivel. Annyit engedünk még itt meg, hogy a csatolt paraméterek közt szerepelhessenek jogosultságok is és azokat így felül tudjuk majd írni. Ezek egyszerű művelet → új jogosultság, kulcs-érték párokban fognak majd szerepelni és felülírják az adatbázisban tartalmazott értékeket.

Ettől sem különbözik nagy mértékben az utolsó módosító műveletünk, melyet definiálunk, ez pedig nem más, mint a törlés, amely a **delete** kulcsszóval kelthető életre. Itt egyszerűen egy útvonalat adunk meg, mely egyértelműen jelöli azt az oldalt, amelyet törölni szeretnénk. Itt is amolyan hibák elkerülése szempontja szerint haladva nem engedjük meg, hogy egyetlen kérés több oldalt is töröljön. Természetesen egy lekérésben több ilyen részkifejezés is szerepelhet, amely lehetőséget ad arra, hogy több törlési műveletet hajtsunk végre egyszerre, de ezek mindegyikét egyenként explicit módon meg kell majd adni.

A módosító műveletekre egy standard válasz érkezik amennyiben sikeresek, amit tekinthetünk üres válasznak is. Amennyiben valamilyen probléma adódott, akkor természetesen a hibával

térünk vissza.

Mindemellett a kliens oldalra bízzuk azt is, hogy megerősítést, esetlegesen további ellenőrzéseket végezzen majd az ilyen műveletek kapcsán. A szerver oldali program szempontjából nem hiba, ha valaki kitörli az egész adatbázist, hiszen joga van hozzá.

3.2.2 Adatbázis kommunikáció Xquery-vel

Egyik legfontosabb pontja jelen dolgozatunknak az, hogy hogyan tárolunk és kérünk vissza XML dokumentumokat vagy részeket. Mint korábban kitárgyaltuk, itt XML kiszolgálási tudással rendelkező adatbázisokat hívunk segítségül. Elhangzott többször is, hogy elsődleges szempont a kiválasztásnál az, hogy a rendszer ismerje az SQL -hez hasonló XML kijelölő nyelvet, az Xquery-t, hiszen ennek a segítségét szeretnénk felhasználni ahhoz, hogy hatékonyan és elegánsan kérjük le adatainkat. Mindemellett a teljesítmény tényezők miatt is megéri ezzel dolgoznunk. Elég csak abba belegondolnunk, hogy amennyiben nem lennénk képesek szerver oldalon sorrendbe helyezni a lekérdezett tartalmat, akkor azt a szerver oldali programban kellene megtenni. Ehhez pedig ott újra értelmezni kellene a visszakapott XML részt és dolgozni vele, ami hatalmas mennyiségű felesleges számításigényt jelent. Arról nem is beszélve, hogy az adatbázisrendszerünk vélhetően sokkal hatékonyabb formában tárolja a dokumentumokat, mint ahogy azt mi visszakapjuk tőle (sima szövegként) és jobban is tudja alkalmazni a teljesítmény optimalizációs eljárásokat mint a rendszerünk, pl. a cache-elést és az indexelést.

Ahhoz azonban, hogy elkezdjük kitárgyalni a konkrét lekéréseinket, először is tisztáznunk kell, hogy mi az az XQuery és miért olyan jó ez a célunkhoz.

3.2.3 XPath és XQuery

Ahhoz, hogy megismerjük az Xquery-t és az általa nyújtott lehetőséget, először is az igen fontos részét képező Xpath technológiát kell körüljárnunk. Bár használata egy XQuery kifejezésnek csupán egy része, de ettől függetlenül kihagyhatatlan.

Az Xpath egy hivatalos szabvány a W3C részéről, amely segítségével csomópontokat jelölhetünk ki egy XML dokumentumban. Egy ilyen „Xpath” nem más mint egy útvonalleírás, melyben különböző kritériumok szerint adjuk meg, hogy pontosan mit is

szeretnénk megtalálni.

Maga a szabvány az XML dokumentumok strukturáltságát használja ki és azt, hogy a csomópontok címkéi alapján egy jól olvasható és használható útvonal adható meg. Ezen útvonal minden eleméhez kritériumokat adhatunk meg, melyek leszűkíthetik az adott szinten kijelölt elemeket.

Az Xpath szintaktikája kifejezetten egyszerű. Szintaktikai elemként egy csomópont neve kiválasztja az összes alatta található gyermekét a fában, amelyet ha egy / jel követ, akkor újabb csomópont nevet adhatunk meg. A / jel mindig a gyökeret jelenti a részfában ahol járunk, így egy csomópont után írva (mivel a részfának az lesz a gyökér eleme) egy szinttel mélyebbre hatolhatunk. A // karaktersorozat segítségével kijelölhetünk a teljes részfában elemeket, nem csak az adott szinten. A . karakter segítségével az aktuális pozíción szereplő csomópontot adjuk meg, a .. segítségével pedig a szülő elemre hivatkozhatunk. Emellett lehetőségünk van még attribútumok kijelölésére is, ezt a @ karakterrel tehetjük meg. Így pl. megadhatunk egy olyan útvonalat, mely a gyökér elem alatt lévő data csomópont alatt lévő összes olyan *node* nevű csomópontot kiválasztja, melynek az *id* attribútuma 1:

```
/data//node[@id='1']
```

Ennek eredményeképpen egy vagy több olyan csomópontot (és azok gyermekeit is egyben) jelölünk ki, melyeket egy az egyben felhasználhatunk. Természetesen ezzel a módszerrel elég nehéz úgy kijelölni elemeket, hogy csak azokat az adatokat kapjuk meg amire ténylegesen szükségünk van. Ilyenkor elég hosszú kifejezéseket is létrehozhatunk, ami nem feltétlen hasznos.

Az Xpath szabványból jelenleg a 2.0 verzióval jelzett a legújabb. Emellett az 1.0-s verzió létezik még. Ezek szoros összhangban állnak a később említésre kerülő XSLT szabvány 1.0-s és 2.0-s verzióival, talán nem meglepő módon azért, mert az azonos verziók egymást használják fel. A második Xpath szabvány már elég sok mindent tartalmaz, amivel a korábban említett korlátozások kikerülhetőek, de mi jelen helyzetben mégis inkább az Xquery-re fogunk koncentrálni. Példának okáért a 2.0-s szabvány már rendelkezik for ciklussal is, mely megfelel egy lebutított XQuery FLOWR lekérésnek is, mindemellett mindenhol szekvenciáként teszi elérhetővé az adatot. Az un. atomi értékek egy elemből álló szekvenciaként jelennek meg. Atomi értékek alatt olyan csomópontokat értünk, melyeknek nincs gyermekük.

Az Xpath emellett lehetőséget biztosít arra, hogy tetszőleges mennyiségű XML dokumentumot felhasználjunk egy kifejezésben. Ugyanis rendelkezik egy doc függvénnyel, mely mindig a paraméterül megadott dokumentum gyökerét adja meg. Így lehetőségünk van akár egyszerre több forrásból is kiválasztani csomópontokat, vagy több forrás alapján. Pl. sokat találkozhatunk majd később a `doc('database.xml')` kifejezéssel, mely nem meglepő módon az adatbázisunkat tartalmazza.

Ennek a lekérésnek a finomítására jelent meg az XQuery szabvány, mely hivatalos W3C specifikáció. Teljes egészében tekintve két részből áll, egy lekérdező nyelvből és a korábban az adatbázisoknál már emlegetett update nyelvből, mely a tartalom frissítésére szolgál. Ez utóbbi még csak ajánlás és nem lépett szabvány szintre, ill. sok helyen hiányos is. Ezért történhet meg az, hogy a különböző rendszerek különbözőféleképpen implementálják le. Emiatt is egy kicsit kevésbé részletesen fogunk foglalkozni az update résszel, minimálisra szorítva a felhasznált részeit. Mint kiderül persze, nem is nagyon lesz szükségünk bonyolult beszúrásokra ill. összetett módosításokra, mivel a kliens-szerver közti kommunikációt is úgy határoztuk meg, hogy a műveletek minél egyszerűbbek legyenek. Mondhatni egy- az-egyben átfordíthatóak a klienstől kapott kérések Xquery-re, némileg a szintaktika cserélésével.

Természetesen ettől függetlenül nem adjuk meg azt a lehetőséget, hogy közvetlen Xquery-t küldjön a kliens rendszerünk a szerver oldali programnak, hiszen ez azt a látszatot keltené, hogy az úgy ahogy van le is fut és annak az eredményét kapjuk vissza. Ilyet mi semmiképpen sem akarunk (ez esetben nem is lenne sok értelme a programunknak), főként biztonsági szempontokból. Persze az is furcsa lenne, hogy egy szabályos lekérésre a rendszer egy XML-ként visszaküldött hibával válaszol, ami az XQuery esetében elég messze áll a szabványtól.

Az XQuery igazából sokkal kiterjedtebb téma, hogy mi minden részét lefedjük ezen gyorstalpaló keretében, ezért csak néhány fontosabb részét emelnénk ki.

Először is, tudni kell, hogy minden szabályos XML rész, szabályos XQuery kifejezésnek tekinthető, így az `<example>Hello World!</example>` is értelmezhető. Talán nem olyan meglepő módon önmagát adja eredményül.

Ami minket viszont igazán érdekez, az az úgynevezett FLWOR forma (hétköznapi szóhasználatban flower-nek ejtik), ami nem más mint a FOR LET WHERE ORDER BY RETURN kulcsszavakból álló betűszó. Ez erősen hasonlít az SQL -féle SELECT-FROM-WHERE módszerre, azonban már az a tudat, hogy XML dokumentumokat kezelünk, felszólít

minket arra, hogy felejtsük el a relációs adatbázisoknál megismert gondolatokat.

Először is nézzük külön, hogy mit jelentenek az egyes kulcsszavak.

A FOR kulcsszó segítségével egy szekvenciát (vagy akár többet is) hozunk létre egy vagy több Xpath kifejezés segítségével. Pontosabban szólva kijelölünk csomópontokat tetszőleges XML dokumentumban, akár egyszerre többen is. Ezeket a szekvenciákat változókhoz rendeljük, melyeket $ prefixel jelölünk. Egy jó példa a FOR demonstrálására:

```
for $szinesz in doc('hamlet.xml')//actors
```

Ennek segítségével kijelöltük az összes szereplőt a Hamlet című műből, melyeket a $szineszek változóhoz kötöttünk. Egy XQuery kifejezés majdnem megfeleltethető egy programozási nyelvekben megszokott ciklushoz, a kijelölt elemeken egyenként végig iterálunk.

A LET kulcsszó segítségével a kijelölés után újabb változókat hozhatunk be a kifejezésbe. Mint alias-ok fognak ezek segíteni nekünk abban, hogy átláthatóbb módon tudjunk később hivatkozni elemekre. Pl.:

```
let $fizetes := $szinesz/salary
```

A WHERE kulcsszó természetesen szűrési feltételeket tartalmaz, melyekkel szűkíthetjük az érintett csomópontok sorát. Itt egy vagy több tetszőleges feltételt adhatunk meg. Pl.:

```
where $fizetes >= 20000
```

Az ORDER BY segítségével sorrendet adhatunk meg a kijelölt és szűrésen átesett csomópontokat tetszőleges szempont szerint. Példának lássuk:

```
order by $fizetes descending
```

A RETURN kulcsszó pedig az így létrehozott szekvencia kimenetét fogja formázni. Talán ez a rész az egyik legfontosabb, hiszen itt fogjuk látni, hogy tetszőleges módon formázott XML szövegrészt legyárthatunk kegy ilyen lekérdezésből. Leginkább ennél fogva akartuk azt elérni, hogy a szerver oldali programunkban ne szerepeljen semmilyen XML feldolgozó rész, mivel az XQuery teljes erejét kihasználva már a lekéréssel együtt megoldhatjuk ezt könnyedén.

Lássunk egy példát rá:

```
return
  <szinesz>
    <nev>{data($szinesz/nev)}</nev>
    <fizetes>{$fizetes}</fizetes>
  </szinesz>
```

Természetesen itt tetszőleges komplexitással építhetjük fel a visszakapott adatokból az XML dokumentumot. Sőt, lehetőségünk van akár egyenesen XHTML gyártására is, egy ilyen XQuery kifejezés segítségével. Azonban ezzel megint csak nem akarunk élni ezen dolgozat során, hiszen ahogy kifejtettük korábban, szeretnénk a megjelenítést teljesen különválasztani a többi résztől. Mindemellett nem feltétlen lenne szerencsés, ha a rendszerhez képest kifejezetten sokat változó megjelenítési formákat bele kellene fogalmaznunk egy lekérésbe. Ezzel vélhetően erőteljesen megnövelnénk a szükséges adatforgalmat is.

Egy ilyen FLWOR kifejezésből természetesen egy lekérésben több is szerepelhet, tetszőleges elosztásban és módon. Lássunk szemléltetésképpen egy bonyolultabb példát, amiben fény derül néhány nagyobb képességére is az Xquery-nek.

(*A példa a wikipedia.org -ról származik*)

```
<html><head/><body>
{
  for $act in doc("hamlet.xml")//ACT
  let $speakers := distinct-values($act//SPEAKER)
  return
    <div>
      <h1>{ string($act/TITLE) }</h1>
      <ul>
      {
        for $speaker in $speakers
        return <li>{ $speaker }</li>
      }
      </ul>
    </div>
}
</body></html>
```

Mint látható, egyszerűen vagyunk képesek tetszőleges bonyolultságú XML vagy XHTML dokumentumot felépíteni csupán a lekérdezett adatok segítségével. A nyelv kapcsos zárójelekkel jelöli az értelmezendő kifejezéseket és tetszőleges számú Xpath vagy XQuery kifejezés szerepelhet benne.

Mind emellé az XQuery az Xpath-al együtt egy igen méretes függvénytárat biztosít számunkra, amelyekkel megkönnyíti életünket. Itt az XML csomópontok kezelésére szolgáló módszereken kívül találhatunk aritmetikai, dátumkezelő, string kezelő és még sok mást is. Ennek tárgyalása már ténylegesen teljes mértékben kívül esik ezen dolgozat témakörén. Erre általában könyveket szoktak szentelni.

Emellett lehetőségünk van még saját eljárásokat, függvényeket is definiálni, melyeket később meghívhatunk. Az XQuery csak kifejezésekkel dolgozik, nem tartalmaz állításokat, még akkor sem, ha néhány helyen egyes kulcsszavak utalnának erre. Így amennyiben egy függvényt akarunk definiálni, azt egyszerűen a következőképpen tehetjük meg:

```
declare function local:doubler($x) { $x * 2 }
```

Emellett ki kell emelnünk ismételten azt, amit korábban az adatbázisoknál elmondtunk, hogy a tárolt függvények, eljárások használatával igen nagy optimalizációs lehetőséget adunk az adatbázis-kezelő szoftverünknek. Így egy esetleges későbbi továbbfejlesztési fázisban természetesen megpróbáljuk kiemelni a lekérések ismétlődő részét.

Kicsit foglalkozunk még az XQuery Update Facility -vel, vagyis az XQuery frissítő nyelvével is (röviden XQUF). Mivel ahogy már sokszor említve volt, ez egy még nem teljesen tisztázott része a specifikációnak, így lehet, hogy az itt leírtak később már nem lesznek használhatóak egy újabb feldolgozó motor esetében, ill. nem úgy fognak viselkedni.

A szintaktikánál elsősorban a használt adatbázisrendszerek által nyújtott dokumentációt vesszük figyelembe, nem pedig a W3C által ajánlott specifikációt, hiszen utóbbival nem sokat érünk, ha az adatbázis szoftverünk máshogy szeretné megkapni a kérést.

Elsőként tartalom beillesztését fogjuk kitárgyalni. Ez angolul az **insert** szóval érhető el. Xquery-ben ez konkrétan a **do insert** kulcsszavakkal használható. Formálisan a beillesztés szintaktikája a következőképpen néz ki:

```
do insert <mit> as <hogyan> into <hova>
```

vagy

```
do insert <mit> (after|before) <hova>
```

Itt értelemszerűen a *mit* helyére egy XML kódrész fog kerülni, amit be akarunk illeszteni a dokumentumunkba. A *hogyan* segítségével az első variációban azt adhatjuk meg, hogy a kijelölt szinten hova kerüljön a megadott XML kódrész, így itt lehetőségünk lesz arra, hogy *first* -t vagy *last*-t írjunk, melynek függvényében a beillesztendő tartalom a szinten a legelső csomópont vagy a legutolsó lesz. Amennyiben ennél pontosabban szeretnénk meghatározni az adott szinten elfoglalt helyét, akkor a második módszert kell alkalmaznunk és egy konkrét csomópontot kijelölni, amely elé vagy mögé tudjuk helyezni. A *hova*, egy Xpath kifejezés lesz természetesen. Ezzel a kifejezéssel tetszőleges számú csomópontot megjelölhetünk és így több helyre is beszúrhatjuk egyszerre a tartalmunkat.

Lehetőségünk van csomópontok kicserélésére is, amikor egy kiválasztott elem helyére tetszőleges XML kódrészt helyezünk be, ezt formálisan a következő módon tehetjük meg:

```
do replace <mit> with <mivel>
```

Illetve még egy nagyon fontos dolog, amit tehetünk az a csomópontok törlése. Itt egyetlen paranccsal, akár tetszőleges számú elemet is törölhetünk a dokumentumból, ezt formálisan a következő féleképpen tehetjük meg:

```
do delete (<mit>[, <mit>...])
```

Összegezve most megismerhettek felületesen az XQuery alapjait. A nyelv maga természetesen ennél sokkal többet nyújt, de maga a CMS rendszer fejlesztése során se fogunk sokkal elmélyültebb és bonyolultabb kifejezéseket használni, mint amiket itt láthattunk.

3.2.4 XQuery gyártása a kliens oldali lekérésekből

Mint említettük a legfontosabb feladata szerver oldali programunknak a kliens oldali lekérések megfeleltetése XQuery nyelvre és az adatok áromoltatása. Az is felmerült, hogy ebben a dolgozatban alkalmazott lekérések a szerver felé meglehetősen könnyen átalakíthatók. A következőkben végig fogjuk tárgyalni a különböző feladatokat, amik érkezhetnek a programunkhoz és azt, hogy azt formálisan milyen módon küldi tovább az adatbázisunk felé.

Természetesen emellett még több feladat is szerepet kell, hogy kapjon egy-egy munkamenet folyamán, amelyeket a lekérések teljesítésével párhuzamosan fogunk ellátni. Ilyen feladat pl. maga a munkamenet kezelése is, melyet nem tudunk megoldani egyszerűen csupán az XQuery motor segítségével (bár könnyen megeshet, hogy megfelelő mennyiségű idő rászánásával lehet találni olyan megoldást, ami ezt is elvégzi). Itt többek közt a munkamenetek létrehozása (egyedi azonosító hozzárendelése és lejárati idő meghatározása) és törlése (lejárati idő vizsgálata minden kérésnél) mellett azt is el kell intéznünk, hogy a munkamenetünket kötni tudjuk egy felhasználóhoz (autentikálás, ill. amikor még nem történt meg, akkor mindig egy speciális felhasználóhoz rendeljük, ez pedig a Guest vagy Vendég lesz), ill. a bekövetkezett permanens változtatásokat le kell mentenünk a felhasználónk adatbázis részébe is.

Ezek kifejezetten egyszerű kifejezésekkel megoldható feladatok, így nem is szánunk rá különösebb figyelmet a későbbiekben. Emiatt könnyen megeshet, hogy egyetlen egy lekérés alatt a szerver oldali programunk több Xquery-t is átküld majd az adatbázisnak, azonban ez teljesen megszokott dolog a CMS rendszerek világában, sőt általában több tíz lekérésről beszélünk. Ettől függetlenül érdemes törekednünk arra, hogy minél kevesebb legyen ebből, hiszen ha feltételezve, hogy kis adatokkal dolgozunk, akkor a legdrágább tevékenységünk maga a lekérés intézése lesz (mellette elhanyagolható időt vesz igénybe az XQuery kifejezés kiszámolása).

Szükségünk van még korábban emlegetett jogosultság kezelésre is. Ezt már valamivel könnyebben bele tudjuk fogalmazni egy XQuery kifejezésbe, így ahhoz, hogy a szerver oldali programunknak ne kelljen közvetlenül XML-el foglalkoznia, egybe fogjuk fonni a többi kéréssel. Erre elég jó lehetőséget ad nekünk a nyelv, hiszen rendelkezik elágaztató konstrukcióval, amit kihasználva máris felvázolhatunk egy egyszerű lekérés mintát, ahol $x az aktuális oldal, $op az elvégzendő művelet, $user pedig a felhasználó aki végre akarja hajtani a műveletet, az is_entitled() pedig egy általunk létrehozott függvény, ami megmondja a számára beadott paraméterek (melyek a korábban felsoroltak) alapján, hogy a művelet szabad-e vagy nem szabad elvégezni.

```
{
  if(cms:is_entitled($x, $op, $user))
    then do <lekérés>
  else return <error>Önnek nincs jogosultsága ezen művelet
végrehajtásához</error>
}
```

Ezzel a módszerrel máris belefogalmaztuk egy XQuery lekérésbe a jogosultság kezelést, így ezzel már nem kell a szerver oldali programunkban foglalkozunk. Továbbiakban az előbbi kódrészben is megtalálható <*lekérés*> résszel fogunk foglalkozni.

Először is tárgyaljuk azt az eshetőséget, amikor egy oldalt szeretnénk lekérdezni. Ekkor rendelkezünk egy útvonallal, egy vagy több szűrőfeltétellel és egy paraméterlistával. Tekintve, hogy az útvonalban nekünk az oldalak nevei találhatóak, amik az adatbázisban attribútumokként vannak jelölve, ezért ez könnyen átfordítható egy egyszerű Xpath kifejezésre, amely pl. root/subpage/somepage esetében a következőképpen néz ki:

```
doc('database.xml')/data/pages/page[@name='root']/page[@name='subpag
e']/page[@name='somepage']
```

Ezzel lényegében ki is tudjuk jelölni azokat a dokumentumokat, amelyeket megcéloztunk. Innentől kezdve könnyedén tudunk létrehozni egy XQuery for állítást, amely a következőképpen néz ki a példánál maradva:

```
for $x in
doc('database.xml')/data/pages/page[@name='root']/page[@name='subpag
e']/page[@name='somepage']
```

Ezután feldolgozzuk a szűrőfeltételeket. A szerver oldali program feladata, hogy megtalálja az itt és a paraméterlistában elhelyezhető nevek adatbázisbeli megfelelőjét. Ezeket ott egyszerűen beleégetjük majd a kódba, elvégre nem lesz belőle túl sok és azokat a jövőben nem kívánjuk változtatni. Így pl. ha van egy feltételünk, mely úgy szól, hogy [page.index >= 0 and page.index <= 20] akkor azt a következőképpen tudjuk egy where ágba átültetni:

```
where $x[position() >= 0] && $x[position() <= 20]
```

A paraméterlistát pedig egyszerűen a return részben fogjuk megadni, így ha lesz nekünk egy olyan listánk, amely a következőképpen néz ki:

```
content => page.content, author => page.author, realname =>
author.realname
```

Akkor azt egyszerűen a következő XQuery return-é konvertáljuk át:

```
return <result>
<content>{data(doc("database.xml")/data/pagecontent/page[@id=$x/data
[@name='lastid']])}</content>
  <author>{data($x/data[@name='author'])}</author>

<realname>{data(doc("database.xml")/data/users/user[@name=data($x/da
ta[@name='author'])]/data[@name='realname'])}</realname>
</result>
```

Ez talán egy kicsit magyarázatra szorulhat a sok bonyolult kifejezést látva. A tartalom, mint ahogy az adatbázis struktúránál említve volt, külön szerepel az oldaltól, ezért úgy szerezzük meg, hogy megkeressük azt a csomópontot, melynek az *id* attribútuma megegyezik a közvetlen a vizsgált oldal alatt található *data* csomóponttal, melynek a neve a *lastid*, amit szintén egy attribútumként tárolunk. Ez a lastid tartalmazza azt az azonosítót, amellyel megtaláljuk az adott oldal éppen aktuális tartalmát, mivel volt arról szó, hogy adatbázis szinten lehetőséget akarunk biztosítani a régi változatok elmentésére is. A tulajdonos lekérdezése nem igénye különösebb magyarázatot és ezek után a tulajdonos adatainak a lekérdezése sem.

Ezzel létre is hoztunk egy általános formát, amely segítségével könnyedén megszerezhetünk bármilyen oldallal kapcsolatos adatot az adatbázisból. Ezek után a felhasználó adatainak a lekérése sem jelenthet problémát (ügyelve természetesen arra, hogy nem közérdekű adatokhoz, ha lehet akkor az adminisztrátorokon és tulajdonoson kívül senki más ne férjen hozzá).

Amennyiben nem konkrét oldalakat akarunk lekérdezni, hanem keresést akarunk végrehajtani, akkor azt is könnyedén megoldhatjuk úgy, hogy a **for** résznél a kijelölést `doc('database.xml')/pages//page` -re módosítjuk és a where résznél elhelyezünk még egy kitételt, mely a keresésre vonatkozik.

Vegyük figyelembe azt, hogy az XQuery FLWOR kifejezések where részét általában teljes

mértékben áthelyezhetnénk a kijelölést végző Xpath kifejezésbe, vagy akár fordítva is. Mi itt nem teszünk különbséget a kettő közt és csak az olvashatóságot meg az elvi különbségeket figyelembe véve döntjük el, hogy hova kerüljön valamilyen szűrési módszer. Ez azért lehetséges, mert az XQuery az XML csomópontokkal kapcsolatos műveleteit teljes mértékben az Xpath 2.0 szabványból veszi át, így ez nem korlátoz minket sehol sem. Vélhetően az adatbázis rendszer számára is közömbös a dolog, hiszen a tárolási forma igen nagy valószínűséggel köszönő viszonyban sem áll a sima XML szöveges állománnyal és maga a kijelölés is teljesen más módon megy végben a háttérben.

Az oldalak létrehozása és módosítása sem jelent ezután problémát számunkra. Itt egy dolgot figyelembe kell vennünk még, hogy nem mindegy, hogy új tartalmat hova helyezünk el. Az adatbázis szerkezetnél nem definiáltunk sorrendet és a lekérésekben sem szerepel (egyelőre) a sorrendbehelyezés. Ez igazából nem azért van, mert nem akarunk foglalkozni vele, hanem azért, mert egy minimális implementációval dolgozunk. Így azt jelentjük ki, hogy a csomópontok sorrendje az adatbázisban időtől függ (a legújabbak kerülnek legelőre) és lekérdezéskor a visszakapott oldalak sorrendje mindig megegyezik a fizikailag jelenlévő sorrenddel. Ezt figyelembe véve írhatunk egy példa kódot arra, amikor oldalt hozunk létre. Feltételezzük, hogy rendelkezésünkre áll minden információ, akkor a következő utasítást szeretnénk majd végrehajtani:

```
{
if(exists(doc('database.xml')/data/pages/page[@name='root']) and
not(exists(doc('database.xml')/data/pages/page[@name='root']/page[@n
ame='új oldal']))) then
  do insert <page name=”új oldal”><data name=”lastid”>15</data><data
name=”author”>admin</data></page> as first into
doc('database.xml')/data/pages/page[@name='root'],
  do insert <page id=”15”><![CDATA[ Ez egy új oldal lesz! ]]></page>
as last into doc('database.xml')/data/pagecontent
else return <error>Az adott oldal már vagy létezik, vagy nincs szülő
felette!</error>
}
```

Emellett természetesen még más adatokat is tároltunk egy oldalról az adatbázisban, de

egyelőre a példa egyszerűsítésért azokat itt nem szerepeltetjük.

Hasonló módon van lehetőségünk az oldalak tartalmának frissítésére és törlésére is, igazából ezek már az eddig látott kódrészletek kisebb nagyobb ismétlései lennének.

3.3 Kliens oldali követelmények

Eljutottunk oda, hogy nagyrészt mindent tisztáztunk a CMS rendszer szerver oldali részéről és az adatbázisról. Meghatároztuk egy szintaktikát melynek segítségével kommunikálni lehet majd a rendszerrel és egyfajta API-t hoztunk létre, mely bárhonnan bármivel használható. Természetesen webes rendszerre gondolván itt elsősorban a felhasználó egy böngésző segítségével fogja megtekinteni a feltöltött tartalmakat és ehhez még további eszköztárat kell nyújtanunk.

Ismerve eddigi gondolkodásunkat tudjuk, hogy a kliens oldalnak a feladata a megjelenítés lesz. A megkapott XML dokumentumokat (szabványnál maradva) XHTML dokumentumokká kell alakítania és biztosítani kell egy felületet a lekérésekhez is. Emellett még rengeteg dolog jöhet szóba, ami szintén a kliens oldalra kerülhet és megkönnyíti a felhasználást, de ezek általában mind-mind a megjelenítési réteghez tartoznak. Itt egy elég csekély részét fogjuk felületesen fedni annak, amit meg lehetne valósítani.

A W3C szerencsére már létrehozott egy ajánlást, amely eredetileg pontosan arra lett szánva, amit itt el akarunk érni. Lássuk ezt részletesebben.

3.3.1 Az XSL és az XSLT

Eredetileg léteztek a HTML oldalak, melyek egy megjelenést írtak le meglévő elemek segítségével. Ehhez később csatlakozott a CSS (Cascading Style Sheets) specifikáció, mellyel stíluslapokat lehetett csatolni a meglévő HTML oldalakhoz. Ezzel a módszerrel nem csak központosítani lehetett a megjelenés részleteit, de akár teljesen át is alakíthattunk már meglévő részeket más formára.

A később megjelenő XML technológiához is létrehoztak egy stíluslap jellegű nyelvet. Ez az XSL, azaz az eXtensible Stylesheet Language névre hallgat. Ez lényegében ugyanazt a célt szolgálja, mint a HTML-nél a CSS, vagyis az egyes elemek megjelenéséért felelős. Mivel XML-ben nincs kötve az egyes címkék jelentése (tekintve, hogy mi adjuk a nevüket), ezért lényegesen különbözik a kettő.

Azt a folyamatot, amikor az XML-t XSL segítségével valamilyen más formára hozzuk, transzformációnak nevezzük és itt érkezünk el a következő technológiához, az XSLT-hez, azaz az XSL Transformation-höz. Ennek segítségével vagyunk képesek egy XML

dokumentumból bármilyen más létrehozni, vagyis inkább transzformálni. Természetesen mi itt elsősorban XHTML-ben gondolkodunk, mint végső kimenet, így erre adunk majd példákat.

Az XSLT-ből jelenleg két verzió létezik, az 1.0 és a 2.0. Az itt későbbiekben felsorolt példáknál a kettő közti különbség nem nagyon fog érződni. Néhány dolgot átgondoltak benne és kicseréltek, ill. talán a legfontosabb, hogy a 2.0-s szabvány az Xpath 2.0-t használja, így annak képességei kiaknázhatóak.

Az XSLT dokumentumok valójában XML dokumentumok lesznek. Rendelkezik egy speciális névtérrel, amely az *xsl* névre hallgat, és ezek segítségével tudunk transzformációs utasításokat megadni. Az összes többi elem bármi lehet lényegében, akár egy szimpla XML, de akár XHTML is. De emellett még rengeteg formát tudunk definiálni ilyen módon, pl. az Office 2007 formátumát az OpenXML-t is.

Ezen dokumentumok a hagyományos XML nyitórész mellett egy speciális nyitó címkével is szerepelnek, amely a következő (itt jelenleg az XSLT 1.0-s szabványt használjuk):

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

vagy

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

A kettő ekvivalens. XSLT-ben un. template-eket (sablonokat) definiálunk, melyeket illesztünk a feldolgozandó XML dokumentum egy részére. Másfelől nézve azt mondhatjuk, hogy XSLT-ben olyan transzformációs szabályokat adunk meg, melyek alkalmazhatóak egy XML dokumentum bizonyos részére (ez a rész akár az egész is lehet). Így pl. nem mondhatjuk azt, hogy egy transzformáció nem volt sikeres, mert nem jó dokumentumra alkalmaztuk, hanem csak azt mondhatjuk, hogy nem történt transzformáció, mert a dokumentumban nem szerepelt olyan rész, amire tudtuk volna alkalmazni. Template-eket a következőképpen tudunk definiálni:

```
<xsl:template match="/">
  <helloWorld>Ez a template tartalma</helloWorld>
</xsl:template>
```

Itt pl. létrehoztunk egy olyan sablont, ami az egész dokumentumra alkalmazható. Ez természetesen nem jó felfogás, mert akármit kap bemenetnek, arra megpróbál transzformációt alkalmazni (mint később látjuk ettől függetlenül még így is hibatűrésen belül vagyunk). Gyakorlatban a sablonok akkor használhatóak nagyon jól, ha egy komplett dokumentumot próbálunk transzformálni és nem egy kis részt, mint ahogy majd mi tesszük. Így itt valószínűleg sokat fogunk találni olyan match attribútumokkal, melyek az egészet kijelölik.

Amennyiben az előző példát alkalmaznánk egy XML-re, akkor értelemszerűen mindig ugyanaz a karakterlánc lenne az eredmény. Mondhatni nagy nehezen már eljutottunk egy hello világ alkalmazásig.

Természetesen ezzel nem érnénk sokat, így lehetőségünk van az XML-ből elemeket kiemelni, a *value-of* művelet segítségével. Ebben egy Xpath kifejezéssel tetszőleges csomópontnak az értékét tudjuk átemelni a kimenetbe. Lássunk egy példát rá:

```
<xsl:template match="/">
  <xsl:value-of select=”valami/semmi” />
</xsl:template>
```

Ennek a helyére kerül majd a kiválasztott elem értéke. XSLT-ben elsősorban értékekkel dolgozunk és nem XML csomópontokkal. Természetesen a szimpla elem kiemeléssel sem kapunk kezünkbe még túlzottan nagy előnyt, ezért lehetőségünk van néhány magas szintű konstrukció megadására is, így pl. ciklusokat írhatunk:

```
<xsl:template match="/">
  <table>
    <tr><td>Név</td><td>Életkor</td></tr>
  <xsl:for-each select="valami/ember">
    <tr>
      <td><xsl:value-of select=”name” /></td>
      <td><xsl:value-of select=”age” /></td>
    </tr>
  </xsl:for-each>
  </table>
</xsl:template>
```

Itt már egy teljes XHTML részletet hoztunk létre, amelyben egy XML dokumentumból kiemeljük az összes ember elemet és annak kiíratjuk a nevét és életkorát. Ezzel a példával már elég jól szemléltethető, hogy mennyire könnyedén használható az XSLT ilyen jellegű átalakításokra, bár azt még a szerző se merné kijelenteni, hogy programozási tudással nem rendelkező embereknek is a kezébe lehetne adni.

Lehetőségünk van még elágaztató utasításokat létrehozni, sorba rendezést meghatározni és akár még több ágú elágaztatást is megadhatunk a *choose* kulcsszó segítségével. Természetesen a kifejezések a feltételekhez mind Xpath kifejezések lesznek. Ezekre most ömlesztve lássunk egy példát:

```
<xsl:template match="/">
  <table>
    <tr><td>Név</td><td>Életkor</td><td>Felnőtt?</td></tr>
  <xsl:for-each select="valami/ember">
    <xsl:if test=”race = 'human'”>
      <tr>
        <td><xsl:value-of select=”name” /></td>
        <td><xsl:value-of select=”country” /></td>
        <td><xsl:choose>
                  <xsl:when test="age &gt 18">Igen</xsl:when>
                  <xsl:when test="age &lt 18">Nem</xsl:when>
           <xsl:otherwise>Éppenhogy</xsl:otherwise>
                </xsl:choose></td>
      </tr>
    </xsl:if>
  </xsl:for-each>
  </table>
</xsl:template>
```

Itt már egy egész összetett példát láthatunk arra, hogy milyen az amikor felhasználjuk az XSLT tudását. Mindemellett az XSLT rendelkezik még egy funkcióval, aminek segítségével lehetőségünk van akár egymásba ágyazni a különböző sablonokat. Ez az *xsl:apply-templates*, mely a jelenleg kijelölésben lévő elemre vagy annak gyermekeire alkalmazza az összes lehetséges sablont. Így külön tudjuk választani a megjelenését az egyes elemeknek, úgy, hogy

közben egymásba ágyazzuk őket.

Ezzel nagyjából áttekintettük azt, hogy mire lehet képes egy XSLT dokumentum. Ha jobban belegondolunk, akkor észrevehetjük, hogy az Xquery-t akár tejes mértékben képes helyettesíteni. Ellenben nem adatok lekérdezésére van kitalálva, hanem transzformációkra. Mi a dolgozatban arra használjuk, amire való.

3.3.2 XML transzformáció böngészővel

Azok után, hogy tisztáztuk az XSLT fogalmát, itt az ideje, hogy az alkalmazását is átnézzük. Mivel az, hogy a rendszerünktől visszakapott XML részeket konkrétan milyen formában öntjük át, az mér tényleg alkalmazás kérdése, ezért nem térnénk ki ilyen példákra. Mivel az egész dolgozat egy elméleti architektúrát valósít meg és azt is csak minimális reprezentatív szinten, ezért nem fogunk azzal foglalkozni, hogy mennyi minden újítást, biztonsági eljárást vagy egyéb tudást lehetne még beleépíteni a kliens oldalba.

Helyette nézzük azt, hogy milyen módokon tudunk XSLT-t használni. A legtöbb nagy böngésző (IE6+, Firefox 3, Opera9 …) mind rendelkezik beépített XSLT feldolgozóval és képes ilyen műveletet végrehajtani weboldalak megjelenítéséért. Természetesen elsősorban nem arra tervezték őket, hogy teljes honlapokat ezzel jelenítsenek meg, de ettől függetlenül képesek rá. Elmondhatjuk azt is, hogy mag a transzformáció nem jelent hatalmas terhelést a böngészőre nézve, általában századmásodpercek alatt végrehajtja akár nagyobb mennyiségű adaton is. Természetesen ennek kipróbálása része volt a dolgozat előtti kutatómunkának, így nem a végén érte a szerzőt a meglepetés, hogy ez működik.

Alapvetően két módon tudunk transzformációt végrehajtatni a böngészővel. Az első megoldás szerint az XML dokumentumba helyezünk el egy hivatkozást a stíluslapunkra. Ezt a `<?xml-stylesheet type="text/xsl" href="xslt_stiluslap.xsl"?>` sor beírásával tehetjük meg. Ekkor a böngésző amennyiben meg kívánja jeleníteni az XML dokumentumot, a jelzett .xsl fájl segítségével fogja megtenni. Ez egy elegáns megoldásnak tekinthető, de nem feltétlen a legjobb. Mivel nem feltétlen akarunk ilyen hivatkozást elhelyezni és az eredeti célunk is az, hogy a válaszként küldött kódrész még köszönő viszonyban sem legyen a megjelenítéssel, ezért nem kívánatos ezt a módszert választanunk.

Ehelyett lehetőségünk van arra, hogy JavaScript-ből közvetlenül kérjünk a böngészőtől transzformációt. Ebben már sokkal nagyobb erő rejlik, hiszen mi magunk választjuk meg,

hogy pontosan mit akarunk mivel átalakítani és akár böngésző specifikus változtatásokat is eszközölhetünk a művelet előtt, ha erre szükség van. A mai napi web fejlesztési tapasztalat pedig azt mondja, hogy erre szükség van. Már eleve ha csak a lehetőségeket figyelembe vesszük, akkor mindenképpen csábítóbb az utóbbi megoldás, hiszen így teljes mértékben vezérlést kap a kliens oldal afelett, hogy miként kívánja életre hívni az XML dokumentumok tartalmát. Így lesz lehetőségünk akár arra is, hogy egy tartalmat több módon is megjelenítsünk, újabb lekérés nélkül és anélkül, hogy a szervernek erről bármit is tudnia kellene.

3.3.3 XML transzformáció a szerver oldalon

Könnyen megeshet az-az eshetőség is azonban, hogy a felhasználónk egy olyan környezettel rendelkezik, ahol nincs lehetősége JavaScript futtatására vagy a böngészője nem képes XSLT transzformációkat kezelni. Ebben az esetben igény kerekedik egy olyan megoldásra a szerver részéről, amelyen keresztül a böngésző már csak a kész XHTML kódot kapja meg.

Annak érdekében, hogy ne kelljen az egészet újraimplementálni, létrehozhatunk egy közbülső klienst is, ami a kliens oldali JavaScript helyett végzi a munkát. A manapság fellelhető hasonló technológiákat használó rendszerekben is ált. szokott lenni un. alap HTML nézet, ami hasonlóképpen működik. Ekkor a szerver oldali „kliensünk” küldi el a lekéréseket a szerver oldali programunknak és ő maga alkalmazza ugyanazon XSLT dokumentumok felhasználásával a transzformációkat, majd a végeredményt elküldi az igazi kliensnek. Ez természetesen nem kívánatos megoldás, hiszen olyan terhelést jelenthet a szervergépnek, amit el akartunk kerülni, de szükséges lehet, amennyiben mindenki számára elérhetővé akarjuk tenni a rendszerünket.

Sajnos a megoldással felmerül több probléma is, mégpedig az, hogy amennyiben a kliens oldalon jelentős mennyiségben alkalmaztunk Javascript-es kódokat, akkor azokat ebben a közbülső részben vagy teljesen újra kell implementálni, vagy szerver oldalon értelmezni ezeket a kódokat. Utóbbi talán elegánsabb megoldás, ellenben megint jelentős terhelést hozhat magával (legalábbis az eddigiekhez képest, amikor is a szerverünk szinte alig dolgozott).

4. Összefoglalás

Dolgozatunk végére érve összefoglalhatjuk a tapasztaltakat és átgondolhatjuk, hogy mennyire érdemes ilyen jellegű próbálkozásokkal továbbiakban foglalkozni.

Ami biztos, az az, hogy a dolgozat előtti kutatómunka kezdetén csupán kísérletinek és gyakorlatban nehezen alkalmazhatónak tartott rendszernek igen is van jövője. Ill. ha magának a rendszernek nem is feltétlen, de a benne felhasznált módszereknek és elveknek mindenképpen.

Az itt tárgyalt technológiák mind használatban vannak már a mai rendszerek nagy részében. Az XML egy kihagyhatatlan elem lett az informatika világában és felhasználása a webes rendszerekből egyszerűen kihagyhatatlan.

A kitűzött feladat az volt, hogy építsünk egy rendszert amely minél több ilyen technológiát használ fel. Ez teljes mértékben sikerült is, hiszen bebizonyosodott, hogy gyakorlatban is működik. Az, hogy alkalmazható-e éles környezetben már más kérdés. Bizony szempontból igen, ugyanis a terhelést ezzel az újfajta architektúrával úgy osztottuk el, hogy az esetleges hátrány amit az XML-el való munkával kapunk bőségesen visszatérítődik azzal, hogy egy egész réteget sikeresen leválasztottunk a szerverünk válláról.

Ellenben meg kell jegyezni azt is, hogy jelenleg nem feltétlen áll készen a felhasználók serege teljes mértékben egy ilyen rendszer használatára. Több technikai probléma is felmerül, egyes esetekben az is megjelenhet, hogy a kliens gépére tett feladatok túlságosan lassúvá teszik egy ilyen weboldal betöltődését, amennyiben az ott lévő hardware nem megfelelő ehhez. Mindemellett az un. teljesen AJAX alapú oldalak (mint amilyen az itt létrehozott is, bár ez nem volt részletezve), rendelkeznek egy pár olyan tulajdonsággal, ami merőben eltérő a hagyományos weboldalaktól. Ezeket nagy általában különböző körüljárásokkal ki lehet kerülni, de egyelőre még mindig nem megszokott az ilyen jellegű rendszerek használata.

Ettől függetlenül maga megoldások egyszerűek és nagyszerűek, hiszen teljesen szabványos eszközökkel építhettünk fel egy olyan rendszert, ami képes ellátni a ma általánosan felmerülő igények igen nagy részét. További finomításokkal akár sokkal többre is képes lehet.

Maga a kutatómunka, amely ezen dolgozat mögött van, bebizonyította azt is, hogy érdemes ilyen technológiákra alapozni. Hiszen folyamatosan fejlődésben vannak, ha maga a szabvány

nem is módosul csak új verzió jön ki hozzá, az implementációi fejlődnek, gyorsabbak lesznek és hatékonyabbak. Így ha jelen esetben problémákba ütközik is az alkalmazás, vélhetően azok később el fognak hárulni.

Amennyiben csak abba gondolunk bele, hogy az egyik jelentősen alkalmazott technológia az XSLT egy olyan prédikációval él ma, hogy a közeljövőben a használata jelentősen megnő majd és a böngészők fejlesztői is ráfekszenek a feldolgozó motor fejlesztésére, akkor már érezzük azt, hogy nem tettünk rossz lóra vele.

Természetesen még mindig rengeteg technológia van, amely most kihasználatlanul maradt, pl. az Xforms, vagy a SOAP, de a realitások és az időkorlátok keretein belül kellett maradjunk. Természetesen ezek mind felhasználhatóak lettek volna, nagyrészt igen hatékonyan (bár pont a SOAP esetében volt megfelelő ellenérv a használata ellen), figyelembe kell vennünk még mindig azt, hogy bár a szabvány az szabvány és az implementáció az implementáció, a kettő viszonya olyan mint az elméleté és a gyakorlaté, ritka ha egyeznek. Ennek következtében fordulhat elő, hogy egy egyszerű XSLT transzformációhoz is több sornyi JavaScript kódra van szükségünk, mert a különböző böngészők különböző módon tették használhatóvá a feldolgozó egységüket.

Természetesen a legtöbb helyen azért előfordul a csoda és teljesen hordozható módon tudjuk használni a megírt dokumentumainkat (pl. XSLT és az XQuery lekérések meglepő módon mindenhol ugyanúgy vannak leimplementálva), de azért rátapintottunk olyan technológiákra, amik még nincsenek is a specifikáció fázisában (lásd XQUF).

Ennek jegyében a mű továbbiakban is kísérleti jellegűnek tekinthető, hiszen egyben egy próba is volt annak megismerésére, hogy az ilyen jellegű tervek megvalósíthatóak a gyakorlatban is.

Ez utóbbi kérdésre igazából csak az informatika jövőbeli fejlődése fog tudni teljes értékű választ adni, jelenleg csak annyit lehet kijelenteni, hogy a merész megoldásokból lettek a legnagyobb újítások.

5. Szakirodalom

A dolgozatban található ismeretanyag elsősorban internetes forrásokból származik, a példák is nagyrészt ezekre támaszkodnak.

Az alkalmazott technológiák és azok specifikációja mind elérhető a World Wide Web Consortium weboldalán: http://www.w3.org/

A technológiák megismerésében és a példákban megfelelő forrást nyújtotta a W3Schools különböző oldalai:

- http://www.w3schools.com
- http://www.w3schools.com/xml/default.asp
- http://www.w3schools.com/xsl/default.asp
- http://www.w3schools.com/xpath/default.asp
- http://www.w3schools.com/xquery/default.asp

Felhasznált irodalom:

- Priscilla Walmsley: *XQuery* (O'Reilly Media, Inc., ISBN: 0596006349)
- Steven Holzner: *XPath Kick Start: Navigating XML with XPath 1.0 and 2.0* (Sams, ISBN: 0672324113)
- Michael Kay: *XSLT 2.0 and XPath 2.0 Programmer's Reference (Programmer to Programmer)* (Wrox , ISBN: 0470192747)