A Neumann János Számítógép-tudományi Társaság
**Webalkalmazások Fejlesztése Szakosztálya**
bemutatja...

# *Magyarországi Web Konferencia 2006*

# *Magyarországi*
# *Web Konferencia*
# *2006*

## Tisztelt Látogató!

Szeretettel üdvözöljük régi-új rendezvényünkön. A konferencia nevének megváltoztatásával szerettük volna jelezni, hogy nem csupán a PHP nyelvre koncentrálunk, teret adva a webes fejlesztésben használt technológiák minél szélesebb körű bemutatásának. Akik rendszeresen látogatták rendezvényeinket, azok tudhatják, hogy nem a *mivel*, hanem a *hogyan* volt az, amire elsősorban koncentráltunk. Konferenciánk erősségét abban látjuk, hogy az egyszerű „ezt hogyan készítem el" bemutatók helyett a témát magasról szemlélő előadásokat részesítettük előnyben. Ennek volt köszönhető, hogy már az első konferenciánkon sem csak a PHP nyelvvel foglalkoztunk, hanem megpróbáltuk felölelni a webes alkalmazásfejlesztés szélesebb spektrumát is. Ennek a törekvésnek esett áldozatul a PHP Konferencia elnevezés, és ez adott lehetőséget arra, hogy megszülessen a 2006. Magyarországi Web Konferencia.

A konferencia célja a szakmai tapasztalatszerzésen túl a témával foglakozó szakemberek és csoportok összegyűjtése és bemutatása, ezért idén is lehetőséget szeretnénk adni a közösségek találkozására, illetve a látogatók ismerkedésére. A második konferencián nagy sikert aratott találkozók keretében idén is lehetőség lesz arra, hogy a levelezőlistákról, IRC csatornákról ismert fejlesztőket élőben is láthassák a résztvevők. A helyszín kiállításain szakkönyveket, szoftvereket, a konferenciához kapcsolódó emléktárgyakat vásárolhatnak az érdeklődők.

Az idei konferencián számos szerver oldali technológia helyet kapott, így a rendezvény lehetőséget ad a különböző megközelítések összehasonlítására, egyfajta útkereszteződésként szolgálva a fejlesztők és döntéshozók számára. Bizonyos bevezető jellegű előadások segítenek az adott technológiákban járatlan érdeklődőknek a terület áttekintésében, de a haladóbb kérdések kedvelői is megtalálják az őket foglalkoztató témákat. Természetesen nem csak a szerver oldali megoldásokkal foglalkozó dinamikus nyelvek, .NET és Java témacsoportok tartanak majd nagy érdeklődésre számot. Az aktív webfelületek modern megoldásainak bemutatásával az asztali programokra egyre jobban hasonlító webalkalmazások fejlesztésének trükkjeivel ismerkedhetnek meg a résztvevők. Idén is fontosnak tartjuk a hozzáférhetőség kérdését, mely nemcsak a látássérült weboldal látogatók számára kulcskérdés, hanem az alternatív elérési eszközök használóinak és a kereső indexeléskor is. Azoknak szólnak a web architektúra témacsoportunk előadásai, akik magasabb szintű koncepcionális meglátásokra és ismeretekre vágynak.

A programfüzet, melyet kezében tart, ezúttal is segítséget nyújt az érdekesebb programpontok kiválasztásához, és hosszú távon is hozzá fordulhat, ha szakmai információkra van szüksége a tárgyalt témákat illetően. Bizonyos előadások meglátogatása előtt különösen ajánlott a kapcsolódó cikk átfutása, hiszen a bemutató megértését jelentősen könnyítheti.

Rendezvényünk nem jöhetett volna létre a számos támogató odaadó segítsége nélkül, melyek lehetővé tették, hogy idén is megvalósíthassuk merész álmainkat. Az NJSZT Webalkalmazások Fejlesztése Szakosztálya és az ELTE Radnóti Miklós Gyakorlóiskola összefogásával és az előadók aktív közreműködésével várakozásunk szerint idén is olyan rendezvényen vehet részt, melyre még sokáig hivatkozni fog.

*Reméljük, hasznos információkkal és új kapcsolatokkal lesz gazdagabb a mai nap végére.*

# Neumann János Számítógép-tudományi Társaság

A hazai informatikai élet meghatározó szereplőjeként a Társaság legfontosabb feladata megőrizni azokat az értékeket, amelyek beilleszthetők a most alakuló új tudásalapú társadalomba, napjaink követelményeinek megfelelő új szakmai irányok kijelölése és a (közel) jövő informatikai társadalmának aktív formálása. Az NJSZT 1968. óta működik, jelenleg 2300 egyéni és száz jogi taggal, 1999. január 1-től közhasznú szervezetnek minősül. Céljai elérése érdekében a Társaság központi, 19 területi-, valamint 24 szakmai szervezetében a következő közhasznú tevékenységeket végzi:

- tudományos tevékenység, kutatás, fejlesztés,
- nevelés és oktatás, képességfejlesztés, ismeretterjesztés,
- szakmai kulturális tevékenység,
- szakmai kulturális örökség megóvása,
- munkaerőpiacon hátrányos helyzetű rétegek képzésének, foglalkoztatásának elősegítése és a kapcsolódó szolgáltatások,
- euroatlanti integráció elősegítése.

A Társaság intézményektől független szakmai fórumként segíti hazánkban, illetve a magyar nyelvterületeken

- az informatika alkalmazását, fejlesztését, az eredmények elterjesztését;
- a szakma presztízsének, minőségi színvonalának és etikájának megőrzését, illetve emelését;
- az informatikával hivatásszerűen foglalkozók, illetve az informatikai eszközöket és módszereket más szakterületen alkalmazók véleményének és szakmai érdekeinek érvényre jutását;
- a széles körű részvételt a nemzetközi szakmai közéletben;
- az informatikai szakemberek tájékoztatását és tapasztalatcseréjét;
- az informatikai kultúra terjesztését, az informatikai oktatást.

### A Társaság tevékenységi köre

A Társaság, célkitűzéseink megvalósítása érdekében közhasznú szervezetként szolgáltatásokat nyújt, illetve vállalkozásoknak ad keretet, ezeken belül:

- szakmai közéleti munkára ad lehetőséget;
- kutatási, fejlesztési, oktatási és továbbképzési programokat véleményez, és részt vállal azok kidolgozásában;
- állami szervek, gazdálkodó szervezetek, társadalmi szervezetek felkérésére, megbízására vagy tagjainak kezdeményezésére állást foglal fontos szakmai és az informatikával kapcsolatos társadalmi kérdésekben, koncepciókat, tanulmányokat, szakvéleményeket dolgoz ki nyilvántartott egyesületi szakértők közreműködésével;
- előadásokat, ankétokat, konferenciákat, kongresszusokat, szemináriumokat, szakmai bemutatókat, kiállításokat, tanfolyamokat rendez;
- szakmai tanácsadást végez, szakértői rendszert működtet, pályázatot hirdet, díjakat alapít és adományoz, célfeladatok elvégzését jutalmakkal ismeri el;
- törekszik arra, hogy a diákokat és a fiatal szakembereket bevonja a szakmai közéletbe;
- tevékenységi területén kapcsolatokat tart fenn különféle bel- és külföldi szervezetekkel, tagként képviseli Magyarországot hazai, ill. nemzetközi tudományos szervezetekben, terjeszti az informatikai írástudást, az ECDL hazai irányítását végzi.

A Társaság testületeiről, bel- és külföldi kapcsolatairól, aktuális szakmai eseményekről részletes információ található a `http://www.njszt.hu/` honlapon és az évenként tíz alkalommal nyomtatásban is megjelenő „Mi Újság” című hírlevélben.

# A World Wide Web Consortium

A World Wide Web Consortium-ot (W3C) 1994-ben alapította Tim Berners-Lee, a Web szülőatyja. A cél elsősorban az volt, hogy a webtechnológiák fejlesztésével foglalkozó vállalatok és kutatóintézetek a jövőben ne forgácsolják szét feleslegesen energiájukat, elért eredményeiket, hanem egymással vállvetve, egymást segítve teremtsék meg az informatika jövőjét.

A W3C egymással együttműködő technológiákat (specifikációkat, szoftvereket) dolgoz ki, ezzel is megkönnyítve a fejlesztők és a felhasználók munkáját.

Ezen fejlesztések alapján születnek meg a W3C ajánlásai, melyekhez bárki hozzáférhet a következő címen: `http://www.w3.org/TR/#Recommendations`. A W3C ajánlások (például HTML, CSS, XML, SOAP) alapján készült termékek szabványosak, széles körben felhasználhatók, ezért a gyártóknak érdekükben áll az ajánlásoknak megfelelő termékeket előállítani, ha vásárlóik igényeit ki akarják szolgálni.

Az elmúlt évben a W3C hat új webes szabványkészletet (Web standards) jelentetett meg, megnyitotta az Indiai Irodát, valamint csökkentette a tagdíjat a fejlődő országokban. Életre hívta a Mobilweb kezdeményezést (Mobile Web Initiative), és tizenegy új csoportot alapított, többek közt a gazdag webkliensek (Rich Web Clients), a hatékony alternatíva az XML-re (Efficient XML Interchange), a szabálykicserélési formátum (Rule Interchange Format) és az egészségügy és élettudományok (Health Care and Life Sciences) területeken.

### A W3C Magyar Iroda

A Konzorciumnak ma már több mint négyszáz tagja van a világ minden részéről (IBM, Microsoft, Nokia, Honda, Fuji, CERN stb.; a teljes taglista: `http://www.w3.org/Consortium/Member/List`). Az MTA SZTAKI 1995 óta tagja a Konzorciumnak. A W3C itt nyitotta meg egyetlen kelet-közép-európai irodáját, a W3C Magyar Irodát. Az Iroda elsődleges célja az, hogy a magyar intézmények és vállalatok minél szélesebb körében megismertesse a W3C tevékenységét, segítse az azokba való bekapcsolódást, illetve azon információkhoz való hozzáférését, amelyek segítségével a jövőben képesek leszünk növelni szerepünket a Web világában. További információ: `http://www.w3c.hu/`

### Mit jelent a W3C-tagság?

A W3C ma közel negyven területen végez fejlesztéseket (HTML, XML, HTTP, P3P, RDF, ...). Ezeket olyan munkacsoportjai végzik, melyekbe a W3C-tagok delegálhatnak munkatársakat, s így részt vesznek a Web jövőjének alakításában, valamint első kézből – jóval a hivatalos bejelentés előtt – jutnak hozzá olyan információkhoz, melyek a webtechnológiákkal kapcsolatos stratégiai döntésekben helyzeti előnyt jelentenek. Ha egy tag a fejlesztésekben nem is vesz részt, a friss információkat a hírleveleken keresztül folyamatosan megkapja és hasznosíthatja, mely már önmagában nagyon értékes lehet. Ezen kívül delegálhat tagot a Tanácsadó Testületbe, s ezáltal beleszólhat az ajánlások kidolgozásába. További információ: `http://www.w3c.hu/forditasok/why_join.html`

## NJSZT Webalkalmazások Fejlesztése Szakosztály

A Neumann János Számítógéptudományi Társaság Webalkalmazások Fejlesztése Szakosztályában immár közel három éve azon munkálkodunk, hogy a webes környezetek fejlesztésével foglalkozó szakemberek és érdeklődők számára a legaktuálisabb és szakmailag legkorrektebb információkat közvetítsük. Az Első Magyarországi PHP Konferencián alapított független, szakmai közösségünk célja a webalkalmazások fejlesztésére használható technológiák népszerűsítése, szakmai anyagok írása, fordítása, azok támogatása, továbbá a programozási nyelvekkel és a kapcsolódó technológiákkal foglalkozó konferenciák és versenyek szervezése és lebonyolítása.

Egyik fontos tevékenységünk a nagy érdeklődéssel kísért budapesti tavaszi konferenciák megszervezése, melyeken a webes technológiák legújabb megoldásairól informálódhatnak az érdeklődők, rálátva a különböző megközelítésekre, a technológia eltérő vetületeire. Ezeket az országos eseményeket a kisebb, vidéken megrendezett roadshow rendezvények kísérik, köszönhetően a helyi lelkes szervezőgárdáknak.

A konferenciákon és roadshow állomásokon túl számos más projekt foglalkoztatja a szakosztály tagjait. A legnagyobb a Weblabor nevű szakmai lapunk folyamatosan aktuális hírekkel és cikkekkel történő megtöltése, és szolgáltatásainak javítása. A Weblaboron olvasható, a közösség által beküldött tartalmak mutatják, hogy a közösség egyes tagjai kis egyedi befektetéssel együttesen komoly eredményeket tudnak elérni. Ezeknek a közösségeknek a katalizátorai a Weblabor keretében működő szakmai levelezőlisták és fórumok, melyek lehetőséget adnak a felmerülő problémák megvitatására, megoldására. Szakosztályunk alapítói is egy ilyen közösségi kemény magból kerültek ki.

Amellett, hogy bevált rendezvényeinket és jól működő közösségi webhelyünket tovább szeretnénk működtetni, számos tervünk van, melyek megvalósításához segítő kezekre van szükség. Szívesen látunk körünkbe bekapcsolódni szándékozó új tagokat. Amennyiben sikerült felkeltenünk érdeklődését szakosztályunk iránt, látogassa meg közösségi találkozónkat az ebédszünetben, kísérje figyelemmel híreinket a `http://weblabor.hu/` oldalon, és látogassa meg a `http://wfsz.njszt.hu/` címen elérhető honlapunkat.

# Tartalomjegyzék

*Az alábbiakban a konferencia tematikus programja olvasható. Minden előadás magyar nyelvű, a konferencián három párhuzamosan futó sávban kerül bemutatásra. A konferencia előadásaihoz, bemutatóihoz és a kerekasztal beszélgetéshez kapcsolódó cikkeket az alábbi oldalakon találja meg.*

# ASP.NET 2.0 és Visual Studio 2005 előnyök a webfejlesztőknek

*Aki lemarad, kimarad – tartja a mondás és ez a webes megoldásokra különösen igaz. Hogyan lehet drasztikusan lerövidíteni a fejlesztési időt olyan szoftverek esetén, amelyek ennyire különbözőek és sokszínűek, mint a webalkalmazások?*

A Microsoft néhány éve komoly szemléletváltáson ment keresztül: a kész szoftverek mellett megoldás platformok szállítójává vált. Ezek a platformok és a hozzájuk kapcsolódó fejlesztőeszközök elsősorban a tipikus igények hatékony kielégítését tűzik ki célul, azaz azt, hogy a gyakori problémákra gyorsan lehessen megoldásokat adni.

## Hogyan ne írjunk saját keretrendszert?

Nincs ez másként a Microsoft webfejlesztői platformja, az ASP.NET esetében sem. Bár az ASP.NET 1.1 verziója jelentősen megkönnyítette a webfejlesztők munkáját, számos olyan ismétlődő feladat maradt, amely arra késztette a web programozókat, hogy ezekhez – a többnyire alapfunkciókhoz – egyedi osztálykönyvtárakat készítsenek. Az ASP.NET 2.0-val a Microsoft elsődleges célja, hogy *platform szintű* támogatást nyújtson a webalkalmazások fejlesztésével kapcsolatos gyakori feladatok elvégzéséhez.

Ezen tipikus feladatok közé tartozik:

- a relációs és hierarchikus adatok, objektum alapú gyorsítótárak kezelése,
- a felhasználók regisztrációjához, bejelentkezéséhez, személyes (profil) adataihoz kapcsolódó szolgáltatások nyújtása,
- a különböző jogosultságokkal rendelkező felhasználók megkülönböztetése és a jogkörökhöz alkalmazkodó HTML kód generálása,
- az oldalak tartalmához igazodó, mégis egységes felhasználói felület biztosítása,
- a kódtól és az oldalaktól független arculat, bőrök (skin) és témák kialakítása,
- a lokalizáció és többnyelvűség támogatása,
- a konfigurációs beállítások egyszerű módosítása és felolvasása,
- a webhely tartalmához és a felhasználói tevékenységhez igazodó navigációs elemek és webhely térképek megjelenítése,
- a webhely állapotának monitorozása, naplózása.

Az ASP.NET 2.0 osztálykönyvtára ezekhez a feladatokhoz beépített támogatást ad, szükségtelenné téve az egyedi API-k és keretrendszerek kifejlesztését.

## Hogyan ne legyenek határok?

Minden keretrendszerrel kapcsolatban az egyik legfontosabb kérdés, hogy hol van az a határ, amin túl már inkább átok, mint áldás az adott környezet használata. Erre gondolva az ASP.NET tervezői olyan *provider* alapú modellt alkalmaztak, amely egységes kiterjesztési mechanizmust biztosít, így a szokásostól eltérő kihívásoknak is *standard módon* lehet megfelelni, ami nem csak a rugalmasságot, hanem a hatékony fejlesztést is garantálja.

## Hogyan ne írjunk kódot?

Még ha van is egy olyan keretrendszerünk, amely segítséget ad programozói feladataink megoldásában, az API felhasználása még mindig kódolást igényel – legalábbis a hagyományos megközelítésben. Ezzel is szakít az ASP.NET, amit két dolog tesz lehetővé:

- A keretrendszer közel 90 (!) beépített vezérlőelemet tartalmaz, melyek többsége összetett feladatokat valósít meg. Minden vezérlőelem felelős azért, hogy szerver oldalon milyen kódot futtat és ennek eredményeként kliens oldalra milyen HTML kódot generál.
- A vezérlőelemek közötti kommunikációt deklaratív módon írhatjuk le, azaz azt határozzuk meg, hogy mit szeretnénk, nem pedig azt, hogy hogyan. A deklaratív leírásból automatikusan kód generálódik.

Ezzel a deklaratív, vezérlőelem alapú megközelítéssel nem csak lényegesen kevesebb kódra van szükség, de a leírás jóval átláthatóbb és karbantarthatóbb is, mint a hagyományos kód.

## Hogyan ne képezzük webtervezővé magunkat?

A generált kóddal – különösen a Microsoftos eszközök által generált kóddal – kapcsolatban gyakran felmerülő kérdés, hogy az eredmény mennyire szabványos, hogyan használható heterogén környezetben? Ezen a területen is komoly előrelépést jelent az ASP.NET 2.0, amelyben egészen mélyen integráltan megtalálható az XHTML, a táblázatok nélküli CSS alapú oldalelrendezések és a böngészőfüggetlen JavaScript támogatása.

A webalkalmazások sokszínűsége általában megköveteli, hogy a webtervezők kiegészítsék a fejlesztőeszközök által generált kódokat. Ebben jelentős segítséget adnak azok az oldal- és webhely sablonok, amelyekből kiindulva egyszerűen készíthetünk látványos és egyúttal a szabványoknak is megfelelő portálokat, valamint alkalmazásukkal megismerhető a „legjobb technológia-alkalmazási gyakorlat”.

## Hogyan ne végezzünk rutinmunkát?

Az elkészítendő ASP.NET 2.0 alkalmazás vagy webhely még ezek után is sok egyedi munkát igényelne, ha nem lenne a webfejlesztés rutinfeladatait magas szinten támogató, integrált fejlesztő környezet (IDE). Még a legegyszerűbb kialakítás, a *Visual Web Developer 2005* is a következőket nyújtja:

- *Vizuális tervezőrendszer:* a WYSIWYG elven alapuló eszközök sokasága, amely a keretrendszer használatához kapcsolódó rutinfeladatok szinte mindegyikét automatizálja.
- *HTML, CSS és JavaScript támogatás:* tervezői üzemmódban is 100%-ban megmarad a kézzel bevitt HTML kód, a HTML forrás érvényességének ellenőrzése külön specifikálható, a HTML jelölőknél is működik az IntelliSense, a beépített vezérlőelemek automatikusan JavaScripttel működnek, ha van értelme, a stílusinformáció HTML-től külön tárolható a webes megoldás könnyű frissítéséhez és a „böngészés" támogatására, végezetül pedig a Style Builder dialógus a HTML elemek stílusának vizuális módon való kialakítását teszi lehetővé.
- *XML, RSS és webszolgáltatás támogatás:* XML fájlszerkesztő séma és IntelliSense támogatással, szintaxis színezés és IntelliSense támogatás az XSLT fájlokhoz, külön vezérlőelem az XML fájl és a vonatkozó XSLT fájl adatengedélyezett vezérlő elemekhez (pl. GridView) kötéséhez, RSS publikálása illetve ezekre való feliratkozás az IDE-ből, szabványos webszolgáltatások publikálása és felhasználása.
- *Vizuális adattervező eszközök:* a fejlesztés közbeni relációs adatbázis funkciók teljességét biztosító SQL Server 2005 Express Edition, az adatbázis lekérdezések vizuális segédeszközökkel való kialakítása, intelligens címkékkel felruházott adatengedélyezett vezérlő elemek, varázsló az adatforrásokhoz való hozzáférések lépésenkénti konfigurálásához, GridView és DetailsView vezérlőelemek a táblázatos illetve egyedi adatok, valamint ezek kombinációinak kódolás nélküli megjelenítéséhez és kezeléséhez.
- *Leegyszerűsített hibaelhárítás:* töréspontok bevezetése és használata, segédeszköz a kivételek kezeléséhez, hibaelhárítási adattippek a kérdéses típusokra mutatva, hibaelhárítás vizualizálási eszközök az összetett adatok megtekintéséhez.
- *Teljes testreszabás:* a fejlesztőkörnyezet személyre szabott kialakíthatósága, a HTML és ASP.NET címkék generálásához finombeállítási lehetőség, a forráskód formázásának beállíthatósága, a betűtípusok és színek állíthatósága, testre szabható kóddarabkák (snippets), import/export beállítások varázslóval, egyedi elemek vagy sablonok létrehozása a másokkal való megosztás céljából.
- *Könnyű konfigurálás és telepítés:* változatos projekttípusok (webszerver, FTP hely vagy közös fájl-hely alapúak), adminisztrációs eszköz a vonatkozó XML címkék kézi kódolásának kiküszöbölésére, MMC snap-in kiegészítés az IIS-hez, beépített FTP támogatás, egyszerű fájl „Copy-Paste"-tel történő telepítés, ASP.NET 2.0 hosztolási támogatás.

## Hogyan ne fizessünk mindezért?

Az ASP.NET alkalmazások fejlesztéséhez és futtatásához egyetlen dologra van szükség: a .NET keretrendszerre – ez pedig ingyenes. Másként fogalmazva, a Microsoft olyan webes fejlesztői platformot és futtatókörnyezetet ad, ami beépítve tartalmazza a leggyakoribb fejlesztői feladatok megoldását, ráadásul ingyen!

A korábban említett deklaratív kód megírásához és felhasználásához elvben nincs szükség fejlesztőeszközre, azonban egészen más a fejlesztői hatékonyság, ha van olyan eszközünk, ami ezt támogatja. Itt is van ingyenes megoldás, még pedig a Visual Web Developer 2005 Express Edition, ami a Microsoft Visual Studio 2005 termékcsaládjának a webes fejlesztésekre korlátozott változata (lásd keretes összefoglalónkat). Fejlesztői adatbázis szerverként ott van az SQL Server 2005 relációs motor Express Edition változata, ami szintén ingyenesen használható. Adott tehát egy ingyenes adatbázis szerver, egy fejlesztőeszköz, egy fejlesztői keretrendszer és futtatókörnyezet – tehát minden, amire dinamikus webalkalmazások fejlesztésénél szükség lehet!

*Balássy György*

*Villamosmérnök, a BME Automatizálási és Alkalmazott Informatikai Tanszékén webportálok fejlesztését oktatja. 2000 óta foglalkozik a Microsoft .NET platformjával, melynek meghonosításában jelentős szerepet vállalt előadóként, konzulensként és A .NET Framework és programozása című könyv társszerzőjeként. Az MSDN Kompetencia Központon belül a Portál Technológiák Csoport vezetője, a www.devportal.hu közösségi honlap szakmai gazdája. 2005 óta a Microsoft magyarországi regionális igazgatója.*

# Felhasználóbarát űrlapok

*avagy mit tegyünk a Web Forms 2-ig és az XForms-ig?*

## Motiváció

Amióta cikkeznek a webes felületek következő generációiról, nagy érdeklődéssel olvasgatok az XFormsról, illetve a Web Forms 2-ről és hasonlókról. Az XForms a W3C gondozásában nagy falat lesz: szinte mindent újra kell tanulnunk: leváltja a mai űrlapokat, máshogy gondolkodik, máshogy működik, más tájszólást ért. Viszont cserébe „szinte mindent" megvalósíthatunk majd vele, ami felett ma bosszankodunk, amiről álmodunk, vagy amit elképzelhetetlennek tartunk egyszerű weblapon. A Web Forms 2 a több böngészőgyártót tömörítő WHATWG munkacsoport gondozásában ezzel szemben reálisabb: a jelenlegi XHTML és JavaScript lehetőségeket bővíti ki, hogy hasonló célokat érjen el, noha alacsonyabbra helyezi a lécet. Célja, hogy ne legyen ilyen nagy váltás és ezért lassú a fejlődés.

Mi tetszik bennük leginkább? Számos új eszköz az egyszerű szöveges beviteli mezők mellé, új hasznos attribútumok és egy csokor új rutin. Csak ízelítőül: dátum- és időpontválasztó, csúszka, minimum és maximum érték, kötelező mező, ismétlődő mezők és sablonok, automata értékellenőrzés és összegzés, űrlap adatok XML formátumba alakítása és küldése.

Reménykedve próbálgatom az Opera9 WebForms 2 képességeit, követem a Mozilla XForms implementálásának lépéseit a fejlesztők blogjában. A HTML világából nézve csodálatosak, innovatívak, nagyon jók, csak késnek, nagyon késnek.

Mit tehet ilyenkor a lelkes fejlesztő? Vagy beáll böngészőgyártónak, vagy pedig fogja a jelenleg is használható megoldásokat és a legjobbat próbálja kihozni belőlük. Mert a Web Forms 2 és az XForms szabványban írtakat ugyan megvalósítják majd, és egy későbbi időpontban már kellően elterjedtek is lehetnek, hogy segítségükkel fejlesszünk, de a mi ügyfeleink vannak olyan pimaszak, hogy a mában éljenek. Vége az álmodozásnak, valamit ki kell találni.

Közel két éve kerültem szembe ezzel a helyzettel. Adott egy jó webalkalmazás, webkőkorszaki felülettel. A „kőkorszaki" persze nem jelent semmi különöset, csak azt, hogy olyan valaki fejlesztette, akinek nem ez a szakterülete. Nevezzük ezt az egyént – egyáltalán nem sértésnek szánva – átlagos szerver oldali programozónak. Ekkor azt a feladatot kaptam, hogy fejlesszek ki egy olyan igényes, „szabványosítható" felületet, amelyet később az átlagos szerver oldali programozó önállóan fel tud használni. A konkrét programból kiemelve, majd kibővítve ez a felület-koncepció a ProForm nevet kapta.

**A köztes megoldásokat általában nem szeretik, de attól még szükség van rájuk.**

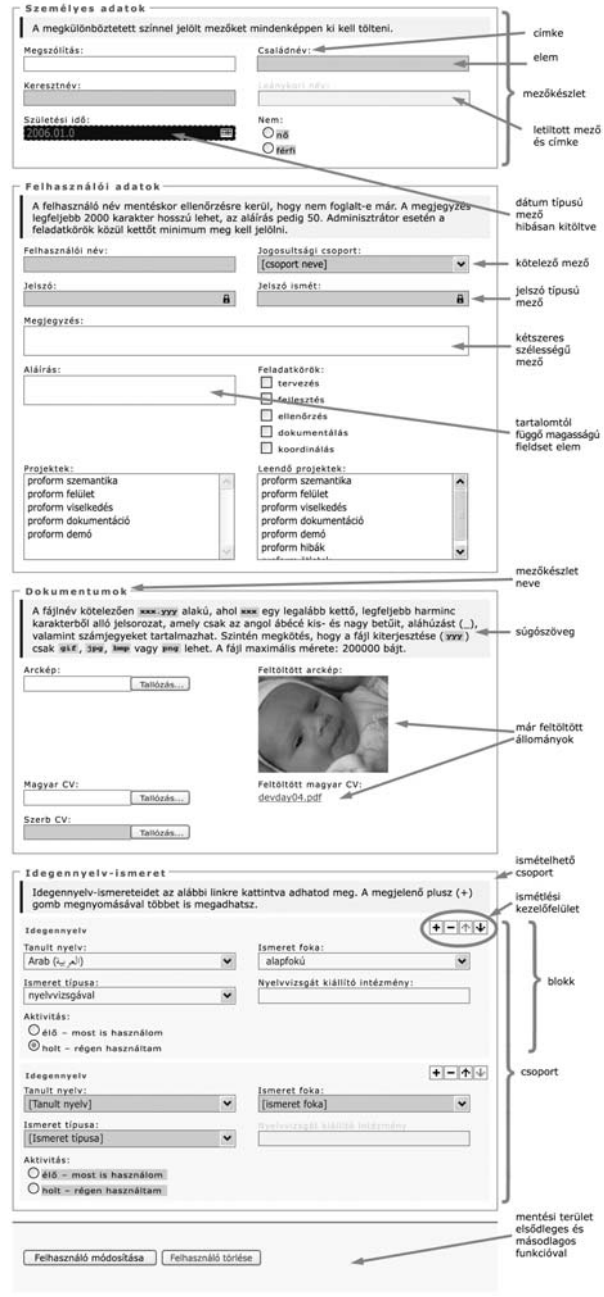
Azóta történt az alábbi eset: az egyik nagyobb megrendelő egy másik céget választott egy projektjükhöz. A mű elkészült. A megrendelő nem vette át, mivel a webes program ugyan úgy működött, ahogy kell, de a felülete tényleg borzasztó volt. Mi volt a megoldás? Megbízott bennünket, hogy készítsük el a program felületét. Micsoda égés, micsoda elismerés…

**Ma nem elég ha működik, jó és szép felület is kell hozzá.**

Jelen előadás célja tehát felismerni a jó felületek fontosságát és néhány konkrét példa segítségével betekintést nyerni a ProForm megoldásba. Az előadás gerince gyakorlati bemutató, nem érthető a ProForm felületek működés közbeni látványa nélkül.

## ProForm

Vannak tehát kőkorszaki felületek, ezekkel nem foglalkozunk. Vannak úgynevezett kreatív webfelületek, ezzel akkor foglalkoznánk, ha mindenki tudna már ProForm szintű egységes űrlapokat építeni. A ProForm tehát egy sablonos felület. Az ereje abban áll, hogy következetesen alkalmazva nagyon letisztult, egyszerű és használható alkalmazásokat kapunk.



Mi valójában a ProForm a fejlesztő számára? Egy XHTML-re, CSS-re, erős DOM szkriptelésre építő felületi keretrendszer, amely egy átlagos webalkalmazás minden elemét tartalmazza. Lényegében tehát egy részletes felületkialakítási koncepció, egy kidolgozott megjelenés CSS-ben, egy JavaScriptben megírt űrlapkezelő objektum érvényesítéssekkel, XForms-szerű ismétlődő mezőkkel, és a legégetőbben hiányzó funkciókiegészítésekkel, valamint egy kódolási megállapodás XHTML-ben. Jelenlegi verziója működik Internet Explorer 6-ban, Firefoxban és Opera 8-ban.

## Rövid áttekintés

### Űrlap

Az űrlap a legalapvetőbb felület, szigorúan hierarchikus felépítésű: mezőkészlet (`fieldset`), csoport (`table`), blokk (`tbody`), mező (`input/textarea/select`) a tagolódása. Ami leginkább megtetszett az Web Forms 2-ben és meg akartam valósítani: az űrlapokban lehetnek ismétlődő mezők – olyan csoportok, amelyek több blokkal rendelkeznek. Ismétlődés hozzáadásakor, ismétlődő csoport törlésekor, csoportsorrend módosításakor az egyes elemek, vagy maga az egész blokk akár változhatnak is (pl. hozzáadáskor módosítás), erre kényelmes JS metódusok készültek. Így például egy felületen lehet megvalósítani az újabb adat hozzáadását és a meglévő adatok módosítását.

Az űrlapok viselkedését JavaScripttel tudjuk konfigurálni. Erre egy interfészt ad kezünkbe a ProForm: alapesetben az űrlap elküldhetősége a kérdés (minden kötelező mező ki van-e töltve és nincs formai hibásan kitöltött). A könnyedsége abban áll, hogy az ismétlődő elemeket is ugyanúgy kezelhetjük, mint az egyszeres elemeket. Elküldéskor – PHP-vel feldolgozva – az ismétlődő mezők tömbként fognak megjelenni.

### Lista

A lista lapozható és rendezhető: fejlécből és elemekből áll. A funkciógombokat a füles böngészést lehetővé téve GET metódussal szokás kialakítani. A lista szűréséhez is egységes felületet kapott.

### Nézet

A nézet a hagyományos lekérdezés megjelenése.

### Információs panel

Az információs panel a visszaigazolási üzenetek, kapcsolódó (összetett) funkciók megjelenésének helye. Használatával könnyen tudunk rendezettséget alkotni, hisz legtöbbször ezek a kapcsolódó funkciók vesznek el valahol az oldal alján.

### Fül

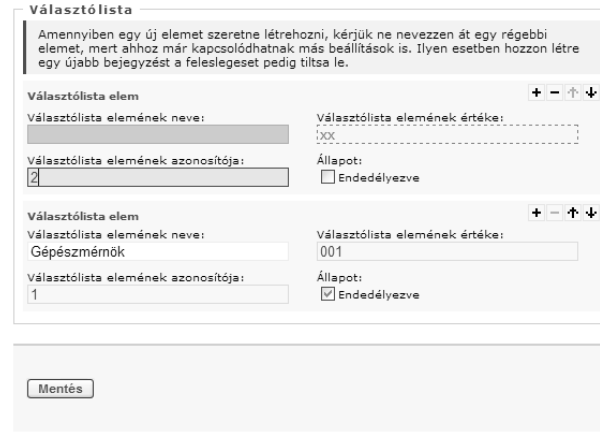
Alkalmazásával egyszerűen tudjuk összefoglalni az egyszerű műveleteket, jellemzően: az első fülön a lista, a második fülön az új elem hozzáadása, a harmadik fülön pedig egy meglévő elem módosítása.

### Összevonás

Az űrlap speciális fajtája, ahol két azonos típusú egyedet tudunk összevonni, az alkotóelemeiknek kiválasztása és módosítása mellett, gyorsan és látványosan. Ez az a felület, ami a legtöbb webes programból kimarad, mondván, hogy „ilyet nem lehet csinálni”.

## Első példa: választólista elemeinek beállítása

Ugorjunk a közepébe. Első feladatként valósítsunk meg egy választólista (`<select>`) lehetséges elemeinek beállítását lehetővé tevő felületet. Hogyan tennénk ezt kőkorszaki módon? Először készítenénk egy listát, ahol minden meglévő elemet felsorolnánk egy-egy módosítás és törlés gombbal. Másodjára egy új elem létrehozása űrlapfelületet, ahol egyenként tudnánk felvinni az `<option>` elemeket. Ennek egy változata lenne a már meglévő `<option>` elemek módosítása. Ráadásnak még készíthetünk egy „sikeresen töröltük az elemet” felületet is. Ez négy felület. Lássuk, hogyan vonjuk össze ezt egyetlen felületté ProFormmal egy ismétlődő csoport segítségével.




*Egy már meglévő és egy hozzáadott elem látható*

Először is elkészítjük a szükséges HTML kódot, melynek csak a váza látható az alábbiakban. A `fieldset class="proform"` fogja a CSS osztályokat érvényesíteni. Az űrlap és a táblázat azonosítóira fogunk hivatkozni a viselkedés JavaScript-tel való leírásakor. A mezők neveiben és azonosítóiban legyünk figyelmesek a sorszámozásra.

```
<form id="main_form">
  <fieldset class="proform">
    <table id="group_select"><tbody><tr>
      <td><label
➥ for="combo_name:0">Név:</label><input type=
➥ "text" name="combo_name[0]" id="combo_name:0"
➥ class="proform-equal" /></td>
      <td><label
➥ for="combo_value:0">Érték:</label><input type=
➥ "text" name="combo_value[0]" id="combo_value:0"
➥ class="proform-equal" readonly="readonly" />
➥ </td></tr></tbody>
    </table>
  </fieldset>
</form>
```

Ez után elkészítjük a viselkedés leírását. Inicializáljuk erre az űrlapra az ProForm-ot (1). Majd beállítjuk a csoportot az azonosítójával hivatkozva (3), megadjuk az ismétlődési jellemzőket (4), felsoroljuk a kezelni kívánt mezőket (5), beállítjuk az eseményfeldolgozót (6), kikapcsoljuk a csoport törlése gombot (11) és egy módosítót is elhelyezünk a csoporton (12). Amikor elindítjuk (18), a ProForm automatikusan létrehozza a kezelőfelületet (+-↑↓), mivel nem engedélyezzük a meglévő elemek törlését, a törlés gombot közben lekapcsolja.

Felismeri a felsorolt azonosítók alapján a mezők ismétlődéseit, majd lefuttatja a talált blokkra a megadott eseményfeldolgozót.

```
 (1) proform.initForm('main_form');
 (2) proform.addGroup( {
 (3)   id : 'group_demo4',
 (4)   repeat : true, repeatMin: 1,
➥ repeatMax : 10,
 (5)   elements : [ 'combo_name', 'combo_value',
➥ 'combo_id', 'combo_enabled' ],
 (6)   processor : function (oT) {
 (7)     var aReturn = [];
 (8)     aReturn.push( proform.condition.required
➥(oT.combo_name.value == '', [oT.combo_name]));
 (9)     aReturn.push( proform.condition.warning
➥ (oT.combo_value.value != '' &&
➥ !proform.condition.checkFormat('number',
➥ oT.combo_value.value), [oT.combo_value],
➥ ['Számmal kell kifejezni.']));
(10)     return proform.condition.totalize
➥ (aReturn); },
(11)   config : { disableButtonRemove : true },
(12)   alter : { onAddRepetitionBlock : function
➥ (elTable, elTBody, oT) {
(13)     oT.combo_value.removeAttribute
➥ ('readonly');
(14)     oT.combo_id.removeAttribute('readonly');
(15)     oT.combo_enabled.setAttribute('checked',
➥ 'checked');
(16)     removeClass(elTBody.panelButtons.remove,
➥ proform.config.disabled);
(17)   }, } } );
(18) proform.init();
```

Amikor a felületből később hozzáadjuk a képen is látható új mezőket, lefut egy belső funkció, ami újraszámozza ezeket. Ez után a megadott módosító funkció hajtódik végre, ami elvégzi a mezőkön a kért változtatásokat (13-15) és bekapcsolja a törlés gombot (16), majd lefut erre a blokkra is az eseményfeldolgozó és kijelöli a kötelező mezőket (8). Amikor begépelünk egy karaktert a létrehozott új mezőkbe, lefut újra az eseményfeldolgozó és történetesen felismeri, ha nem számmal adtunk meg egy értéket (9).

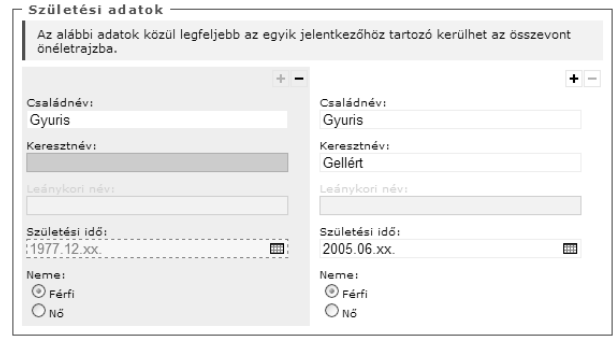
Figyeljük meg a még ki nem töltött, a hibásan kitöltött, az éppen szerkesztés alatt lévő, és a csak olvasható (readonly) mezők megjelenését.

## Második példa: kétszer regisztrált személy adatainak összevonása

No lássuk, mi az, amit nem szokás megvalósítani. Elég gyakori eset, hogy valaki elfelejti a jelszavát és kétszer beregisztrál egy portálon, kétszer jelentkezik egy állásra vagy egy konferenciára. Az adminisztrátor szeretné törölni az adatait, de látja, hogy az egyikben nem adott meg minden adatot, amit a másikban. A ProForm erre is ad egy felületet.

Létrehozzuk a HTML kódot az előzőekhez hasonlóan, de két táblázattal, amelyek azonosítóiba :a és :b utótagok kerülnek. A JavaScript viselkedésleíró is szinte teljesen azonos, csupán egy további rejtett mezőt kell megadnunk, amely kapcsolóként tárolja, hogy melyik oldalt választjuk ki. Figyeljük meg hogy az eseményfeldolgozó csak azon az oldalon jelöli meg a még nem vagy hibásan kitöltött mezőket, amelyet éppen megjelöltünk. És természetesen ez működik ismételhető mezőkre is, ahol nem csak az egyik oldalt, hanem akár minden tételt kiválaszthatunk.

**A következetes felület szép és használható.**




*Összevonási űrlap*

## Következtetések

Tehát adott egy felületi keretrendszer gyakorlati oktatóanyaggal társítva. Csak bemásolom a CSS és JavaScript állományokat, a megadott útmutatók alapján építem fel a felületeket XHTML-lel (és egy kis JavaScript-tel is) és máris egy színvonalas felületet készítettem. Egyszerű.

Aktuális-e a ProForm? Tippem szerint még két év kell, mire az XForms vagy a Web Forms 2 egyike az alternatív böngészők között teljes támogatottságot nyer, a most vezető böngésző gyártójánál pedig isteni csoda kell, hogy natívan implementálják bármelyiket. Szerintem, két évig még biztosan a ProFrom és hasonló megoldások maradnak a porondon. Lehet-e fejleszteni? Nekem is vannak új ötleteim, és alakulóban van egy jókora frissítés előkészítése, de érdemes megvárni hozzá az Internet Explorer 7 megjelenését.

Manapság egyre nagyobb figyelmet kapnak a kódgeneráló segédletek. Bár a készítéskor ez nem volt szem előtt, elmerengtem már azon, hogy mi lenne ha például a Ruby on Rails scaffold funkciója ilyen minőségű felületeket generálna. Sok átlagos szerver oldali programozó életét könnyítené meg.

Amikor összeállítottam a ProForm-ot, azért tettem, mert nem volt hasonló más megoldás. Vedd, tanuld meg, használd, fejleszd tovább, alkoss jobbat!

További példák és forráskód:
`http://opensource.nexum.hu`

*Gyuris Gellért*

*Szegeden él, jelenleg a webfejlesztéstől visszatérve eredeti hivatásához. Egyetemi évei alatt hobbiból ismerkedett a webbel, majd öt évig volt aktív webfejlesztő a nexum MarsNet Kft.-nél. Nem a szerver oldali programozás a szakterülete, hanem a webfelületek kialakítása és a böngészők. Az utóbbi két évben szinte csak webalkalmazások felületeivel foglalkozott. Honlapja az arcok.ujevangelizacio.hu/bubu címen található.*

# Flex: csak rugalmasan!

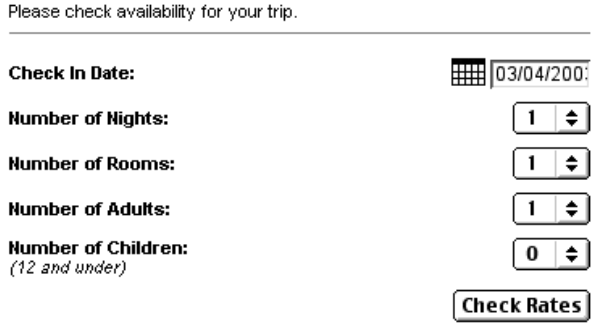
*Webfejlesztőként nap mint nap találkozunk a web korlátaival, melyek bizony eléggé megkeserítik mindennapi munkánkat. Bevethetjük a JavaScriptet űrlapjaink használhatóbbá tétele érdekében, de ha már egy kicsit is többet szeretnénk kihozni alkalmazásainkból e téren, hatalmas mennyiségű JavaScript és DHTML kódot kell írnunk. Még ha szakértők vagyunk is a témában, rendkívül sok időt igényel, hogy interaktívvá és felhasználóbaráttá tegyük alkalmazásainkat. A kérdés adott: miért pazarlunk oly sok időt a felhasználói felület kifejlesztésére, amikor erre kész megoldások vannak, és nekünk inkább az üzleti logika és a hatékonyság a fontosabb?*

Az Adobe/Macromedia Flex elvezet bennünket a webalkalmazások következő generációjának fejlesztéséhez, a gazdag internetes alkalmazásokhoz (Rich Internet Applications – RIA). Korábban is láthattunk már gazdag funkcionalitással, lenyűgöző felhasználói élménnyel bíró Flash alkalmazásokat, melyeknek megfelelő felületek megvalósítása HTML-ben szinte lehetetlen. Köszönhető ez annak, hogy Flash lejátszóval az internetezők 98%-a rendelkezik. A Flex azonban elfeledteti velünk a Flash hagyo-mányos használatakor alkalmazott időegyenesek, jelenetek, alakzatok fogalmát, s ehelyett kontrollokban, eseményekben gondolkodhatunk.

A cikkhez kapcsolódó előadásból tehát megtudhatjuk, hogy hogyan fejleszthetünk RIA-kat a Flex legújabb, 2.0 Beta 1 verziójának segítségével, milyen eszközök állnak rendelkezésünkre. A bemutató egyúttal összefoglalót is ad a jelenleg elérhető alternatív technológiák előnyeiről és hátrányairól: AJAX, Avalon, XUL, XAML, SVG, Java appletek, OpenLaszlo.


## Az élmény számít

Képzeljük el a következő helyzetet: szobát szeretnénk foglalni egy hotel weboldalán. Először ellenőrizzük a szobakapacitást (első weblap), majd várunk. Utána kiválasztjuk a szobát (második weblap). Megint várunk. A kiválasztott dátumra már nincs szabad szoba, irány vissza, kiválasztani egy másik időpontot. És ismét várunk. Néhány oldalon még az előtt kell megadnunk a hitelkártyánk adatait (harmadik weblap), mielőtt az árakat közölné a weboldal. Megadjuk az adatokat, ugyancsak várunk. Ez után összegezve láthatjuk a foglalásunk eredményét (negyedik weblap). Elborzadunk az ártól, nagyon magas! Előről kezdjük az egészet. Ezt olvasni is unalmas, hát még végigvinni… Akkor miért használja ezt mindenki?




Felmérések szerint az e-kereskedelem hozzávetőlegesen 44%-nyi potenciális bevételtől esik el azért, mert a cégek nem képesek a felhasználók igényeire koncentrálni. A hotel idejében felismeri a problémát, megújítja foglalási rendszerét, áttér a RIA-k használatára. A négy oldalt megszüntetik és helyettük már csak egy interaktív oldal várja a foglalni vágyókat. Kiválaszthatunk egy hétvégét a családunknak a nekünk tetsző szobával. A válogatás során megjelenik a szoba képe, anélkül, hogy az oldal újratöltődne.

A teljes oldal sohasem töltődik újra, nem kell visszamenni az előző oldalra, még csak várni sem kell. Ugye, hogy mennyivel jobb élmény most már a foglalás is?

## Mik azok a RIA-k?

A RIA-k átmenetet képeznek a webalkalmazások és a hagyományos asztali alkalmazások között. Céljuk, hogy az asztali alkalmazásokkal megegyező felhasználói élményt nyújtsanak. Futtatásukhoz szükségünk lehet egy webböngészőre, de nem kizárólag ebben a környezetben lehetnek használhatóak, hiszen léteznek olyan RIA-k melyek PDA-ra, mobiltelefonra illetve egyéb eszközre is elérhetőek.

A hagyományos webalkalmazásoknál minden folyamat a szerveren megy végbe, a kliensen csak a statikus HTML tartalom jelenik meg. Ennek a rendszernek a hátulütője tehát az, hogy minden egyes általunk kiváltott interaktivitásnak keresztül kell mennie a szerveren. Ez úgy valósul meg, hogy a webalkalmazás adatokat küld a szervernek, és a szerver válaszul visszaküld egy új oldalt, ami letöltődés után megjelenik a felhasználó böngészőjében. Ez történik akkor is, mikor egy linkre kattintunk: várunk, a szerver előállítja a statikus tartalmat az adatbázis és a szerveroldali környezet segítségével, majd megjelenik a képernyőn az új oldal.

A RIA-k ezzel szemben ki tudják használni a kliens processzorának lehetőségeit is, így valós idejű felhasználói felületet biztosítanak, ami a hagyományos HTML oldalakkal nem oldható meg. Ennek köszönthetően bármilyen felhasználóbarát elemet be tudunk vezetni a kliensoldalon, anélkül, hogy alkalmazásunk kommunikálna a szerverrel: fogd és vidd módszert (drag and drop), új adatok háttérben történő letöltését, kliensen végzett számításokat. A RIA-k másik előnye a teljesítményben keresendő: nem kell a szervernek feleslegesen teljes weblapokat generálnia; a RIA egyszer töltődik le a felhasználó számítógépére, utána már csak kisebb adatcserék szükségesek. Az eredmény: a hálózati forgalom drasztikusan csökkeni fog, gyorsabban tudjuk teljesíteni a kéréseket.

## Adobe Flex – áttekintés

A Flex első, 1.0-ás verziója 2005. márciusában jelent meg, tehát viszonylag új technológiával állunk szemben. Az azóta folyó intenzív fejlesztés gyümölcseként 2006. február 1-én megjelent a Flex ezen cikk írásakor legújabb, 2.0 Beta 1-es verziója. A Flex 2.0 alkalmazások futtatásához a Flash Player 8.5-ös verziójára van szükségünk, az 1.5-tel készített RIA-k a 7.0-ás verzióval is megelégszenek. A Flex 2.0 Beta 1 a következő termékekből áll:

- Flex Framework 2.0
  Maga a Flex keretrendszer
- Flex Builder 2.0
  Eclipse alapú integrált fejlesztői környezet, mellyel hatékonyan fejleszthetünk RIA-kat a Flex alá. Felépítése hasonlít a Dreamweaver-ére, tehát hamar elsajátítható a kezelése. Alkalmazásainkat a hagyományos forráskód és a Macromedia-tól már megszokott vizuális nézetben is elkészíthetjük.
- Flex Enterprise Services 2.0
  Segítségével adat importálási réteggel ruházhatjuk fel a RIA-kat.
- Flex Charting Components 2.0
  Látványos grafikonok, diagramok készítésére használhatjuk.
- Flash Player 8.5
  A Flex 2.0-ás RIA-k futtatásához nélkülözhetetlen

## Flex Framework

A Flex Framework tulajdonképpen a Flex lelke, ez szolgáltatja azokat az elemeket, melyek elengedhetetlenek a Flash platformra építkező RIA-k fejlesztéséhez. A Flex Framework három, egymástól jól elkülöníthető komponensből tevődik össze:

- MXML (Macromedia Flex Markup Langauge)
  A Flex XML alapú leírónyelve, mely segítségével meghatározhatjuk, hogy hogyan nézzenek ki az alkalmazásaink, hol helyezkedjenek el az objektumok a képernyőn. Mivel ez egy szabványos leírónyelv, az MXML fájl ugyanúgy szerkeszthető egy egyszerű szövegszerkesztőben (akár jegyzettömbben is), mint az e célra kifejlesztett Flex Builder-ben. Segítségével maximálisan kihasználható az egyes részek újrahasznosítása, hiszen általa erősen elkülönített a megjelenítés és az üzleti logika.
- ActionScript 3.0
  Az ActionScript egy ECMAScript alapú, objektumorientált, JavaScript-hez hasonló programnyelv, mely kiválóan alkalmas a Flex alkalmazások kliens oldali logikájának megírására. A fordítás során az MXML kódból is ActionScript kód lesz, így könnyedén megírhatjuk komponenseinket az előbbiben is. ActionScript-en keresztül tudunk kommunikálni a külvilággal is, webszolgáltatásokat vagy távoli objektumokat elérve.
- Flex osztály könyvtár (Flex class library)
  Tartalmazza a beépített osztályokat, így ezeket nem kell magunknak megírnunk. Ismerős lehet a Java és .NET fejlesztőknek, ahol a `java.*` illetve a `System.*` jelöli az alaposztályakat. Flex-ben ezeket az `mx.*` tartalmazza.

A következő példában egy termékkatalógus listáját fogjuk megjeleníteni, az adatokat egy XML fájl szolgáltatja majd. A felület és az adatok leírása a következő:

```
termeklista.mxml:
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
➥ xmlns:mx="http://www.macromedia.com/2005/mxml"
➥ xmlns="*" pageTitle="Terméklista">
 <mx:Model id="termekek" source="termekek.xml"/>
 <mx:DataGrid dataProvider="{termekek.termek}"/>
</mx:Application>
```

| ar | nev | raktaron |
|---|---|---|
| 1000 | Törölköző | igen |
| 900 | Sampon | igen |
| 800 | Tusfürdő | nem |
| 100 | Szappan | nem |

```
termekek.xml:
<?xml version="1.0" encoding="utf-8"?>
<termekek>
 <termek>
  <nev>Tusfürdő</nev>
  <ar>800</ar>
  <raktaron>nem</raktaron>
 </termek>
 <termek>
  <nev>Sampon</nev>
  <ar>900</ar>
  <raktaron>igen</raktaron>
 </termek>
 <termek>
  <nev>Szappan</nev>
  <ar>100</ar>
  <raktaron>nem</raktaron>
 </termek>
 <termek>
  <nev>Törölköző</nev>
  <ar>1000</ar>
  <raktaron>igen</raktaron>
 </termek>
</termekek>
```

Mint láthatjuk, ennél egyszerűbb már nem is lehetne! A termékeket anélkül tudjuk rendezni név, ár és a raktári elérhetőség szerint, hogy ezt a funkciót külön megírtuk volna. Az adatokat tartalmazó XML fájlt természetesen bármilyen szerver-oldali nyelv előállíthatná, akár adatbázisból is.

## ActionScript 3.0

A következő egyszerű példában az ActionScript használatára láthatunk egy alkalmazást. Az `onCreateComplete`

függvény fordításkor automatikusan lefut, jelen esetben beállítja a `cimke` nevű felirat szövegét.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
➥ xmlns:mx="http://www.macromedia.com/2005/mxml"
➥ xmlns="*" pageTitle="ActionScript"
➥ creationComplete="onCreationComplete()" >
 <mx:Script>
  <![CDATA[
    private function onCreationComplete() {
      cimke.text='Hello World!';
    }
  ]]>
 </mx:Script>
    <mx:Label id="cimke"/>
</mx:Application>
```

## RIA-t, de másképp!

Ezek az egyszerű példák természetesen nem mutatják meg a Flex igazi képességeit, de azt már el tudjuk dönteni, hogy szimpatikus-e számunkra ez a megközelítés, vagy más terület irányába mozdulnánk inkább. Ha nem nyerte el tetszésünket a Flex, RIA fejlesztésekhez további technológiákban is gondolkozhatunk.

### AJAX

Az AJAX (Asychronous JavaScript and XML) a következő elemeket foglalja magában:

- szabványalapú megjelenítőréteg, azaz az XHTML és a CSS használata
- dinamikus felhasználói felület és interakció a DOM segítségével
- aszinkron adatátvitel a böngésző és a szerver között az `XMLHttpRequest` objektummal
- adatcsere és manipuláció XML és XSLT alapokon
- a JavaScript intenzív használata, amely mindezeket összekapcsolja

Az AJAX célja, hogy rugalmasabb oldalat (melyeknek csak egyes részei töltődnek újra a szerverről), gyorsabb interakciót és gazdagabb élményt kínáljon a látogatónak.

- Előnye: erőteljes, sok böngésző és platform támogatja
- Hátránya: nagy projekteknél bonyolult a használata

### XUL

A XUL (ejtsd: zúl) egy sokoldalú leíró nyelv a gazdag felhasználói felületek készítéséhez. A Mozilla böngészők és az ahhoz kapcsolódó alkalmazások része (például Firefox).

- Előnye: magas teljesítmény, gyors, JavaScript-tel működik, XML-en alapul
- Hátránya: csak XULRunner alapú alkalmazások használhatják

### XAML/Avalon

A XAML a XUL-hoz hasonló leíró nyelv, a Microsoft következő generációs felhasználói felületének, az Avalonnak része, és az Internet Explorer következő verziói is támogatni fogják.

- Előnye: magas teljesítmény, erőteljes, jól konfigurálható
- Hátránya: Nem platformfüggetlen, a Windows Vista megjelenéséig nem elérhető

### Java Appletek

A Java Appletek Java-ban írt programok, melyeket be tudunk ágyazni a weboldalainkba. Futtatásukhoz szükségünk van a Sun Java Plugin-re.

- Előnyök: gyors, a legtöbb platformon elérhető (Java Plugin-nel)
- Hátrányok: szükségünk van a Sun Java Plugin-jére, melyet sok vállalat biztonsági okok miatt nem engedélyez. Az Appletek első betöltése jelentős időt vesz igénybe.

### SVG

Az SVG (Scalable Vector Graphics) egy szöveg alapú grafikus nyelv, mellyel képeket tudunk leírni vektoros alakzatokkal, szöveggel és beágyazott grafikákkal. XML adatforrásokkal és SMIL-el (Synchronized Multimedia Integration Language) is integrálható. A CSS-sel és a JavaScript-tel jól együttműködik.

- Előnye: gyors és rugalmas
- Hátránya: megfelelő plugin-re van szükség használatához, melyet sok cég nem engedélyez telepíteni, a nyelv még elég éretlen és fejlesztés alatt van.

### OpenLaszlo

Az OpenLaszlo egy nyílt forrású platform RIA-k készítéséhez, mely két részből épül fel:

- Az LZX egy programozási nyelv. Ez egy XML és JavaScript alapú leíró nyelv, hasonlít a XUL-hoz és a XAML-hez.
- Az OpenLaszlo szerver fordítja le a forráskódot bináris SWF mozivá – ugyanúgy, mint a Flex –, így bárhol futtathatóak az OpenLaszlo-val készült RIA-k, ahol van Flash Player.

Láthatjuk tehát, hogy sokmindenben hasonlít az OpenLaszlo a Flex-hez. Melyiket válasszuk? A következő összehasonlítás segíthet dönteni.

- Mind az OpenLaszlo, mind a Flex J2EE alapú alkalmazásszerver, mindkettő SWF mozit generál.
- Az OpenLaszlo nyílt forráskódú, a Flex használata ingyenes otthoni használatra.
- Az OpenLaszlo JavaScript-et, a Flex ActionScript-et alkalmaz.
- Az OpenLaszlo nem támogatja a Unicode-ot, míg a Flex igen.
- Amíg az OpenLaszlo megelégszik a Flash Player 5-ös kiadásával, addig a Flex-nek 8.5-ös verzió kell.
- A Flex jobb teljesítményt nyújt, mint az OpenLaszlo.
- A Flex JRun4-en is alkalmazható.
- A Flex együttműködik a Flash 8-cal. A Flash 8 a FlashType lévén a legjobb szöveg renderelést éri el, amit nem lehet összehasonlítani az OpenLaszlo renderelésével.

## Összefoglalás

A cikk csupán ízelítőként készült, teljes összképet az előadáson igyekszem adni. A fenti példa, és az előadáson bemutatott minták, valamint a technológiákhoz kapcsolódó linkek elérhetőek az előadás weboldalán:

`http://mmflex.be`

*Kiss-Tóth Marcell*

*A tiszaújvárosi Eötvös József Gimnázium informatika tagozatos tanulója. Szabadidejében ismerkedett meg a Macromedia Flex-szel és a Macromedia Cold Fusion-nel, az utóbbiról már cikket is publikált. A Tiszaújvárosban rendezett második PHP RoadShow főszervezője. Az előadásához kapcsolódó weboldala az mmflex.be címen tekinthető meg, személyes weboldala a kiss-toth.hu címen érhető el.*

# JavaScript ereje – Textarea++

*A web fejlődésével, a Web 2.0 kifejezés megjelenésével a tavalyi évben (illetve idén még hangsúlyosabban) újra előtérbe kerültek a web alkalmazások. A webalkalmazások bár általában az alapvető HTML elemekre építenek, nem árt, ha felhasználnak JavaScript segítségével egy kicsit többet nyújtó, úgynevezett webkomponenseket is. Előadásomban azokat az ötleteket szeretném felvillantani, melyek segítségével egy hagyományos <textarea /> elem kiegészíthető, a felhasználó számára használhatóbbá tehető.*

A webalkalmazások az AJAX technológia széles körű elterjedésével (és elfogadásával), továbbá a web értelmezését érintő szemléletváltozással nagyon sokat változtak az utóbbi időben. A korai és viszonylag régóta jelen levő webalkalmazások (mint a webes levelezők, szerkesztőségi felületek, vagy például a phpMyAdmin) egyre másra változnak át sokkal rugalmasabban, gyorsabban és kényelmesebben használható eszközökké. Ezt a szemléletváltozást hívhatjuk Web 2.0-nak (hívjuk így!), s nevezhetjük másképp is, mint például „a legújabb trendek, megoldások"-nak. Valójában nem a címke a lényeg, hanem hogy mit értünk mögötte.

Az asztali alkalmazások lehető legjobb utánzásához, s a legjobb felhasználói élmény megteremtéséhez a JavaScript használata jelentősen előtérbe került, s egyre több megoldás, ötlet is ismertté vált webfejlesztői körökben a JavaScript megfelelő felhasználását illetően. Mindezek az újdonságok és célok rendkívül jól használható megoldásokat teremtettek.

Egy egyszerű szövegmező számos módon egészíthető ki új funkciókkal – olyanokkal is, melyekről nem is gondolnánk, hogy megoldható. Előadásomban ezekből szeretnék felvonultatni néhányat, ha nem is lépésről-lépésre bemutatva a konkrét kivitelezésüket, de egy átfogó képet adva arról, hogy milyen lehetőségeink vannak ezen a téren. A következő ötletek egy részére kész megoldást kapunk a `http://livsey.org/experiments/textareatools/` címen, a továbbiak pedig saját fejlesztéseim és elgondolásaim alapján kerültek képbe.

A bemutatott megoldások mind nyilvános, mind adminisztrátori felületen hasznosak lehetnek, így ha nem feltételezhetjük egy konkrét böngésző használatát, célszerű az összes elterjedt böngészőre felkészülni (Internet Explorer, Mozilla Firefox, Opera, Apple Safari). Az előadásban példakódok szemléltetésével fogom bemutatni a megoldásokat, az egyszerűség kedvéért azonban csak a szabványos, vagy ha ilyen nincs (erre felhívva a figyelmet), akkor a Firefoxban működő megoldást preferálva. Nem árt tudni viszont, hogy a példák mindegyike megvalósítható úgy, hogy legalább a két legelterjedtebb böngészőben működjön, ráadásul diszkrét JavaScript segítségével úgy, hogy a többiben se adjon hibaüzenetet.

## Szavak száma

Bemelegítésül vegyünk egy egyszerű feladatot: jelezzük ki, hogy hány szót tartalmaz egy szövegmező!

Mit kell ehhez tennünk? Mindenekelőtt rájönni, hogy hogyan lehet megszámlálni a szavakat egy karakterláncban. A megoldás pofon egyszerű (kis egyszerűsítéssel): daraboljuk fel a `split()` metódust felhasználva a szöveget a „nembetű" karakterek mentén, s a kapott elemek száma lesz a szavak száma. Ez után ki kell még tenni egy „kijelzőt", ami lehet egy beviteli mező, vagy egy span elem is, és ha változik a szövegmező tartalma, akkor frissíteni.

A gépelést a szövegmező keyup eseményeit figyelve kaphatjuk el. Lássuk be azonban azt is, hogy a dolog így nem teljes, ha valaki egérrel, a vágólapról másol be új értéket, akkor nem fog sohasem lefutni a kódunk. Ezért nem árt a change eseményt is figyelni, s ha kell, hasonlóan korrigálni az értéket.

## Maximális hossz

Sokszor előfordul, hogy egy adatbázisban korlátozott egy mező hossza – esetünkben mindegy, hogy szándékunk szerint, vagy tőlünk teljesen független okokból. Ilyenkor ha nem szeretnénk, hogy a felhasználó véletlenül is kicsússzon a rendelkezésre álló helyből, s emiatt adatvesztés léphessen fel, esetleg hibaüzenet jelenjen meg, akkor figyelhetjük a mező hosszát, s korrigálhatjuk JavaScript segítségével. Akkor is használhatjuk ezt a megoldást, ha nem szeretünk sokat olvasni például egy kétsoros kapcsolatfelvételi űrlapba beírt látogatói esszét fogadva.

A dolog kivitelezése most sem túl nehéz, hisz a szövegmező minden keyup és change eseményénél meg kell nézni, hogy hány karaktert tartalmaz a szövegmező, s ha túl sokat, akkor egy alert ablak társaságában le kell vágni a kívánt hossznál a tartalmat. A megoldás úgy szép, ha a még éppen beférő karakterek számát kijelezzük, hasonlóképpen, mint az előbb a szavak számának kijelzésekor.

## Betűméret

A betűméret állítása kényelmi szolgáltatás lehet. Egyes felhasználóink talán szívesebben használnak nagyobb betűket íráskor, nem kell hozzá gyengénlátónak vagy idősebbnek lenni, egyszerűen sokkal kényelmesebbek lehetnek a nagyobb betűk. Ha a betűk különböző méreteit szeretnénk lehetőségként felkínálni a felhasználónk számára, a feladat egyből következik: gombok segítségével befolyásolni, hogy a szövegmezőben látható karakterek mekkora betűkkel jelenjenek meg. A feladat értelmezhető úgy is, hogy az oldalon levő összes szövegmező betűmérete befolyásolandó (és akár megjegyzendő), il-

letve úgy is, hogy csak az éppen aktuális szövegmező betűméretét kell megváltoztatni. Az előbbi a betűméret választó megoldásokhoz hasonlóan (melyek általában az oldalon levő szöveg betűinek nagyságát szokták befolyásolni) valósítható meg, vagyis úgy, hogy alternatív CSS fájlokat rendelünk az oldalhoz a kívánt betűnagyságtól függően. Ez egy jól működő megoldás lehet, sőt javasolt összekapcsolni az oldalon levő többi betű méretének állításával is. Egy másik megoldási lehetőség, hogy az éppen aktuális szövegmező `style` tulajdonságát állítgatjuk, bár használhatóság szempontjából kérdéses, hogy van-e értelme külön-külön betűnagyságot állítani szövegmezőnként. Az állapot mentése mindenképpen követendőnek látszik, hiszen ha valaki átállítja a méretet, valószínűleg azért teszi, mert általában is szeretne nagyobb betűket, s nem csak abban a pillanatban. Általában az emberek azon tulajdonsága, hogy mekkora betűket látnak jól, idővel nem javulni, hanem romlani szokott.

## Magasság állítás

A szövegmező mérete a legtöbb weboldalon rögzített méretű. De mi a helyzet akkor, ha az oldal alkotója, a tisztelt fejlesztő nem gondolta végig jól a felmerülő igényeket, s bár csak négy sort adott a cikk bevezető szövegének, az átlag felhasználó igénye hat sor lenne? Természetesen a négy soros szövegmezőbe belefér a hat sor (csak megjelenik a görgetősáv), de lehetne a sorok száma a felhasználó által állítható, testre szabható lehetőség is, amit elmentünk, s legközelebb már úgy tesszük ki az oldalt. Így a hat sor egyben átláthatóvá válik a használó számára, ezzel időt takarítva meg, nem beszélve az így születendő jobb minőségű cikkekről.

Egy szövegmező méretét kétféleképpen állíthatjuk. Egyrészt a `rows` tulajdonságát növelve, másrészt pedig a stílusán keresztül a magasságot átállítva. Mivel az előbbi hatástalan, ha már be van állítva CSS segítségével a magasság, ezért az utóbbi használata lehet az igazán javasolt. A magasságot növelhetjük rögzített értékkel, de a betűméretet is kiolvashatjuk, s növelhetjük aszerint. Illetve rendelkezésünkre állnak a relatív betűméretek is. Az előzőleg felvetett betűméret állítás esetén is változtathatjuk a magasságot, hogy ugyanannyi sor férjen ki a betűméret átállítása után is.

## HTML szerkesztő

HTML szerkesztő alatt jelen esetben nem a számos kész megoldással kecsegtető WYSIWYG megközelítésű, hanem kód szintű szerkesztés megvalósítására gondolok. A feladat röviden az lenne, hogy egy CTRL-B billentyűkombináció hatására az éppen kijelölt karaktersor körül jelenjen meg a `<b>` és `</b>` pár, ha pedig már ott van, akkor tűnjön el. Ezt ki lehet persze bővíteni más billentyűkombinációkra és funkciókra, de ha ezzel megvagyunk, a többi már nem lesz nehéz számunkra. Erre eddig még csak félmegoldásokat láttam, de egyik projektem kapcsán egy elég okos megvalósítás kerekedett ki a dologból, melyet végül a Weblaboron is elkezdtünk alkalmazni, először csak szerkesztők számára, majd a közelmúltban bevezetve immár nyilvánosan is.

Vegyük sorra a problémákat, lehetőségeket!

Először is kérdés, hogy hol áll a kurzorunk, van-e kijelölés, s ha igen, akkor mi a tartalma. A modern böngészők – bár nem túl ismert a dolog – egy kiváló lehetőséget biztosítanak ezeknek a kérdéseknek a megválaszolására a `selectionStart` és `selectionEnd` tulajdonságok segítségével. Ha ezt a két számot lekérdezzük, egy egyszerű `substring` metódushívással máris rendelkezésünkre áll, hogy mi van éppen kijelölve. Ha a kettő megegyezik, értelemszerűen nincs kijelölve semmi.

A következő feladat a tartalom elemzése lehet, erre egyké mintaillesztő kifejezés kifejezetten alkalmas. Ha pedig eldöntöttük, hogy mi a teendő, pár karaktersorozat művelet, és rendelkezésünkre is áll az új tartalom, melyet vissza tudunk írni.

Tudva az új tartalom határértékeit, utolsó lépésként még módosítani tudjuk a kijelölést, hogy az újonnan beszúrt karaktersorozat legyen kiválasztva.

## Az előadásban részletesebben

Ebbe a rövid összefoglalóba nem fért bele minden, amit igyekszem majd az előadás negyven percében átadni, s magába az előadásba sem fog beleférni az, amit mind-mind el lehetne mondani a téma kapcsán. A célom a haladóbb web programozók számára olyan útmutató átadása lesz, mely segítségével el tudnak indulni akár egy profi „szövegszerkesztő" kialakítása felé, vagy ha ilyen ambícióik nincsenek is, akkor is egy kicsit könnyebbé tudják tenni felhasználóik életét. Bár a célpont nem a kezdő felhasználó, igyekszem úgy alakítani a mondandómat, hogy az számára is érthető legyen, s az utólag letölthető csomagot kis módosítással ő is fel tudja használni.

A bemutatandó konkrét megoldások kapcsán kisebb-nagyobb problémák, illetve további megoldandó feladatok is felmerülhetnek. A weben találhatunk elég sok leírást, tippet a témakörben, illetve az előadásban is igyekszem majd kitérni a buktatókra. Egy ilyen probléma például, hogy a szövegmező szerkesztési története elveszhet ha közvetlenül állítjuk az értékét, és emiatt nem működik a CTRL-Z visszaállítás funkció, amit a felhasználó kifejezetten hiányolni fog, ha komolyan a szövegmezőben szeretné szerkeszteni a szöveget.

A tapasztalat szerint a felhasználó (újságíró) életét kifejezetten megkönnyíthetik ezek a változások, s hálával fogadják az ilyen jellegű támogatást. Tegyük jobbá a felhasználói interfészeket, s nyújtsunk testreszabott megoldást ügyfeleinknek!

Találkozunk az előadáson!

*Bártházi András*

*A Wish Internet Consulting ügyvezetője, a Weblabor egyik főszerkesztője, az NJSZT WFSZ titkára. Cégének vezetőjeként ars poeticája, hogy mindig a legfrissebb technológiák segítségével, az ügyfelek igényeit messzemenőkig kielégítő megoldásokat nyújtsanak, lehetőségek szerint alternatívákat kínálva. Számos portál, webalkalmazás létrehozásában és megújításában vett részt. Cége hosztolja a Weblabor.hu, Drupal.hu, Firefox.hu és RubyOnRails.hu oldalakat, a kapcsolódó levelezőlistákat.*

# Webrádió – AJAX alkalmazási bemutató

*Sokat hallani manapság az AJAX-ról, melyen az aszinkron működésű JavaScript alapú böngésző-szerver kommunikációs technológiát értjük. Az alábbi írásban szeretném bemutatni azt, hogy én hogyan kerültem kapcsolatba az AJAX-szal egy egyszerű, de szerintem látványos felhasználáson keresztül.*

Egyik ismerősöm csapata készített egy webes rádiót. A feladatom egyszerű volt: a csatorna aktuálisan lejátszott számát kell a weblapra kiírni, így a látogató egyből látja, hogy milyen számot hallhat éppen. A weblap már rendelkezett egy egyszerű zeneszám cím kiírással, de az nem frissült a számok váltáskor, csak az oldal újratöltésekor. Gondoltam, ezt meg lehetne oldani elegánsabban is.

A technológia adott: linuxos szerver gépen egy Icecast szerver (démon) fut. Ez az úgynevezett streaming szerver, a rádiót hallgató ügyfelek ehhez kapcsolódnak. Az icecast több rádió több csatornáját képes kiszolgálni. Ezeket a csatornákat egy speciális program, az Ices2 vezérli. A rádió ogg kódolású számokat képes lejátszani, a csatorna paramétereit pedig XML fájlon keresztül lehet megadni.

Az automatikus szám frissítéshez arra van szükség, hogy a webszervertől a HTML oldal JavaScript segítségével kérje el az aktuálisan játszott szám metaadatait (a szám címe, előadója), amit utána be kell illesztenünk a lapba. A programba végül belekerült egy-két apró trükk is, például gyorsítótár a szerver terhelés csökkentése végett. A szerver oldali programot PHP nyelven írtam.

## Főbb lépések

- A szerveren meghatározni, hogy melyik számot játssza épp a rádió
- Szerver oldali válasz XML összeállítása az adatokkal
- Gyorsítótár készítése
- Frissítés megvalósítása a HTML oldalon

### Melyik számot játssza éppen a rádió?

Az hamar kiderült, hogy az *ices2* példánytól közvetlenül nem lehet lekérdezni az aktuális szám adatait. Igaz, szolgáltat webes felületen keresztül információt arról, hogy melyik előadó melyik száma szól éppen, de nekem több adatra volt szükségem ahhoz, hogy tudjam, mikor vált a rádió az egyikről a másikra. Ez az időpont közvetlenül nem határozható meg, de egy apró trükkel kideríthető. Ha meghatározható az időpont, amikor kezdődött a szám és megvan a hossza is, akkor abból már gyerekjáték megmondani, hogy mikor kell a weblapon a címet frissíteni. Az ices példány logba írásának szintjét kellett egy kicsit feljebb venni, mert így a naplóba már beteszi azt az információt, hogy mikor cserélt számot és ez melyik fájl is éppen.

Az ices log részlete:

```
[2005-09-18  21:59:49] INFO playlist-
➥ builtin/playlist_read Currently playing
➥ "relaxmp3/13 - The Eternal's Song of the Ocean
➥ Waves (part 1).ogg"
```

Ahhoz, hogy megtudjam, hogy melyik az utolsó bejegyzés, vagyis melyik számot játssza éppen a rádió, egy egyszerű dolgot tettem. Nem akartam az egész állományt beolvasni és a végére ugrani, hiszen a log mérete igen nagy is lehet.. A szabványos unix `tail` parancsot használtam, mely a fájl végéről listáz néhány, megadott számú sort. Ez után vettem ezt a pár sort, majd visszafelé elindulva az első olyan sort kerestem, amelyikben a *„Currently playing”* szerepel. A `tail` programot az `exec()` PHP paranccsal hívtam meg. A függvény használ néhány, általam definiált állandó is, melyeket a kényelem miatt vezettem be:

```
define('TAILPROG', '/usr/bin/tail');
define('LINESFROMLOG', 4);
```

A `TAILPROG` a `tail` unix parancs elérési útját adja meg. A `LINESFROMLOG` pedig azt, hogy a `tail` mennyi sort listázzon a naplófájlból. Ezt tapasztalati úton vettem akkorára, hogy egy nekem szükséges sor mindig benne legyen a listában. (Az *ices* loglevel beállításától függ.).

A kész függvény a következőképpen fest:

```
function getCurrentStreamName() {
  $cmd = TAILPROG . ' ' .
         LINESFROMLOG . ' ' . LOG;
  $lines = array();
  exec($cmd, $lines);
  $lines = array_reverse($lines);
  $matches = array();
  foreach ($lines as $line) {
    if (preg_match('!\[(.*?)\].*Currently playing
➥ "(.*?)"$!', $line, $matches)) {
      $GLOBALS['startTime'] =
➥ strtotime($matches[1]);
      $GLOBALS['currentPlay'] = $matches[2];
      break;
    }
  }
}
```

Az `exec()` a `$lines` változóba a sorok tömbjeként beteszi a log sorait. Ezt megfordítom, mivel nekem az utolsó játszott szám kell, majd elindulok, és megnézem, melyik sorban van meg a megfelelő információ. Az egyszerűség végett a kinyert eredményeket a `$GLOBALS` tömbbe rakom be. Most már megvan, hogy melyik számot mikor kezdte játszani a szerver. Azt, hogy meddig tart a szám, tehát azt az időpontot, amikor számot vált a rádió, kiszámolhatjuk úgy, hogy a szám hosszát egyszerűen hozzáadjuk a kezdő időponthoz. A szám hossza meghatározható úgy, hogy megnézzük az ogg állomány belső információit. Ehhez az *ogginfo* parancs használható. Egy ogg állományt paraméterként megadva az alábbi kimenetet szolgáltatja:

```
New logical stream (#1, serial: 00003434): type
➥ vorbis
Vorbis headers parsed for stream 1, information
➥ follows...
Version: 0
Vendor: Xiph.Org libVorbis I 20020717 (1.0)
Channels: 2
Rate: 44100
```

```
Nominal bitrate: 64.001000 kb/s
Upper bitrate not set
Lower bitrate not set
User comments section follows...
        title=The Eternal Song of the ??????????
➥ Ocean Waves Part 1
        artist=Gandalf ????????????
        comment=(rádio url)
Vorbis stream 1:
        Total data length: 2236829 bytes
        Playback length: 5m:44.502s
        Average bitrate: 51.943349 kbps
Logical stream 1 ended
```

Minket most csak a *Playback length:* sor érdekel. Valójában érdekelne a szám előadója és a szám címe is, de ezt az információt, mint a fenti példából is látszik, nem sikerült maradéktalanul, ékezethelyesen kinyerni. A hiba abból adódott, hogy az *ogginfo* program futtatása során nem állítottam be a helyes nyelvi környezeti változókat.

Az újabb állandók közül, az `OGGINFOPROG` az *ogginfo* elérhetőségét adja meg, az `OGGDIRROOT` pedig a csatorna számainak a lelőhelyét a fájlrendszeren.

A szám hosszának meghatározása:

```
$cmd = OGGINFOPROG . ' -v  ' .
       escapeshellarg(
         OGGDIRROOT . $GLOBALS['currentPlay']
       );
$lines = array();
exec($cmd, $lines);

$foundBitRate = false;
$foundLength = false;
$searchPattern = "!bitrate:(.*?)[^\d]*$!i";
foreach ($lines as $line) {
  if (!$foundBitRate &&
➥ (preg_match($searchPattern, $line, $matches))) {
    $foundBitRate = true;
    $GLOBALS['bitRate'] =
➥ intval(trim($matches[1]));
  }
  if (!$foundLength && (preg_match('!Playback
➥ length: *(\d*)m:(\d*).*$!i', $line, $matches))) {
    $foundLength = true;
    $GLOBALS['length'] = intval(trim($matches[1]))
➥ * 60 + intval(trim($matches[2])) + 1;
  }
}
```

Itt még kiemeljük a bitrate adatokat is, hátha jó lesz még valamire. A szám előadóját és a címét végül az ices program webes felületéről szerezzük meg. A `STATURL` tartalmazza ennek webcímét. Sokkal egyszerűbb lenne a dolgunk persze, ha az *ogginfo* használatával mindent megkapnánk egyből...

```
$lines = file_get_contents( STATURL);
// Current Song:</td><td class="streamdata">Color
➥ - Álmatlanul</td>
if (preg_match('!Current Song:\s*</td><td
➥ class="streamdata">(.*?)-(.*?)</td>!i', $lines,
➥ $matches) ) {
  $GLOBALS['artist'] =
➥ htmlspecialchars(trim($matches[1]), ENT_QUOTES);
  $GLOBALS['title'] =
➥ htmlspecialchars(trim($matches[2]), ENT_QUOTES);
}
```

## Szerver oldali válasz XML összeállítása

Az aszinkron kliens-szerver kommunikáció XML fájlokat használ az adatok továbbítására. Nincs is más hátra, mint hogy a válasz XML-t előállítsuk a megfelelő HTTP fejlécekkel együtt. A fejlécek kiküldése nagyon fontos, mert különben előfordulhat az, hogy a böngésző nem frissíti a szám információkat, mivel az információs XML-t saját belső gyorsítótárából szedi elő, a friss adatokat tartalmazó helyett. Az itt használt fejlécek biztosítják azt, hogy ha a böngésző „ránéz" a program kimenetére, akkor mindig azt lássa, hogy az információ friss.

A böngészőknek elküldött fejlécek:

```
// Date in the past
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
Header("Cache-control: private");
// always modified
header("Last-Modified: " . gmdate("D, d M Y
➥ H:i:s", time()-10) . " GMT");

// HTTP/1.1
header("Cache-Control: no-store, no-cache,
➥ must-revalidate");
header("Cache-Control: post-check=0, pre-check=0",
➥ false);
header("Cache-Control: max-age=1, s-maxage=1,
➥ no-cache, must-revalidate");

// HTTP/1.0
header("Pragma: no-cache");
```

A válasz XML összeállítása, ahol `NL` az újsort tartalmazó állandó:

```
// content type
header('Content-type: text/xml');
echo '<?xml version="1.0" encoding="UTF-8"?>' . NL;
echo '<response>' . NL;
echo '<artist>' . $GLOBALS['artist'] . '</artist>'
➥ . NL;
echo '<title>' . $GLOBALS['title'] . '</title>'
➥ . NL;
echo '<bitrate>' . $GLOBALS['bitRate']
➥ . '</bitrate>' . NL;
echo '<length>' . $GLOBALS['length'] . '</length>'
➥ . NL;
echo '<nexton>' . ($GLOBALS['startTime'] +
➥ $GLOBALS['length']) . '</nexton>' . NL;
echo '<expire>' . $expire. '</expire>' . NL;
echo '</response>';
```

A válasz XML-ben a már korábban beállított globális változókat írjuk ki. A `text/xml` fejléccel biztosítjuk, hogy a választ XML-ként azonosítsa majd a böngészőben futó programunk.

#### De mi is az az expire?

Röviden a számcsere időpontja. Ez mondja meg, hogy az aktuális dátumhoz képest hány másodperc múlva fog befejeződni az éppen játszott szám.

```
$expire = $GLOBALS['startTime'] +
➥ $GLOBALS['length'] - $now;
```

A feladat szerver oldali részét ezennel meg is oldottuk. A program már képes szabványos XML-t válaszolni az AJAX lekérdezésekre.

## Gyorsítótár készítés

Vegyük észre, hogy a fenti megoldás működőképes, de nem elég hatékony. Jelen állapotában minden, a weboldalra érkező lekérdezésre meghatározza az aktuálisan futó szám jellemzőit, ami teljesen felesleges dolog, mert az nem változik, míg a következő számot nem kezdi el játszani a rádió. Ezért egy nagyon egyszerű gyorsítótárat

készítettem: egy PHP include fájlt a megfelelő változó értékekkel, mely így fest:

```
<?php
$GLOBALS['artist'] = 'Hungária';
$GLOBALS['title'] = 'Koncert a Marson';
$GLOBALS['bitRate'] = '64';
$GLOBALS['length'] = '344';
$GLOBALS['startTime'] = '1127073589';
$GLOBALS['length'] = '344';
?>
```

A szerver oldali program elején egyszerűen beillesztem ezt az állományt és rendelkezésre is állnak az aktuális szám jellemzői a megfelelő értékekkel a globális változókban. Ez után ellenőrizni kell a gyorsítótárban lévő adatok érvényességét. A vizsgálat során megnézzük, hogy `startTime + length` időpontja kisebb-e, mint `time()` értéke. Magyarán, ha a szám elkezdésének időpontjához hozzáadjuk a szám hosszát, akkor, ha a mostani időpontnál kisebb értéket kapunk, az azt jelenti, hogy a gyorsítótárban lévő adatok elavultak, mivel a rádióban már másik sláger hallható. Ilyenkor le kell kérdeznünk az új szám adatait, majd ezt be kell írnunk a gyorsítótárba. Egyébként csak egyszerűen a válasz XML-t kell kiírni a már rendelkezésre álló adatok alapján. Az idő értékeket mindenhol másodperc alapon tároljuk, így nem kell foglalkozni semmiféle konverzióval, a mértékünk alapja a Unix időbélyeg.

## Frissítés megvalósítása a HTML oldalon

A HTML oldalon `XMLHttpRequest` segítségével meghívjuk a szerver oldalon található `getStreamData.php` programunkat, majd a kapott adatokat feldolgozzuk és kiírjuk a weblapra. Az oldalban a megfelelő helyre betett `currentStream` azonosítójú `div` elembe írjuk be az információkat.

```
function refreshTune() {
 var element =
➥ document.getElementById('currentStream');
 element.innerHTML = '<p>Loading ...</p>';
 xmlhttp.open("GET", "getStreamData.php", true);
 xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4) {
   var xml = xmlhttp.responseXML;
   var node = xml.getElementsByTagName("artist")[0];
   var textNode = node.childNodes[0];
   var artist = textNode.nodeValue;
   node = xml.getElementsByTagName("title")[0];
   textNode = node.childNodes[0];
   var title = textNode.nodeValue;
   node = xml.getElementsByTagName("expire")[0];
   textNode = node.childNodes[0];
   var expire = textNode.nodeValue;
   element.innerHTML = artist + ' - ' +
                       title + ' refresh: ' +
                       expire;
   setTimeout(refreshTune, expire * 1000);
  }
 }
 xmlhttp.send(null);
}
```

Természetesen, az adatok megjelenését az oldal kinézetéhez kell illeszteni.

Az `xmlhttp` objektum az AJAX kommunikáció alapköve, felelős a JavaScript és a szerver közötti kommunikációért. Ennek az objektumnak a létrehozása böngészőfüggő módszerrel történik, melyben különböző AJAX függvénytárak lehetnek segítségünkre. Létrejöttét követően használata szerencsére egységes.

A függvény végén a `setTimeOut()` függvénnyel beállítom ugyanennek a függvénynek a meghívását. De mikorra is? Pont annyi másodperc múlva kell ezt a függvényt meghívni, amikor az aktuális szám lejár. Így nyer értelmet a visszaadott `expire` érték.

Nincs több feladat hátra, mint a tesztelés: az oldal letöltésével egy AJAX hívás elkéri az aktuális szám adatait. Ezt követően a szám címe mindig automatikusan frissül, amikor szükséges.

## Néhány megjegyzés

- Olyan környezetben ahol biztonsági vagy más okokból nem működik a `file_get_contents()` függvény távoli webcímekre, ott a `CURL` parancsokat kell használni az *ices2* web információk beolvasására,
- `safe_mode`-ban futó PHP esetén a `tail` és `ogginfo` parancsokhoz, illetve a loghoz, és az ogg fájlokhoz külön hozzáférést kell biztosítani a megfelelő beállításokkal (`safe_mode_exec_dir`, stb). Jobb ha felvesszük a kapcsolatot a rendszergazdával.

## További információk:

- Magyarockradio: `http://magyarockradio.hu/`
- Icecast streaming server: `http://www.icecast.org/icecast`
- *tail* dokumentáció: `http://linuxcommand.org/man_pages/tail1.html`
- *ogginfo* dokumentáció: `http://linuxcommand.org/man_pages/ogginfo1.html`

*Szabó Dénes*

*1997 óta foglalkozik webes fejlesztésekkel, elsősorban PHP nyelvű programozással. Ezen felül a Flash kivételével mindenfélébe beleártja magát, ami szükséges egy weblap illetve webhely elkészítéséhez: design, HTML, CSS. Pécsett él, ahol InterNode Bt nevű saját cégének dolgozik. Elkötelezett híve a webes szabványok betartásának, a mindennapi életben Linuxot használ. Az Első PHP Konferencián Smarty témakörben adott elő.*

# Django: Python on Rails

*Manapság a webes keretrendszerek virágkorukat élik. A Ruby on Rails indította hullám végigsöpört a web tengerén, nem kis vihart kavarva ezzel. Pro és kontra sok kritikát kapott, de annyi elmondható, hogy számtalan esetben kitűnő hasznát lehet venni egy ilyen komponens-gyűjteménynek. A RoR példáján felbuzdulva egy kansas-i lapnál dolgozó programozócsoport úgy gondolta, a Rails szellemében épít hasonló keretrendszert Python nyelven. Így született a Django.*

A Python végtelenül rugalmas, erősen tipizált, objektumorientált nyelv. Sajátos nyelvtana kényszeríti a programozót a strukturált kód megírására. Hordozhatóságát biztosítja, hogy interpretált nyelv, de a hatékonyabb teljesítmény végett az első futást követően bájtkódban tárolja az objektumokat. Szükség szerint C nyelven írt modulok tapaszthatók hozzá, ezzel is fokozva a teljesítményét. A Python értelmezője a legtöbb korszerű operációs rendszer (Mac OS X, Linux, FreeBSD, Windows) alá elérhető, de léteznek speciális implementációi is: a Jython a Java VM-ben fut, míg az IronPython a .NET (és Mono) platformokon teszi elérhetővé az óriáskígyót. Elterjedtségének köszönhetően kitűnően használható a LAMP/WAMP modellekben.

Korábban az egyetlen jól támogatott Python keretrendszer a Zope volt, amely a Plone tartalomkezelő rendszer alapjául is szolgál. A Ruby on Rails berobbanása azonban felpezsdítette az állóvizet, és egyre-másra jelentek meg különböző nyelveken hasonló koncepciójú keretrendszerek. A Python világában két újonc, a TurboGears és a Django vetette meg a lábát. A TurboGears fejlesztői főképpen már meglévő, kiforrott komponenseket hangoltak össze, míg a Django fejlesztői nulláról írták meg a kódot.

A Django mögött egy kisebb webfejlesztő csapat áll, eredetileg egy hírszerkesztő rendszer alapjául fejlesztették ki. Fő filozófiája a DRY, Don't Repeat Yourself, azaz olyan tervezés, amellyel elkerülhető bármilyen típusú redundancia. A Django legfőbb varázsa abban rejlik, hogy egyszerű megoldást nyújt a webalkalmazások fejlesztése során gyakran felmerülő problémákra, mintákra. A keretrendszer felépítése nem a tipikus MVC modellt követi, ugyanis a vezérlőnek a nézetek, a nézeteknek pedig a sablonok felelnek meg (a Django FAQ-jában viccesen a fejlesztők ezt MTV-nek nevezik). Természetesen ez nem azt jelenti, hogy ne lenne egymástól jól elkülönítve a megjelenítés az alkalmazás logikától, sőt!

## A Django telepítése

A Django telepítéséhez legalább 2.3-as verziójú Python szükséges. OS X és Linux felhasználóknak feltehetően ez már telepítve van a gépükre. A Django egy beépített webszerverrel is érkezik, amely ragyogó eszköz a fejlesztéshez, azonban ez a kiszolgáló éles használatra nem javasolt, produktív környezetben az Apache mod_python moduljával érdemes összekötni. Adatbázis használatához szükséges a kapcsolódó csatolók beszerzése. Jelenleg a MySQL, PostgreSQL és SQLite motorok támogatottak.

Töltsük le az archív állományt a Django oldaláról, majd tömörítsük ki. Lépjünk be a létrejött könyvtárba, és adjuk ki a `sudo python setup.py install` parancsot. A Django e sorok írásakor a 0.91-es verziónál tart. Nyugodtan használható éles környezetben is, de az 1.0-s kiadásig a fejlesztők fenntartják az API változtatásának jogát. Aki szeretné a legfrissebb újdonságokat kipróbálni, SVN-ből is beszerezheti a keretrendszert.

Mielőtt elkezdhetnénk az alkalmazásunk fejlesztését, szükséges a munkakörnyezet beállítása. Django terminológiában ezt projektnek nevezzük. Egy projekt tartalmazhat több alkalmazást, és egy alkalmazás nem szükségszerűen egy projekthez tartozhat csak. A Django számára nem szükséges, hogy a programkódjaink a webszerver dokumentum gyökere alatt helyezkedjenek el, így biztonsági megfontolásból helyezzük például a saját mappánk alá.

```
django-admin.py startproject webconf
```

A fenti utasítás hatására a következő könyvtár szerkezetű webconf projekt jön létre.

```
webconf/
   __init__.py
   manage.py
   settings.py
   urls.py
```

A munkakörnyezetünkre jellemző beállításokat – így például adatbázis paraméterek, sablonok – a `settings.py` modul tárolja, az `urls.py`-ban az egyes alkalmazásokhoz tartozó URL-ek szerepelnek. A `manage.py` a 0.91-es verzióban bevezetett burkoló, amelyen keresztül a továbbiakban a parancssorban dolgozhatunk.

Módosítsuk szükség szerint a beállításokat, majd hozzuk őket érvényre a `python manage.py init` paranccsal. (A továbbiakban, ha külön nem jelöltetik, valamennyi utasítást a projektkönyvtárban adjuk ki.) Ennek hatására létrejöttek az adatbázisban a szükséges relációk. Ezeket a lépéseket legtöbb esetben csak egyszer kell megtennünk. A továbbiakban a kényelmes használathoz célszerű két környezeti változó bevezetése. A `PYTHONPATH`-t állítsuk be a projektünk könyvtárába, míg a `DJANGO_SETTINGS_MODULE` változó jelölje a konfigurációs modult. Windows alatt az `export` helyett használjuk a `set` utasítást.

```
export PYTHONPATH="/home/gabor/webconf"
export DJANGO_SETTINGS_MODULE=webconf.settings
```

## Alkalmazás fejlesztése

Elérkeztünk ahhoz, hogy elkezdhetünk egy alkalmazást fejleszteni. Célunk egy egyszerű blogmotor megépítése: szeretnénk, hogy az egyes bejegyzéseket felcímkézhessük, érkezhessenek rájuk hozzászólások, valamint mindezekről nyújtsunk tetszőleges hírcsatornát. A teljes forráskód ismertetésére nem kerül sor helyszűke miatt, azonban a konferencia weblapjáról letölthetők lesznek az előadás anyagai.

Először hozzuk létre az alkalmazás keretéül szolgáló modulokat: `python manage.py startapp blog`. A létrejött `views.py` tartalmazza értelemszerűen a nézeteket, míg a `models/blog.py`-ban tárolhatjuk majd az adatmodelleket. A modellek a Djangóban igen fontos szerepet játszanak. A Django automatikus adminisztrációs felülettel rendelkezik, mert a generált űrlapfelületeket a modellek tulajdonságai alapján nyeri ki a motor. Szintén az attribútumok segítségével lehetséges felhasználótól érkező adatok validálása, végső soron pedig magát az adattáblák felépítését és megszorításait is az adatmodellünktől kölcsönzi a keretrendszer, így annak gondos megtervezése a legfőbb feladat egy alkalmazás fejlesztése során. Egy blogbejegyzés példáján tekintsük át a modellek felépítését.

```
class Post(meta.Model):
 title = meta.CharField(maxlength=255)
 content = meta.TextField()
 is_published = meta.BooleanField("Is it
➥ published")
 published_at = meta.DateTimeField("It was
➥ published at", auto_now_add=True)
 slug = meta.SlugField("Post slug",
➥ prepopulate_from=("title",))
 tags = meta.ManyToManyField(Tag,
➥ verbose_name="Tags")

 def __repr__(self): return self.slug
 # Szeretnénk tetszetős /20060116/elso_bejegyzes/
➥ sémájú címeket
 def get_absolute_url(self): return
➥ "/%d%02d%02d/%s/" % (self.published_at.year,
➥ self.published_at.month, self.published_at.day,
➥ self.slug)

 class META:
  admin = meta.Admin(
   fields = (
    ('Basic', {'fields': ('title', 'content',
➥ 'tags')}),
    ('Advanced', {'fields': ('slug',
➥ 'is_published', 'published_at'), 'classes':
➥ 'collapse'}),
    ),
    list_display = ('title', 'published_at'),
    list_filter = ['tags', 'published_at'],
    search_fields = ['content', 'title'],
   )
  ordering = ('-published_at',)
```

A Django modellek a `meta.Model` osztály leszármazottjai, attribútumai lesznek az egyes adatbázis mezők. Az attribútumokat létrehozó `*Field()` függvények az adattípustól függően változó paraméterlistával dolgoznak. Minden `*Field()` metódus első opcionális paramétere egy ember számára olvasható név, amelynek az adminisztrációs felületen vesszük hasznát. Ha nem adunk, az attribútum nevét fogja olvasható formába konvertálni.

A modellre jellemző metaadatokat az ActiveRecord tervezési mintának megfelelően egy belső `META` osztályban adhatjuk meg. Itt definiálható az admin attribútum is, amely jelzi a keretrendszer felé, hogy ezt a modellt szeretnénk az adminisztrációs felületen keresztül karbantartani. Az `Admin()`-nak átadott mezők az admin felület felépítését részletezik: milyen mezők jelenjenek meg, milyen sorrendben, milyen mezők alapján szeretnénk szűrést végezni, és milyen tartalmakban kívánunk keresni. A modellünknek definiálhatjuk további tetszőleges függvényeit: a `get_absolute_url()` például az adott blogbejegyzéshez tartozó webcímet adja vissza. Ennek jó hasznát vehetjük majd például a sablonokban.

Miután létrehoztuk valamennyi adatmodellt, be kell jegyeznünk a projektbe, csak így lesznek elérhetők. A `settings.py` állományba fűzzük be a modell elérését:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'webconf.blog',
)
```

Az adminisztrációs felület a Django számára szintén alkalmazáscsomagként jelenik meg, így annak használatához fel kell vennünk azt is a listába. Következő lépés a modellekhez tartozó adatbázisok kialakítása. A `python manage.py sql blog` utasítás a konzolra dobja azokat az SQL parancsokat, amelyeket a modelleink alapján generált. Ezt a kódot kell beszúrnunk, végrehajtanunk az adatbázisunkban, de erre kínáltatik egyszerűbb megoldás: `python manage.py install blog`. Hasonlóképpen kell eljárnunk az admin modullal is: `python manage.py install blog`.

A Python egyik erőssége interaktív értelmezőjében rejlik, amelyet a Djangónál is kihasználhatunk. Adjuk ki a `python manage.py shell` utasítást, és rögvest tesztelhetjük frissen létrejött moduljainkat. Importáljuk be őket, majd játsszunk egy kicsit az automatikusan generált adatbázis API-val. A modelleink az osztálynévből származtatott többesszámú alakban érhetők el.

```
>>> from django.models.blog import posts
>>> ff
>>> posts.get_list()
[]
>>> p = posts.Post(title="Hurrá, van blogom",
➥ content="A Django nagyon klassz.",
➥ published_at=datetime.now(), slug=
➥ 'hurra-van-blogom')
>>> p
Hurrá, van blogom
>>> dir(p)
[..., 'add_comment', 'delete', 'get_absolute_url',
➥ 'get_comment', 'get_comment_count',
➥ 'get_comment_list', 'get_next_by_published_at',
➥ 'get_previous_by_published_at', 'get_tag_list',
➥ 'save', 'set_tags']
>>> p.get_absolute_url()
'/20060115/hurra-van-blogom/'
>>> p.save()
>>> posts.get_list()
[Hurra, van blogom]
```

Az adminisztrációs felület birtokba vételéhez létre kell hoznunk egy felhasználót (`python manage.py createsuperuser`), aki teljeskörű jogosultsággal fog rendelkezni, majd indítsuk is el a beépített webszervert (`python manage.py runserver`). Alapértelmezetten a 8000-es porton a 127.0.0.1-es címen fogja a kéréseket fogadni. Nézzük meg, mit sikerült idáig alkotnunk, lépjünk be az adminisztrációs felületre (`http://127.0.0.1:8000/admin/`). Ha mindent jól csináltunk, egy 404-es „Page not found" hibaüzenet fogad minket. Ne csüggedjünk, a hiba nem bennünk van, csupán egy apró kiegészítésről feledkeztünk el. A Django egy jól átgondolt, letisztult webcím kezeléssel rendelkezik. A beérkező kérések webcíme alapján kiválasztja az `urls.py` modulból a megfelelő mintát, és meghívja a hozzá tartozó nézetet. Egy-egy ilyen webcím deklaráció baloldalán egy mintaillesztő kifejezés áll, melyre ha illeszkedik a webcím, betölti a jobboldalon feltüntetett nézetet. Ahhoz, hogy beléphessünk az adminba, fel kell vennünk ezt is az URL modulba. A gyakorlatban csupán egy kettőskeresztet kell kivenni a megfelelő sor elől.

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
 (r'^admin/',
➥ include('django.contrib.admin.urls.admin')),
 (r'^$',
➥ 'django.views.generic.list_detail.object_list',)
➥ {'app_label': 'blog', 'module_name': 'posts'},
 (r'^tag/(?P<tag>.*)/$',
➥ 'webconf.blog.views.posts_by_tag'),
 (r'^(?P<year>\d{4})
➥ (?P<month>\d{2})(?P<day>\d{2})/(?P<slug>
➥ [-\w]+)/$', 'webconf.blog.views.single'),
 (r'^(?P<year>\d{4})
➥ (?P<month>\d{2})(?P<day>\d{2})/(?P<slug>[-\w]+)
➥ /comment/$', 'webconf.blog.views.single'),
)
```

A Django tucatnyi előre elkészített nézettel rendelkezik, így az egyszerűbb feladatokhoz – például a blogbejegyzések listázása a főoldalon – nem szükséges saját nézetet kreálnunk, csupán paraméterként kell átadnunk alkalmazásunk és a kérdéses modulunk nevét. Saját nézeteket a korábban említett `views.py`-ban deklarálhatunk.

```
from django.models.blog import posts, tags
from django.core.template import Context, loader
from django.utils.httpwrappers import HttpResponse
def posts_by_tag(request, tag):
 post_list = get_list_or_404(posts,
➥ tags__label__iexact=tag)
 t = loader.get_template('blog/posts_list')
 c = Context({ 'object_list': post_list,})
 return HttpResponse(t.render(c))
```

A függvény első paramétere a kérelem objektum lesz, a további opcionális értékek pedig a mintaillesztésünk gyűjtőiből érkeznek. Betöltjük a kérdéses `posts_list.html` sablont, majd átadjuk a `Context` objektumnak, amely a feltétel szerint kinyert bejegyzésekkel feltölti azt. Ha megpróbálnánk betölteni egy ilyen oldalt, hibát kapnánk, hiszen még nem készítettük el a sablont. A sablonkönyvtárat helyezhetjük tetszőleges helyre, igényünktől függően célszerű az alkalmazáshoz vagy projekthez kapcsolni. Hogy végül miképpen döntöttünk, a `settings.py` modul megfelelő szekciójában hozhatjuk a Django tudtára.

```
{% extends "blog/base" %}
{% block content %}
{% for post in object_list %}
<h2><a href="/{{ post.published_at|date:"Ymd"
➥ }}/{{ post.slug }}/">{{ post.title }}</a></h2>
<div class="post">
{{ post.content }}
<p>Kelt: {{ post.published_at }}</p>
</div>
{% endfor %}
{% endblock %}
```

A Django beépített sablonrendszerrel rendelkezik, melynek nyelvtana és viselkedése hasonlatos a Smarty és CheetahTemplate megvalósításokhoz. Minden sablon egyszerű szöveges állomány `.html` kiterjesztéssel. Az alapgondolat az volt, hogy a lehető legegyszerűbben lehessen ugyanabból a sablonból különféle formátumokba (HTML, XML stb.) exportálni. A Django sablonkezelése tisztán objektumorientált. Minden esetben létezik egy alap váz, amelyet a nézetekhez csatolt sablonok kiterjesztenek. A jelen példa a `base.html`-t örökíti, amelynek `content` nevű blokkjába fog beékelődni az itt generált tartalom. A sablonok szintaktikája végtelenül egyszerű: változó értékét a {{ és }} jelek között lehetséges kiértékelni, míg különféle nyelvi elemeket, deklarációkat a {% és %} határolók közé szúrhatunk.

## Függöny

A Django ugyan jelenleg még nem érte el az 1.0-s stabil változatot, de több neves oldal is már Djangóval szolgálja ki lapjait. A témával ismerkedő programozók számára legértékesebb az a bőséges tudásbázis, amelyet a `http://djangoproject.com` oldalon halmoztak fel a fejlesztők. Cikkemben noha törekedtem a teljességre, hely hiányában nem idézhettem valamennyi forráskódot, amely néhány esetben nehezíti az érthetőségét, de bízom olvasóm szemfülességében.

## Kapcsolódó linkek

- A Django projekt oldala:
  `http://www.djangoproject.com/`
- A Python honlapja:
  `http://www.python.org/`
- Django Lighttpd-vel és FastCGI-vel:
  `https://simon.bofh.ms/cgi-bin/trac-django-projects.cgi/wiki/`
- Django Ebook referencia:
  `http://e-scribe.com/news/124`
- További írások a témában:
  `http://www.technorati.com/tag/django`

*Török Gábor*

*Jelenleg műszaki informatikus hallgató. Programozással saját kedvtelésből kezdett el foglalkozni. Mintegy öt éve fejleszt PHP-ben, Pythonnal az utóbbi időkben kötött szorosabb ismeretséget. Fontosnak tartja a szabványok és ajánlások ésszerű követését. Rajong a nyíltforrású megoldásokért. Több éven keresztül részt vett a LinuxPortál című televízió- és rádióadás szerkesztésében. Több oktatási központban is megfordult már, jelenleg szabadúszóként dolgozik. Szabadidejében rendszeresen gazdagítja a Weblabor tartalmát hírekkel, blogmarkokkal.*

# Raise Your Own PET

*A „PET" web-fejlesztői keretrendszer ismertetése*

## A rendszer létjogosultsága

2004-ben, a II. Perl Találkozó alkalmából rövid előadást tartottam, melyben kétségeimet fogalmaztam meg az általam sokat használt és nagyra becsült programozási nyelv, a Perl – pontosabban a Perl 5 – jövőjével kapcsolatban.

Abban az időben a Perl közösség a következő, hatalmas mérföldkőnek számító Perl verzió, a Perl 6 lázában égett. Ez érthető, hiszen a Perl 5 egyes részei lassan elavulttá válnak, a számtalan ráncfelvarrás után eljött az ideje egy nagyobb lépés megtételének, mielőtt a nyelv végleg elvesztené támogatói bázisát. Felmerül azonban a kérdés: a Perl 5 hanyatló népszerűségének, de legalábbis a népszerűség stagnálásának az okát vajon tényleg a nyelv szintaktikájában és architektúrájában, illetve az ott tapasztalható, részben szubjektív hiányosságokban találjuk meg?

Véleményem szerint egy nyelv sikerességét formalizmusa és a hozzá köthető paradigmák csak kis mértékben határozzák meg. Ezzel szemben – a „marketinget" most eltekintve – jelentős hajtóerőt képviselhet professzionális, integrált fejlesztőrendszerek megléte, vagy az adott nyelven íródott „sikeralkalmazások" száma, melyek alapvetően magát a nyelvet is sikeressé (értsd: elterjedtebbé) tehetik. Gondoljunk példának okáért a MySQL adminisztrációját lehetővé tévő, PHP alapú phpMyAdmin-ra, vagy a szintén PHP-ben készült, számtalan helyen használt webes levelező alkalmazásra, a SquirrelMail-re! Függetlenül a PHP mint nyelv „jóságától", ezen nyílt forráskódú, ingyenes alkalmazások fontos mozgatórugóit képezik a PHP nyelv és futtatókörnyezet elterjedésének, ezen keresztül – közvetve – sikerességének.

Ezen gondolatok tükrében 2004-ben egy olyan alkalmazás elkészítésének az igénye fogalmazódott meg, amely hozzájárulhatna a Perl népszerűségének növeléséhez. A lehetőségeket figyelembe véve hamar kiderült azonban, hogy a problémák még mélyebben gyökereznek: elég csak azt figyelembe venni, hogy míg PHP futtatókörnyezetet szinte mindenhol gond nélkül találunk, ugyanez nem igaz a Perl-re, hiába létezik szinte minden operációs rendszeren maga a Perl VM, webes környezetben ez nem elegendő.

Sok évvel ezelőtt, az Internet születésekor a Perl szkriptek jelentették „a" webes programozást – ezzel szemben ma CGI szkriptként futtatni valamit már kifejezetten „elavultnak" számít. Bár léteznek alternatívák – lásd például a mod_perl-t –, ezek jellemzően nem állnak rendelkezésre mindenhol, illetve szintén jellemzően teljesítménybeli, biztonsági hiányosságokkal is bírnak.

Végezetül arra jutottam, hogy egy – egyébiránt munkámban is remekül hasznosítható – Perl alapú keretrendszert, tulajdonképpen *alkalmazásszervert* készítek, amely a PHP-hez hasonló egyszerűségű, könnyen megtanulható és bővíthető, ámde Perl alapú fejlesztői környezetet biztosít – ezzel nemcsak a saját igényeimet kiszolgálva, de jó esetben új Perl programozókat is toborozva, hiszen a Perl 5-ben történő webes alkalmazásfejlesztés megtanulásának egyik legnehezebb része éppenhogy az első lépések megtétele, melyet egy felhasználóbarát rendszerrel könnyíteni lehetne. Így született meg a PET.

## A tanulási görbe

A PET tervezésénél talán az egyik legfontosabb szempont egy meredek tanulási görbe kialakítása volt: olyan rendszert szerettem volna létrehozni, amelynek használatát – már konfigurált környezet esetén – egy „zöldfülű" is percek alatt elsajátíthatja, hiszen vegyük észre: egy PHP-s weblap elkészítéséhez ad absurdum elég egy jegyzettömb és egy FTP program. A PHP programozását tanulva első, jelszó fájlunkat az internetre kihelyező programunk megírása már rég megtörtént, miközben első Perl programja elkészítéséhez ilyenkor még az SSH klienst sem töltötte le az átlagos, kezdő, Windows-t használó programozójelölt.

Az egyszerűséget alátámasztandó lássunk egy-két példát! Először is – feltételezve, hogy keretrendszerünk és webszerverünk már installálva, konfigurálva van – írassuk ki a számokat 1-től 100-ig, sortörés elemekkel elválasztva!

A feladat megoldásához mindössze a megfelelő könyvtárba kell feltöltenünk a következő, a példa kedvéért `index.pet`-nek nevezett fájlt:

```
<html>
<body>
[% FOREACH a = [1..100] %]
E: [% GET a %]<br />
[% END %]
</body>
</html>
```

Mint látható, a PET használatához ebben a formában még a Perl nyelv ismeretére sincs szükség: a szintakszis a `Template.pm` sablonkezelőn alapul, pontosabban konkrétan a nevezett modult használja. Nincs „dollárjel, meg kukac", megúsztuk a Perl szintakszist.

Egy hasonló bonyolultságú példában növeljünk és írassunk ki egy session változót az adott oldal minden egyes letöltésekor, ha létezik a bejövő `count` paraméter :

```
<html>
<body>
[% IF QUERY.count %]
 [% SESSION.set("variable",
SESSION.get("variable")+1) %]
[% END %]
Session variable value:
➥ [% SESSION.get("variable") %]
</body>
</html>
```

Kézenfekvő, olvasható – de legalábbis vetekszik a

```
<?php
 echo "Szasz, itt a Chello, vaz!";
?>
```

egyszerűségével.

## Az MVC modell

A fenti példákhoz mindenképpen hozzáteendő, hogy ezt a fajta „lineáris" programozást leginkább a kezdő, a PET-tel még csak ismerkedő programozók használják. A PET ajánlott felhasználási illetve konfigurálási módjában a webcímeket Perl metódusokra képezzük le, ezzel kicsit komolyabb, MVC alapú architektúrát kialakítva. Így például a `http://domain/path/file.pet` webcím automatikusan hív(hat)ja a `MyClass::API::path::ACTION_file()` metódust. (Erről lásd még a későbbiekben is.) Így tehát kedvében járunk mind a „kezdő", mind a „profi" programozóknak!

Írassuk ki a fenti webcímen található oldalra a rendszeridőt!

A `file.pet` tartalmából:

```
...
A rendszeridő: [% subst.time %]
...
```

`MyClass/API/path.pm` tartalmából:

```
...
sub ACTION_file {
 # az egyszerűség kedvéért - K.I.S.S. -
 # semmilyen paramétert nem veszünk át,
 # minden a $self -en át érhető el:
 my $self = shift;
 # a template "subst" tömbjét adjuk vissza itt
 return {
  'time' => localtime(time)
 };
}
...
```

A metódus által előállított értékeket a PET sablonkezelője automatikusan behelyettesíti a `file.pet` sablonba – ezzel máris különválasztottuk az alkalmazáslogikát a megjelenéstől.

## A CPAN

Larry Wall-nak, a Perl atyjának tanítása szerint a jó programozó három legfontosabb tulajdonsága közül az egyik a lustaság. Ez tulajdonképpen a szoftver újrahasznosítás egyik megfogalmazása. Ezt az elvet követi a PET is, amely – kissé leegyszerűsítve – tulajdonképpen nem más, mint egy igen tömör, mindössze néhány ezer sorból álló „modulgyűjtemény", amely azonban számtalan tesztelt, megbízható, széles körben elterjedt és hordozható CPAN modult használ. Itt megjegyezném, hogy a PHP esetében oly gyakran előforduló probléma, mikor bizonyos kiegészítéshez újra kell fordítani az értelmezőt, a PET esetében gyakorlatilag ismeretlen.

Az eredmény egy „pehelysúlyú", saját forrását tekintve PurePerl rendszer, amit karbantartani, továbbfejleszteni így – már csak méreteiből adódóan is – egyszerű és kényelmes.

## Tesztelhetőség

A webes alkalmazások egyik gyakori problémája, hogy a tesztelés néha nehezen automatizálható, hiszen végsősoron weblapokról van szó, többnyire HTML kimenettel – a fejlesztés végeredménye néhány dinamikus oldal, melyek tesztelése manuálisan, tehát drágán és heurisztikusan, vagy bonyolult segédprogramok felhasználásával végezhető csak el.

Mivel esetünkben a PET magját egy *különálló* Perl modul, a `PET::Dispatcher` adja, ezért egy egyszerű segédszkript segítségével parancssori (CLI, vagyis command line) módban is futtathatóak az egyes oldalak, ezzel jelentősen egyszerűsítve az automatizált, algoritmikus és reprodukálható tesztelést.

A `domain/my/relative/url/file.pet` webcímen található oldal parancssori tesztje például egyszerűen így végezhető:

```
bash$ pet_run_cli /path/to/my/install/
➥ my/relaitve/url/file.pet
```

ahol a kimenet egyszerű, szabványos kimenetre kerülő szöveg, ami már könnyedén feldolgozható és kiértékelhető.

A PET eltakarja az alkalmazás felé, hogy milyen környezetben fut: nem kell speciális programozási technikákat, feltételeket programunkra erőltetnünk ahhoz, hogy a tesztelést el tudjuk végezni. Ez nem is annyira a komolyabb rendszerek fejlesztőinek könnyebbség – ahol egy test-suit megléte szinte kötelező –, hanem a tesztelhetőséget hozza közelibbé, elérhetőbbé az „átlagprogramozók" számára.

## Profiling

Szintén gyakori gond webes alkalmazások esetén a teljesítményproblémák okának felderítése, illetve a terhelés csökkentése, az optimalizálás – vagyis a profiling. A PET itt is segítséget nyújt: az opcionálisan kérhető, paraméterezhető `performance.log` naplófájl többek között lehetőséget ad mikroszekundum felbontású valós és virtuális processzoridők mérésére, melynek segítségével megkereshetőek az alkalmazás szűk keresztmetszetét okozó pontok.

Magas terhelés, lassan letöltődő oldalak, „elszivárgó teljesítmény" – mindenki találkozott már a problémával, aki készített nagy forgalmú webes alkalmazásokat. Több másodperc alatt készül el egy oldal, de mégis alig foglal processzoridőt? A `performane.log` segítségével villámgyorsan megtalálható az a bizonyos SQL lekérdezés, amely LOCK-ot hoz létre a legtöbbet használt adattáblánkon.

## Action Mapping

Külön plugin-ek gondoskodnak az egyes webcímek Perl metódusokhoz rendeléséről, vagyis arról, hogy melyik, illetve milyen kód kerüljön futtatásra egy adott webcím behívásakor. Kiválaszthatóak a felhasználni kívánt action mapper osztályok, illetve speciális igények esetén magunk is készíthetünk ilyeneket.

Gyakori megoldás, hogy az összes oldalt egyetlen Perl osztály szolgáljon ki: ilyenkor például a `domain/path/morepath/file.pet` webcím a konfigurált osztály (legyen mondjuk `MyApp.pm` a neve) `ACTION__path__morepath__file` metódusa kerül meghívásra.

Nagyobb projekteknél a webcímeket „alkönyvtárak" szerint bonthatjuk osztályokra: ilyenkor például a `domain/path/test.pet` cím a `MyApp/path.pm` osztály `ACTION__test` metódusát hívja.

Az action mapping gyakorlatilag tetszőleges lehet: egy néhány soros Perl osztály viszonylag bonyolult leképezéseket is megoldhat, például egy portálmotornál használhatja az egyik webcím paramétert, amit egy adatbázistáblával is összevet. A lehetőségek szinte korlátlanok, és relatíve „alacsony tudásszinttel" is felprogramozhatóak.

## Terheléselosztás

A használt FastCGI felület architektúrának mintegy „mellékhatásaként" egyszerűen megoldható a PET-re épített rendszerek terheléselosztása: a viszonylag egyszerűnek tűnő rendszer tehát nem korlátozza egy alkalmazás életútját, nincs szükség a kód újraírására, véget nem érő optimalizálására csak azért, mert elfogynak a futtatószerver erőforrásai. Külső (external) módban üzemeltetve a FastCGI modul lehetőséget ad egyszerre több alkalmazásszerverre történő kapcsolódásra, egyszerű round robin módon: így a HTTP kéréseket fogadó webszerver a dinamikus tartalmat generáló alkalmazásszerverekre gyakorlatilag véletlenszerűen szórhatja a kéréseket. Ilyen megoldást szolgáltat például az Apache webszerver, illetve a `mod_rewrite` modul `RewriteMap` direktívája. Így – bár ritkán ideális megoldás – az erőforrások kibővítésével, vagyis további szerverek beállításával könnyen javítható egy PET alkalmazás áteresztőképessége.

## UTIL

A PET alkalmazásszerver indításakor betölti az úgynevezett UTIL osztályokat. Ezeket a későbbiek során a `$UTIL` változón keresztül bárhol elérhetjük. Ez biztosítja azt, hogy csupán a szükséges modulok legyenek a memóriában, továbbá lehetővé teszi saját eszközkészletek felépítését – ráadásul adott esetben a PET rendszertől függetlenül, netán régebbi projektekből átvéve, ezzel támogatva újrahasznosítható modulok készítését.

A példa kedvéért a PET-hez jelenleg „hivatalosan" rendelkezésre álló MySQL kezelő csomag jellemzően a `$UTIL::mysql` objektumreferencián keresztül érhető el:

```
$UTIL:mysql->updateRowById (
 # table.primary_key
 'users.id' => $session->{userid},
 # DBI resource descriptor
 'main',
 'username' => $form->{username},
 'password' => md5_hex($form->{password}
);
```

## Konfiguráció: global.conf

A PET beállításait egy általános konfigurációs fájl, a `global.conf` tárolja, mely az Apache webszerver szintakszisához hasonló módon tárolja a beállításokat. A konfiguráció globális (site-wide), azonban lokálisan (per-page), azaz egyes oldalakra nézve is módosíthatók bizonyos beállítások. A konfigurációs adatok a `getConf` metóduson keresztül bármikor elérhetőek, ami nagyon jól hasznosítható például több példányban, más környezetben futó alkalmazásoknál – adott esetben egy webhely teszt és éles példányában.

## Többnyelvűsítés

Komolyabb nemzetközi, illetve több nyelvi verzióban megjelenő weblapok esetében gyakran van igény a többnyelvűsítés valamilyen támogatására. A PET ezt elsősorban a pre – illetve postparsing segítségével oldja meg. Példaimplementációként a `PET::Util::PreMultiLang` osztály a meglévő PET fájlokat dolgozza fel előzetesen (preparsing), különálló, egyszerűen szerkeszthető szövegfájlokból a sablonokba helyettesítve a nyelvfüggő tartalmat. Ezen megközelítés előnye az, hogy semmilyen sebességcsökkenést nem okoz.

## ...és így tovább

Mivel a PET egy igen nagy rendszer, ezért részletes ismertetése helyett csupán néhány érdekesebb tulajdonságát, jellemzőjét emeltem ki.

Remélem, az érdeklődés felkeltésére ez éppen elegendő lesz.

*Fagyal Csongor*

*1997. óta Perl programozó, a Concept Online Kft. ügyvezető igazgatója és vezető szoftvertervezője. Fő profilja: vékonykliens alapú megoldások, ezen belül is kiemelve a közösségformáló, nagy forgalmú, úgynevezett „önmenedzselő" tartalommal bíró Perl-ben készült weblapokat. Csongor a nyilt forráskódú rendszerek és a Linux/POSIX környezet üzleti alkalmazásokban történő felhasználásának szószólója, elkötelezett Perl „guru".*

# Ruby On Rails: Fenntartható Egyszerűség

## Mi is az Egyszerűség?

Alkalmazásaink a valós világra készülnek, melynek bonyodalma meghaladja a saját képességeinket. Gyakorlatilag a munkánk egy folytonos küzdelem a bonyolultság ellen, hiszen az eredmény eleget kell tegyen a szerteágazó követelményeknek, miközben számunkra is érthetőnek kell maradnia.

A Ruby on Rails (RoR) egyszerűbbé teszi a megoldásaink hosszútávon fenntartható kialakítását. Ez az egyszerűség az alapgondolatból indul ki: „konfiguráció helyett konvenciók". Más keretrendszerek konfigurációs állományok tömkelegével próbálnak rugalmasak lenni. Ezzel szemben a RoR céltudatosan törekszik az egyszerűségre, megfelelve a természetes nyelvi intuícióknak.

## Model-View-Controller

Az MVC nem újdonság, a koncepciója több mint húsz éve létezik. A Rails viszont forrás állományok, osztályok, metódusok elnevezésére vonatkozó szabályaival megkönnyíti ennek megértését és áttekintését. Például a `http://domain.com/articles/edit/1` webcím a következőket árulja el nekünk:

- az `app/controllers/articles_controllers.rb` állomány tartalmazza a controllert melynek neve `ArticlesController`
- az `ArticlesController#edit` metódus hívódik meg automatikusan
- az `app/views/articles/edit.rhtml` készíti el az `edit` funkcióhoz tartozó HTML választ mely beszúródik az `app/views/layouts/articles.rhtml` általános sablonba

Persze mindez módosítható, de alapértelmezett viselkedésként lényegesen csökkenti az entrópiát.

## Domain-specifikus modellezési nyelv

A Rails egyik szépsége a modellek közötti kapcsolatok egyszerű definiálásában nyilvánul meg, egy a *Lisp* makrók világára emlékeztető, domain specifikus nyelv implementálásával. Így explicit SQL parancsok nélkül (is) tudjuk kezelni adatainkat. Például vegyük a következő modell osztályokat:

```
#article.rb
class Article < ActiveRecord::Base
  belongs_to :author
end

#author.rb
class Author < ActiveRecord::Base
  has_many :articles, :dependent => true
  has_many :drafts, :class_name => "Article",
➥ :condition => 'is_draft=1'
end
```

A modell osztályhoz – nevéből kiindulva – automatikusan megfeleltethető egy adatbázisbeli tábla. Például `Article` típusú objektumok az `articles` táblában lesznek eltárolva. A modell kontextusában hívott függvények `has_many` és `belongs_to`, 1:N-hez relációt definiálnak a két osztály között. Ez alapján a RoR legenerál egy sor könnyen megjegyezhető, illetve értelmezhető metódust. Lássunk egynéhány legenerált metódust, valamint az ezek alatti SQL parancsokat.

```
author = Author.find(1)
# SELECT * FROM authors WHERE (id=1);

author.articles
# SELECT * FROM articles
#     WHERE (articles.author_id = 1);

author.articles
# alapértelmezetten nincs ismételt lekérdezés

author.articles.find_by_title("My life without me")
# SELECT * FROM articles
#      WHERE (articles.author_id = 1
#      and articles.title = "My life without me");

author.drafts.create("New Draft")
# INSERT INTO articles (title, is_draft)
#      VALUES ("New Draft", 1);

author.destroy
# BEGIN
#  SELECT * FROM articles WHERE
➥ (articles.author_id = 1)
#  DELETE FROM articles WHERE id = 4143
#  DELETE FROM articles WHERE id = 1323
#  DELETE FROM articles WHERE id = 9428
#  DELETE FROM authors WHERE id = 1
# COMMIT
```

Az utolsó törléskor az adott szerző összes cikke is automatikusan törlődik. Persze ezt meg tudtuk volna oldani külső kulcsos megszorításokkal is adatbázis szinten, viszont így a törlés mikéntjébe Ruby alkalmazásunk kevésbé szólhatott volna bele:

```
class Article < ActiveRecord::Base
  def after_destroy(article)
    File.rm(article.lead_image)
  end
end
```

A példában kifejeztük, hogy törlés után szeretnénk egy plusz tevékenységet is végrehajtani: például törölni az adott cikkeknek megfelelő lead képet. Megjegyzem, hogy a feltölthető csatolmányok problémakörét elegánsabban lehet megoldani a *FileColumn* nevű RoR kiegészítővel.

## Automatizált tesztelés

Az automatizált tesztelés szükséges a gyors és fenntartható fejlesztéshez. Nélküle hajlamosak vagyunk időnk jelentős részét hibakereséssel tölteni, rettegve minden módosítástól valamint ezek beláthatatlan következményeitől.

A RoR köszönhetően a Ruby rendkívül dinamikus tulajdonságainak egyszerűvé teszi a tesztek írását. Minden RoR projekt esetén a tesztek egyetlen parancs segítségével lefuttathatok:

```
$ rake
```

A tesztek fontosságát bizonyítja az a tény, hogy míg a legtöbb (jól megírt) programcsomag esetén ezt a szerepet a `make test` vagy az `ant testTask` és nem a `make`, illetve az `ant` maga tölti be, addig a RoR-nál a `make` automatikus tesztelést biztosít.

Két típusú tesztelésről beszélhetünk: unit valamint funkcionális tesztelés. Az első a modell objektumokra korlátozódik, míg a második a teljes MVC rendszert átjárva gyakorlatilag megfelel egy valódi felhasználónak. Például:

```
class AccountControllerTest < Test::Unit::TestCase
 def test_index_without_super_user
  @request.session[:user] = users(:average_joe')
  get :index
  assert_response :success
  assert_no_tag(
   :tag => 'a',
   :attributes => {
     :id =>'create-new',
     :href =>/signup/
   },
   :content => "Create a new user"
  )
  end
end
```

A fenti kódrészletben szimuláltunk egy HTTP GET-et az `/account/index` webcímre nézve, miközben a session módosításával beléptetünk egy *átlag felhasználót*. Majd ellenőrizzük, hogy a visszakapott HTTP állapotkód sikeres lefutást jelez-e, illetve azt is, hogy egy nem engedélyezett funkcionalitásra mutató link hiányzik-e a generált HTML-ben.

## „Hibák nélkül"

Avagy tartsuk az alkalmazásunkat mindig működő állapotban. Mikor észreveszünk egy hibát, azonnal tesztet írunk rá és kijavítjuk. A továbbiakban ez a hiba nehezen fordulhat elő, hisz ott a teszt, mely folytonos felügyeletet biztosít. Nincs „ismert hibák listája".

A RoR eme magatartásunkat nagyfokúan támogatja, hisz teszteket írni a Ruby dinamikus természetének köszönhetően egyszerű. Könnyen kicserélhetünk egyes metódusokat tesztelés céljából úgy, hogy a kívánt viselkedésük legyen. Például:

```
class PublicRelationsAlerter
  def  find_top_ten_technorati_hits(term)
    #hálózati hívások sorozata
  end
end
```

A fenti osztály adatelemzést végez a Technorati webhelyen adott kulcsszóra keresve. Tesztelés során a kivételes esetek, mint például a hálózati hibák szimulálhatók egyszerűen újra nyitva az osztályt:

```
class PublicRelationsAlerter
  def  find_top_ten_technorati_hits(term)
    #mesterkélt hálózati hibák
  end
end
```

A Ruby nyelv hasznos tulajdonsága hogy a fenti két `PublicRelationsAlerter` osztály deklaráció egymás mellett szemantikailag elfér: a második deklaráció újranyitja az osztályt, módosítva a metódusát. Így tesztelni tudjuk az egész osztály vagy más osztályok viselkedését olyan kivételes események esetén, mint például a hálózati kapcsolat megszakadása.

Még az alapos és folytonos automatizált tesztelés mellett is előfordul, hogy a rendszer éles használata során bukkannak fel bizonyos hibák. Ilyen esetekben a legjobb, ha azonnal értesülünk róluk, lehetőleg nem az ügyféltől. Erre kínál megoldást a Rails egy könnyen működésbe helyezhető kiegészítője, az *ExceptionNotifier*. Minimális konfiguráció után minden kivételes eseményről levelet kapunk, mely tartalmazza a webcímet, session, cookie és stacktrace adatokat, valamint minden egyéb információt, mely segítségével a hibát könnyen reprodukálhatjuk.

## Félóránként egy kiadás

Ez a web egyik nagy előnye. Gyorsan tudunk (hiba)javított verziókat kiadni anélkül, hogy a felhasználó bármit is letöltene, illetve fárasztó kérdéssorozat után telepítene. És persze mindez nekünk is kevesebbe kerül: nem kell egyszerre ugyanazon termék több verzióján dolgoznunk. Az olyan webalkamazások, mint a Basecamp, Flickr, Odeo stb. teljes mértékben élnek ezzel a lehetőséggel. A RoR egyik kiegészítője a *Switchtower* a kiadás folyamatát rendkívüli mértékben automatizálja:

```
$ rake deploy
```

Egyetlen parancs hatására SSH kapcsolatot épít ki, melyen keresztül az éles környezetben telepíti a legújabb kódverziót és tetszőleges beállítások elvégzése után újraindítja az alkalmazást, lehetőleg anélkül, hogy egyetlen felhasználó is átérezné a váltást. Miért is jó ez? Mert közel tart a valósághoz, nem engedve, hogy hosszabb időt eltöltsünk *l'art pour l'art* avagy *fejlesztés a fejlesztés kedvéért* üzemmódban.

Amilyen egyszerűen működésbe helyeztünk egy új verziót, hasonlóan visszaállíthatjuk az alkalmazásunkat az előző verzióra, ha hibát észleltünk.

```
$ rake rollback
```

Ezek után a hibát a fejlesztői gépen kereshetjük, miközben a webfarmunk a régebbi, bevált verziót futtatja.

## Összefoglaló

Érdemes a Ruby On Rails rendszerrel megismerkedni, hiszen a fent említett előnyökön túl számos más kézreálló szolgáltatást biztosít a webfejlesztőknek. A RoR eredetileg a Basecamp nevű valós alkalmazásból lett kiemelve keretrendszerré, jelenleg pedig egy módszertani letisztulást és kiegészülést mutató folyamatnak lehetünk tanúi, mely során a rendszer bővülése a gyakorlatiasság erős befolyása alatt van. Ezért a további RoR verzióktól is hasonló egyszerűséget várhatunk el.

*Deé Zsombor*

*Magát lelkes fejlesztőnek vallja, aki tekintélyes tartozást halmozott fel a Ruby valamint a RoR közösségek iránt. Nekik köszönheti a bátorságot, hogy külföldről hazatérve Székelyudvarhelyre, társával saját vállalkozásba kezdjen. Különös szeretettel gondol kedvenc projektjére, a SuperGlu-ra, mely felhasználók ezreinek biztosít identitás kifejezést. A közösség felé felgyűlt „tartozásából" próbál kicsit lefaragni ezzel az előadással.*

# Webes Alkalmazások és Kódkönyvtárak Terjesztése PEAR 1.4 Segítségével

A PEAR 1.4 megjelenésével lehetővé vált, hogy saját (nem kizárólag PHP-ben íródott) webes alkalmazásainkat és kódkönyvtárainkat könnyen csomagokká alakíthassuk. Ezeket a csomagokat egy saját PEAR szerver felállításával elérhetővé tehetjük nyilvánosan, vagy felhasználhatjuk cégen belül, hatékony frissítési lehetőségeket biztosítva ezzel a párhuzamosan futó projektjeink egyes moduljainak.

## A PEAR-ről általában

A „PEAR" rövidítés a `http://pear.php.net` webhelyen elérhető PHP nyelvű kódraktárat és az ezt kiszolgáló infrastruktúrát takarja. A cikk írásakor 36 kategóriában összesen 354 csomag áll rendelkezésre a PHP fejlesztők számára. Ezek a csomagok tömörített (`tar.gz`) formátumban kerülnek terjesztésre, kézi letöltésük és kibontásuk helyett azonban erősen ajánlott a webhelyen elérhető automatikus telepítő használata. Minden csomag tartalmazza a telepítéséhez szükséges információkat, amelyek alapján a telepítő az adott rendszer környezetére érzékenyen a megfelelő helyre képes telepíteni őket, a művelet végrehajtásához esetlegesen szükséges adatokat a felhasználótól a telepítés során bekérve.

A PEAR telepítő (és infrastruktúra) 1.4-es verziójának megjelenése azért számít mérföldkőnek a projekt életében, mert ezentúl már nem csak a PEAR központi webhelye szolgálhat forrásként a csomagok letöltéséhez. Lehetővé válik mind nyilvános, mind „privát" PEAR szerverek felállítása, amelyeken saját csomagjainkat terjeszthetjük, vagy egyszerűen kezelhetjük a köztük lévő függőségeket akár saját célra is. Az 1.4-es verzió emellett egyéb új lehetőségei miatt is várhatóan rövid időn belül népszerű és elterjedt lesz, hiszen a függőségek kezelésén kívül az egyes fájlokhoz és fájltípusokhoz saját szerepeket és műveleteket rendelhetünk, valamint lehetőségünk van a telepítést követő úgynevezett „post-install" szkriptek készítésére is.

## A PEAR parancssori telepítője

A PEAR háromféle telepítőt kínál, ezek közül a parancssori változat (CLI Installer) a legelterjedtebb. Egy alap PHP disztribúció telepítésekor általában ez is automatikusan a rendszerre kerül, és a `pear` parancs segítségével indítható. Leggyakrabban használt lehetőségei a *list*, *install*, *run-scripts*, *list-upgrades*, *upgrade* és *upgrade-all*. A telepítési feladatokon kívül ez az eszköz használható a legkönnyebben csomagok előállítására is.

## Csomagok készítése

Saját bináris csomagfájl generálásához készítenünk kell egy `package.xml` leírófájlt a csomagról. Ezt a fájlt viszonylag könnyen előállíthatjuk a PEAR által biztosított `PEAR_PackageFileManager` csomag segítségével. Korábban a leírófájl 1.0-ás változata volt népszerű, ma már a 2.0-s használata javasolt, amelyet a `PEAR_PackageFileManager2` osztály segítségével le is generálhatunk. Ehhez egy rövid PHP nyelvű szkriptet kell írnunk. Amennyiben nem fejlesztünk PHP-ben, a szükséges `package.xml` fájlt természetesen kézzel is előállíthatjuk. A pontos formátumról bővebb információt a PEAR Kézikönyvben találunk.

Az 1.0-ás verziójú csomagleíró-fájl a következő elemeket tartalmazhatja: csomagnév, rövid leírás, teljes leírás, licensz megnevezés, karbantartók neve, szerepe, email címe, és az adott verzió száma, állapota, dátuma, tetszőleges szöveges információi, a verzió fájljainak teljes listája, függőségei, és a változásnapló (changelog) a korábbi megjelenésekre nézve. Fontos, hogy érvényes (valid) XML fájlt hozzunk létre. A fájlban a betűk közül csak az angol ABC betűit használhatjuk. Amennyiben egyéb nemzetközi karakterekre van szükségünk, a megfelelő HTML-entitásokat kell alkalmaznunk. A `PEAR_PackageFileManager` csomag automatikusan elvégzi a nemzetközi karakterek HTML entitássá történő alakítását, amennyiben a PEAR legalább 1.4.0a2 verziója van telepítve a rendszerünkön.

Az egyes fájlokhoz a következő szerepeket rendelhetjük: *php, ext, doc, data, test, script, src* és *extsrc*. A csomag megjelenéséhez a következő függőség-típusokat adhatjuk meg: *pkg, ext, php, prog, os, sapi, zend*. A `rel` attribútum segítségével definiálhatjuk az igényelt függőségek viszonyát az adott rendszeren lévőhöz képest: *has, eq, lt, le, gt, ge, not*. Emellett megadhatjuk, hogy a függőségeket milyen konkrét értékekhez kívánjuk viszonyítani, valamint hogy a függőség kötelező vagy opcionális. Ezen információk pontos jelentéséről az előadáson tudhatnak meg többet az érdeklődők.

A `package.xml` fájl második verziója elődjét csatornák megvalósításának támogatásával, bináris PECL csomagok támogatásával, valamint sokkal specifikusabb függőségfeloldással egészíti ki (korábbi csomagfájljaink automatikusan erre a formátumra konvertálhatók a *pear convert* parancs segítségével).

## Szerepek, műveletek, telepítés-utáni szkriptek

Csomagjaink minden egyes fájljához definiálnunk kell egy szerepet (role), amely meghatározza, hogy az adott fájl hogyan és hová kerül telepítésre. A fent ismertetett beépített szerepeken kívül saját szerepeket is létrehozhatunk. Ilyen esetekben a leírófájlban azt is meg kell adnunk, hogy az adott szerep melyik csatorna melyik csomagjában van megvalósítva. A teljesség kedvéért a csomag függőségeinél célszerű lesz ezt a csomagot is függőségként megjelölnünk.

Példa egy tetszőlegesen definiált szerep használatára:

```
<usesrole>
    <role>foo</role>
    <package>Foo</package>
    <channel>pear.example.com</channel>
</usesrole>
<file name="myclass.php" role="php">
<file name="arbitrary.file" role="foo">
```

A fájlok telepítési módszerét műveletek (tasks) definiálásával befolyásolhatjuk. A PEAR-rel három egyszerű, és egy összetettebb művelet kerül terjesztésre. A `<tasks:replace>` művelet a telepítés során a telepítendő fájlokban karaktersorozatok kicserélésére képes. A `<tasks:windowseol>` és a `<tasks:unixeol>` a forrásfájlok sorait lezáró soremelések formátumát tudja telepítés közben befolyásolni.

Végül, a `<tasks:postinstallscript>` tetszőleges testreszabást tesz lehetővé, telepítés utáni szkriptek megadásával.

A telepítés utáni szkriptek saját magunk által elkészített PHP nyelvű programok, amelyeknek a telepítő a `package.xml` fájlban meghatározott paramétereket adja át a telepítést követően. Mielőtt ezek a paraméterek átadásra kerülnének, sorban megjelennek a telepítést végző felhasználónak jóváhagyásra. Ezáltal lehetőségünk van a csomagjaink telepítési folyamatához alapértelmezett értékeket rendelni, amelyeket egy adott rendszerre történő telepítéskor felül lehet bírálni.

Mivel a telepítés utáni szkriptekben tetszőleges kód helyezhető el, ez egy potenciális biztonsági résként szolgálhatna ártó szándékú fejlesztők számára. Emiatt a telepítés utáni szkriptek nem futnak le automatikusan, azokat kézzel kell meghívni a telepítést végző felhasználónak a `pear run-scripts <csomagnév>` parancs segítségével, miután meggyőződött róla, hogy az adott csomag megbízható forrásból származik.

## Saját csomagszerver felállítása

PEAR szervert igen könnyen telepíthetünk, mivel maga a szerver is egyetlen PEAR csomagként áll rendelkezésre, és a `pear.chiaraquarter.net` csatornáról szerezhetjük be. A telepítendő csomag neve `Chiara_PEAR_Server`. A szerver telepítéséhez a PHP legalább 5.0-s verziójára és MySQL adatbázisszerverre van szükségünk.

Saját szerverünket legegyszerűbben egy webes felületen keresztül adminisztrálhatjuk. Egy ilyen felület rendelkezésre áll csomag formájában, a `pear.crtx.org` csatornán. A telepítendő csomag neve `Crtx_PEAR_Channel_Frontend`.

A webes felületen a szerver adminisztrálása céljából tetszőleges számú felhasználót hozhatunk létre és tetszőleges számú csomag megjelentetésére van lehetőségünk. Ahhoz, hogy a csomagokat könnyen tallózhassák látogatóink a böngészőjükben, kategóriákba sorolhatjuk őket. Miután ezeket a beállításokat elvégeztük új csatornánkon, nekiláthatunk a csomagok első verzióinak megjelentetéséhez.

A megfelelő beállítások elvégzése után egy csomag megjelentetése mindössze a csomag bináris fájljának (`tar.gz`) feltöltéséből áll. A szerveren ez automatikusan kibontásra kerül, így a benne található leírófájlnak megfelelően a szükséges adatbázis- és fájlrendszerműveletek azonnal végre tudnak hajtódni további beavatkozás nélkül.

Amennyiben elkészítettünk egy saját csatornát és azt megnyitjuk a nagyközönség számára, célszerű regisztrálnunk egy népszerű, csatornákat összegyűjtő portálon. Ilyen portál például a `pearadise.net` címen található, ahol a cikk írásakor tizennégy csatorna tartalma jelenik meg tallózható formában.

Ha egy létező nyílt forrású alkalmazást szeretnénk PEAR csomag formájában megjelentetni, előbb érdemes körülnéznünk a meglévő csatornákon, hogy nem tette-e meg már ezt valaki előttünk. A `pearified.com` szerver például kizárólag erre a célra jött létre, több elterjedt nyílt forrású projekt is elérhető rajta PEAR csomagként (FCKeditor, phpMyAdmin, JPSPAN, SimpleTest, Smarty stb.).

## Helyi PEAR telepítések

Amennyiben több projekten dolgozunk, és mindegyikben használunk PEAR-t, célszerű a csomagokat projektenként külön-külön telepíteni. Ezzel megóvhatjuk magunkat az esetleges kellemetlenségektől, amik a csomagok tömeges frissítése során érhetnének.

Az egyedi telepítésekhez először szükségünk van egy globális telepítésre, amelyből rendelkezésünkre áll a PEAR parancssori telepítője. Ez után egy egyedi, projektre korlátozott telepítést például a következőképp végezhetünk el:

```
mkdir external
pear config-create /myproject/external
➥ project1.ini
pear -c project1.ini channel-update pear
pear -c project1.ini list
pear -c project1.ini install PEAR
pear -c project1.ini install --onlyreqdeps
➥ Mail_Mime
pear -c project1.ini install --onlyreqdeps Mail
pear -c project1.ini install --alldeps Net_SMTP
pear -c project1.ini list
pear -c project1.ini list-upgrades
pear -c project1.ini upgrade-all
```

Ha a PEAR-t projektjeinkhez lokálisan telepítettük, be kell állítanunk az egyes projektekhez tartozó `include_path` direktívákat. Ezeknek a megfelelő PEAR telepítésre kell mutatniuk.

Ezt követően bátran frissíthetjük csomagjainkat bármelyik projektünkben anélkül, hogy aggódnunk kellene a többi projekt működőképességének fennmaradását illetően.

*Mocsnik Norbert*

*Rengeteg különböző nyelv és módszer megismerése után hat éve a PHP mellett döntött. Az utóbbi négy évben a nemzetközi közösség nyílt forrású fejlesztései foglalkoztatják leginkább, 2003 óta vesz részt a PEAR, másfél éve a WACT projektben. Munkájában főként a nemzetközileg elismert legfrissebb módszereket és technológiákat igyekszik minél szélesebb körben bevezetni, tökéletesítve és megkönnyítve ezzel partnerei mindennapi munkáját.*

# W3C WAI

*avagy Weblapok Akadálymentesítése*

## Bevezetés

Az Interneten minden információ megtalálható, csak rá kell találni, szokták mondani, és az élet nagyon sok területére ez az állítás igaz is. Mindenki, aki megtanulja hatékonyan használni a Világhálót, rátalálhat ezek közül az információk közül a keresettre. Vagy mégse ilyen egyszerű a képlet?

Sokszor nem is gondolunk arra, hogy mennyire különbözők vagyunk, mennyire mások az igényeink, képességeink és lehetőségeink. Ami az egyik embernek jó illusztráció, szép ábra, az a másiknak átláthatatlan bitsorozat, vagy több tíz percig töltődő fölösleges adat. Amit az egyik ember táblázatnak lát, a másik soronként rendezett adathalmaznak. Ami az egyik kijelzőn nagynak tűnik, a másikon túl kicsi. Amit valaki jól elkülönülő színes vonalaknak lát, azt más esetleg azonos árnyalatú, megkülönböztethetetlen szürke vonalaknak. És még sorolhatnánk a számtalan példát, amely azt bizonyítja, hogy mind mások vagyunk, és ugyanazt az információt mennyire máshogy fogjuk fel.

Egy akadálymentesített honlap létrehozása esetén nem csak a fogyatékossággal élőket kell figyelembe venni, hanem az egyéb, valamilyen szempontból különleges igényű csoportokat is. Vegyük sorra a lényegesebbeket.

## Fogyatékossággal élők

A testi vagy lelki fogyatékossággal élők nemcsak nagy csoportját alkotják az akadálymentesített honlapok felhasználóinak, hanem sokkal jobban igénylik is azokat. Hiszen például egy mozgássérültnek sokkal nagyobb szüksége van arra, hogy otthonról intézhesse banki ügyeit, mint mondjuk egy ép embernek, aki régi, elavult böngészője miatt nem éri el a bank webhelyét.

### Vakok és gyengénlátók

Sokan azt gondolják, hogy az akadálymentesített honlapok egyetlen célcsoportja a vakokból áll. Ez az állítás semmiképp sem állja meg a helyét, de igaz, hogy a vakok és gyengénlátók igen nagy felhasználói csoport. Nagyon jó példa erre a gondolkodásra a Budapest Portál [BpPort], amelynek külön felülete van csak vakok és gyengénlátók számára. Ahhoz, hogy egy honlap tartalmát a vakok felolvasóprogramja vagy kijelzője is értelmezni tudja, a teljes tartalomnak elérhetőnek kell lenni e szöveges formában is.

A gyengénlátók igényei nem teljesen azonosak a vakokéval, hiszen nagy, jól szerkesztett ábrákat képesek lehetnek látni, sőt segít is nekik, mint mindenki másnak is, a szöveg megértésében. Azok, akik nem használnak felolvasóprogramot, igénylik, hogy a betűtípust illetve a kontrasztot állíthassák az oldalon.

### Színvakok és hallássérültek

Fontos alapszabály akadálymentesített honlap készítésekor, hogy szín és hang ne hordozzon fontos információt. A szín nem csak a (szín)vakokat akadályozhatja a megértésben, hanem a színtévesztőket, régi monokróm monitort, esetleg modern, de kevés színnel rendelkező mobiltelefonos böngészőt használókat is. Hang esetében a hangkártyával illetve hangszóróval fel nem szerelt, például irodai gépekre is gondolni kell.

### Mozgássérültek

Ebbe a felhasználói csoportba azok tartoznak, akik valamilyen alternatív kezelőfelülettel kezelik a számítógépüket. Ez néha azt jelenti, hogy nem rendelkeznek egérrel, csak billentyűzettel vagy fordítva. Nagyon nagy segítség egy mozgássérült számára, de mindenki másnak is hasznos, ha nem kell sokat gépelni. Az egyszer a regisztrációnál már megadott email-címet például a belépésnél nem kell újra kitölteni, vagy esetleg nem kell minden alkalommal begépelni a jelszót.

### Értelmileg visszamaradottak

Komoly kihívást jelent egy olyan átlátható és könnyedén használható felhasználói felület kialakítása, amelyet értelmi fogyatékosok is tudnak használni. Ez a feladat két részre szedhető, az egyik az átláthatóság és egyértelműség, amely mindenkinek előnyös, de a fogyatékosoknak elengedhetetlen. Erre mindig törekedni kell. A másik már kicsit nehezebb, a felhasználót lépésről lépésre vezetni kell, hogy mindig értse és tudja, hogy pontosan hol tart az adott folyamatban. Míg ez megkönnyíti egyes emberek dolgát, másokat felidegesít, ezért szokták ezeket a szolgáltatásokat kikapcsolhatóvá tenni.

## Technológiailag megkülönböztetettek

Ebben a fejezetben nem a felhasználó személye szerinti csoportokat vesszük sorra, hanem az általuk használt eszközök képességeiben rejlő különbözőségekből adódó csoportokat tekintjük.

### Eltérő képernyőméret

Felhasználónk képernyőmérete lehet nagy és kicsi is. A világ a felé halad, hogy mindkét szélsőségből egyre több lesz, hiszen sorban jelennek meg a böngészésre képes mobiltelefonok és PDA-k (kézigépek), valamint a 19 colos és annál nagyobb monitorok is. A CNN honlapján például, egy 19 colos monitoron a teljes képernyő kevesebb mint kétharmadát teszi ki a fix szélességű oldal. Ez még a kisebbik baj, de ráadásul a tényleges hír beszorult ennek a résznek a negyedébe. A hír maga a teljes képernyő 17%-át foglalja el vízszintesen (220 képpont).

Kis képernyők esetén ugyanakkor gyakori a vízszintes görgetés. Ezt mindenképpen el kell kerülni, nagyon átláthatatlanná teszi az oldalt és nagyon nehézkes lesz a kezelése is. Amennyiben PDA-ra is szeretnénk optimalizálni az oldalunkat, és nem szeretnénk az újratördelést rábízni a gép böngészőjére, ezt stíluslapokkal megoldhatjuk.

### Elavult böngésző

Nem szabad azt feltételezni, hogy mindenki rendelkezik a legújabb vagy egy bizonyos böngészővel. Sokan használnak olyan régi számítógépet és operációs rendszert, amely nem is teszi lehetővé a böngésző frissítését. Ezek a böngészők például nem képesek JavaScript vagy Flash animációk futtatására, esetleg még a stíluslapokat se támogatják. Az is előfordul, hogy bizonyos cégeknél bizton-

sági okokból vannak ezek letiltva. Gyakran sokkal szebb lesz tőle az oldal, esetleg plusz funkcionalitást ad (például kliens oldalon ellenőrzi az űrlapba írt adatokat), de az oldal működőképessége nem függhet ezek meglététől.

#### Gyenge hardver

Ez is lényeges szempont, és talán Magyarországon sokkal lényegesebb, mint azt gondolnánk. Egy régi gép kevés memóriával rendelkezik, nem tud bizonyos méretűnél nagyobb oldalakat megjeleníteni, és természetesen régi böngészővel rendelkezik, ami további korlátozás. A leggyakoribb hardveres „probléma" ugyanakkor a lassú Internet-kapcsolat. Amikor a fejlesztő készíti a honlapot, azt vagy a saját gépén teszi, vagy egy, a szerverrel helyi hálón összekötött gépen. Ilyenkor könnyen abba a hibába esik, hogy nem veszi észre, hogy milyen nagy adatállományokat, képeket, táblázatokat szúrt be az oldalba. Egy olyan oldalon, ahol a menü képekből áll, és szépen megváltozik a színe, ha a felhasználó fölé viszi az egeret, könnyen több száz kilobájtosra is dagadhat az oldal mérete. Ez egy régi modemnél párperces töltögetést jelent, amit csak igazán fontos és tartalmas honlap esetében vár ki a felhasználó, legtöbbször inkább odébbáll.

## Speciális célcsoportok

#### Gyerekek

A gyerekek sokkal többre képesek, mint azt gondolnánk, három éves kortól már könnyedén tudják használni az egeret, viszont – mivel nem ismerik a betűket – a billentyűzet használata náluk a speciális gombokra korlátozódik. A figyelmük könnyen irányítható, viszont könnyen el is kalandozik, azaz az oldalon lévő legszínesebb, legérdekesebb, vagy mozgó elemre fognak figyelni, rákattintani. További fontos igény, hogy a gyerekeket vezetni kell, lépésről lépésre, ha végzett egy feladattal, oldallal jön a következő, szépen sorban.

#### Idősek és informatikai szempontból alacsony képzettségűek

A mai nyugdíjas generáció életében a technika hatalmas fejlődésen ment keresztül, ennek a korosztálynak nagyon lényeges a korábban már többször is hangoztatott átláthatóság és a lépésenkénti vezetés. Így körülbelül érti, hogy mi történik, de ha mégis elveszne, biztos lehet benne, hogy a program végigvezeti, és a végén kilyukad valahol. A manapság olyan közkedvelt felugró (popup) ablakok például teljesen elrontják ezt az átláthatóságot. Mindenkinek jól jön egy okosan felépített súgó, de azok számára, akik nem annyira értenek az informatikához, ennek megléte és használhatósága létfontosságú lehet.

## W3C – WAI

A Word Wide Web Consortium [W3C], a webes szabványosítással (ajánlásokkal) foglalkozó szervezet, amely a mindenki számára jól ismert HTML, XML, CSS stb. ajánlásokon kívül még sok egyéb, kevésbé ismerttel is rendelkezik. Ezek közé tartozik a Web Accessibility Initiative [WAI] által létrehozott, 1999-ben ajánlássá nyilvánított, Web Content Accessibility Guidelines 1.0 [WCAG1] is.

Ebben három csoportra osztott feltételek vannak, amelyek szükségesek egy akadálymentesített honlap létrehozásához, ezeket úgy is felfoghatjuk, mint egy vonalvezetőt, amely segíti a fejlesztőt egy akadálymentes honlap létrehozásában.

Az első csoport (priority 1) olyan feltételeket (checkpoint) tartalmaz, amelyek betartása kötelező egy akadálymentesített honlap létrehozása esetén. Ha egy honlap betartja az összes priority 1 feltételt, akkor kiteheti a W3C WAI-A logót. A priority 2 feltételek betartása csak ajánlott, míg a priority 3 csak lehetőség. A priority 1 és 2 együttes betartása a WAI-AA, míg mindhárom együttes betartása a WAI-AAA konformitást jelenti.

## Konklúzió

Ezt a cikket gondolatébresztőnek szántam, hogy akik honlapokat készítenek, gondolják át a lehetőségeiket, és azt, hogy milyen széles közönséghez szeretnének szólni. A téma még rengeteg érdekes kérdést vet fel, melyekre itt most hely hiányában nem tudtunk kitérni, de kis odafigyeléssel és alternatív például szöveges böngészőkkel, gyengébb gépekkel és más operációs rendszerekkel való tesztelés után a legtöbb hibára fény derül.

## Irodalomjegyzék


- [BpPort] Budapest Portál, `http://www.budapest.hu`, `http://latasserult.budapest.hu`
- [CNN] Cable News Network, `http://www.cnn.com`
- [W3C] World Wide Web Consortium, `http://www.w3c.org`
- [W3HU] W3C Magyar Iroda, `http://www.w3c.hu`
- [WAI] Web Accessibility Initiative, `http://www.w3.org/WAI`
- [WCAG1] Web Content Accessibility Guidelines 1.0, `http://www.w3.org/TR/WCAG10`


*Pataki Máté*

*Három éve az MTA SZTAKI Elosztott Rendszerek Osztályán dolgozik. A webes szabványokkal foglalkozó nemzetközi szervezet, a W3C egyetlen közép európai irodája is itt működik (W3C Magyar Iroda), melynek egy éve koordinátora. A DSD sok hazai és nemzetközi projekt tagja, webes rendszerekkel foglalkozik. Úttörő szerepet játszott a magyar Web kialakításában, az első intézményi honlap is a nevéhez fűződik. Több ingyenes szolgáltatást is biztosít a magyar netezőknek, olyanokat mint például a KOPI Plágiumkereső Portál, vagy a közkedvelt SZTAKI Szótár.*

# Weboldalak a vakok „szemével”

## Bevezetés

Az internet napjainkra nagyon fontos eszközzé vált. Rengeteg törekvés van arra, hogy egyre több mindent intézhessünk az internet segítségével: vásárolhatunk, banki és egyéb hivatalos ügyeinket intézhetjük, hovatovább nem szabad megfeledkezni arról sem, hogy az egyetemista diákok már sok esetben csak az internet segítségével tudnak például kurzusokra, vizsgaidőpontokra jelentkezni. Az interneten nagyon sok információt meg lehet találni, így aztán bármi új ismeretanyagra van szükségünk, szintén az internethez fordulhatunk, gondoljunk csak az internetes lexikonokra, amilyen például a Wikipédia. Ezen fontos és alapvető tények miatt sem zárhatunk ki senkit a hálózatról azzal, hogy az általuk is reprezentált csoport vagy csoportok számára nehezen vagy egyáltalán nem használható módon helyezzük el anyagainkat, tartalmainkat és/vagy szolgáltatásainkat az interneten. Ezért fontos a weboldalak akadálymentessége vagy akadálymentesítése.

Előadásunkban bemutatjuk, hogy miképpen használják az internetet a vak emberek. Megmutatjuk, hogy milyen egyszerű majdnem teljesen akadálymentes oldalakat készíteni. Arra is kitérünk, hogy mennyivel „könnyebb dolga van” egy képernyőolvasónak akkor, ha akadálymentes oldalakat kell feldolgoznia, mintha olyan oldalról kell információt adni, melynek készítése során nem vették figyelembe az akadálymentesség irányelveit.

Kijavítunk egy akadályoktól hemzsegő weboldalt úgy, hogy a konkrét példában általános kódolástechnikai segítséget adunk a vakok számára is fogyasztható tartalom publikálásához. Habár nem a teljes akadálymentesítési folyamatot fedi le a vakok számára történő hibajavítás, mégis hasznos tanácsokat kaphat a teljes akadálymentesítés iránt érdeklődő webgrafikus vagy programozó is.

*PAC Mate termékek*

*PAC Mate QX420 (ezt az eszközt bemutatjuk)*

Az előadás során a vakok által használt eszközöket is ismertetni fogjuk a közönséggel. Bemutatkozik a JAWS és működés közben egy braille-kijelző is segít elképzelni azt, amivel eddig sokan csak olvasmányaikban találkozhattak. Összehasonlítjuk a különböző operációs rendszereken elérhető, vakok által használható böngészésre alkalmas szoftvereket és a jövőre vonatkozó előrejelzést is adunk.

## A képernyőolvasókról

Akik látnak, a szemükkel nézik a monitort, így kapnak információt a számítógéptől. A látássérülteknek szükségük van arra, hogy a képernyőn megjelenő információ számukra is érzékelhető módon jelenjék meg. A teljesen vak emberek esetén általában kétféle információátadás jöhet szóba a hang és a braille írás.

A vakok által használt eszközök információközlése igen bonyolult. A tévhit szerint a képernyőolvasó felolvassa a képernyő egész tartalmát, s ezzel elvégezte munkáját. Ez használhatatlan volna, gondoljunk csak a weboldalak navigációs eszközeire, vagy az operációs rendszer kezelésére. Egy ilyen program több összetett funkciót is tartalmaz, például feladata eldönteni azt, hogy milyen információt kell közölnie a felhasználóval. Mindez bizonyára első hallásra bonyolultnak tűnhet, elég annyit megjegyezni, hogy a képernyőolvasók jól átgondolt, de viszonylag bonyolult módszerrel igyekszenek mindig a legfontosabb információt közölni a felhasználójukkal. Feladatuk azonban nem korlátozódik a weboldalak szövegének felolvasására, hanem funkciókat és navigációs lehetőségeket is kínálnak azon túl, hogy a teljes rendszert elérhetővé teszik a vakok számára.

A képernyőolvasó rendszerek napjainkban nem számítanak újdonságnak, hiszen a békebeli DOS parancssorát is fel lehetett vértezni a beszélés adományával. Természetesen ezek a rendszerek lényegesen primitívebbek voltak napjaink korszerű technológiájánál. A ma népszerű operációs rendszerek szinte mindegyikére létezik képernyőolvasó, ám ezek használhatósága és minősége még nem minden esetben teszi lehetővé a vakok maradéktalan információhoz jutását. Hazánkban a Windows-hoz kifejlesztett programok örvendenek nagyobb népszerűségnek

a vakok között, ennek főbb oka, hogy magyar nyelven csak erre a platformra készítettek megfelelő minőségű képernyőolvasót. A magyarországi vakok túlnyomó többsége a JAWS for Windows-t használja, ezért is ezt a programot fogjuk példaként tüzetesebben megvizsgálni.

Előző verzióiban a JAWS kizárólag a Microsoft Internet Explorerrel működött együtt egy MSAA nevű technológia segítségével, ezt egy speciális FS-DOM-technikával kibővítve válik lehetővé a felhasználói felület átadása más böngészőprogramokból is a beszédszintetizátornak, vagy braille eszköznek.

Néhány technikai (vagy ideológiai?) fanatikus látássérült DOS vagy Unix alatt (csak karakteres felületen) használja a webet. Sajnos ez a fajta böngészés és felhasználói interfész sokkal kevesebb lehetőséget nyújt, mintha egy grafikus rendszeren működő böngészővel olvasnák a weboldalak tartalmát, valamint a karakteres módban működő képernyőolvasók kevesebb tudása miatt a karakteres böngészők kezelése is nehézkesebb. Előadásunkban ezekkel az eszközökkel nem foglalkozunk bővebben.

*Braille írógép*

*Húsz karakteres braille-kijelző*

Amikor megnyitunk egy weboldalt, azt a JAWS megkapja a rendszertől feldolgozásra, majd átrendezi úgy, hogy az használható legyen a látássérültek számára. A képernyőn természetesen a weboldal eredeti képét láthatjuk, de ami a látássérült számára „megjelenik", az teljesen más, ő hallhatja többek közt a menüket, és a címsorszinteket is. A „megjelenik" azt jelenti, hogy a JAWS egy nem látható, de hallható, vagy braille írással elolvasható felületre elhelyezi a weboldal átrendezett formáját, mint egy szöveget, így a látássérült felhasználó egy az ő számára átalakított tartalmat ér el. Eközben látó társa azt látja, amit neki megjelenít a böngésző. Ez lehetővé teheti a közös projektek lebonyolítását látó és nemlátó kollégák részvételével, és megmagyarázza a külön elkészített vakbarát oldalakkal kapcsolatos ellenszenvünket. Ezzel kapcsolatosan megjegyezzük, hogy a látássérültek előszeretettel választják a külön vakbarát verzióval felszerelt portálok „normális" változatát, mert azok tartalma az esetek többségében jóval felülmúlja az elszeparált fogyatékos oldal tartalmát (gettósítás).

## Hogy látja a vak?

Mivel a képernyőolvasók mindkét típusú kimenete lineáris természetű (hang és karaktersor a braille esetében), a vakok nem érzékelhetik a weboldalak egyes elemeinek a képernyőn elfoglalt területét, vagyis nem tudják, hogy mi van bal, és mi van jobb oldalon, nem látják, hogy a menü felül, vagy alul van, hiszen a felolvasás során nem a képernyőn való megjelenés szerint találkoznak a weboldallal, hanem a forráskód alapján kapják meg a feldolgozott kimenetet. Egy táblázatot, vagy egy űrlapot például többféle módszerrel is bejárhat a képernyőolvasó program felhasználója, és számtalan az átlagos felhasználó által nem is képzelt segítséget kaphat az alkalmazástól. Ilyen és hasonló érdekes és tanulságos tapasztalatokat is átadunk az akadálymentes web ügyét szívükön viselő hallgatóinknak.

## A jövő

Milyen reményteli szikrák mutatkoznak az alternatív operációs rendszerek iránt érdeklődő látássérültek számára? Milyen (számottevő) új projektek indultak a közelmúltban, milyen más rendszerek érkezését várhatjuk? Ezek a kérdések is terítékre kerülnek előadásunk során.

A világon elsők között mutatjuk be az új JAWS (7.0 változat) és a Mozilla Firefox böngésző egy közös projektjének eredményét, amelynek köszönhetően a magyar vakok is használhatnak a jövőben alternatív böngészőprogramot.

*Torma Zsolt*

*Harminc éves látássérült informatikus. Rendszergazdaként dolgozik, valamint a JAWS for Windows képernyőolvasó program honosítási projektjének vezetője.*

*Károly György Tamás*

*Programozó és webdesigner. 1993-ban kezdett weblapokat tervezni hobbiból, ma már ez a fő tevékenysége. Jelenleg a weboldalak akadálymentesítését és akadálymentes honlapok tervezését tartja a legfontosabb ügynek. Honlapja a kgyt.hu címen található. Szabadidejében feleségével, Bertával tervezgeti a jövőt.*

# Webre, de gyorsan

A Java a C++ nyelvhez képest egy fiatal programozási nyelvnek tekinthető. Első megjelenésétől számítva igen nagy fejlődésen ment keresztül, a legmodernebb technikákat alkalmazva. Nem csoda, hogy a projektek egyre nagyobb hányada alapoz erre a platformra, nem is említve a Java weben betöltött szerepét. Érdemes tehát górcső alá venni a Java technológiát és megvizsgálni az általa nyújtott lehetőségeket.

A korai, egyszerű JVM implementációk általában interpreteres utat követték, modern változataik azonban a jóval nagyobb futtatási sebesség elérését lehetővé tevő JIT technikát alkalmazzák. A JIT fordítók az interpretált nyelveken készült alkalmazások futtatását gyorsítják fel olyan módon, hogy végrehajtás előtt elvégzik a program forráskódról tárgykódra történő átalakításának folyamatát. Így a program a tényleges végrehajtás során már natívan, gépi kódban működik, és végrehajtásához nincs szükség az időigényes értelmezési folyamatra, miközben továbbra is módosítható marad, közvetlenül forráskódban.

A másik gyorsítási lehetőség a Java Hotspot technikája. Ez a technológia a felhasználó „szokásait" veszi figyelembe. Az úgynevezett 80/20-as szabályra alapoz: a kód végrehajtása során 80%-ban, csak a kód 20%-a fut. Tehát egy adaptív algoritmussal vizsgáljuk a kódrészletek használatát és a leggyakrabban végrehajtott kódot és környezetét optimalizáljuk.

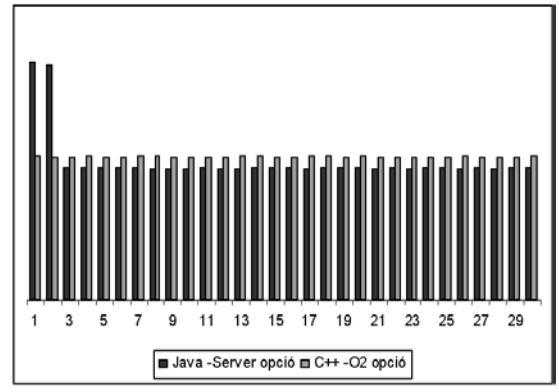
A modern objektum-orientált nyelvekben egyre nagyobb hangsúlyt kap a programozók munkájának támogatása. Ennek tipikus példája a Garbage Collection, azaz szemétgyűjtés. A Java (hasonlóan a .Net-hez) nyelvi szinten támogatja a Memory Management-et, és ezzel igen nagymértékben segít a kódolás során elkövetett hibák elkerülésében. Ez a projektek megfelelő időn belül történő befejezése szempontjából igen figyelemreméltó. A Java memóriakezelése során igyekszik lecsökkenteni a heap-töredezettséget, illetve összefüggővé tenni a használt memóriaterületet. Ez gyors memóriafoglalást eredményez.

A Java platformfüggetlensége mellett fontos megemlíteni testreszabhatóságát és optimalizálási lehetőségeit:

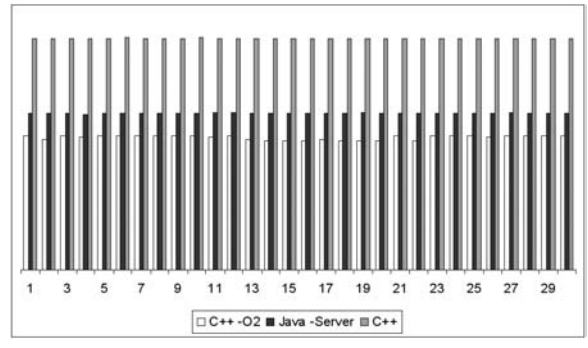
- A Garbage Collection során lehetőségünk van a megfelelő stack-méret beállítására, a többgenerációs másoló- és tömörítő kollektoroknak köszönhetően megszabhatjuk a generációk méretét és arányát. Ezzel meghatározzuk a GC futásának gyakoriságát, továbbá az algoritmus futási idejének hosszát is.
- A Java által használt GC technika nagyban alapoz az objektumok élettartamára vonatkozó megfigyelésekre. Például minél fiatalabb egy objektum, annál rövidebb az élettartama.
- Lehetőség van a különböző optimalizációs paraméterek kiválasztására. (`-server`, `-client`, `-hotspot` stb.) amelyek alkalmazásával az igényekhez mérten alakíthatjuk a kód futását. Egy szerver alkalmazás például lassabban indulhat, de legyen nagy a rendelkezésre állása, és adjon minél hatékonyabb szolgáltatást. A kliens alkalmazásnál viszont a gyors indulás a főbb szempont. Ezeket is igyekeztek a Java fejlesztők szem előtt tartani, így a modern JVM -ben már több modul is található az igényeknek megfelelően.
- A legújabb fejlesztésekben egyre nagyobb hangsúlyt fordítanak a minél jobb hardverkihasználásra. A JRE 1.5-ös verziójában például a Celeron processzorok minél hatékonyabb regiszter kihasználására törekedtek.

Sokan lassúnak tartják a Java-t például a C++-szal szemben, mondván a JVM többrétegű architektúrája lassú működést eredményez. Ez a kijelentés eléggé idejétmúltnak tekinthető, hiszen mérésekkel bizonyítható, hogy a Java teljesítménye igencsak összemérhető a C++-szal. A Java nagyobb memóriaigénye pedig a hardverköltségek mérséklésével már nem jelent olyan nagy hátrányt.

Az első ábrán harmincezer elem buborékrendezéssel történt sorba állításának időigénye látható. A Java `-server` opciója hatékonyabban oldotta meg a feladatot a C++-nál. Érdemes megfigyelni a Java –server opciójának alkalmazása során nyújtott optimalizálás mértékét is. Az ábrán a vízszintes tengelyen a futtatások száma látható. Minden egyes paraméter során harmincszor ismétlődött meg a mérés, ugyanazokra az elemekre, ugyanolyan körülmények között. A Java átlagosan **3.668%**-kal teljesített jobban, mint a C++, mindkét esetben a legjobb optimalizációs paramétereket használva.




A második ábrán látható adatok alapja húszmillió elem rendezésének feladata quicksort algoritmussal. Itt is összemérhető a Java teljesítménye a C++-éval, és nem marad el tőle lényegesen. Ne felejtsük el továbbá a cseréknél nyújtott kényelmes programozhatóságot, hordozhatóságot és könnyebb kódellenőrzést. Ennél a rendezésnél a Java **17,2%**-kal lassabb volt, mint a C++, szintén a legjobb optimalizáció használata esetén. A Java így is **32.3%**-kal hatékonyabb a legjobb optimalizációt használva az optimalizáció nélküli C++ kódnál.



Erre a teljesítményre szükség is van, mert az emberek hamar rájöttek, hogy milyen nagyszerű dolog a web, és egyre többmindent szeretnének a böngészőkön keresztül elintézni. Banki, üzleti ügyek intézéséhez, webes felületű levelezéshez és még sok minden máshoz. Mindenféleképpen olyan technológiát kell alkalmazni, amely lehetővé teszi, hogy dinamikusan generáljuk külső állapotoktól, eseményektől függően az aktuális tartalmat. A CGI volt az első erre alkalmas technológia. Ennél az eljárásnál a kiszolgáló a hozzá érkezett kéréseket egy külső programhoz továbbította. Ez a program feldolgozta a kérést, és az eredményt visszaküldte a felhasználónak. Minden olyan kérés hatására, ami CGI programot igényel, a kiszolgálónak új rendszerfolyamatot kell indítania a program futtatásához, majd az utolsó bitig át kell adnia az összes olyan információt, amelyre a válasz elkészítéséhez szükség van. Egy-egy ilyen művelet ezért rendkívül költséges. A másik probléma, hogy egy CGI program nem tud interaktív módon együttműködni a webkiszolgálóval és nem képes kihasználni annak képességeit, mert különálló rendszerfolyamatban fut.

Később a CGI fejlesztéseként jött létre a FastCGI, mely nagyjából ugyanúgy működik mint a CGI, viszont minden egyes programhoz a memóriában maradó rendszerfolyamatot hoz létre. Így nem kell minden kéréshez külön folyamatot indítani. Ez egy nagy előrelépésnek számított, de az alapvető probléma továbbra is megmaradt: programonként egy folyamatot kell futtatni. Ráadásul a FastCGI továbbra sem segíti semmiben sem a program és a kiszolgáló szorosabb együttműködését.

Ezzel szemben a Java Servletek egy szervlet konténeren belül futnak. Az ilyen konténerek gyakorlatilag egy Javaképes webszervernek felelnek meg. A konténer és a szervletek között olyan szoros együttműködés van, hogy ez utóbbiak képesek egy rendszerfolyamaton belül külön szálakon futni. Így megspórolható a folyamat indítási és átadási költség. Mivel Java alapúak, az operációs rendszerek és a webkiszolgálók között is hordozhatók.

Miután betöltődött egy szervlet, egyetlen objektumpéldányként a kiszolgáló memóriájában marad. Ezt követően a kiszolgáló egy nagyon egyszerű metódussal hívja meg a szervletet egy kérés kiszolgálásához. Mivel nincs szükség további rendszerfolyamatokra vagy az értelmező program meghívására, ezért a szervlet gyakorlatilag a kérés beérkezésekor azonnal megkezdheti annak feldolgozását. Az egyidejűleg beérkező kéréseket különböző szálakban kezeli a kiszolgáló, ami nagyfokú méretezhetőséget tesz lehetővé. Szervletet készíteni nagyon egyszerű!

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class WebConfServlet extends HttpServlet {
 protected void doGet(
  HttpServletRequest request,
  HttpServletResponse response)
 throws ServletException, IOException {
  response.setContentType("text/html;charset=
➥ UTF-8");
  PrintWriter out = response.getWriter();
  out.println("<html><body>");
  out.println("<h1>WebConf</h1>");
  out.println("</body></html>");
  out.close();
 }
}
```

Csupán bővíteni kell a `HttpServlet` osztályt azokkal a képességekkel, amikkel fel szeretnénk ruházni az osztályunkat. Amikor egy olyan kérés érkezik a kiszolgálóhoz, ami a szervletünkre hivatkozik, akkor a konténer automatikusan meghívja az adott kérés típusához tartozó metódust. Például egy egyszerű GET kérésre a `doGet()`, egy POST-ra a `doPost()` hívódik meg automatikusan. A kérést kiszolgáló metódus két paramétert kap: egy `HttpServletRequest` és egy `HttpServletResponse` osztályt. Az első tartalmaz minden olyan információt, ami a kéréshez tartozik, például a fejlécet, a paramétereket, a sütiket, a session objektumokat. A `HttpServletResponse` osztályú paraméterünkön keresztül tudjuk eljuttatni a választ a felhasználóhoz.

Ezek után már csak a konténerünk tudtára kell adni, hogy hol található és hogyan érhető el a szervletünk. Ehhez létre kell hoznunk egy webalkalmazást, majd annak `WEB-INF/web.xml` fájlját legalább az alábbi minimális módon meg kell adni:

```
<web-app>
 <servlet>
  <servlet-name>WebConfServletAlias</servlet-name>
  <servlet-class>WebConfServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>WebConfServletAlias</servlet-name>
  <url-pattern>webconfservleturl</url-pattern>
 </servlet-mapping>
</web-app>
```

A `<servlet>` elem szolgál a szervlet osztállyal kapcsolatos adatok megadására. Minden esetben meg kell adni egy álnevet (`<servlet-name>`) amivel később hivatkozni fogunk rá és annak az osztálynak a nevét amit meg kell a konténernek hívnia (`<servlet-class>`). A `<servlet-mapping>` elemen belül a szervlet elérésével kapcsolatos információk kapnak helyet, melyek közül az `<url-pattern>` a szervlet elérését határozza meg. Ebben az esetben ha a böngészőnkből meghívjuk a `http://serverneve/webconfservleturl` címet, akkor az általunk megírt `WebConfServlet doGet()` metódusa fog lefutni.

A szervletek legnagyobb hátránya az, hogy a megjelenítés és a feldolgozás egybe van építve, sőt mindezt még le is kell fordítani. Így ha a felhasználói felületet meg szeretnénk változtatni, akkor ahhoz át kell írni a Java forrást, majd le kell fordítani. Sok esetben nem biztos, hogy a grafikusok rendelkeznek a szükséges Java ismeretekkel, hogy a munkájukat szervletek alkalmazása esetén is kifogástalanul el tudják látni. Ilyenkor szegény Java fejlesztőt fogják hátráltatni, hogy segítsen nekik a munkájuk szervletekbe foglalásában. Ezen probléma elkerülésére kiválóan alkalmazható a JSP technológia. A JSP segítségével különválasztható a megjelenítés az adatok kezelésétől, az alkalmazáslogikától.

A JavaServer Pages célja, hogy mind a programozók, mind az egyéb felhasználók számára könnyebbé tegye a webtartalmak elkészítését. Mindezt a jól bevált szervlet technológiára építve. Mivel a JSP segítségével teljes mértékben szét lehet választani az ügyfél oldali felület és az üzleti logika fejlesztését, a kettőt akár párhuzamosan is végezhetjük, ezáltal lényegesen lerövidül a webalkalmazások fejlesztési ideje.

A technológia egyszerűségét mutatja hogy egy JSP oldal nem más mint egy egyszerű szöveges dokumentum, ami tartalmazhat statikus HTML elemeket és dinamikus részeket is. A dinamikus tartalom előállítására speciális JSP komponenseket kell használnunk. Ezeket az elemeket biztosíthatják úgynevezett JavaBean-ek illetve TagLibrary-k.

A JavaBean egy egyszerű objektum ami adatok és műveletek összefogására használható. Egy ilyen JavaBean-nek implementálnia kell a `java.io.Serializable` felületet, valamint meg kell felelnie a JavaBean-ekre vonatkozó névadási konvenciónak. Ez a konvenció rögzíti hogy az objektum tulajdonságaihoz hogyan kell lekérdező (get) és beállító (set) metódusokat készíteni.

Az alábbi példa megmutatja, hogy hogyan kell felépíteni egy egyszerű `KonferenciaBean`-t úgy, hogy annak `name` tulajdonságához a JSP lapunkról hozzáférjünk mind olvasásra mind írásra.

```
public class KonferenciaBean implements
➥ java.io.Serializable {
 private String name=new String();
 public String getName() {return this.name;}
 public synchronized void setName(String pName)
➥ {this.name = pName;}
}
```

A JSP lapokon használható elemeket három csoportba sorolhatjuk.

- A direktívák azok az elemek, amelyek elérésről-elérésre azonos viselkedésűek.
- Az akcióelemek valamilyen eseményt hajtanak végre az éppen rendelkezésre álló adatok alapján. Ez az esemény szinte bármi lehet, így érhetjük el a JavaBean-jeinket is.
- A szkriptelemek helyes Java kódot tartalmaznak.

Amikor egy felhasználó megnyit egy JSP lapot, a futtató konténer megvizsgálja hogy történt-e változás a lap tartalmában, illetve megnézte-e már valaki. Ha ez az első látogatás az oldalon vagy valamelyik fejlesztő megváltoztatta a tartalmat akkor a konténer automatikusan készít a lapból egy szervlet forrást és lefuttatja azt. A forrás elkészítésére a szkriptelemek miatt van szükség. Ez az eljárás az első futtatáskor valóban némi többlet időt eredményez, de ezek után már a jól megszokott szervlet sebességgel működik az oldalunk.

A JSP lehetővé teszi a jelölő kód és a programozási nyelven megírt kód szétválasztását. Ezáltal míg a HTML kód a weboldal külső megjelenítéséért felelős, addig a programkód feladata a dinamikusan változó rész megvalósítása. A szétválasztás egyik módja a JSP standard és egyedi akcióelemeinek használata. Ezeket az elemeket programkódok valósítják meg, és ugyanúgy használhatók, mint egy normál weboldal szokásos jelölőelemei. A másik lehetséges eljárás a JSP kombinálása más Java technológiákkal.

Elérkezett az idő, hogy használjuk is a `KonferenciaBean` objektumot.

Adjunk az egyetlen tulajdonságának ertéket majd olvassuk ki azt:

```
<jsp:useBean
  id="rendezveny"
  scope="application"
  class="KonferenciaBean" />
<jsp:setProperty
  name="rendezveny"
  property="name"
  value="WebConf" />
<jsp:getProperty
  name="rendezveny"
  property="name" />
```

A `jsp:useBean` hatására létrejön a `class` paraméterben megadott osztály a `scope`-ban jelzett életciklussal. Az *application* azt jelenti, hogy amíg a konténer működik, ez az objektum is létezik és eléri minden más objektum az adott JVM-en belül. A *session* objektumok addig léteznek amíg a HTTP session, a *request* addig, míg az adott kérés kiszolgálásra kerül, a *page* addig, amíg az adott JSP lap feldolgozása folyik (mivel egy kérést több lap is kiszolgálhat). A `setProperty` hívásra a `property`-ben jelzett tulajdonsághoz tartozó setter (`setName`) metódus hajtódik végre míg a `getProperty` a getter (`getName`) metódust hívja meg. Mindebből a weboldalon csak a WebConf felirat fog megjelenni a `getProperty` hatásásra.

Abban az esetben ha a Bean-ünk valamelyik tulajdonságának nem statikus értéket akarunk megadni hanem a HTTP kérés valamely paraméterének az értékét, akkor a `value` helyen a `param` kulcsszót kell használnunk, melynek az értéke a kívánt paraméter neve lesz. Ennek például az űrlapok feldolgozásánál van szerepe, mert így az űrlaptól érkező adatokat közvetlenül be tudjuk tölteni egy objektumba. Azonban nagyon kényelmetlen lenne egy nagyobb űrlap esetében minden beviteli elemhez egy-egy külön sort megfeleltetni a JSP lapunkon, ezért erre van egy sokkal kényelmesebb megoldás is.

```
<jsp:useBean
  id="rendezveny"
  scope="page"
  class="KonferenciaBean" >
<jsp:setProperty
  name="rendezveny"
  property="*" />
</jsp:useBean>
```

Ebben az esetben a kérés összes GET és POST paraméterének az értéke belekerül a `KonferenciaBean`-be, feltéve hogy a Bean-nek van az egyes paraméterek nevével megegyező tulajdonsága.

Vannak esetek amikor nem elég egy vagy több Bean, összetettebb eszközre van szükségünk. Például szeretnénk egy adatbázis-lekérdezés eredményét megjeleníteni. Ekkor a mások vagy saját magunk által készített TagLibraykat hívhatjuk segítségül. Ezek gyakorlatilag újra felhasználható webkomponensek (egyedi akcióelemek).

Az interneten keresgélve sok ilyen kész TagLibrary-t találunk, melyek közül a JSTL-t emelném ki. Ha nem akarjuk a JSP lapjainkat teletűzdelni szkriptelemekkel, akkor a ciklusokat és elágazásokat kiválthatjuk JSTL akciókkal. Sőt, a JSTL nem csak egy lista bejárására, hanem akár SQL lekérdezések végrehajtására is alkalmas. Az alábbi egyszerű módon jeleníthetjük meg JSTL-lel egy SQL lekérdezés eredményét a JSP lapunkon.

A tomcat `server.xml` konfigurációs fájljában a `Host` elemen belül hozzunk létre egy új kontextust.

```
<Context path="/webconf" docBase="webconf"
debug="5" reloadable="true" crossContext="true">
 <Logger className=
➥ "org.apache.catalina.logger.FileLogger"
➥ prefix="localhost_DBTest_log." suffix=".txt"
➥ timestamp="true"/>
 <Resource name="jdbc/TestDB" auth="Container"
➥ type="javax.sql.DataSource"/>
 <ResourceParams name="jdbc/TestDB">
  <parameter>
   <name>factory</name>
   <value>org.apache.commons.dbcp.
➥ BasicDataSourceFactory</value>
  </parameter>
  <parameter>
   <name>maxActive</name>
   <value>100</value>
  </parameter>
  <parameter>
```

```
    <name>maxIdle</name>
    <value>30</value
   </parameter>
   <parameter>
    <name>maxWait</name>
    <value>10000</value>
   </parameter>
   <parameter>
    <name>username</name>
    <value>userneve</value>
   </parameter>
   <parameter>
    <name>password</name>
    <value>userjelszava</value>
   </parameter>
   <parameter>
    <name>driverClassName</name>
    <value>com.mysql.jdbc.Driver</value>
   </parameter>
   <parameter>
    <name>url</name>
    <value>jdbc:mysql://dbhost:3306/
➥ database?autoReconnect=true</value>
   </parameter>
  </ResourceParams>
</Context>
```

Ezzel megmondtuk hogy a *jdbc/TestDB* hivatkozással szeretnénk elérni a *dbhost*-on a *database* adatbázist a *userneve/userjelszava* felhasználóval. Ezt követően a JSP lapon már használhatjuk is.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
➥ prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql"
➥ prefix="sql" %>
<sql:query var="rs" dataSource="jdbc/TestDB">
 select nev from webconf where eloadas=
➥ "Webre, de gyorsan"
</sql:query>
 <c:forEach var="row" items="${rs.rows}">
  ${row.nev}<br />
 </c:forEach>
```

Ha elfogy a türelmünk és a keresgélés és több órányi próbálgatás után sem találunk magunknak megfelelő Library-t, akkor nincs más választás, mint írni kell egyet. Ebben a TagSupport és BodyTagSupport osztályok, valamint a Magyarországi Web Konferencia 2006 *Webre, de gyorsan* című előadása nyújthatnak hathatós segítséget.


### *Soós István*

*Tizenhárom éve foglalkozik programozással, nyolc éve webes technológiákkal (C++, Perl, PHP, ASP.NET, Java). Jelenleg a Budapesti Műszaki és Gazdaságtudományi Egyetem Automatizálási és Alkalmazott Informatikai Tanszékén doktorandusz és a Sun Magyarországnál Java/J2EE konzulens.*

### *Karóczkai Krisztián*

*A PHP Roadshow sorozat dunaújvárosi állomásának egyik fő szervezője és előadója volt 2004-ben és 2005-ben is. A 2004-es Dunaújvárosi Java szakmai nap szervezője és előadója. Jelenleg az MTA SZTAKI Párhuzamos és Elosztott Rendszerek Laboratóriumában a portál csoport vezetője. Munkatársaival azon dolgozik, hogy a világ különböző grid-jeinek szolgáltatásait a felhasználók könnyedén egy Java alapú webes felületen tudják igénybe venni. A webes rendszerek elkötelezett híve, több kereskedelmi rendszer megalkotásában vett részt. Ezelőtt nyolc évvel került kapcsolatba először a Java technológiával. A 2005-ös Sun Java Fejlesztői Verseny Web-szolgáltatások megvalósítása J2EE 1.4 alapon kategória nyertes csapatának, a JavaLightTeam-nek a tagja.*

# Kreáljunk egy kis webet!

A HTTP protokoll és a rá épülő HTML technológia lehetővé teszi egyszerű, kérdés-válasz alapú webes alkalmazások készítését. A legtöbb webes technológia ezeket az alapokat helyezi át valamilyen programozási környezetbe, legyen az CGI, PHP vagy éppen a Java alapú Servletek, JSP-k. A fejlesztő számára nyújtott beépített szolgáltatások szintje, újrahasználhatóságuk nagysága függ az adott platformtól, de feldolgozási logikájuk kérés-válasz alapú. Hamar jelentkeznek a bonyolultabb rendszereknél felmerülő problémák: nehéz módosítani, karbantartani a kódot éveken keresztül, teljes újraírás nélkül. Grafikus felhasználói felületek tervezésekor és implementálásakor már korábban (SmallTalk) bebizonyosodott, hogy hatékony megoldás a Model-View-Controller (MVC) tervezési minta használata.

Az MVC alapú grafikus alkalmazás három fő feladatkört (és komponens-csoportot) különít el: a *Model* felel az alkalmazás adataiért, ez fedi le a belső struktúráját; a *View* komponensek formázzák meg és tálalják a felhasználó elé a megfelelő és éppen igényelt adatokat; míg a *Controller* felügyeli a folyamatot: ez dolgozza fel a felhasználó eseményeket, módosítja/lekérdezi a modellt, majd meghatározza, hogy milyen felületet jelenítsen meg mindezek alapján.

## JavaServer Faces

A webes fejlesztők is elkezdték alkalmazni az MVC alapú keretrendszereket, a Java világban az elmúlt néhány év legnépszerűbb nyílt forráskódú rendszere a Struts volt. A J2EE platform azonban elsősorban a szabványosított specifikációkat szereti, így elindult – a Struts vezető fejlesztőjének a bevonásával – a Java webes MVC keretrendszer specifikációja, melynek eredménye a JavaServer Faces (JSF) szabvány lett. A JavaServer Faces kialakításakor a fő cél az eddigi tapasztalatok felhasználása, hiányok pótlása, valamint a vizuális eszközzel történő fejlesztés támogatása volt, a forráskód alapú hatékony szerkeszthetőség megtartása mellett. Az ipar (például Sun, IBM, Oracle) valamint a nyílt forráskódú közösség (például Apache MyFaces) nagyon hamar a specifikáció mögé állt.

A JSF egy szerver oldali, komponens alapú felhasználói felület-keretrendszer, webes és általános környezetre. Felépítése független a kliens alkalmazástól, azaz ugyanazt az alkalmazás-logikát használhatjuk webes vagy mobil felületre egyaránt – sőt elméletileg készíthető hozzá vastag kliens alkalmazás is. A felületi elemek (címkék, szövegmezők, gombok, jelölőmezők) állapottal rendelkeznek a szerver oldalon, amely állapot az események feldolgozásakor az MVC-ben elvárt módon mindig a megfelelő lesz. A felületi komponensek állapota, eseménymodellje és a megjelenítési környezet jól specifikált.

Bár a JSF használatának előnye leginkább bonyolultabb oldalak szerkesztése során jön elő, terjedelmi okokból most csak egy egyszerűbb példát mutatunk be. A GuessNumber példaalkalmazás célja, hogy a felhasználó által tippelt számra megmondja, hogy tényleg arra „gondolt-e" a szerver vagy sem.

Először definiáljuk az oldal tartalmát (`pages/guess.jsp`):

```
<%@ taglib uri="http://java.sun.com/jsf/html"
➥ prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core"
➥ prefix="f" %>
<f:loadBundle basename="guess.messages"
➥ var="msg"/>
<html>
 <f:view>
  <h:form id="inputNumbers">
    <h:outputText value="#{msg.how_to_play}"/>
    <br/>
    <h:outputText value="#{NumberBean.message}"/>
    <br/>
    <h:inputText id="userNumber"
➥ value="#{NumberBean.userNumber}" required="true">
       <f:validateLongRange minimum="0"
➥ maximum="100"/>
    </h:inputText>
   <br/><br/>
   <h:commandButton
➥ value="#{msg.makeguess_button}"
➥ action="#{NumberBean.checkGuess}"/>
   <br/>
   <h:commandButton
value="#{msg.trayagain_button}"
➥ action="#{NumberBean.playagain}"/>

  </h:form>
 </f:view>
</html>
```

A `guess.jsp` egy egyszerű űrlapot ír le, amelyen szerepel egy beviteli mező (`h:inputText`), két gomb (`h:commandButton`) és néhány kiírásra szolgáló címke (`h:outputText`). A JSP elején definiáljuk a használt JSF tag library elemeket, majd beállítjuk a többnyelvű feliratokat tartalmazó erőforrást (`f:loadBundle`). A JavaServer Faces – hasonlóan a Java platform más felületeihez – teljes mértékben támogatja a többnyelvű felhasználói felületek egyszerű összeállítását. Ennek kihasználására az alkalmazáshoz mellékelni kell egy-egy property fájlt minden nyelvhez (`guess/messages.properties`):

```
how_to_play=Please pick a number between 0
➥ and 100.
makeguess_button=Guess
trayagain_button=Again!
success_text=You have guessed the number,
➥ {0} is correct!
tryagain_smaller=Incorrect guess. Please try
➥ a smaller number.
tryagain_bigger=Incorrect guess. Please try
➥ a bigger number.
```

Amennyiben például magyarul is el szeretnénk érni az alkalmazást, nem kell mást tennünk, csak az előbbi feliratokat lefordítani és a `guess/messages_hu_HU.properties` fájlba elmenteni. Amennyiben a böngésző jelzi, hogy magyar nyelven szeretné elérni a tartalmat, automatikusan magyarul jelenik meg, de authentikációval rendelkező rendszerben felhasználói profilhoz is rendelhető a megjelenítendő nyelv választása. Fontos kiemelnünk a beviteli mezőhöz tartozó validátor használatát. Többféle (értéktartomány, mintaillesztő kifejezés, dátumformátum, saját magunk által definiált) validátort helyezhetünk az egyes beviteli mezőkre, amelyek segítségével a hibaellenőrzés

szerver oldalon és (generált JavaScript segítségével) kliens oldalon is egyszerűsödik.

Az oldal működéséhez még meg kell írnunk a JSF oldalhoz tartozó úgynevezett backing beant, azaz a háttérosztályt (`NumberBean.java`):

```
package guess;

import javax.faces.context.*;
import javax.servlet.http.*;
import java.util.*;

public class NumberBean {

  public NumberBean () {
     playagain();
  }

  // változók az oldal adatainak eléréséhez
  // A felhasználó által beírt szám
  Integer userNumber;
  // A kisorsolt véletlenszám
  int randomNumber ;
  String message = "";

  // get és set bean-metódusok kellenek,
  // de ez fejlesztői eszközzel generálható
  public Integer getUserNumber ()
➥ {return this.userNumber;}
  public void setUserNumber (Integer value)
➥ {this.userNumber = value;}
  public String getMessage()
➥ {return this.message;}

  public String playagain () {
    randomNumber = (int)(Math.random()*100);
    this.message = "";
    return "playagain";
  }

  public String checkGuess () {
    if ( userNumber.intValue() == randomNumber )
      return "success";

    FacesContext context =
➥ FacesContext.getCurrentInstance();
    ResourceBundle bundle =
➥ ResourceBundle.getBundle("guess.messages",
    context.getViewRoot().getLocale());
    if (userNumber.intValue() > randomNumber) {
      this.message =
➥ bundle.getString("tryagain_smaller");
    } else {
       this.message =
➥ bundle.getString("tryagain_bigger");
    }
    return "tryagain";
  }
}
```

A backing bean tehát egy egyszerű Java osztály néhány mezővel, a hozzájuk tartozó get/set valamint a JSF oldalon is hivatkozott eseménykezelő metódusokkal. Az eseménykezelő metódusok szöveges értékkel térnek vissza, amelyeket a navigáció továbblépési szabályainál fogunk felhasználni. Mivel most csak egy nagyon egyszerű példát készítettünk, a továbblépés most csak egy statikus HTML oldalra vezet. Az utolsó megírandó rész a JavaServer Faces konfigurációs állománya (`faces-config.xml`):

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces
➥ Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
<faces-config>
  <navigation-rule>
   <from-view-id>/pages/guess.jsp</from-view-id>
   <navigation-case>
     <from-outcome>success</from-outcome>
     <to-view-id>/pages/success.html</to-view-id>
   </navigation-case>
  </navigation-rule>
  <managed-bean>
   <managed-bean-name>NumberBean</managed-bean-name>
   <managed-bean-class>guess.NumberBean</
➥ managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

A backing beanek többféle környezetben helyezkedhetnek el: az *application scope* minden felhasználó által elérhető és tartalma folyamatosan megőrződik, a *session scope* az adott felhasználói folyamathoz tartozik és csak annak élettartalmáig marad érvényben, míg a *request scope* csak az adott lekérdezés folyamán lesz elérhető.

Első ránézésre a JSF felesleges teljesítmény-overheadnek tűnhet a hagyományos Szervletekhez, JSP-hez képest, azonban az alkalmazás telepítésekor (illetve az egyes oldalak első elérésekor) minden JSF oldal Szervletté fog fordulni, az egyes kötések közvetlen metódushívások, végső soron pontosan címzett memória-elérések lesznek. Ezzel kiküszöbölhető bármilyen esetleges interpreteres végrehajtásból származó hátrány, ugyanakkor megőrződik a fejlesztés egyszerűsége, átláthatósága, a felület és a kód szétválasztott, könnyen karbantartható állapota.

## Sun Java Studio Creator

Bár JSF szerkeszthető a fenti módon, kézileg előállított forrásfájlok segítségével, a legtöbb fejlesztő idő és gépelésigényesnek tartaná. Mivel felépítése jól formalizálható, könnyű hozzá eszköztámogatást nyújtani. A Sun Java Studio Creator egy integrált fejlesztői környezet JSF webes alkalmazások gyors, vizuális fejlesztéséhez. Magába foglalja az alkalmazásszervert is, így további komponens, plugin vagy program letöltésére nincs szükség. A szerkesztőfelületén drag-and-drop módszerrel húzhatunk komponenseket az oldalra, dupla kattintással el is kezdhetjük írni az oldalhoz tartozó eseménykezelőt, illetve grafikus oldal-navigáció segítségével követhetjük az alkalmazásunk irányítási folyamatának vázlatát. Adatbázis-intenzív alkalmazásokhoz beépített SQL lekérdezés építővel, automatikus táblázat-kialakítóval és értékbetöltésekkel rendelkezik.

Az előadásunk végén látható demonstrációban bemutatjuk az előző részben ismertetett JSF programot, valamint ízelítőt adunk a Creator gyors fejlesztést támogató képességeiből.

*Zsemlye Tamás*

*Huszonkét éve foglalkozik programozással, ezen belül több mint tíz éve foglalkozik a Java Platformmal a Sun Magyarországnál.*

*Soós István*

*Tizenhárom éve foglalkozik programozással, nyolc éve webes technológiákkal (C++, Perl, PHP, ASP.NET, Java). Jelenleg a Budapesti Műszaki és Gazdaságtudományi Egyetem Automatizálási és Alkalmazott Informatikai Tanszékén doktorandusz és a Sun Magyarországnál Java/J2EE konzulens.*

# Osszuk meg a tudást!

Az internetes technológiák terjedésével egyre több alkalmazást ruháztak fel olyan képességgel, hogy valamilyen webes felületről el lehessen érni bizonyos funkcióit. Elterjedtek az olyan alkalmazások mind internetes mind intranetes környezetben, melyek már teljes értékű webes kliensfelülettel rendelkeztek. Ez vezetett a webalkalmazások megszületéséhez.

Egy webes alkalmazást karbantartani nagyságrendekkel egyszerűbb, mint egy hagyományos vastag klienst. Amikor egy webes alkalmazást bővítünk vagy módosítunk, elég egy helyen, azon a gépen megtenni, ahonnan ezt elérik a felhasználók. A vastag klienst alkalmazó megoldásoknál azonban esetenként az összes klienst le kell cserélni. Ráadásul a webes alkalmazásokhoz kapcsolódó böngészők egy nyelvet (HTML/XHTML) beszélnek. Igaz, előfordulnak itt is olyan problémák, hogy valamelyik egy bizonyos tájszólást jobban támogat, mint a másik. Viszont egy böngészőn keresztül nem csak egy, hanem számtalan ilyen alkalmazást tudunk használni, ellentétben a vastag klienseket alkalmazó megoldásokkal, ahol minden kiszolgáló csak a saját kliensével akar szóba állni.

A komolyabb webes alkalmazások már jóval összetettebbek lehetnek egy hagyományos dinamikus weblapnál. Ezeket klasszikus webtartalom előállítási technológiákkal megvalósítani nehezebb feladat ugyan, de nem lehetetlen, sőt nagyon sokan készítenek keretrendszereket olyan módszerek alkalmazásával, melyeket nem igazán erre fejlesztettek. Ezek a keretrendszerek annyira elterjedtek, hogy már fel sem tűnik, ha ilyenekkel találkozunk. Egy ilyen nagy alkalmazás több kisebb, önálló egységből épül fel. Ezeket az építőelemeket nevezzük portleteknek. Annak érdekében, hogy ezek a portletek könnyen átvihetőek legyenek egy másik környezetbe, megszületett a JSR 168-as ajánlás, melynek megfelelő porleteket írni és használni nagyon egyszerű. Az előadás során bemutatásra kerül néhány alapfogás a témában, úgy mint:

- konténerek konfigurálása
- mások portletjeinek telepítése
- saját portletek készítése
- JSF használata portletekkel

Korunk komolyabb szoftvermegoldásai tartalmaznak vagy tartalmazhatnak olyan komponenseket, melyek oly módon párhuzamosíthatóak, hogy az egyes részek nem egy, hanem több különböző számítógépen futnak. Ennek nem csak programozástechnikai, hanem gazdasági előnyei is vannak. Olcsóbb több, kisebb teljesítményű számítógépből összeállítani egy nagy számítási erőforrást, mint beruházni egy szuperszámítógépbe. Ugyanakkor előfordul, hogy bizonyos adatok csak megadott helyekről hozzáférhetőek, vagy az eléréshez szükséges egyéb feltételek túl drágák. Ezeknek az eltérő helyen különböző környezetben futó részeknek azonban szükségszerűen adatokat kell cserélniük egymással. Erre már régen rájöttek a nagy szoftvergyártók, és elkezdték kifejleszteni saját megoldásaikat. Az OMD létrehozta a CORBA-t, a SUN az RMI/IIOP-ot, az IBM a DSOM-ot, a Microsoft a DCOM-ot. Ezek a saját környezetükben nagyon jól működtek, de némelyiküket nagyon nehéz volt közös munkára bírni, és csak úgynevezett híd alkalmazások segítségével lehetett. Ezen alkalmazások természetesen plusz erőforrásokat igényeltek, és ha valamelyik oldalon változott a kommunikációs felület, akkor ezt az alkalmazást módosítani kellett. Ezek a problémák az üzleti folyamatok integrációjának legfőbb akadályaivá váltak.

A webszolgáltatások (web services) elosztott rendszerek együttműködő-képességének javítása érdekében jöttek létre. A webszolgáltatások leírása tulajdonképpen egy nyílt szabványhalmaz, mely komponensalapú és szolgáltatás-orientált alkalmazások létrehozását teszi lehetővé. Mivel a webszolgáltatások szigorúan definiált felületekkel az interneten keresztül könnyedén elérhető funkciókat biztosító alkalmazások, melyek felépítését teljes egészében nyílt és szabványos elemek írják le, a működésük a heterogén internetes környezetben is garantált. Nem változtattak semmit a már bevált távoli metódushívás alapkoncepcióján, csupán azt teljesen új, nyílt alapra (HTTP, XML) helyezték. Minden egyes implementáció (XML-RPC, SOAP, WSDL, UDDI) ezekből indul ki. Ezért gyakorlatilag bármi lehet web service, ami elérhető az interneten keresztül szabványos XML alapú üzenetekkel és független mind operációsrendszerektől, mind programnyelvektől.

Az XML-RPC az egyik legegyszerűbb XML alapú üzenetküldő protokoll. Alaptípusok átvitelére ideális, összetettebb adatszerkezetek közül még tömbök és sruktúrák átadására is képes, viszont objektumokról nem tud információkat kezelni.

SOAP a komolyabb webszolgáltatások alapköve, lehetővé téve szolgáltatások és alkalmazások számára az egyszerű adatcserét. A SOAP üzenet az adattípusok olyan ábrázolását tartalmazza XML formátumban, mely könnyen leképezhető implemetáció-specifikus formára, ráadásul nem csak a HTTP protokollon keresztüli metódushívást támogatja.

A nagyobb szoftvergyártók és a témával foglakozó nyílt forrású közösségek elkészítették már azokat az eszközöket, melyek segítségével nem kell nekünk saját erőből megvalósítanunk a teljes protokollt a szabványos kommunikációhoz. Az Apache SOAP gyakorlatilag két szervletet hozott létre, amelyek biztosítják a szükséges eszközöket a szabványos kommunikációhoz, így a tényleges webszolgáltatásunk csak egy egyszerű osztály.

```
public class WebConfService
{
    public WebConfData sendActualData(String
➥ pEloadas)
    {
        WebConfData res=new WebConfData();
        res.setEloadas(pEloadas);
        return res;
    }
}
```

Ezt a webszolgáltatás be kell jegyeznünk. Erre a célra vagy a webes adminisztrációs felületet vagy egy parancssori programot használva egy egyszerű WSDL fájlt használhatunk. A WSDL (Web Services Description Language) a webszolgáltatások XML nyelven történő leírását definiálja. Ez a leírás tartalmazza

- Az elérhető függvények listáját.
- A bemeneti és kimeneti adattípusukra vonatkozó információkat.
- Az adatátviteli protokollt.
- Az adott szolgáltatás eléréséhez szükséges címet.

Ezen leírás alapján a kliens alkalmazásunk meg tudja találni és meg tudja hívni a kívánt szolgáltatás nyílvános metódusait. Így a WSDL-lel a kliens alkalmazásainkat egy kisméretű szöveges fájl segítségével interaktívan tudjuk eltérő szolgáltatásokhoz csatlakoztatni.

Az UDDI (Universal Discovery Description and Integration) az egyes szolgáltatások regisztrálását és felkutatását specifikáló eljárás.

## Sun eszközök a Java fejlesztéshez

A Java fejlesztés alapvetően platform és eszközfüggetlen, minden részletét egyszerű szövegszerkesztőkkel össze tudjuk állítani, nincs semmilyen bináris fájl, amely információt tárolna. A fejlesztők – néhány kivételtől eltekintve – azonban jobban szeretnek valamilyen eszköztámogatást igénybevenni a munkájukhoz. A következő részben a Sun eszközök ingyenesen elérhető, már nyílt forráskódú, vagy rövidesen nyílt forráskódúvá váló termékeit mutatjuk be röviden, amelyek hatékonyan segítik a korábban bemutatott technológiákra történő fejlesztéseket.

A Sun fejlesztőeszközök alapját a nyílt forráskódú Netbeans adja. Nemrég kiadott 5.0-ás változata megjelenésének időpontjában az egyik legtöbb funkciót tartalmazó eszköznek nevezhető. Beépített grafikus felület tervezőjéhez hasonlót más eszközökben nem találunk, de azonnali támogatást kaphatunk mobil fejlesztésekhez és szerver oldali (többek között webes) fejlesztéshez, futtatáshoz és hibafelderítéshez egyaránt. A beépített dinamikus profiler segítségével könnyen felderíthetőek a lassan lefutó, tovább optimalizálható kódok, feleslegesen beragadt referenciák, valamint könnyen meg tudjuk tekinteni egy összetett felépítésű program karakterisztikáját. A Netbeans két részből áll: az IDE tartalmazza a fejlesztői eszközöket és felületet, míg a Netbeans Platform egy pluginekkel bővíthető általános keretrendszert biztosít vastag kliens programok fejlesztéséhez. Gyakorlatilag az IDE is a Platform kiegészített változata.

A Netbeans eszközre épül a Sun Java Studio Enterprise és a Sun Java Studio Creator is. Mindkettő eszköz ingyenesen elérhető, letölthető és használható kereskedelmi célokra is. A Sun Java Studio Enterprise a Netbeans alapokra építve többek között a következő támogatást biztosítja a fejlesztőknek:

- Vizuális XML séma szerkesztés és a sémához tartozó XML példány kezelése
- Üzleti folyamatok leírására BPEL alapú, grafikus folyamatábra szerkesztő
- Integrált BPEL futtató-környezet és a folyamatok tesztelése
- Webszolgáltatások egyszerű készítése, alkalmazása, SOA-architektúrák fejlesztése
- Portlet készítés
- UML alapú modellezés, kódgenerálás
- Fejlesztői kapcsolattartást segítő üzenetküldő eszköz

A Java Enterprise Edition környezetek, így a webes felületek is szerver oldali konténerben, alkalmazásszervben futnak. A Sun Application Server legújabb változata Glassfish néven nyílt forráskódúvá vált, szabadon elérhető és használható. Más rendszerekhez hasonlóan a használata ingyenes, támogatást lehet hozzá vásárolni. Végül a Sun Java Enterprise System teljes vállalati megoldásként minden felmerülő feladatra biztosít szoftvert, portáltól kezdve az identity manageren keresztül a message queue megoldásokig. Legacy rendszerek integrációjához, szolgáltatás-orientált architektúrák kialakításához pedig a Sun Integration Suite (korábban: SeeBeyond) ajánlható.

*Karóczkai Krisztián*

*A PHP Roadshow sorozat dunaújvárosi állomásának egyik fő szervezője és előadója volt 2004-ben és 2005-ben is. A 2004-es Dunaújvárosi Java szakmai nap szervezője és előadója. Jelenleg az MTA SZTAKI Párhuzamos és Elosztott Rendszerek Laboratóriumában a portál csoport vezetője. Munkatársaival azon dolgozik, hogy a világ különböző grid-jeinek szolgáltatásait a felhasználók könnyedén egy Java alapú webes felületen tudják igénybe venni. A webes rendszerek elkötelezett híve, több kereskedelmi rendszer megalkotásában vett részt. Ezelőtt nyolc évvel került kapcsolatba először a Java technológiával. A 2005-ös Sun Java Fejlesztői Verseny Web-szolgáltatások megvalósítása J2EE 1.4 alapon kategória nyertes csapatának, a JavaLightTeam-nek a tagja.*

*Zsemlye Tamás*

*Huszonkét éve foglalkozik programozással, ezen belül több mint tíz éve foglalkozik a Java Platformmal a Sun Magyarországnál.*

# Bevezetés a WebObjects alapú webalkalmazás-fejlesztésbe

## Bevezető

Idén ünnepli tizedik születésnapját a WebObjects, amely a hazai webalkalmazás-fejlesztők körében viszonylag kevésbé ismert. Nem megérdemelten, mert igen erőteljes és hatékony, Java technológiára épülő alkalmazásfejlesztő környezet. Ebben a cikkben röviden bemutatjuk a WebObjects történetét, felhasználási lehetőségeit és felépítését.

## A WebObjects története

A WebObjects története az 1990-es évek közepére nyúlik vissza, amikor a NeXT Computer által fejlesztett NeXTSTEP operációs rendszer a fénykorát élte. A NeXTSTEP-hez kapcsolódó fejlesztői eszközök segítségével gyorsan és egyszerűen lehetett grafikus felhasználói felülettel rendelkező alkalmazásokat készíteni. Ez hamar felkeltette a nagyvállalati szektor érdeklődését, különösen az adatbázisokhoz kapcsolódó kliens-szerver alkalmazások fejlesztése kapcsán. Ezt a piacot célozta meg a NeXT, amikor megalkotta az Enterprise Objects néven ismert technológiáját, amely az első kereskedelmileg sikeres objektum-relációs leképezést biztosító keretrendszer volt. Az Enterprise Objects segítségével az adatbázis hozzáférés különböző feladatait objektumokra lehet rábízni. 1995-ben a NeXT felismerte, hogy várhatóan a web lesz a kliens-szerver megoldások domináns közege, így mérnökei az Enterprise Objects objektum-orientált szemléletének mintáján és sikerén felbuzdulva kifejlesztettek egy keretrendszert, amely a HTTP kéréseket és a dinamikus HTML generálást kezeli. Erre a keretrendszer gyűjteményre alapozva 1996 januárjában a NeXT bemutatta a WebObjects 1.0-ás verzióját, a világ első objektumorientált webalkalmazás-szerverét.

1997-ben az Apple Computer felvásárolta a NeXT-et, elsősorban a UNIX alapú operációs rendszeréért, amely az Apple operációs rendszerének, a Mac OS X-nek az alapját biztosította. A bekebelezés során természetesen a WebObjects és annak fejlesztőgárdája is átkerült az Apple szárnyai alá. Az Apple internetes megjelenésében központi szerepet kapott a WebObjects, hiszen az Apple online boltja, valamint a .Mac névre hallgató portálja is ebben készült el. A WebObjects értékesítése azonban nem volt igazán sikeres. A közvetlen értékesítés, illetve a közel 50.000 dolláros licensz ár továbbra is csak a nagyvállalati ügyfelek körét célozta meg. A WebObjects-et például az AT&T, a BBC, az Adobe, a Fleet Bank, a Standard & Poor, a Dell, a Disney és a Deutsche Bank használta.

2000 májusában, az 5.0-ás verzió bejelentésekor az Apple stratégiát váltott azzal, hogy a WebObjects árát drasztikusan csökkentette (50.000 dollárról 699 dollárra!), valamint hagyományos módon megvásárolható, dobozos termékként kínálta tovább. Ezzel a lépéssel megnyitotta az utat a fejlesztők, a kis- és közepes vállalatok, és az oktatási intézmények felé. 2005-ben ezt még tovább erősítette azzal, hogy a WebObjects 5.3-as verzióját szabadon letölthetővé tette az Apple fejlesztői honlapjáról. Ma talán a legismertebb WebObjects alapú alkalmazás az iTunes Music Store, a világ legnagyobb online zeneáruháza.

## Mire jó a WebObjects?

Történetéből következően a WebObjects legnépszerűbb felhasználási területe az adatbázis hátteret igénylő internetes/intranetes alkalmazások, ahol az alkalmazás kliens oldali felhasználói felületét dinamikusan generált HTML biztosítja. Ez persze nem jelenti azt, hogy az alkalmazás csak HTML-t szolgálhat ki. Dinamikusan generálhat például XML, PDF, SVG, WML vagy SMIL dokumentumokat is. Ha pedig a HTML-nél dinamikusabb felhasználói felületre van szükség, akkor képes böngészőn belül futó Java Applet-et vagy asztali Java kliens alkalmazást is előállítani. A WebObjects viszonylag új alkalmazási területe a SOAP alapú webszolgáltatások (web services) biztosítása. Fontos megjegyezni, hogy elvileg ugyanaz a WebObjects alkalmazás képes ezeket a típusokat párhuzamosan is kiszolgálni. A továbbiakban – az egyszerűség kedvéért – csak a HTML felülettel rendelkező alkalmazásokkal foglalkozunk.

## A WebObjects felépítése

A WebObjects mint termék három fő komponensre bontható. Az első komponenst a már említett objektumorientált keretrendszerek (frameworks) alkotják. A másodikat az ezekre épülő fejlesztői eszközök, míg a harmadikat a kifejlesztett alkalmazások futtatási (deployment) infrastruktúrája adja. Ha a fejlesztői eszközök felől tekintünk rá, akkor azt mondhatjuk, hogy a WebObjects egy alkalmazás fejlesztői környezet. Ha viszont a futtatási környezet felől, akkor egy alkalmazás szerver.

A WebObjects ma már teljes egészében a Java-ra épül. Ez azt jelenti, hogy a keretrendszerek osztályai Java-ban készültek, a futtatási környezet JVM (Java Virtual Machine) meglétét feltételezi, és természetesen az alkalmazásunk egyedi kódját is Java-ban kell megírnunk. A Java elterjedtsége és nyitottsága a biztosíték arra, hogy alkalmazásunk egyedi igényeknek is megfeleljen, hiszen segítségül hívhatjuk a különböző kereskedelmi és nyílt forráskódú Java-s kiegészítéseket. A WebObjects könnyen együttműködik más, Java alapú futtatási megoldásokkal is, így például EJB konténerekkel, szervletekkel, webszolgáltatásokkal.

A következőkben röviden áttekintjük a főbb keretrendszereket és a hozzájuk kapcsolódó fejlesztői eszközöket.

#### Az Enterprise Objects Framework (EOF) és az EOModeler

Ezt az objektumorientált keretrendszert a WebObjects koronaékszerének is szokták nevezni. Önmagában az EOF ismertetése is meghaladná ennek a cikknek a terjedelmi korlátait, így csak ízelítőt adunk belőle. Az EOF fő feladata az adathozzáférés menedzselése valamilyen külső

adatforráshoz kapcsolódva. Az adatforrás jellemzően egy relációs adatbázis, de lehet LDAP adattár is. Az EOF az adatbázisban tárolt adatokat úgynevezett Enterprise Objects (rövidítve EO) objektumokba képezi le. Ezt a technikát szokás objektum-relációs leképezésnek hívni. Az általunk írt WebObjects alkalmazás objektumorientált módon, közvetlenül ezeket az EOF által biztosított EO objektumokat használja, ha adattárolásra illetve adat visszakeresésre van szüksége. A WebObjects terminológiáját használva, az EOF egy réteg (layer) az alkalmazás és a relációs adatbázis (vagy egyéb adatforrás) között. Az elválasztás révén az alkalmazás nem áll közvetlen kapcsolatban az adatforrással. Relációs adatbázis esetén ez például azt jelenti, hogy az alkalmazás kódjába nem kell (és nem is ajánlott) SQL parancsokat írni.

Ahhoz, hogy az objektum-relációs leképezés megfelelően működjön, definiálnunk kell az EOF számára egy „térképet", ami megmondja, hogy az adatforrás adatai mely EO objektumokba kerüljenek majd bele. A WebObjects-ben ezt a térképet EOModel-nek hívjuk, és az EOModeler nevű fejlesztői eszközzel készítjük el. Az EOModel fájl több adatot is tartalmaz. Többek közt ebben van, hogy az adatforrás milyen URL címen érhető el. Relációs adatbázis esetén ez jellemzően egy JDBC (Java Database Connectivity) URL, hiszen a WebObjects JDBC kapcsolaton keresztül kommunikál az adatbázissal. Szinte az összes ismert adatbázishoz létezik JDBC illesztő, így ezek közül bármelyiket használhatjuk a WebObjects-es alkalmazásunk adatbázis hátteréül. Fontos megemlíteni, hogy az EOF párhuzamosan több, akár különböző gyártótól származó adatforrásból is képes dolgozni, és az objektum-relációs leképezést megvalósítani. Így elképzelhető, hogy az EO objektumunk bizonyos adatai az egyik adatforrásból, bizonyos adatai pedig másik adatforrásból származnak. Azt, hogy melyik honnan, sem az EO objektum, sem az alkalmazásunk nem tudja. Ez pusztán az EOF felelőssége.

Az EOModel-ben megtalálható az adatforrás struktúrájának összes jellemzője is. Így például a relációs adatbázis tábláinak, oszlopainak neve, valamint az egyes oszlopok SQL adattípusa. Ha az adatbázis struktúra már létezik – azaz nem most kerül kialakításra –, akkor az EOModeler képes a struktúrát „visszafejteni", és ez alapján az EOModel-t automatikusan létrehozni.

Végezetül az EOModel tartalmazza a leképezés tényleges térképét is, vagyis azt, hogy az adatbázis oszlopai mely EO objektumok mely változóihoz vannak hozzárendelve. Az EOModeler az így felállított modell alapján legenerálja a relációs adatbázist, és az EO objektumok Java kódjának alapját.

Felvetődhet a kérdés, hogy miért jó az, ha az adatforrásunk adatai objektumokban vannak elhelyezve, majd az alkalmazásunkból ezeket az EO objektumokat használjuk, ahelyett, hogy közvetlenül az adatforrást használnánk? A WebObjects egyik fő koncepciója, hogy élesen szétválasztja az alkalmazás megjelenését, az alkalmazás funkcionalitását (logikáját), és az alkalmazás adathozzáférését. Sőt, igyekszik elkülöníteni egymástól az alkalmazás szakterületének megfelelő üzleti logikát (business logic) az alkalmazás egyéb logikájától (application logic). Ez a gondolat részben a SmallTalk Model-View-Controller paradigmájában, részben a NeXTSTEP objektumorientált fejlesztői környezetében gyökerezik. Az adathozzáférést az EOF segítségével már elkülönítettük, lényegében adatbázis-függetlenné tettük az alkalmazásunkat. Az üzleti logikát az EO objektumok kódjába helyezzük el, így az EO-k már nem pusztán adathordozók, hanem önálló viselkedéssel rendelkező objektumok lesznek. Az üzleti logika ilyesfajta elhelyezéséből több előnyünk is származik. Az elkészült EO-kat például újra felhasználhatjuk a szakterületnek megfelelő összes alkalmazásunkban. Adott feladathoz például készíthetünk egy HTML felületű alkalmazást, és egy asztali Java adminisztrációs alkalmazást. Természetesen mindkettőben ugyanazon EO-kat használva. Ha az üzleti logikában később változtatni kell, akkor a karbantartás is könnyen elvégezhető, hiszen elég az adott EO-ban végrehajtani a változást, majd az összes őt használó alkalmazás automatikusan átveszi azt.

Érdekes, hogy az objektumorientált programozásban megszokott öröklődés az EO-k esetén is használható. Különböző lehetőségeink vannak az EO öröklődés modellezésére, köztük olyanra is, amikor az EO több adatbázistábla leképezéséből áll össze.

Adatbázisok kezelésénél kiemelt fontosságú az elsődleges és az idegen kulcsok létrehozása és kezelése. A WebObjects esetén ezzel sem kell különösebben foglalkoznunk, hiszen az EOF ezeket a feladatokat automatikusan elvégzi helyettünk. Vagy magától generálja az elsődleges kulcsot, vagy valamilyen általunk megírt egyedi kódot, illetve tárolt eljárást hív meg az előállításához.

Az EOF saját tranzakció menedzsmenttel rendelkezik. Minden általunk aktuálisan használt EO a memóriában egy elkülönített területen, az úgynevezett editing contextben (szerkesztési környezetben) található. Az editing context felelőssége, hogy folyamatosan figyelje a benne lévő EO objektumok állapotváltozásait, az új EO-k létrejöttét, a meglévők módosításait, törlését. Ha végeztünk a változtatásokkal, akkor kiadhatunk az editing context-nek egy mentés parancsot. Az EOF ekkor megvizsgálja, hogy milyen változtatások történtek az editing context-ben, és dinamikusan generált SQL parancsok segítségével átvezeti azokat az adatbázisban. Az átvezetést egyetlen adatbázis tranzakció keretében hajtja végre, így ha a tranzakció sikertelen lenne, az adatbázis nem kerül inkonzisztens állapotba. Az editing context-ben visszavonás lehetőség is van. A módosítások lépésenként visszagörgethetők a szerkesztési verem kiindulási állapotáig.

A memóriában tárolt EO objektumok bizonyos mértékben gyorsítják is az alkalmazás működését, hiszen jóval gyorsabb adatelérést biztosítanak, mint a közvetlen adatbázis hozzáférés. Néhány probléma azonban felmerül, amire a WebObjects elegáns megoldásokat ad. Ha egy objektumot leképezünk az adatbázisból, akkor ahhoz kapcsolatokon keresztül további objektumok tartozhatnak. Ezeket szintén be kell olvasni a memóriába. A gond csak ott jelentkezik, hogy ezeknek a kapcsolódó objektumoknak további kapcsolatai lehetnek, így könnyen eljuthatunk egy olyan szituációba, hogy az egész adatbázis a memóriába kerül, egy EO objektum beolvasása miatt! Ezt az EOF úgy oldja meg, hogy a kapcsolódó objektumnak csak az üres vázát hozza létre, de nem tölti fel adatokkal. Csak abban a pillanatban fordul az adatbázishoz

az EOF, ha az alkalmazás adatokat kér a kapcsolódó objektumtól (egy-a-több kapcsolat esetén az objektumoktól). Abban is biztosak lehetünk, hogy bármely, a memóriába került EO objektumból csak és kizárólag egy példány létezik. Az EOF ugyanis leellenőrzi, hogy az adott objektum nem szerepel-e már a memóriában, amikor az adatbázis sorokból objektumokat hoz létre.

#### A WebObjects Framework (WOF) és a WebObjects Builder

A WebObjects Framework felelőssége az alkalmazás felhasználói felületének biztosítása, valamint az állapotmenedzsment. A felhasználói felületet komponensekből állíthatjuk össze. Egy komponens egy teljes oldalt takarhat, de lehet annak csak egy apró része is. A komponensek tetszőleges módon egymásba ágyazhatók. A komponensek megjelenéssel és önálló viselkedéssel is rendelkeznek. A korábban említett elkülönítés itt is megjelenik, hiszen minden komponens tulajdonképpen három fájlból áll: egy HTML sablonból (ami tisztán a megjelenést definiálja), egy Java kódból (ami tisztán a viselkedést definiálja), és egy köztes deklarációs fájlból, ami az első kettő közötti kapcsolatot térképezi le. Ezzel a megoldással azt nyerjük, hogy a megjelenés bármikor megváltoztatható a kód módosítása nélkül, és viszont. Sőt, a HTML sablon elkészítését akár egy harmadik félre is rábízhatjuk, miközben mi az alkalmazás kódját írjuk.

A WebObjects rengeteg előre gyártott komponenssel rendelkezik. Ezek között egyszerűbbek és bonyolultabbak is vannak. Az igazi csemegét azonban a saját komponensek gyártása jelenti. Itt mindig arra törekszünk, hogy ezek minél több helyen és minél több alkalmazásban újrahasznosíthatóak legyenek. E célt szolgálja az is, hogy a komponensek saját API-val rendelkezhetnek. A komponens ezen keresztül adatcserét végezhet a külvilággal (rendszerint a többi komponenssel). Az adatcserét a WOF meghatározott módon és időben, automatikusan szinkronizálja.

A komponensek elkészítésében a WebObjects Builder nevű grafikus eszközt használjuk. Ez a szoftver a komponenst alkotó mindhárom fájlt egyidőben képes kezelni. Segítségével a hagyományos webszerkesztő programokhoz hasonlóan, egy grafikus felületen állíthatjuk össze a komponens HTML sablonját. Folyamatosan látjuk és szerkeszthetjük a komponens kódjának változóit és metódusait, valamint nagyon egyszerűen elvégezhetjük a HTML sablon és a kód összekapcsolását is.

Mivel a HTTP egy állapotmentes protokoll, ezért minden többfelhasználós webalkalmazásnál nagy kihívást jelent az állapot megőrzése. Ezt a WebObjects úgy oldja meg, hogy minden felhasználó számára létrehoz egy önálló Session objektumot, aminek egyedi, véletlenszerűen generált azonosítót ad. Az egyes felhasználók ez után másmás webcímet látnak, amiben a session azonosító és az éppen lekért oldal komponenseinek azonosítói vannak belekódolva. A beérkező HTTP kérésről a WebObjects így pontosan meg tudja állapítani, hogy az melyik felhasználótól és melyik komponensről érkezett. A Session objektum szabadon bővíthető. Az itt elhelyezett változó például a felhasználó kosarának tartalmát hordozhatja, ha egy e-bolt alkalmazásról van szó. Természetesen más állapotmenedzsment megoldásokat is támogat a WebObjects, köztük a session azonosítójának cookie-ban történő elhelyezését.

#### A Direct To Web Framework (D2W) és a Rule Editor

Ha az EOF-re a koronaékszer minősítést használtuk, akkor a D2W-re a „titkos fegyvert" lehetne. Ez a keretrendszer az úgynevezett Szabály Alapú Gyors Alkalmazásfejlesztés (Rule-Based Rapid Application Development – RBRAD) technológián alapszik. Ennek lényege, hogy a kliens oldali HTML felület, az EOModel és egy szabálygyűjtemény alapján önállóan, futásidőben jön létre. Megfigyelhető ugyanis az a tendencia, hogy míg az adatmodellezés és az üzleti logika lefektetése viszonylag rövid idő alatt elvégezhető, addig a hozzá tartozó HTML felület és alkalmazás logika – a modellben szereplő entitások számának arányában – sokkal lassabban. Azt mondhatjuk, hogy minden entitáshoz kell minimum egy kereső-, egy listázó-, egy megtekintő- és egy szerkesztőoldal, ráadásul két entitáshoz már ennek duplája, tízhez ennek tízszerese szükséges. A D2W lényege, hogy az entitásokkal végzendő feladatot (keresés, listázás, megtekintés, szerkesztés) tekinti állandónak, és a konkrét entitás ismeretében ehhez gyártja le ő maga az oldalakat. Ezt a gyártási folyamatot általunk megadott szabályokkal (rules) tovább finomíthatjuk. Ezek a szabályok egyszerű logikai állítások. Például: „Ha az aktuális objektum egy Személy, és a vezetéknevéről van szó, és jelenleg a megtekintés fázisában vagyunk, akkor a vezetéknév megjelenítéséhez a WOTextField nevű komponenst használd." A szabályokat a Rule Editor nevű eszközzel adhatjuk meg. A D2Wben egy tucatnyi szabály alapkiépítésben is megtalálható, ami azt eredményezi, hogy egészen rövid (akár órákban mérhető) idő alatt is egy működő webalkalmazást tudunk létrehozni. Fontos, hogy a D2W nem egy „alkalmazás-varázsló", hiszen kódot nem generál. Ugyanakkor a D2W nem is egy merev technológia, hiszen teljes egészében testreszabható.

## Összefoglalás

Szinte minden webalkalmazásnak szüksége van egy adatforrásra, illetve valamilyen webes integrációra, ami legtöbbször a HTTP kérések kezelését és a dinamikus HTML oldalak előállítását jelenti. A WebObjects objektumorientált keretrendszerei kiforrott infrastruktúrát kínálnak e két lényeges összetevő magabiztos kezelésére. Olyan szilárd alapot adnak, hogy segítségükkel akár további kód írása nélkül is létrehozhatunk egy többfelhasználós, adatbázis alapú webalkalmazást.

## Kapcsolódó linkek:

- `http://www.apple.com/webobjects`
- `http://developer.apple.com/tools/webobjects`
- `http://wodev.spearway.com`
- `http://wocode.com`
- `http://www.omnigroup.com/developer/mailinglists/webobjects-dev/`

*Szántai Károly*

*Informatikus mérnök, szaktanácsadó. A Tactus Multimédia Stúdió valamint a Martin & Charles Webstandards Consulting társtulajdonosa és munkatársa. Hobbiszinten húsz, hivatásszerűen tizenkét éve foglalkozik informatikával. Több neves hazai és nemzetközi cégnek készült multimédiás szoftver, oktatóprogram, interaktív érintőképernyős kioszkrendszer és webalkalmazás vezető fejlesztőmérnöke. Webes fejlesztései kapcsán néhány éve érdeklődésének középpontjába a webszabványok és a hozzáférhetőség került. A Martin & Charles szaktanácsadójaként az internet webszabványokra épülő minőségellenőrzési és minőségbiztosítási kérdéseivel foglalkozik.*

# Kerekasztal beszélgetés a webalkalmazások jövőjéről

A webes űrlapok 1993-ban jelentek meg, és azóta nem is változtak szinte semmit. A mai webalkalmazás igények azonban a hagyományos űrlapok által nyújtottaknál sokkal kifinomultabb interakciós lehetőségeket feltételeznek. A dokumentum alapú web egyre inkább alkalmazás kialakítási felületté alakul, amelyen az alkalmazásokat akár emberek, akár gépek számára készíthetjük. Az utóbbi esetben a különböző programozói felületek összekapcsolására kell törekednünk, a felhasználókra gondolva viszont megszokott, használható megoldásokat kell nyújtani.

Ezen a kerekasztal beszélgetésen célunk annak körbejárása, hogy milyen kliens oldali támogatás illetve milyen szerver oldali elemek állnak ma illetve a jövőben rendelkezésünkre, hogy a weben is az asztali programok élményét nyújtó alkalmazásokat fejleszthessünk ki, a felhasználók megelégedésére. Ennek érdekében természetesen érdekes a webes ajánlásokat kidolgozók szemlélete, gondolva az XForms vagy Web Forms 2.0 ajánlásokra. De mindezek a megvalósító böngésző és más webkliens program gyártó cégek, közösségek aktív részvétele nélkül elképzelhetetlenek. A ma szélsebesen terjedő AJAX technika szerver oldali támogatásának megvalósítása is sok esetben még a jövő zenéje, de ennek hibái, rossz alkalmazásai is mutatkoznak már. Jó kérdés, hogy merre mutat a jövő, az ezen munkálkodók milyen megoldásokat szeretnének bevezetni, milyen új kihivásokat látnak az AJAX-szal kiegészített űrlapokon túl.

Beszélgetésünkön egy témavezető és néhány meghívott vendég beszélget a felmerülő kérdésekről, megvillantva saját álláspontjaikat, elképzeléseiket, vízióikat. A beszélgetés iránt érdeklődő hallgatók is aktív részesévé válhatnak az eszmecserének.

*A Konferencia Szervezői*

# Magas rendelkezésre állású rendszerek a gyakorlatban

Akármilyen szolgáltatási vagy termelési tevékenységet is végez egy cég, egyre nagyobb mértékben támaszkodik az informatikára és így válik függővé belső informatikai rendszerének működőképességétől.

Ezek gyakran a cég legfontosabb vagy akár a teljes működését szabályozzák, így az informatikai rendszerekben bekövetkező bármilyen szolgáltatás kiesés vagy adatvesztés komoly anyagi következményekkel járhat. Az informatikai eszközök ugyanakkor egyre nagyobb versenyben kerülnek kifejlesztésre, ami gyakran együtt jár azzal, hogy a régebbi típusoknál ugyan lényegesen gyorsabbak, nagyobb kapacitásúak és többet tudnak (amire természetesen szükség is van), de ezzel együtt egyre megbízhatatlanabbak. Ráadásul minél több eszközből áll a rendszerünk, annál több ponton következhetnek be problémák.

Ezzel együtt kell élni, viszont fel lehet rá készülni. Ebben segítenek azok a technológiák és az a szemlélet, amelyet összefoglalóan a HA – (a magas rendelkezésre állás, angolul a **High Availability** rövidítése) szóval szoktunk emlegetni.

Az előadás kitér a hamis biztonságérzetet keltő megoldásokra: biztonsági mentések, osztott adatbázis, cluster struktúrák. A HA rendszerek lényegi tárgyalása során kiderül, hogy miként mérhetjük fel cégünket üzemeltetésbiztonsági szempontból, a szolgáltatás-szemléletet összehasonlítjuk a szerver-szemléletű megközelítéssel.

Konkrét gyakorlati példán tekintünk át egy megvalósítást, mely napi ötmillió oldal kiszolgálását végzi 99,9%-os vállalt rendelkezésre állási szinten. Szó esik a konfigurációs sablonokról, és a HA rendszerek másodlagos hasznáról a skálázhatóság és terheléselosztás terén.

Sok cégnél az üzemeltetés-biztonság mindössze annyit jelent, hogy „mentik" a legfontosabb adatokat, és baj esetén ezekre számítanak mint mentőövre.

Ez igen fontos, de üzemeltetés-biztonsági (rendelkezésre állási) szempontból ez bizony az esetek többségében elégtelen megoldás. Ha elromlik egy szerver, akkor annak javítása vagy cseréje alatt elérhetetlen a rajta futtatott szolgáltatás. Ilyenkor általában installálni kell az új gépet, de már önmagában a nagy mennyiségű adat visszatöltése is igen időigényes folyamat.

Ezen kívül még nem beszéltünk finom-beállításokról, a jogosultságok visszaállításának gyakori problémájáról. Hab lehet a tortán az alkalmazás esetleges hardverkulcsos védelme, ami tovább lassítja ezeket a folyamatokat és az üzemkiesés idejét napokra növelheti.

Annyit még meg kell említeni, hogy a jó és használható biztonsági mentéseknek és az előállításukat végző folyamatoknak sok, nemegyszer bonyolult kritériumnak is meg kell felelnie. Futó rendszereken végzett mentések esetén biztosítani kell a mentett adatok konzisztenciáját. Egyes operációs rendszerekhez adott backup szoftverek illetve egyes kereskedelmi termékek bizony hiányosságokkal és komoly hibákkal sújtják használóikat. Végezetül a mentés önmagában sohasem elégséges megoldás, folyamatosan ellenőriznünk kell annak visszaállíthatóságát is!

## Az üzemeltetés biztonsága

Az informatikusok gyakran úgy próbálják kivédeni a lehetséges kiesést, hogy az adott szerver mellé beállítanak még egyet. Ez már előrelépés ugyan, de egyáltalán nem elégséges megoldás és szintén hamis biztonságérzetet ad.

Ennek oka a következő: magának a kulcsfontosságú szolgáltatásnak a kliensek általi elérhetősége más szolgáltatások megbízható működésétől is függ. Nem elég az adott szervert tartalékgéppel kiváltani, hiszen ha a fő szolgáltatás által igényelt egyéb szolgáltatásokban következik be a probléma, akkor hiába van a fő szolgáltatás tartalékszervere, maga a szolgáltatás mégis elérhetetlenné válik.

Példán keresztül érzékeltetve ez a következőket jelenti. Képzeletbeli cégünk, egy bank, a központi szerverén tárolja ügyfelei adatait, tehát létkérdés (kellene legyen), hogy ezek folyamatosan rendelkezésre álljanak a bank fiókjai vagy a homebanking rendszerek számára.

Nyílván ez a központi adatbázisszerver szünetmentes tápegységen a legmodernebben felszerelt és legjobban védett szerverterem mélyén dolgozik.

Vásároltak hozzá egy második, úgynevezett „tartalék" gépet is. Ha az első gép meghibásodna, akkor ezt a tartalékgépet, mint hideg vagy meleg tartalékot használják fel. A valamilyen módon clusterbe kötött tartalékgép az elsődleges gép feladatát azonnal átvenné, és a legkisebb fennakadás nélkül folyhatna tovább a megszokott mederben a munka.

Most tegyük fel, hogy a hálózat kliensgépei a hálózaton keresztül kapják az IP címeket a központi DHCP szervertől. Ha a DHCP szerver mondja fel a szolgálatot, akkor a hálózati kliensek ugyanúgy nem érik el többé a központi adatbázisszervert. A szolgáltatás elérhetetlenné válik, ilyenkor mit sem ér a központi adatbázis-szerver megkettőzése.

Talán rövid gondolkodás után vagy anélkül is választhatnánk ennek a megkettőzését is, és így tovább. (Valaho-

gyan előállítjuk ezek közös adatrendszerét, ami nem is olyan egyszerű feladat.)

Sajnos ez sem elég. Gondoljunk csak bele, hogy ha a „szuperbiztos" szerver termünkben valami miatt elromlik az egyik router, akkor máris megállt a szerverek közötti teljes adatforgalom. Minden áll, amíg az új routert üzembe nem helyezik és a szükséges konfiguráló programokat be nem töltik rá.

Mindezen problémákat tehát folyamatosan a legdrágább eszközök (szerverek, routerek) megkettőzésével oldottuk meg és még nem vagyunk készen, hiszen az egyes eszközök kiesésekor (pl. router csere esetén) a szolgáltatás igenis leáll, márpedig pontosan ezt akartuk elkerülni.

## Mi tehát a megoldás?

Természetesen bizonyos eszközökből sajnos tényleg többet kell vásárolnunk, mint azt a szolgáltatás maga önmagában indokolná, ám ezek ügyes szervezésével valamint az egész szolgáltatási struktúra (a szolgáltatások egymásra épülésének) feltérképezésével ez nem jelenti az eszközök automatikus megduplázását.

Jól tesszük, ha nemcsak mint „vasat" vásároljuk meg ezeket az eszközöket, hanem az egész rendszernek a működését átgondolva a szerverek közötti szolgáltatás-átvételt teljesen, vagy majdnem teljesen automatizáljuk. Ennek eredményeként létrejön tehát egy olyan hálózati szolgáltatás, ami a kliensek számára tényleg transzparensen átterhelhető a szerverek között. Az így kialakított és okosan megszervezett rendszert hívjuk HA rendszernek.

## Hogyan tervezzünk, készítsünk ilyen rendszert?

Kezdeti lépésként meg kell határozzuk, hogy mik a legfontosabb szolgáltatások és ezektől a szolgáltatásoktól milyen rendelkezésre állási szintet várunk el. Ezzel együtt fel kell térképezzük azt is, hogy az egyes szolgáltatások miként épülnek egymásra, ebből végső soron meghatározzuk, hogy a kritikus szolgáltatások milyen más „kisebb" szolgáltatásoktól függnek.

Nyílván ezek együttesen lesznek felelősek a „fő" szolgáltatásokért, tehát amilyen szintet elvárunk a „fő" szolgáltatástól, olyan szintet várunk el az őt kiszolgáló egyes kisebb szolgáltatásoktól is.

Egy ismerős gyakorlati példában legyen a fő szolgáltatás egy webes alkalmazás futtatása.

Nyílván ez több kisebb szolgáltatás megbízható működését igényli maga alatt. Ezek a teljesség igénye nélkül:

- PHP
- Apache
- MySQL
- BIND
- hálózat és ennek elemei stb.

Ezek mindegyikének a fő szolgáltatástól elvárt szinten kell működnie ahhoz, hogy a fő szolgáltatásunk, a webes alkalmazás, teljes mértékben hozzáférhető legyen a felhasználó számára.

A felkészülés abból áll, hogy átvizsgáljuk a teljes rendszer elemeit, és megállapítjuk, hogy melyek a legkritikusabb pontok, hogyan lehet felkészülni a meghibásodásokra, amelyek önmagán a meghibásodáson túl milyen egyéb következményekkel járhatnak.

Mivel ennek a rendszernek tényleges bemutatása igencsak nagy terjedelemmel bírna, egy webes működő rendszer elemekre bontását az előadáson ismertetem.

Végezetül még egy kedvcsináló gondolat: Az átgondolt szervezés megalapozta HA rendszer kialakításával együtt gyakran még egy fontos problémára is megoldás születhet: a skálázhatóságra. Ez azt jelenti, hogy mivel a szolgáltatás nem szerverhez kötött, így a terhelést szét lehet osztani az üzemben lévő gépek között.

Ha például egy szolgáltatás öt gépen üzemel, és szükség van további 10% teljesítménynövelésre, akkor nem kell a régi szervereket kiselejtezni és helyette nagyobb(ak)at vásárolni, hanem még egy gép üzembe helyezésével a teljesítményt 20%-kal növelhetjük, rögtön 10% kapacitástöbbletet hagyva a rendszerben.

*Fischer Zsolt*

*A Publishing Factory Kft. ügyvezetője. A magas rendelkezésre állású rendszerek és a biztonságtechnika megszállottja. Legnagyobb ügyfelei között van a Tv2, a Nemzeti Sport, a Blikk, a Honvédelmi Minisztérium, az Allianz Biztosító, nagyobb bankok és egyetemek.*

# Szemantikus Web: egy rövid bevezetés

## Bevezetés

A Weben lévő információk típusa igen nagy változatosságot mutat. A klasszikus szöveges információk mellett találhatunk fényképeket, kettő és három dimenziós grafikát, videót, zenét, stb. A keresett információk (például egy utazáshoz szükséges tények összessége) sokszor különböző weblapokon találhatók csak meg (egy részük a légitársaság lapján, mások egy adott szálloda saját lapján, stb.). Ez utóbbi esetben a problémát nehezítheti, ha a különböző lapok különböző terminológiákat használnak ugyanarra a fogalomra. És persze ne felejtsük el a természetes nyelvek (magyar, angol, kínai) gazdagságát sem. Mindezek ellenére a Web csodálatos módon működik. Ennek egyik magyarázata, hogy mi, emberek, rendkívül könnyen tudunk inkoherens vagy nem teljes információkat kombinálni. Általában megértjük egy fénykép tartalmát (hacsak nincsenek látási problémáink), könnyedén áthidalunk terminológiai különbségeket, félkész információkat tudunk teljessé tenni, és sokszor különböző nyelveken is beszélünk.

Ez azonban a számítógépekre nem igaz. Mint ahogy ezt mindannyian tudjuk, a számítógépek alapvetően buták. Ahhoz, hogy intelligens programok (ágensek) működni tudjanak, a rendelkezésekre álló adatokról, az adatok közötti összefüggésekről, azok esetleges ekvivalenciájáról, nyelvéről, stb. sokkal precízebb információkat kell biztosítani. Csak ezek alapján lehet elvárni, hogy az ágensek képesek legyenek kihasználni azt a fantasztikus mennyiségű adatot, amely ma a világhálón rendelkezése áll. Márpedig az ágensek már most rengetegen vannak, és az ezekre alapozott alkalmazások száma, és főleg az irántuk támasztott felhasználói igények állandóan növekednek.

Íme néhány példa lehetséges alkalmazásokra, melyek ma még vagy csak papíron léteznek vagy, habár léteznek, jelentős problémákkal küszködnek. M. Dertouzos, az MIT Számítástudományi Laboratóriumának valamikori vezetője már 1995-ben (vagyis, ami a Webet illeti, az ősidőkben…) több izgalmas alkalmazást ír le az „Unfinished Revolution" című könyvében [1] (a könyv egyébként magyarul is megjelent „Félkész forradalom" címmel [2]), mint például egy automatizált repülőjegy foglaló rendszert, amely figyelembe veszi a különböző légitársaságok információit, az utas kívánságait stb. Hasonló alkalmazásokat ír le T. Berners-Lee szerzőtársaival egy sokszor idézett cikkben [3], illetve könyvében [4]. Szeredi Péter és kollégái, egy nemrégiben megjelent könyvükben [5], hosszan elemzik a keresőrendszerek problémáit; hasonló problémák adódnak a webszolgáltatások tömeges elterjedésének esetén (például „találd meg a legelegánsabb differenciálegyenlet megoldó webszolgáltatást, és mondd meg, hogy hogyan kombinálhatom ezt a grafikus kijelző szolgáltatással"). Rosszul, vagy egyáltalán nem látó felhasználóknak egy kép részletesebb, szövegalapú leírása elérhetővé tehet egy, a Weben lévő grafikát [6]. Megint más jellegű probléma vetődik fel mikor nagy cégek összeolvadnak (például mikor a HP felvásárolta a Compaq céget): milyen módon kombinálható a két cég adatbázisa, amelyek végül is hasonló (pénzügyi, technikai, jogi, személyzeti stb.) információkat tartalmaznak de különböző konvenciókkal, terminológiákkal, szokásokkal. És a sort az intelligens ágensek alkalmazására lehetne még vég nélkül folytatni.

Mindezen példák egyik közös jellemzője, hogy egy alkalmazáson belül kombinálni kell, méghozzá koherens módon, Weben lévő adatokat. Itt az „adat" lehet egyszerűen a Weben lévő fájl (például egy online címlista), elérhető adat (például egy adatbázisban lévő rekord elérése valamilyen alkalmazási felületen keresztül), vagy *meta-adat*, vagyis adat az adatról (például egy könyvtári katalóguscédula elektronikus változata). A Szemantikus Web ezt a problematikát próbálja megoldani. Ha röviden akarjuk jellemezni, akkor azt mondhatjuk, hogy a *Szemantikus Web célja egy olyan infrastruktúra létrehozása, amely lehetővé teszi a Weben lévő adatok integrálását, a közöttük levő kapcsolatok definiálását és jellemzését, illetve az adatok értelmezését.*[1]

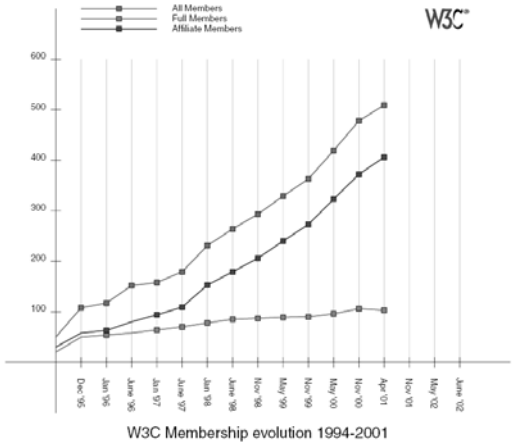
Ahhoz, hogy ezt az infrastruktúrát létrehozzuk, néhány fontos építőelemre van szükség. Ezek a következők (zárójelben azon W3C specifikációk, amelyek már léteznek vagy fejlesztés alatt vannak, és amelyek az adott építőelemhez alapvető szerepet játszanak):

1. Az adatokat egyértelműen meg kell „címezni" a hálón, vagyis el kell őket nevezni (URI=URL+URN)
2. Szükség van egy precíz adatmodellre, amely formális keretet ad az adatok egymáshoz való kapcsolására, azok leírására, meta-adatok definiálására (RDF [7])
3. Az adatokat (az adatmodell alapján) el kell tudni érni, le kell tudni kérdezni (SPARQL [8])
4. Az adatok leírására szolgáló terminológiát kell tudni definiálni (RDFS [9], OWL [10], SKOS [11])
5. Az adatokon, illetve azok leírásán, logikai következtetéseket kell tudni levonni (OWL [10], RIF [12])

Az URI használata természetesen nem specifikus a Szemantikus Webre; a Szemantikus Web itt felhasznál egy elterjedt és jól definiált technológiát. A többi (RDF, OWL stb.) a Szemantikus Webre lett kifejlesztve, illetve áll fejlesztés alatt: vannak közöttük már szabványosított specifikációk (RDF, OWL), vannak, amelyek már nagyon közel állnak egy végleges formátumhoz (SKOS, SPARQL), és megint mások még csak a fejlesztés legelején tartanak (RIF). A továbbiakban mindezekről a technológiákról adok egy nagyon rövid áttekintést.

## RDF (Resource Description Framework, Erőforrás Leíró Nyelv)

Az RDF a Szemantikus Web alapköve és legrégebbi specifikációja. Legegyszerűbb egy példán keresztül bemutatni.




*1. Ábra: egy egyszerű business grafika*

Az első ábra egy jellegzetes business grafikát mutat, amelyhez hasonlót igen sokat lehet a Weben találni. A tartalma világos, látszólag nem igényel további magyarázatot. A probléma akkor merül fel, ha ezt az ábrát egy vak Web felhasználó számára akarjuk elérhetővé tenni.

Mint ahogy erre már utaltunk, a megoldás az adat szöveges leírása lehetne. De ahelyett, hogy ezt a specifikus ábrát írnánk le részletesen, ki szeretnénk fejleszteni egy olyan intelligens programot, amely képes egy meta-adatot automatikusan értelmezni, és abból egy olvasható, vagyis, egy vak felhasználó számára felolvasható szöveget létrehozni. Ez azt jelentené, hogy az ábra létrehozójának csak néhány, jól definiált meta-adatot kell az ábrához hozzárendelni, a többi a program dolga. Mivel a Weben vagyunk, feltételezhetjük, hogy az ábra SVG-ben van, vagyis egy grafikus XML leírásban; ennek annyi itt a jelentősége, hogy a grafika minden eleme megcímezhető egy URI segítségével. (Az alkalmazás egy részletesebb leírása megtalálható [6]-ban).

Milyen állításokat tartalmazna egy meta-adat ebben az esetben? Íme néhány példa:

- „A teljes ábra egy business grafika"
- „A business grafika típusa vonalgrafika"
- „Ezen a címen az ábra képaláírása található"
- „A képaláírás egyben egy link is a Weben"
- „A képaláírás erre és erre a URI-re utal"

Az összes példa egy hármas tagolást mutat. Valamiről állítunk valamit („a teljes ábra", „egy adott cím", „a képaláírás") és ezt összekapcsoljuk egy egyszerű szövegelemmel („vonalgrafika") vagy egy külső adattal („URI"). Maga a kapcsolat is jelentőséggel, „szemantikával" bír („típusa", „utal"). Természetesen egy alkalmazásban bonyolultabb állítások is szükségesek lehetnek, de a gyakorlat azt mutatja, hogy ezek is lebonthatók ilyen hármas tagolású alapelemekre.

Az RDF az ilyen „hármasok" matematikai modellje. Egy RDF hármas alanyból, állítmányból (vagy tulajdonságból, mindkét fogalom használatos), és tárgyból áll[2]; ebben a megközelítésben például a második példamondat a következőképpen absztrahálható:

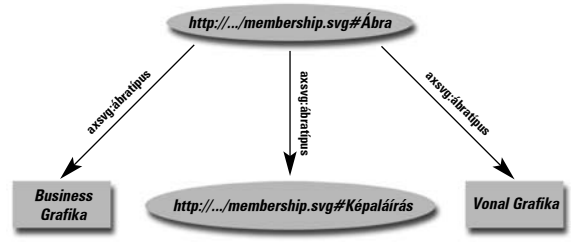
```
(A business grafika címe,típus,"vonalgrafika")
```

A hármas mindhárom eleme egy URI (amely, általánosságban, valamilyen „erőforrásra" utal[3] mely lehet egy adatforrás a Weben, egy email cím stb.) vagy, a tárgy esetében egy szövegelem. Vagyis, ha az ábrához és a „típus" tulajdonsághoz egy-egy jól meghatározott URI-t rendelünk hozzá, akkor a hármas precízebben leírható a következő módon:

```
(http://.../Ábra.svg,http://.../típus,"vonalgrafika")
```

Az RDF (Resource Description Framework, vagyis Erőforrás Leíró Nyelv) az ilyen hármasok (vagy állítások) összességét írja le. Az RDF specifikáció tartalmaz néhány további fogalmat, amelyekre most nem tudunk kitérni, de ezek nem igazán fontosak az alapfogalmak megértéséhez.

Az RDF állítások összességét tekinthetjük egy irányított, címkézett gráfnak is. A gráf csomópontjai az alanyok és tárgyak (vagyis a rájuk hivatkozó URI-k); a gráf élei pedig az állítmányok (tulajdonságok), ahol is az állítmány neve (vagyis az azt leíró URI) adja az él címkéjét. Az RDF állítások gráfban való ábrázolása nagyon áttekinthető; érdemes gráfokban „gondolkozni" az RDF állítások létrehozásakor és alkalmazásakor. A probléma persze az, hogy a gráfoknak kell hogy legyen egy géppel olvasható formája is: az idők folyamán több, egymással ekvivalens szöveges reprezentációforma is született. A W3C által definiált RDF/XML formátum XML alapú, de létezik más, nem XML alapú szintakszis is. És ez rendben is van: nem a szintaxis a lényeg, hanem a mögötte álló adatmodell.




*2. Ábra: egy egyszerű RDF gráf*

A második ábra egy egyszerű RDF gráfot mutat, amely három állításból áll: lényegében az első háromból a fenti listán. (A tulajdonságokat az egyszerűség kedvéért névterekkel adtuk meg). A három állítás XML változata a következőképpen néz ki:

```
<rdf:Description rdf:about="http://...#Ábra">
  <axsvg:ábratípus xml:lang="hu">Business
grafika</axsvg:ábratípus>
  <axsvg:aláírás
rdf:resource="http://...#Képaláírás"/>
  <axsvg:grafikatípus xml:lang="hu">Vonal
grafika</axsvg:grafikatípus>
</rdf:Description>
```

A szintaktikus részletek itt most nem érdekesek, a példa remélhetőleg, magáért beszél.

A modellben kitüntetett szerep jut az URI-nak. Az URI-k használata biztosítja, hogy az RDF állítások *bármilyen* adatra vonatkozhatnak a Weben. Így például egy adott grafikáról nemcsak a grafika létrehozója publikálhat meta-adatot, hanem bárki. De talán még fontosabb, hogy az RDF hármasok nemcsak meta-adatokra használhatók, hiszen például az alany lehet bármilyen adat, amelynek URI címe van, vagyis amely a Weben „van". RDF hármasokkal a Weben lévő adatokat lehet egymáshoz rendelni, közöttük lévő kapcsolatokat definiálni és jellemezni, hűen a bevezetésben leírt célokhoz. Bizonyos értelemben azt is lehet mondani, hogy az URI-k használata „ágyazza" az RDF hármasokat a Webbe.

Az URI-k használatának van egy további, rendkívül jelentős következménye. Igen egyszerűvé válik különböző forrásokból származó RDF hármasok összekombinálása egy alkalmazáson belül. Ha két, egymástól különböző hármas halmaz azonos csomópontokat vagy éleket tartalmaz (merthogy ugyanazokat az URI-ket használják), akkor ezen csomópontok az RDF szempontjából *ugyanazok*. Magyarán, a gráfokat triviális módon egybe lehet olvasztani. Vagyis egy alkalmazás minden további nélkül felhasználhat és összekombinálhat Weben lévő meta-adatokat, RDF hármasokat, függetlenül attól, hogy azokat ki hozta létre[4]. Végeredményben ez annak az analógja, amikor az emberi felhasználó koncepcionálisan összekombinál Weben lévő információkat. Nem véletlen, hogy gyakorlati alkalmazások manapság rutinszerűen használnak több tízezer, százezer, vagy akár millió hármasból álló adatokat, amelyeket gyakran így kombinálnak össze.

Az RDF specifikációnak már két generációja is létezik, a legutóbbi 2004-ben jelent meg [7]. Mivel az első verzió még a 90-es évekből származik, bőven volt idő RDF programrendszerek, programozási környezetek kifejlesztésére. Ma lényegében az a helyzet, hogy szinte minden programnyelvhez létezik legalább egy RDF programozási környezet (a berlini Frei Universität által fenntartott Web lap [13] gazdag információforrás a rendelkezésre álló eszközöket illetően). E programozási környezetek tartalmaznak egy vagy több szövegelemzőt az RDF adatok beolvasására, többé-kevésbé szofisztikált módon tárolják a hármasokat, amelyeket le lehet kérdezni, ki lehet törölni, új hármasokat lehet létrehozni, stb. A programozási környezetek döntő többsége nyílt forráskódú fejlesztés.

Mint már fentebb jeleztük, gyakorlati alkalmazások adott esetben milliós nagyságrendben kezelnek RDF hármasokat. Ez több problémát is felvet. Egyrészt szükség van adatbázisokra amelyek képesek ilyen nagyságrendeket kezelni, hiszen ilyen hármas halmazokat nem lehet egyszerűen a memóriában tárolni. Másrészt szükség van valamiféle lekérdező nyelvre bonyolultabb információk lekérdezésére.

Az első probléma inkább gyakorlati, és ma már több RDF adatbázis rendszer létezik, mind a szabad szoftverek piacán, mind üzleti termékként. Így például az Oracle legfrissebb verziója (10*g*) képes RDF adatokat tárolni, és egy RDF absztrakciós felületet is nyújt a felhasználónak. Mások egy teljesen új fejlesztéssel alakítottak ki adatbázis rendszereket kizárólagosan az RDF céljára.

A lekérdező nyelv kérdése nyilvánvalóan más jellegű. Általánosságban azt lehet mondani, hogy egy adatbázis-lekérdező nyelv ahhoz az adatmodellhez kapcsolódik, amelynek az adatait el kívánjuk érni. Így például az SQL a relációs adatmodellhez kapcsolódik, míg az XML Query az XML adatmodelljéhez. A SPARQL [8] ezeknek az analógja az RDF világban: ez az RDF adatmodelljéhez kapcsolódó lekérdező nyelv[5].

A SPARQL alapelve voltaképpen igen egyszerű: mivel az RDF hármasok halmaza egy gráf, egy SPARQL lekérdezés egy *gráfmintát* tartalmaz. Egy gráfminta olyan (RDF) hármasok halmaza, amelyek változókat is tartalmazhatnak az alany, az állítmány vagy a tárgy helyén. A lekérdezés során a rendszer a változókat behelyettesíti a hármas halmazban lévő URI-kal vagy szövegelemekkel, és ellenőrzi, hogy az így kapott gráf az eredeti RDF gráf részgráfja-e. Ha igen, a változó behelyettesítés sikeres, és a változók a lekérdezés egyik lehetséges eredményét adják. Így például definiálhatjuk a következő SPARQL lekérdezést:

```
SELECT ?u, ?v
WHERE { ?g axsvg:aláíras ?u.
        ?g axsvg:grafikatípus ?v.
      }
```

Ha ezt a lekérdezést a 2. ábrán lévő gráfra alkalmazzuk, akkor a

```
?u = http://.../membership.svg#Képaláírás
?v = "Vonal grafika"
```

hozzárendelés egy lehetséges (ebben az esetben az egyedül lehetséges) válasz a lekérdezésre. Természetesen ez egy leegyszerűsített példa; a gyakorlatban mind a hármas halmazok, mind a lekérdezések sokkal bonyolultabbak. Érdemes megjegyezni, hogy bár a SPARQL fejlesztése még nem záródott le, a [13]-ban felsorolt programozási környezetek egy jelentős része már implementálta a SPARQL valamely változatát (vagyis például a memóriában tárolt hármas halmazokat is lehet SPARQL lekérdezésekkel kezelni), és az RDF adatbázisok is kezdik beépíteni a rendszerükbe. Ez jól mutatja a SPARQL jelentőségét.

## Ontológiák

Az RDF, illetve újabban az RDF+SPARQL kombináció, már érdekes és fontos alkalmazási lehetőségeket rejt magában. A teljes Szemantikus Web azonban, ahogy ezt a bevezetésben leírtuk, ennél többet igényel. Két fő terület van, amely további technológiákat kíván, nevezetesen:

- Milyen terminológiát használhatunk egy alkalmazáson belül? A fenti példában használtuk az `axsvg:aláírás` tulajdonságot; felmerül a kérdés, hogy jogos-e a használata, ha igen, milyen körülmények között (pl. mely

[4] Aki megpróbált már különböző forrásokból származó XML hierarchiákat egy alkalmazásban összekombinálni, az tudja, hogy XML esetén ez milyen bonyolult tud lenni.

[5] Ki kell hangsúlyozni, hogy míg az RDF egy lezárt, publikált specifikáció, a SPARQL még fejlesztés alatt áll és 2006 vége előtt nem valószínű, hogy be lenne fejezve. Vagyis a részletek még itt-ott változhatnak.

erőforrásokra vonatkoztatva)? Egyáltalán, hogyan lehet osztályozni a rendelkezésre álló erőforrásokat?
- Hogyan jellemezhetjük az erőforrások közötti logikai kapcsolatokat? Mikor mondhatjuk, hogy két erőforrás (például két különböző nevű tulajdonság) ugyanazon fogalomra utal? Lehet-e a tulajdonságok logikai viselkedését jellemezni (például hogy szimmetrikus vagy tranzitív, hogy valójában egy függvény stb.)?

Világos, hogy ezek a kérdések felmerülnek minden komolyabb Szemantikus Web alkalmazás esetén, de az is világos, hogy ezen kérdések megválaszolására egy bonyolultabb rendszer kell, amely ki kell hogy egészítse az RDF egyszerű adatmodelljét. Az is viszonylag hamar kiderül, hogy az összes felmerülő kérdés megválaszolása rendkívül bonyolulttá válhat, ezért érdemes technológiák egy családját kifejleszteni, amelyek közül a felhasználó igénye szerint választhat. Így fejlődött ki a W3C keretében az RDFS, az OWL és a SKOS, valamint indult el 2005 legvégén az RIF fejlesztése. Ezek közül az RDFS és az OWL technológiák a legkiforrottabbak, és a továbbiakban elsősorban ezekre fogunk koncentrálni.

Az alapvető megközelítés fogalomjegyzékek, ontológiák használata, illetve a tradicionális ontológiai módszerek adaptálása RDF környezetekre. Tekintsük a következő egyszerű példát egy fogalomjegyzékre:

- létezik az „emlős" fogalma
- minden „delfin" egyben „emlős" is
- „Flipper" egy delfin

Ebben a fogalomrendszerben, az „emlős" egy osztály, amelyhez különböző egyedek tartozhatnak; az „emlős" osztályhoz való tartozás „jellemzi" az egyedeket. A második állítás egy logikai kapcsolatot teremt két osztály között, hiszen azt jelenti, hogy minden egyed, amely az egyik osztályhoz tartozik, automatikusan a másikhoz is tartozik. Végül a harmadik állítás egy egyed hovatartozását definiálja.

Az RDFS (RDF Vocabulary Description Language, magyarul RDF Szókészlet Leíró Nyelv) [9] egyik célja az ilyen fogalomrendszerek absztrahálása. Az RDFS bevezeti az erőforrások *osztályának* fogalmát, mint „egyedek lehetséges összességét", továbbá a *típus* fogalmát, vagyis egy egyednek egy osztályhoz való tartozásának jelzését. Egy tetszőleges erőforrás típusa tehát egyszerűen azt jelzi, hogy az erőforrás egy adott osztályhoz tartozik. Végül bevezeti az alosztály relációt két osztály között, az értelemszerű definícióval.

Első pillanatra talán meglepőnek tűnhet, hogy az RDFS fogalmak és definíciók *RDF-ben vannak megfogalmazva*. Vagyis az általános osztály nem más mint egy speciális erőforrás, amelynek egy meghatározott URI-je van; a típus és az alosztály pedig szintén egy-egy (RDF) tulajdonság jól meghatározott és szabványosított szemantikával. Emlékezzünk arra a tényre, hogy az RDF hármasok *bármilyen* URI-re, vagyis bármilyen erőforrásra vonatkozhatnak, másszóval az RDFS definíciókhoz nem kell az RDF adatmodelljéből kilépnünk. Mindezek előrebocsájtásával a fenti három állítást a következőképpen fordíthatjuk le RDF-re:

```
(http://....#emlős, rdf:type, rdfs:Class)
(http://....#delfin, rdf:type, rdfs:Class)
(http://....#delfin, rdfs:subClassOf,
➥ http://....#emlős)
(http://....#Flipper, rdf:type,
➥ http://....#delfin)
```

Itt is, mint az előző példákban, névtereket is használtunk a URI-k rövidítésére; az egyedüli különbség az, hogy az rdf és rdfs névtereknek megfelelő URI-k ebben az esetben a W3C szemantikára vonatkoznak. Az első két hármas azt jelenti, hogy az „emlős" és „delfin" egyedek önmagukban osztályok (vagyis az absztrakt RDFS osztály egyedei), a harmadik hármas az alosztály kapcsolatot hozza létre a két osztály között, míg a negyedik hármas a Flipper, mint egyed hovatartozását definiálja.

A fenti hármasok egy nagyon egyszerű következtetési sémát is lehetővé tesznek. Valóban, ha egy RDF rendszer a fenti négy hármast kapja bemenő adatként, és a rendszer „érti" az RDFS szemantikát is, akkor a rendszer *következtetheti* a következő hármast is:

```
(http://....#Flipper, rdf:type, http://....#emlős)
```

Vagyis a rendszer képes logikai következtetéseket levonni a rendelkezésére álló adatokból. Habár természetesen ez a példa a trivialitás határát súrolja, más esetekben a következtetések sokkal bonyolultabbak és hasonló alapokon nyugszanak.

Az osztálydefiníciók a tulajdonságok jellemzését is lehetővé teszik. Az osztályok ismeretében lehetővé válik egy tulajdonság, mint bináris reláció értéktartományának és érvényességi körének (másszóval értékkészletének) definiálása: pontosan definiálni lehet, hogy az állítmány alanya vagy tárgya mely osztályokhoz tartozhat. A fenti példára visszatérve: az RDFS dokumentum természetesen, meghatározza a saját maga által definiált tulajdonságok értéktartományát és értékkészletét, és nyilvánvaló módon az `rdfs:subClassOf` állítmány alanya egy osztály egyed kell hogy legyen. Ezért voltaképpen a második hármas felesleges, hiszen a harmadikból következik. (Természetesen használata attól még nem hiba, csak redundáns!) Végül: az RDFS lehetőséget ad egy al-tulajdonság definiálására is: ha P a Q al-tulajdonsága, és fennáll (x, P, y) akkor fennáll (x, Q, y) is.

Miként az RDF-re, az RDFS-re is igaz, hogy tartalmaz még további konstrukciókat, amelyre itt nem tudunk kitérni. De remélhetőleg az eddigiek már adtak egy képet arról, hogy az RDF+RDFS kombináció mire képes. Meg kell jegyezni, hogy az RDF+RDFS formális szemantikája (például annak precíz leírása, hogy egy RDFS-t értő rendszer milyen következtetéseket vonhat le a rendelkezésre álló bemeneti adatokból) messzemenően nem triviális, és megértése alapos modellelméleti tudást igényel. De a felhasználók számára ez nem feltétlenül fontos, ugyanúgy ahogy a felhasználók döntő többsége nincs tisztában mondjuk az általuk használt programozási nyelv formális szemantikájával.

Az RDFS az ontológiák definiálásának egy szintjét jelenti; azt mondhatnánk, hogy az alapszintet. Az RDFS definíciói annyira alapvetőek, hogy a gyakorlatban az RDFS nehezen választható el az alap RDF modelljétől. Így például szinte nincs olyan komolyabb RDF alkalmazás, amely ne használná az osztály és a típus fogalmát, még akkor is, ha a programozási rendszer nem „érti" az RDFS precíz szemantikáját. Az RDFS azonban alapszint, és a komplikál-

tabb definíciók esetén szükség van a következő szintre; itt lép be az OWL specifikáció (Web Ontology Language, magyarul Web Ontológia Nyelv) [10]. Lássuk mit ad az OWL mint „pluszt" az RDFS-hez képest!

Az első érdekes terület az osztályok fogalma. Mint fentebb láttuk, RDFS-ben osztályokat lehet definiálni (lényegében elnevezni), és az osztályok között alosztálykapcsolatokat lehet létrehozni. Mást azonban nemigen. Az OWL az RDFS-re épül, vagyis mindazt, amit RDFS-ben lehet, azt OWL-ben is lehet, de a OWL-ben osztályokat lehet *konstruálni* is. Osztályokat lehet más osztályok uniójaként, metszeteként, komplemenseként definiálni; lényegében rendelkezésre állnak az alap halmazelméleti műveletek. Lehetséges az osztályok egyedeinek explicit felsorolása; és osztályokat lehet a tulajdonságok segítségével is definiálni.

Érdemes kitérni ez utóbbi alternatívára, mert ez a legbonyolultabb, és talán informatikusok számára a legkevésbé természetes művelet. Eredete a tradicionális ontológiákra nyúlik vissza. Ha még visszaemlékszünk a delfin ontológiánkra, abban az alábbi mondattal is definiálhatnánk egy delfint:

> a delfin olyan emlős amely a vízben él

(most egy pillanatra tekintsünk elattól a ténytől, hogy a delfin nem az egyetlen ilyen emlős). Ez a mondat a delfinek osztályát definiálja a következő módon:

- vegyük az emlősök osztályát
- vegyük az „él" tulajdonságot
- tekintsük az emlősök osztályának azon egyedeit, amelyre az „él" tulajdonságot alkalmazva a hármas tárgya a „víz" (hasonló módon definiálhatnánk a denevéreket, mint olyan emlősöket amelyek a levegőben élnek). Ez definiálja (OWL-ban) a delfinek osztályát.

Az OWL ezt az osztálykonstruálási formát értéktartomány-korlátozásnak nevezi.

Mivel az OWL az RDFS kiterjesztése, az OWL fogalmak ugyanúgy RDF hármasokban írhatók le, beleértve a tulajdonság korlátozásokat, de a szükséges hármasok száma és struktúra komplexitása kicsit már bonyolultabb. Az érdeklődő olvasó utánanézhet az OWL Overview nevű dokumentumban [14], illetve annak magyar fordításában [15].

Az OWL a tulajdonságok terén is tovább bővíti az RDFS által adott lehetőségeket. Lehetővé teszi ugyanis a tulajdonságok logikai jellemzését, ahogy erre a fejezet elején utaltunk. Tulajdonságokat lehet szimmetrikusnak, tranzitívnek, függvénynek stb. deklarálni. Egy OWL-t „értő" környezet egy sor nem triviális logikai következtetést képes levonni ezen definíciók alapján. Hogy egy példával szolgáljunk: az előző fejezetben leírt alkalmazás esetén szeretnénk, ha az alábbi következtetés lehetséges lenne: „ha 'A' balra van 'B'-től, és 'B' balra van 'C'-től, akkor 'A' balra van 'C'-től". Egy OWL környezetben lehetséges a 'balra van' tulajdonság tranzitivitásának definiálása, ami a fenti következtetést lehetővé teszi.

Az OWL más területeket is érint, amelyekről az RDFS nem beszél. Ilyen például az osztályok, tulajdonságok vagy egyedek *ekvivalenciájának* a problémája. Az osztályok és osztálydefiníciók különböző forrásokból származhatnak; egy alkalmazás adott esetben különböző ontológiákat kombinálhat össze (ne felejtsük el, hogy az RDFS és az OWL is RDF-ben van megfogalmazva, vagyis az előző fejezetben leírt gráfkombinációs lehetőség ontológiákra is érvényes!). Szükségessé válhat tehát annak a kérdésnek a megválaszolása, hogy két osztály azonos-e, vagyis azonosak-e az egyedei avagy sem. Vagy hogy éppen szigorúan különböznek egymástól. Ugyanez a kérdés felmerül a tulajdonságok kapcsán is. Az OWL tartalmaz erre egy sor konstrukciót. Így például lehetséges a következő hármas, mint OWL állítás:

```
(http://....#delfin, owl:equivalentClass,
➥ http://....#dolphin)
```

vagyis az angol „dolphin" nevű osztály ugyanaz mint a magyar nevű „delfin". Emlékezzünk vissza a bevezetésben felvetett problémára: „Ez utóbbi esetben a problémát még nehezítheti ha a különböző lapok különböző terminológiákat használnak ugyanarra a fogalomra." Ezek az OWL konstrukciók ezen probléma megoldására lettek kialakítva.

Tisztában kell lenni azzal, hogy egy OWL alapú következtetési rendszer *lényegesen* bonyolultabb mint egy RDFS alapú. Az OWL eredete a tudásreprezentáció több évtizede kutatott és fejlesztett területére nyúlik vissza. Az OWL az első olyan nyelv, amely a klasszikus tudásreprezentációs logikákat (úgynevezett „leíró logikákat") a Web dimenzióira alkalmazta; rövid idő alatt az OWL a legelterjedtebb tudásreprezentációs nyelvvé vált. De ezen új dimenziók új gyakorlati problémákat vetettek fel; az egyre hatékonyabb következtetési algoritmusok kifejlesztése még mindig aktív kutatási irány, de már megjelentek az első programozási rendszerek, amelyek képesek az OWL (precízebben fogalmazva: az OWL résznyelvének) interpretálására, és a megfelelő következtetések kezelésére.

## Új problémák, fejlesztések

Az RDFS és az OWL kifejlesztésével a munka korántsem állt le. Bármennyire is gazdag, az OWL rendszer nem teljes; egy sor tulajdonságot, logikai szabályt nem lehet OWL-ban kifejezni. Így például nem képes valószínűségeket kifejezni, pedig sokszor fordul elő, hogy egy alkalmazás egy hármasokkal leírt kapcsolatot csak úgy tud értelmezni, hogy „ez a kapcsolat egy adott valószínűséggel áll fenn". Az RDFS és az OWL által leírt kapcsolatok viszont szigorúan igen/nem jellegűek. Tipikus ilyen alkalmazási terület erre a problémára a biológia, biotechnológia. A valószínűségek (vagy, analóg módon, a fuzzy kapcsolatok) kifejezése nagyon sokakat érdekel: a legutóbbi nemzetközi Szemantikus Web konferencián egy külön munkaértekezlet szerveződött ennek a problémának a vizsgálatára [16], és ez természetesen csak az első lépés.

Bizonyos következtetési láncok definiálása és kifejezése szintén túlmutat az OWL lehetőségein. A klasszikus eset a nagybácsi példája: ha a hármas halmazunk tartalmazza a következő két hármast

```
(Dávid, apja, Iván)
(Iván, testvér, János)
```

akkor szeretnénk, ha a rendszer a következő hármasra is következtetne:

```
(János, nagybácsi, Dávid)
```

Sajnos egy ilyen szabályt az OWL-ban leírni nem lehet, ez az úgynevezett Horn következtetési szabályokat igényli amely másfajta logikákhoz kapcsolódnak. Komoly érdeklődés van az ilyen következtetési szabályok szabványosítására, illetve a Szemantikus Web általános struktúrájába való beillesztésére, de mindez nem jelentéktelen elméleti és gyakorlati problémákat vet fel. Egy tipikus alkalmazás lehet egy email kliensben használatos szűrődefiníciók kicserélése szabványosított formában vagy biológiai/biokémiai adatok értelmezéseinek leírása és publikálása. A W3C 2005 áprilisában szervezett egy nagyon sikeres munkaértekezletet [17], és ez vezetett a W3C egyik legfrissebb munkacsoportjához Rule Interchange Format néven. Ez a csoport csak néhány hónapja kezdte el az érdemi munkát [12].

Egy másik érdekes terület a biztonság, bizalom kérdése. Milyen módon bízhatunk meg egy hármas halmazban: például igaz-e, hogy a halmaz létrehozója valóban az, akinek mondja magát? Lehetséges-e a hármas halmazokat (vagy azok egy részét) digitálisan aláírni? Ha igen, milyen protokoll mellett? Lehetséges-e egy hármas halmaz titkosítása?

Az ontológiák fejlesztése is jelentős feladat. Egy-egy nagyobb ontológia (például egy gén-ontológia) kialakítása közösségi munkát igényel, ami felvet egy sor, a kooperatív programfejlesztés problémáira emlékeztető gyakorlati problémát. Az OWL léte ebben voltaképpen csak katalizátor szerepet játszik, hiszen az igazi probléma egy adott terület fogalmainak ismerete, kategorizálása, szisztematikus leírása. Ehhez az OWL csak szintakszist és nem jelentéktelen mértékben motivációt nyújt.

Mindez csak néhány példa a Szemantikus Web által felvetett kutatási-fejlesztési problémák közül. Világszerte sok kutató és fejlesztési csoport dolgozik ezeken a problémákon, egyetemi és ipari környezetben egyaránt; érdemes tehát ezt a területet figyelemmel kísérni!

## Eszközök, alkalmazások

Az előbbiekben már hivatkoztunk a nagy számban rendelkezésre álló programozási környezetekre. Ezek közül néhány (például a Java-ra írt Jena rendszer) nemcsak RDFS, hanem (bizonyos korlátozás mellett) OWL alapú következtetést is megenged. OWL következtetési rendszereket egyébként külön is el lehet érni és más környezettel kombinálni.

Mint mondtuk, a rendelkezésre álló fejlesztési rendszerek, eszközök egy része nyílt forráskódú, habár adott esetben nagy, jól ismert cégek laboratóriumaiban fejlesztették őket (Nokia, HP, IBM, Adobe stb.). Megint más eszközök piaci termékek (például az Altova XML Spy-hoz kapcsolódó szerkesztője, az Oracle vagy a Northrop Grumman adatbázisa vagy, hogy magyar példát is mondjunk, a `progos.hu` fejlesztési környezete). Összességében elmondhatjuk, hogy a Szemantikus Web eszközök piaca az utóbbi években igen jelentősen fejlődött.

Ami az alkalmazásokat illeti, a Szemantikus Web követi azt a jól ismert utat, amelyet más Web alkalmazási területek is bejártak. Az egyetemi, kutatóintézeti fejlesztések száma ma már nagyon jelentős és nagyon tág alkalmazási területeket lefed, elég csak megnézni az utóbbi négy-öt év európai uniós kutatás-fejlesztési projektjeit. De például RDF-et használ a Mozilla Firefox böngészője is, amely tartalmaz egy teljes, Javascript-ben írt RDF környezetet, és az RSS formátum is voltaképpen RDF. Kis méretű cégek, spin-off vállalatok, melyek erre a technológiára alapozzák egzisztenciájukat szinte naponta születnek. És ma már a nagy világcégek is érdeklődnek a Szemantikus Web iránt. Nemcsak az eszközfejlesztők (mint a fennt említett HP, Nokia, IBM, vagy Oracle) hanem az alkalmazók is (Siemens, Nokia, Vodafone, Sun).

Hozzá kell azonban tenni, hogy a nagy cégek alkalmazásai jelenleg elsősorban az „intranet"-en belül maradnak. Így például a Nokia, Vodafone, vagy a Sun portálokat fejlesztettek ki egy-egy specifikus célra (a Sun esetében például a rendszerkarbantartók számára fenntartott információs portált, a Vodafone esetében a telefonok csengőhangjait áruló portált). Anélkül, hogy a felhasználó ezt tudná, ezek a portálok „belül" Szemantikus Web eszközöket használnak. Ez a fejlődési fázis nem csak a Szemantikus Web jellemzője; a 90-es években a bonyolultabb Web felhasználások szintén az intranet-en működtek először és csak később jelentek meg „igazi" Web alkalmazások, és voltaképpen még mindig ez a helyzet a Web Szolgáltatások területén is.

A sok lehetséges példa közül emeljünk ki egyet: ez a „Baby Care Link" alkalmazás, amelyet egy angol cég fejlesztett ki Boston-i megrendelésre. A feladat egy tanácsadói központ létrehozása koraszülött gyerekek ápolására. A múltban Boston-ban létezett egy ilyen központ, amelyben napi 24 órában emberek válaszoltak kórházi telefonhívásokra, hogy orvosi, gyógyszerekkel kapcsolatos, biztosítási, és egyéb információkkal szolgáljanak. Ez nyilvánvalóan nagyon nagy igénybevétellel járó, fárasztó és felelősséggel teli munkát jelentett. A „Baby Care Link" ennek egy Web változata. A tudást OWL-ban modellezték, a kérdéseket egy dedikált Web szolgáltatáson keresztül lehet feltenni, és a rendszer az OWL következtetéseket felhasználva továbbítja a válaszokat. Ha valakinek erre megfelelő joga van (például orvosok, kutatók), akkor a tudásbázist bővítheti is.

## Információk, dokumentumok, könyvek

A rendelkezésre álló cikkek, Web portálok, leírások, blogok száma ténylegesen óriási, és nagyon nehéz benne eligazodni, hiszen a Szemantikus Web, mint mondtuk, aktív kutatás-fejlesztési téma is. A David Beckett által fenntartott „Planet RDF" Web lap [18] talán a legjobb kiindulópont.

A könyvek piaca is jelentős. Természetesen a könyvek kiválasztása kicsit személyes ízlés kérdése is; Powers [19] vagy Antoniu és van Harmelen [20] könyvei csak néhány a sok közül. De mindenképpen érdemes megjegyezni, hogy, néhány hónap eltéréssel, két könyv is megjelent 2005-ben magyarul; míg Gottdank Tibor könyve [21] inkább gyakorlati és bevezető jellegű, addig Szeredi Péter és kollégái inkább az elméleti háttérre koncentrálnak [5].

Természetesen az alapforrás a W3C ajánlások sorozata. Van közöttük inkább bevezető jellegű (mint például a már említett OWL áttekintés [14]) míg mások nehezebben olvashatóak. De mindenképpen ki kell emelni, hogy a W3C magyar irodájának, és elsősorban a fordítónak,

Pataki Ernőnek köszönhetően a teljes RDF, RDFS és OWL dokumentum sorozat elérhető magyarul is, például a magyar iroda honlapjáról [22]. Ugyanezen a honlapon fel lehet jelentkezni a W3C Magyar Iroda hírlevelére is, amelyen keresztül a legfrissebb W3C-val kapcsolatos információkhoz lehet igen könnyen hozzájutni.

Végül, de nem utolsósorban, érdemes a W3C Szemantikus Web honlapját [23] is rendszeresen látogatni. Ott nemcsak eszközökre vonatkozó információkat lehet találni, hanem a látogató követheti a W3C-n belüli legújabb fejlesztéseket is.

## Irodalom


[1] Michael L. Dertouzos, *The Unfinished Revolution*. Harper Business, New York, 1995.
[2] Michael L. Dertouzos, *A félkész forradalom*. Typotex, Budapest, 2002.
[3] Tim Berners-Lee, James Hendler and Ora Lassila, *The Semantic Web*. In: *Scientific American*, **156**(5), 2001, pp. 35-43.
[4] Tim Berners-Lee, *Weaving the Web*. Harper, San Francisco, 1999.
[5] Szeredi Péter, Lukács Péter, Benkő Tamás, *A szemantikus világháló elmélete és gyakorlata*. TypoTex, Budapest, 2005.
[6] Ivan Herman and Daniel Dardailler, *SVG Linearization and Accessibility*. In: *Computer Graphics Forum*, **21**(4), 2002, pp. 777-786.
[7] http://www.w3.org/RDF
[8] SPARQL Query Language for RDF, E. Prud'hommeaux, A. Seaborne (Eds.), W3C Working Draft, http://www.w3.org/TR/rdf-sparql-query/, 2005.
[9] RDF Vocabulary Description Language 1.0: RDF Schema, D. Brickley, R.V. Guha (Eds.), W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-schema, 2004.
[10] http://www.w3.org/2004/OWL/
[11] SKOS Core Vocabulary Specification, A. Miles, D. Brickley (Eds.), W3C Working Draft, http://www.w3.org/TR/swbp-skos-core-spec, 2005.
[12] http://www.w3.org/2005/rules/
[13] http://www.wiwiss.fu-berlin.de/suhl/bizer/toolkits/.
[14] OWL Web Ontology Language – Overview, D. L. McGuiness, F. van Harmelen (Eds.), W3C Recommendation, http://www.w3.org/TR/2003/owl-features
[15] Az OWL Web Ontológia Nyelv – Áttekintés, D. L. McGuiness, F. van Harmelen (Eds.), W3C Ajánlás, http://www.w3c.hu/forditasok/OWL/REC-owl-features-20040210.html
[16] Uncertainty Reasoning for the Semantic Web (Workshop at ISWC2005), P.G. Costa, K.B. Laskey, K.J. Laskey, M. Pool (Eds.), http://ite.gmu.edu/~klaskey/URSW_Proceedings.pdf, 2005.
[17] W3C Workshop on Rule Languages for Interoperability, S. Hawke, C. de Sainte Marie, S. Tabet (Eds.), http://www.w3.org/2004/12/rules-ws/program2, 2005.
[18] http://planetrdf.com/guide/
[19] Shelley Powers, *Practical RDF*. O'Reilly, Cambridge, 2003.
[20] Grigoris Antoniu and Frank van Harmelen, *A Semantic Web Primer*. The MIT Press, Cambridge, Massachusetts, 2004.
[21] Gottdank Tibor, *Szemantikus web*. ComputerBooks, Budapest, 2005.
[22] http://www.w3c.hu
[23] http://www.w3.org/2001/sw/


*Herman Iván*

*Herman Iván matematikusként végzett a budapesti Eötvös Lóránd Tudományegyetemen, 1979-ben. Rövid párizsi tartózkodás után az MTA Sztaki Elektronikus Főosztályán kezdett dolgozni tudományos munkatársként, ott is vált számítástechnikussá. 1986-ban Münchenbe költözött, ahol egy (azóta eltűnt) szoftverházban dolgozott. 1988-tól Amszterdamban a „Centre for Mathematics and Computer Science" nevű kutatóintézetben dolgozik tudományos munkatársként. 1990-ben letette a PhD fokozatot a Leideni Egyetemen, Hollandiában. 2001-től a World Wide Web Consortium állandó stábjához is tartozik; ebben a szerepében a W3C Szemantikus Web koordinációs csoportjának tagja és egyben a W3C világot átfogó hivatalhálózatának vezetője. Rendszeresen tart előadásokat világszerte különböző Web technológiákról, mostanában elsősorban szemantikus webről és SVG-ről, és aktív fejlesztője néhány, szemantikus webbel kapcsolatos szoftvernek (Python nyelven).*

*W3C-s munkáját megelőzően Iván fő tevékenységi területe a számítógépes grafika és vizualizáció volt. E területen részt vett különböző szabványosítási munkákban mint a magyar, később mint a holland szabványosítási hivatalok képviselőjeként. 15 évig volt a Eurographics Association Executive Committee-nek tagja, annak utolsó két évében a szervezet alelnöke is volt. Több nemzetközi konferenciának volt elnöke vagy aktív szervezője; így például elnökölte a World Wide Web konferenciát 2000-ben Amszterdamban, és aktív szervezője volt a Budapesten tartott World Wide Web konferenciának, 2003-ban.*

*További részletek a http://www.w3.org/People/Ivan és http://homepages.cwi.nl/~ivan/ weblapokon.*

**A konferencia főszervezői:**

*Balogh Tibor Csaba - dr. Baranyai László - Bártházi András - Bóna László Márton*
*Granc Róbert - Heilig Szabolcs - Hodicska Gergely - Hojtsy Gábor - Járja Norbert*
*Karóczkai Krisztián - Károly György Tamás - Kiss-Tóth Marcell - Palócz István*
*Papp Győző - Simon Benjamin - Szabó Dénes*

*Köszönjük az ELTE Radnóti Miklós Gyakorlóiskola és lelkes diákjainak segítségét a konferencia szervezésében.*


**http://web.conf.hu/2006**