www.phpconf.hu

2004 második magyarországi

PHPKONFERENCIA

2004 második magyarországi

PHPKONFERENCIA

március 27. szombat

Budapest, Cházár A. u. 10.

Tisztelt Látogató!

Szeretettel köszöntjük a 2004. évi Második Magyarországi PHP Konferencián. Rendezvényünk programjának összeállításakor több, a tavalyi konferencia résztvevői által megadott szempontot vettünk figyelembe. Számos téma iránt érdeklődtek a megkérdezettek, szerencsénkre ezek nagy részére érkezett előadói jelentkezés, illetve olyan megnyerő témákkal leptek meg bennünket az előadók, melyeknek idei látogatóink biztosan nem tudnak majd ellenállni. Igyekeztünk a különböző területekről más-más előismerettel érkezőknek is megfelelő programot biztosítani, így bevezető előadások ugyanúgy helyet kaptak, mint haladó témájú bemutatók.

Széles körben mutatkozott az az igény, hogy gyakorlatiasabb bemutatók is legyenek a konferencián, amelyeken kérdésekkel, javaslatokkal be lehet avatkozni az előadás menetébe. Nagyon fontosnak tartjuk, hogy ne csupán a PHP-re koncentráljunk, hanem széles körű ismeretekkel szolgáljunk. Ezért helyet kaptak olyan témák is, melyek más technológiák alkalmazása esetén is közvetlenül hasznosíthatóak: szó lesz verziókezelésről, szabványkövető oldalkialakításról, megfelelő programozási környezet megválasztásáról, illetve nagy rendelkezésre állású rendszerek tervezéséről és üzemeltetéséről egyaránt.

Szeretnénk, ha a konferencia nem csak szakmai feltöltődésre adna lehetőséget, hanem a különböző interneten szerveződő közösségek találkozóhelyévé is válna. Rendezvényünk azért is különösen alkalmas erre, mert a környező országokból is érkeznek résztvevők, így az online csoportosulások komoly esélyt kaphatnak az összegyűlésre. Ennek érdekében kellő időt szántunk a programban a személyes ismerkedésre, beszélgetésre. A levelezőlistákon, fórumokon, IRC csatornákon megismertek könnyebb azonosítását pedig a névkártyákon szereplő becenevekkel is segíteni szeretnénk.

Ezzel a konferencia kiadvánnyal kettős célunk volt. Egyrészt szeretnénk megkönnyíteni a választást a párhuzamosan futó programpontok között, másrészt szeretnénk egy, a konferencia után is értékes, tartalmas kiadványt az Ön kezébe adni. Érdemes egy-egy előadás meglátogatása előtt átfutni a hozzá kapcsolódó cikket, hiszen így csak azokat az információkat kell jegyzetelni, amelyek a füzetben még nem szerepelnek.

Nem hagyhatjuk köszönet nélkül rendezvényünk számos támogatójának odaadó segítségét, mellyel lehetővé tették, hogy beteljesítsük nagyszabású terveinket. Az NJSZT Webalkalmazások Fejlesztése Szakosztály lelkes tagjainak összefogásával és az előadók aktív közreműködésével várakozásunk szerint úgy tudjuk lebonyolítani a rendezvényt, hogy Ön erre még sokáig hivatkozni fog.

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

A Konferencia Szervezői

NJSZT Webalkalmazások Fejlesztése Szakosztály

Az Első Magyarországi PHP Konferencián alakítottuk meg az NJSZT Weblalkalmazások Fejlesztése Szakosztályt 2003. március 29-én. Független, szakmai közösségünk célja a webalkalmazások fejlesztésére használható technológiák (mint például a PHP) népszerűsítése, szakmai anyagok írása, fordítása, azok támogatása, továbbá a programozási nyelvekkel és a kapcsolódó technológiákkal foglalkozó konferenciák szervezése és lebonyolítása. A megalakulásunk hathatós támogatásáért köszönet illeti Remzső Tibort.

A Második PHP Konferencia résztvevőinek az már nem újdonság, hogy évente visszatérő konferenciánkkal igyekszünk összehozni a szakmai közösséget, a terület iránt érdeklődő diákokat és tanárokat egyaránt. Szakosztályunk működése azonban közel sem korlátozódik erre, tagjaink számos érdekes projekten dolgoznak.

Weblabor (http://weblabor.hu/) néven webes technológiákkal foglalkozó hírmagazint működtetünk, mely foglalkozik a legfrissebb trendekkel és alakuló szabványokkal éppúgy, mint az újonnan felbukkanó termékekkel és technológiákkal. A webhely PHP konferenciához időzített megújulása jelentős tartalom-bővülést is hoz, mely fejlesztett közösségi szolgáltatásokat, és még időtállóbb, hosszabb írásokat is jelent. Éppen a Weblabor keretében működnek szakmai levelezőlistáink, melyek lehetőséget adnak a felmerülő problémák megvitatására, megoldására. Szakosztályunk alapítói is a PHP levelezőlistán formálódott „kemény magból" kerültek ki.

Az időtálló segédletek készítésének érdekében hirdettük meg cikk- és hírbeküldő versenyünket, melynek keretében számos értékes írást kaptunk. A PHP dokumentáció fordítása nem állt meg, bár jelenleg kisebb érdeklődés mutatkozik a munkában való részvételre. A PHP 5 érkezésével számos érdekes friss téma kerül a kézikönyvbe, melyet első kézből a fordítás által megismerni véleményünk szerint igazán jó lehetőség. Turcsányi Tamás elkészítette a „WFSZ biztonsági ajánlásai webfejlesztők számára" című dokumentumot.

Külső kapcsolatainkat tekintve elsőként a web szabványait kialakító World Wide Web Consortium Magyar Irodát kerestük fel, akik üdvözölték közeledésünket, és azóta is nagyon jó kapcsolatot tartunk fent.

Kisebb rendezvényekben sem volt hiány az elmúlt évben. Magyarországra látogatott a MySQL cég két prominens személyisége David Axmark és Zak Greant, akik két átfogó előadást tartottak a megjelent népes érdeklődő tábornak a MySQL legújabb fejlesztéseiről. A szakosztály tagjai számos alkalommal előre meghirdetett helyen és időben nyilvános találkozókat tartottak, megvitatva a lehetséges feladatokat, a konferenciával illetve más projektekkel kapcsolatos kérdéseket. Palócz Istán a Fix.Tv Kreatív Klub című műsorában rendszeresen jelentkezik a PHP Suli részeivel.

Amennyiben érdeklődik a szakosztály munkája iránt, arra kérjük, hogy kísérje figyelemmel híreinket a Weblabor oldalon. Szívesen látunk bekapcsolódni szándékozó új tagokat is, számos érdekes tervünk van, melyek megvalósításához elkél a segítség. Bővebb információt talál a szakosztályról és projektjeinkről a http://wfsz.njszt.hu/ címen.

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 évenként tíz alkalommal nyomtatásban is megjelenő „Mi Újság" című hírlevélben.

Program

Az alábbiakban a konferencia programtervezete olvasható. A táblázatban sötét színnel szerepelnek a gyakorlati bemutatók, az oldalszámok zárójelben találhatók.

8:30–10:00	**Regisztráció**		
10:00–10:05	**Megnyitó**		
10:05–11:05	Heilig Szabolcs, Hojtsy Gábor, Illés Szabolcs, Palócz István **CMS Maraton (4)**		
11:05–11:30	Kolman Nándor **A PHP 5 újdonságai (12)**	Bóna László Márton **Objektum-orientált programozás a gyakorlatban (10)**	
11:30–11:50		Ercsey Balázs **Zenetár a webszerverünkön, avagy XML használata a PHP 5–ben (14)**	
11:50–12:05	**Szünet**		
12:05–12:45	Mocsnik Norbert **PEAR – a PHP gyümölcse (16)**	Horváth Zoltán **Fejlesztés PHP-Nuke rendszerre (8)**	Kovács Zsolt **Segítség! Felnőttem! (34)**
12:45–13:25	Bártházi András **Így készült a Weblabor (30)**	Károly György Tamás **PHP és Perl – két dudás egy csárdában (7)**	
13:25–14:55	**Szünet, közösségek találkozói**		
14:55–15:35	Papp Győző **PHP Chemotox (22)**	Nováki Szilárd **A Gomba for PHP fejlesztési környezet (26)**	Mocsnik Norbert **Webes alkalmazás-fejlesztés PEAR csomagokkal (16)**
15:35–16:00		Szalai Ferenc Attila **PHP a grid technológiában – egyszerűen és célratörően (32)**	dr. Baranyai László **Web-szabványok hazai alkalmazásának statisztikai elemzése (25)**
16:00–16:40	Kolman Nándor **Általános űrlapkezelés (20)**	Forstner Bertalan **Elterjedt technológiákra építő webes fejlesztőrendszer (28)**	Mocsnik Norbert **Hatékony fejlesztést segítő további PEAR csomagok (16)**
16:40–16:55	**Szünet**		
16:55–17:25	**Fórum** (a hallgatók kérdéseket tehetnek fel)		
17:25–18:00	**Eredményhirdetés, ajándéksorsolás, zárás**		

CMS Maraton

A web kezdetén a statikus HTML oldalak hosztolási lehetősége külön kiváltságnak számított, ám ahogy egyre terjedt a technológia, és az otthonokban is megjelentek a web böngészésére alkalmas eszközök és programok, a cégek is egyre inkább felismerték, hogy elengedhetetlen a webes jelenlét.

Talán közhelynek számít, hogy a cégek az internetben rejlő lehetőségeket jócskán túlértékelték, melynek kellemes mellékhatásaként – az alapvetően üzleti célból létrehozott – webes portálok segítségével kialakultak olyan közösségek, melyeknek tagjai az internet nélkül az életben sohasem találkoztak volna. Egy-egy ilyen portál mágnesként gyűjtötte össze az azonos téma iránt érdeklődő embereket, akikben felmerült az igény egy saját, független „webes találkahely" létrehozására. Ez a hajtóerő szükségszerűen magával hozta a webhelyek gyors kialakításának és változtatásának igényét, melynek a statikus HTML technológia már nem tudott megfelelni. Elkezdődött a tartalomkezelő rendszerek fénykora.

Tartalomkezelő rendszerek

A CMS (Content Management System) rendszerek lehetővé teszik a felhasználóik számára, hogy komolyabb technikai tudás nélkül kialakítsák és karbantartsák webhelyüket, legyen az akár egy tartalmasabb személyes honlap, egy vállalat intranetes oldala vagy egy termékportfóliót bemutató katalógus. A piacon számos kereskedelmi termék érhető el, ám a szabadon használható és terjeszthető megoldások széles skálája is rendelkezésre áll (http://opensourcecms.com/). Tulajdonképpen bárki írhat saját tartalomkezelő rendszert, a megmaradás szempontjából valójában csak az a kérdés, hogy a fejlesztő milyen követő tábort képes kialakítani a technológiával és a felhasználói támogatással.

Egyetlen megoldás sem lehet minden problémára alkalmazható, így logikus, hogy különböző igényekkel érkező felhasználók más és más rendszereket részesítenek előnyben. Reméljük, hogy rövid összehasonlításunk segít a választásban, és ha nem is a tárgyalt három rendszer valamelyikét választja, a vázolt szempontok és kritériumok segítenek más programok értékelésekor is.

Komolyabb tapasztalattal rendelkező fejlesztők nagy reményekkel saját megoldásuk kialakításába is belefognak, időnként jól meghatározott okok miatt, máskor csupán szakmai önérzetből. Úgy gondoljuk, hogy a meglévő rendszerek működésének, megoldásainak vizsgálata mindenképpen előnyösen hathat a saját tervek és programok kialakításának menetére. Az előadás a rendelkezésre álló időnek megfelelően egy bővebb és technikailag mélyebb összehasonlítással szolgál a három rendszerről.

PHP-Nuke

Az egyik legrégebb óta fejlesztett és legismertebb tartalomkezelő rendszer, a vizsgáltak közül a legtöbb példányban használt. Ezt a sikert elsősorban egyszerű telepíthetőségének és használhatóságának köszönheti, melynek eredményeképpen rendkívül sok modult és megjelenést fejlesztettek hozzá. A 2000-es év augusztusában Francisco Burzi (http://www.php-nuke.org/) kezdte el fejleszteni a Mandrake Linux több mint egy éven át tartó anyagi támogatásával. A 2003-as év közepén alakult meg a PHP-Nuke hivatalos fejlesztői csapata a NukeCops (http://www.nukecops.com/).
A fejlesztői gárdában több neves rendszerszervező, programozó és szakember vesz részt. A második magyarországi PHP Konferencia szervezése ideje alatt a hivatalosan közreadott 7.0-ás verziót érhetik el az érdeklődők a GNU GPL licenc 2.0-ás verziója alatt.

A rendszer világszerte elérhető hatalmas közösségi táborral rendelkezik, ennek köszönhetően már 33 nyelven érhető el a programcsomag és a dokumentáció. Magyarországon több mint négyezer regisztrált tagja van a közösségnek, és közel tízezer honlap működik intra, -és interneten egyaránt. Nagyon könnyen feltelepíthető Linux és Windows operációs rendszerekre, számos beépített modullal érkezik (hírek, letöltések, keresés, statisztikák, szavazások, linkek, stb.). Az ADODB illesztőn keresztül számos adatbázisszerverhez tud csatlakozni. A PHP-Nuke magyar nyelvű dokumentációját Arany Sándor, Illés Szabolcs és Ökrös Attila készítette el 2002-ben. A dokumentáció modulon keresztül beágyazódva már a letölthető telepítőben is közvetlenül elérhető.

Nem csak előnyei vannak azonban a PHP-Nuke rendszernek, egy ilyen széles kört kielégíteni szándékozó megoldásnál elkerülhetetlen, hogy hátrányokról is beszéljünk. A számos elérhető modul nem minden esetben érkezik biztonságos kóddal, az alaprendszer biztonságára az utóbbi időben már vigyázó szemek figyelnek. A szerteágazó külső fejlesztésű modul kör támogatása érdekében több tervezési döntés megmásíthatatlanná vált a későbbi verziók eljövetelével, a visszafelé kompatibilitást megőrző kódok időnként feleslegesen lassítják a webhely futását. A PHP-Nuke történetének korábbi szakaszában Francisco ragaszkodott egyszemélyes fejlesztői státuszához, és ez a rendszer jobbítására törekvő jópár fejlesztőt késztetett arra, hogy különböző mellékágakon folytassák a PHP-Nuke fejlesztését, esetenként komplett átalakítással. Ezeken az utakon alakult ki a PostNuke, a My-PHPNuke és több más csomag. Hasonló okokból vált ki később a PostNukeból a Xaraya és a Xoops fejlesztőgárdája. Időközben a PHP-Nuke is jelentős fejlődésen ment keresztül, és megtartva a felhasználói tábort versenyképes alternatíva maradt.

Drupal

A Drupal (http://drupal.org/) legszívesebben kollaboratív tartalomkezelő rendszernek hirdeti magát, melyet a teljes

felhasználói körre kiterjedő szerzői lehetőségeknek köszönhet, akár webhelyeket is átívelő felhasználói regisztrációval. Éppen ezen tulajdonságai miatt választotta Howard Dean kampányának alaprendszeréül.

A fejlesztés során központi figyelmet kap az elérhetőség és a szabványos kódok alkalmazása, fontos cél a kimenet XHTML és CSS 2 szabványoknak megfelelő generálása a lehető legkevesebb HTML szeméttel. Ehhez járul a legrövidebb URL elérési módok támogatása, mely nemcsak a vizuális, hanem a webcímeken keresztül megnyilvánuló felületet is barátságossá teszi. Mindezek jótékony mellékhatása a keresőbarát oldalkialakítás.

Filozófiája a „többet-kevesebbel", azaz a kész kódok minél nagyobb mértékű újrahasznosítása. Nem ritka, hogy a különböző kiterjesztések egymás mellé települve hirtelen sokkal több funkcionalitást nyújtanak, kihasználva egymás képességeit. A cikk írásakor elérhető 4.4.0-ás kiadásra jelölt verzió letöltési mérete mindössze 430Kb.

A rendszer egyik legnagyobb erénye a mindent átható úgynevezett taxonómia kezelés, ami az összes kategorizálást igénylő problémára megoldást ad (hírrovatok, fórum kategóriák, szójegyzék csoportok, stb.). A taxonómia sokoldalú farendszerek létrehozását teszi lehetővé szinonímákkal, a végpontok közötti kapcsolatokkal, és számos más kellemes szolgáltatással. Egy másik fontos alapkoncepció az, hogy szinte minden tartalom úgynevezett node-okban tárolódik. Egy node a taxonómiák szerint kategorizálható, lehetnek különböző verziói, hozzászólások tartozhatnak hozzá, és még számos előnyös tulajdonsággal rendelkezik, mely a tartalmak kezelésénél jelentősen lecsökkenti a tanulási időt. Az alaprendszer az általánosságban használt modulokkal érkezik (cikkek, fórumok, hozzászólások, bloggerAPI támogatás). A csomag része egy nagyon sokoldalú RSS-aggregátor, és a rendszer arra is fel van készülve, hogy a webhely tartalmait RSS formában is elérhetővé tegye.

Sokszor éri az a vád a Drupal-t, hogy kockafejűek számára készült, ami bizonyos mértékben elismerhető. A telepítést egyelőre nem segíti varázsló, az SQL táblákat a parancsok importálásával magunknak kell létrehoznunk. Az alaptelepítés egy figyelmeztető üzenet mellett nem sok segítséget ad az elindulásban. A taxonómia rendszer valóban tanulást igényel, ám rendkívül sokoldalú képességeivel jól használható eszközt ad a kezünkbe. A rendszerhez magyar dokumentáció nem érhető el, a felület nyelvének magyar fordítása a Weblabor Drupal-ra adaptálása kapcsán készült el nemrég.

eZ publish

Az eZ publish és a mögötte álló eZ Systems cég (http://ez.no/) 1999-ben indult útjára Norvégiában. A cikk írásakor a 3.3-3-as változat tölthető le a projekt weboldaláról. A tekintélyes, tömörítve 5MB-os csomag egy igen széleskörűen alkalmazható rendszert ad kezünkbe, ha élünk vele.

Az eZ publish sikeres próbálkozás arra, hogy a különféle típusú tartalmak kezelésére és publikálására egy általános megoldást nyújtson. Egy cikk, egy blog bejegyzés vagy egy személyi adatlap mind kiszolgálható egyazon motorral, nincs szükség külön modulokra ezek kezeléséhez. A tartalom egy fastruktúrába rendezett objektumhalmazként ábrázolható, az objektumokat adatosztályokból hozhatjuk létre. Néhány alapvető osztály a telepítés során bekerül a vérkeringésbe, de könnyedén létrehozhatunk egy újat magunk is. Egy-egy ilyen osztály alapvető adattípusokból állítható össze, mint például egy, vagy több soros szöveg, kép, bináris állomány, egész szám, URL, stb. Az egyes objektumok tulajdonságain felül az is információval bír, hol helyezkedik el a fastruktúrában az adott elem. Egy cikkhez könnyűszerrel csatolhatunk mozit, vagy megjegyzéseket, csak a cikk mint szülőobjektum alá kell ezeket elhelyeznünk.

Mindezen adatokból az igen erősen konfigurálható, sablonelvű megjelenítőmotor varázsol végül kimenetet. A sablonok segítségével a megjelenítés minden apró kis részlete megváltoztatható. Maga a sablonnyelv igen hasonlít a Smarty nyelvezetére, ami nem csoda, hisz valamikor a kezdetek kezdetén innen merítették az alapokat a fejlesztők. Hogy mely csomóponthoz melyik sablon kerüljön feldolgozásra, a beállításoktól függ, amiket „ini" formátumú fájlokban tudunk módosítani. Minden a legapróbb részletekig beállítható, ami kellemes, viszont a bonyolult beállítási lehetőségek miatt néha nehéz rájönni, hol is van valami félrekonfigurálva.

A rendszer annyira rugalmas, hogy egy sor programozás nélkül létrehozható benne például egy vendégkönyv. Eme rugalmasságot mi sem támaszthatja jobban alá, mint az, hogy magát a tartalmakat adminisztráló felületet is ugyanaz a motor szolgálja ki, mint a nyilvános oldalakat. Az összetettséggel sajnos nagy erőforrás igény is jár. A rendszer bekapcsolt debug funkcióval a kezdetekben kevesli a PHP alapbeállításaiban szereplő 8MB-os memóriafelhasználási határt. Ráadásul jelentős feldolgozás szükséges, mire eljut a rendszer ahhoz, hogy az adott URL-ből kimenetet gyártson. Mindezeken a beépített gyorsítótár funkció segít.

Az eZ publish-sal nehéz a kezdet, sok-sok ismerkedésre szánt idő és energia szükséges ahhoz, hogy a teljes rendszert átlássuk, megismerjük erősségeit és korlátait. Mindezért cserébe egy hatékony webhelyépítő eszköz kerül a kezünk közé.

Összegzés

Egy tartalomkezelő megismerése nem egy-két perces feladat, azonban mindig mérlegelni kell, hogy az igényeinknek legjobban megfelelő programcsomag kiválasztása és testreszabása, vagy egy teljesen új rendszer fejlesztése a kifizetődő. Bárhogyan is döntsünk, ne felejtsük el, hogy egy egy ilyen rendszer kialakítása rengeteg munkába és fáradtságba kerül, melytől egy megfelelő választás esetén megkímélhetjük magunkat. Természetesen nem állítjuk, hogy léteznek tökéletes, minden igényt kielégítő megoldások, csupán azt, hogy érdemes körülnézni, mielőtt egy saját rendszer kifejlesztésébe fogunk. Számos nyílt forraskódú CMS létezik, melyekből ötleteket meríthetünk, és hibáikból okulhatunk, amennyiben egy új rendszer elkészítésének feladatát vennénk vállunkra.

Konklúzióként megemlítenénk, hogy az interneten a legfontosabb három összetevő az információ, a navigáció/funkcionalitás és a grafika/megjelenés. A tartalomkeze-

lő rendszer e hármas elegy kialakításának eszköze kell legyen, nem pedig öncélú szakmai magamutogatásunknak színtere.

Heilig Szabolcs

1997-ben a Veszprémi Egyetem hallgatójaként ismerkedett meg a PHP nyelvvel, rövid „perles" múlttal a háta mögött. Erre a veszprémi VLUG tagjaként nyílt lehetősége. Az elmúlt években foglalkozott online PHP oktatással, részt vett a PHP dokumentációt magyarító csapat munkájában. 2001 decemberében jelent meg a nyelvvel foglalkozó cikksorozatának első darabja a Linuxvilág magazinban. Egyik szervezője az idei, valamint a 2003-as konferenciának.

Hojtsy Gábor

A Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója, a magyar PHP dokumentáció és a magyar PHP levelezőlista elindítója, a Weblabor egyik vezéralakja, Goba néven ismert. A php.net webmester csoport aktív tagja, a dokumentáció keretrendszerének egyik fejlesztője. A 2002-es frankfurti PHP konferencia előadója dokumentáció és PHP-GTK témában, a 2004-es amszterdami PHP konferencia előadója dokumentáció témában. A Drupal tartalomkezelő rendszer egyik közreműködő fejlesztője, a felület lefordítását lehetővé tevő alapmodul hivatalos felelőse. Az első és második magyarországi PHP konferencia egyik szervezője.

Illés Szabolcs

2000-ben a Nyugat Magyarországi Egyetem hallgatójaként ismerkedett meg a PHP nyelvvel, szakdolgozat témájaként választva készített intézményének egy hallgatói portált és webes felületű levelező-rendszert. A PHP-NUKE magyarországi képviselőjeként hozta létre és üzemelteti a Magyar PHP-NUKE Közösségi portált. 2002-ben részt vett a PHP-NUKE rendszer magyar nyelvű dokumentációjának elkészítésében, azóta is folyamatos csoport-koordinációs feladatokat lát el a PHP-NUKE Magyarországnál. A Netkey-Group vezetőjeként fejleszt webes alkalmazásokat. A PHP és mySQL használatát, hazai terjesztését a PHP-NUKE rendszer használatával és a hozzá fejleszthető modulokkal szemlélteti a Fix.tv Kreatív klub című műsorban látható PHP-NUKE oktató sorozatban.

Palócz István

Az ELTE Radnóti Miklós Gyakorlóiskola tanára, az NJSZT Webalkalmazások Fejlesztése Szakosztály alapító tagja és elnöke. 1998-ban ismerkedett meg a PHP nyelvvel és kezdte el oktatását. Igyekszik az oktatók figyelmét a webes alkalmazás fejlesztés felé irányítani, többek között módszertani tapasztalatainak átadásával. Az INFO konferenciákon több előadásával népszerűsítette a PHP nyelvet. A Fix.tv Kreatív klub című műsorában a PHP Suli oktatója. Az első és második magyarországi PHP konferencia főszervezője.

PHP és Perl – két dudás egy csárdában

Előszó helyett: Perl vagy PHP?

Igen. Lehetne ez is a válasz, hiszen a kérdés rossz. Talán nem lehet választani a kettő közül? De, természetesen lehet, azonban a választás eredménye nem mindig azonos.

Különbségek – hasonlóságok

A két nyelv teljesen más, mégis majdnem egyforma. Ez önellentmondásnak tűnik, ám mégsem az, mert egyes szempontok szerint az állítás első fele, míg más szempontok szerint annak második fele bizonyul igaznak. Miben állnak ezek a különbségek és hasonlóságok? Erről szól ez az előadás.

Miből lesz a cserebogár?

Természetesen a C-ből… Persze ez csak vicc. Világegyetemünk nem úgy teremtetett, hogy magában foglalta az új típusú szkripting nyelvek megvalósítását is, tehát valaki(k)nek létre kellett hozniuk ezeket a ma már a kort meghatározó szellemi termékeket. A Perl és a PHP keletkezéstörténete ma már technológiai értelemben történelem, hiszen három-négy év történelmi időnek számít információs társadalmunk életében… Meglepő és mulatságos párhuzamokat fedezhetünk fel e fent említett műalkotás születésének folyamatában, főleg a keresztelő volt kalandos ez esetben.

Kapcsolatok

A két nyelv kompatibilitási és integrációs problémáit feszegetve érdekes tanulságokat szűrhetünk le. Beszámolok néhány közös partnerről a modulok közül (gd, tk, adatbázis illesztők stb.), és megemlítek egy készülő Perl-PHP interoperabilitást lehetővé tevő kiterjesztést is.

Melyik a jobb?

Erre ismét csak azt felelhetem, igen. Egy összehasonlítás során azonban megállapíthatjuk, hogy egy adott célra, egy adott időben és helyen melyik módszert válasszuk. Előadásomban nem kívánom meghatározni ezeket a területeket, csupán egy szemléletet fogok bemutatni, miszerint a cél határozza meg az eszközt, és nem az eszköz a célt.

Átállás, vagy párhuzamos használat?

Vannak területek, ahol érdemes átállni Perl-ről PHP-re, mint például az egyszerűbb weblapok kivitelezése, hiszen akár felére, harmadára is csökkenhet a munkával töltött idő, azonban egy komolyabb portált elkészíthetünk vegyes technikával is, főleg, ha már megírt kódjaink is vannak. Webes felületre való programozásnál könnyebb és gyorsabb lehet a PHP, más fejlesztésekhez a Perl, vagy egy másik programnyelv lehet a hatékonyabb megoldás. Mindig válasszuk a feladathoz és önmagunkhoz mérten legjobb megoldást, ne szűküljünk be egy nyelv használatába, ne restelljünk mindig új technológiákat tanulni.

Károly György Tamás

Szabadúszó webdesigner. 1993-ban kezdett weblapokat tervezni hobbiból, ma már ez a fő tevékenysége. A PHP-val viszonylag friss a barátságuk, 2003 előtt szinte kizárólagosan Perl-ben dolgozott. A tavalyi PHP konferencián döntött úgy, hogy kipróbálja komolyabb feladatok megoldására is ezt a nyelvet, azóta szinte kizárólag PHP-t használ webes munkáihoz. Honlapja a kgyt.hu címen található.

Fejlesztés PHP-Nuke rendszerre

A gyakorlati bemutató során az érdeklődők megtudják, hogy miként tudnak saját elképzeléseiknek és elvárásaiknak megfelelő egyedi modult készíteni, és azt PHP-Nuke rendszerükbe illeszteni.

Mint minden fejlesztés során, az első feladatunk, hogy felmérjük a pontos igényeket, és meghatározzuk a fejlesztendő rendszerrel szemben támasztott elvárásokat. Ezután a rendelkezésre álló fejlesztői eszközöket, az üzemeltetés helyét, és saját képességeinket figyelembe véve kell kialakítanunk a megvalósításhoz szükséges terveket. A feltárás, elemzés és tervezés során meghatározott információk ismeretében állhatunk neki a modul elkészítésének.

Gyakorlati példaként egy egyszerű kvíz modul kerül bemutatásra, ezen keresztül ismerhetjük meg egy PHP-Nuke modul elkészítésének alapvető lépéseit. A modult a teljesség igénye nélkül készítjük el, hiszen az előadás célja nem a modul megírása, hanem annak PHP-Nuke rendszerbe történő beillesztése.

Adatbázis-tervezés és megvalósítás

Az adatbázissal rendelkező modulok esetében először meg kell tervezni az adatbázis szerkezetét, ki kell alakítani a szükséges adattáblákat, azok kapcsolatait és beállításait. Példánkban egyetlen adattáblával fogunk dolgozni, melynek felépítését a következő táblázat tartalmazza:

1. táblázat: nuke_kviz adattábla szerkezeti felépítése

id	Egyedi azonosító
question	Kvíz kérdés
option1	Válaszlehetőség 1
option2	Válaszlehetőség 2
option3	Válaszlehetőség 3
option4	Válaszlehetőség 4
correct_option	Helyes válasz sorszáma (1-4)

Az adattáblák megtervezése után létre kell hoznunk fizikailag a táblákat a PHP-Nuke rendszerünkben. A Nuke rendszer többféle adatbázis-kezelő rendszert támogat, amelyek közül példánkban a MySQL rendszert használjuk. Az integráció többféleképpen is megvalósítható, automatizálhatjuk előre megírt utasításokkal, vagy manuálisan is létrehozhatjuk a MySQL szerverünkön. A `nuke_kviz` adattáblát létrehozó SQL parancs:

```
CREATE TABLE `nuke_kviz` (
        `id` INT( 11 ) NOT NULL AUTO_INCREMENT ,
        `question` VARCHAR( 255 ) NOT NULL ,
        `option1` VARCHAR( 100 ) NOT NULL ,
        `option2` VARCHAR( 100 ) NOT NULL ,
        `option3` VARCHAR( 100 ) NOT NULL ,
        `option4` VARCHAR( 100 ) NOT NULL ,
        `correct_option` INT( 1 ) NOT NULL ,
        PRIMARY KEY ( `id` )
);
```

Felhasználói felület

Az adatbázis elkészítése után létre kell hoznunk a modul működését megvalósító állományokat. A rendszer gyökerében lévő `modules` könyvtárban hozzunk létre a modul számára egy alkönyvtárt, és a fájlokat ebben helyezzük el. Kisebb modul esetén (mint példánkban is) ez akár egyetlen állomány is lehet (`index.php`), nagyobb és összetettebb modul esetén több fájlból is állhat.

Vannak olyan utasítások, amelyek megteremtik a kapcsolatot a rendszerünkkel, ezeket a modul fájljaiba kell beletennünk, ahhoz, hogy megfelelő legyen az együttműködés a Nuke-kal. Ezek az utasítások sorrendben a következők

1. a fájl direkt elérését megakadályozandó:

```
if (!eregi("modules.php", $_SERVER['PHP_SELF'])) {
    die ("You can't access this file
➥ directly...");}
```

2. a rendszer motorját az alábbi utasítással használhatjuk:

```
require_once("mainfile.php");
modul nevének meghatározása:
$module_name = basename(dirname(__FILE__));
```

3. nyelvi fájl megnyitása:

```
get_lang($module_name);
```

4. az oldal keretének első része (fejléc és bal oldali rész):

```
include("header.php");
```

5. saját program elemek
6. az oldal keretének második része (jobb oldali rész és lábléc):

```
$index = 0;
include("footer.php");
```

A modul saját könyvtárában helyezzük el a nyelvi fájlokat is a `languages/` alkönyvtáron belül. A nyelvi állományok képzésének szabályait figyelembe véve kell azokat létrehozni, de erre most nem térünk ki. A magyar nyelvű fájl neve a következő: `lang-hungarian.php`.

Ahhoz, hogy a modul elérhető legyen, be kell kapcsolnunk a PHP-Nuke adminisztrációs felületén a Modulok menüpont alatt. Ezzel már készen is vagyunk, és a Kviz modul a `modules.php?name=Kviz` hivatkozás alatt elérhetővél vált.

Adminisztrációs felület

Amennyiben a modulunk adminisztrációs felületet is igényel, akkor azt külön ki kell alakítanunk hozzá. Először is a karbantartó oldalt megvalósító állományt kell elkészíteni, amit az admin/modules/ könyvtárban kell elhelyezni: ez a példánkban a `kviz.php`. A fájl szerkezeti követelménye, hogy csak függvényeket, és a függvények eléréséhez szükséges elágaztató feltételeket, azaz switch vagy if utasításokat tartalmazhat. Az elágaztatás a kapott `op` paraméter alapján történik. Természetesen a fájl elkészítése során itt is ügyelnünk kell arra, hogy egy meglévő rendszerbe illesztjük azt be. A következő utasítások teremtik meg a kapcsolatot az adminisztrációs alrendszerrel:

1. a fájl direkt hívását megakadályozandó:

```
if (!eregi("admin.php", $_SERVER['PHP_SELF'])) {
 die ("Access Denied"); }
```

2. azonosításhoz szükséges beállítás:

```
$aid = trim($aid);
```

3. az adminisztrátor jogosultságainak lekérdezése:

```
$result = sql_query("select radminarticle,
radminsuper from ".$prefix."_authors where
aid='$aid'", $dbi);
list($radminarticle, $radminsuper) =
sql_fetch_row($result, $dbi);
```

4. a lekért jogosultságok vizsgálata:

```
if (($radminarticle==1) OR ($radminsuper==1)) {
```

5. saját programrészek
6. a jogosultságvizsgálat másik ága:

```
} else { echo "Access Denied"; }
```

Az előző pontban ismertetett fájlt a `admin/case/` könyvtárban elhelyezett fájl hívja meg. Tehát el kell készítenünk ezen fájlt is, aminek neve a példánkra levetítve: `case.Kviz.php`. Ebben meg kell adni az admin modul által várt `op` paramétereket, és magát a modul állományt megnyitó utasítást, természetesen itt is a megfelelő szabályok szerint.

Utolsó lépésként egy menüpontot kell létrehozni az adminisztrációs menüben, amivel az általunk írt oldalra tudunk ugrani. Ezt szintén egy állomány létrehozásával tehetjük meg.

Biztonság

A biztonság kérdésénél alapvetően két dologra kell kitérnünk, az egyik a biztonságos működés kérdése, a másik pedig az illetéktelenekkel szembeni védelem a megfelelően kialakított jogosultságokkal.

A biztonságos működés egy eléggé tág területet fed le, a funkcionalitástól kezdve egészen a programhelyesség bizonyításáig. A következőkben csak a legalapvetőbb pontokat szedjük össze. Ezek a következők:

1. A rendszer egy adott funkciójának végrehajtása során azokat a tevékenységeket hajtsa végre, amiket elvárunk tőle.
2. A vissza nem fordítható tevékenységekről értesítse a felhasználót, esetleg kérjen megerősítést a végrehajtás előtt.
3. Az ember-gép kommunikáció zavarainak elkerülése érdekében a rendszer mindig kellő mennyiségű és tartalmú információt közöljön a felhasználóval.

A jogosultsági kérdéssel nem igazán kell foglalkoznunk, hiszen ha megfelelően illesztjük egyedi modulunkat a Nuke rendszerbe, akkor azt a rendszer elvégzi helyettünk.

Horváth Zoltán

2002-ben került kapcsolatba a PHP nyelvvel, akkor kezdett el webes alkalmazásokat fejleszteni. A nyelvet a PHP-Nuke használatával ismerte meg, amelyhez számos egyedi modult készített, és a meglévő modulokat a megrendelői igényekhez alakította. A fejlesztett alkalmazások adatbázisháttérét MySQL adatbáziskezelő rendszerben valósítja meg. Kapcsolatba került a Netkey-Grouppal, ahol jelenleg weboldalak programozási munkáit látja el. Tanulmányait a Széchenyi István Egyetemen végezte, és szakdolgozatának témája is a PHP nyelvre épült: Általános Internetes Jelentkező Rendszer. A jövőben továbbra is ezen a területen szeretne tevékenykedni, és az internet terjedésével növekvő webes igényeket minél szélesebb körben kielégíteni.

Objektum-orientált programozás a gyakorlatban

Az objektum-orientáltság

Először tisztáznunk kell, hogy mit nevezünk objektumnak, valamint miért érdemes használni. A PHP 4-ben az objektum típusokat a `class` (osztály) kulcsszóval definiáljuk. Az osztály összetartozó változókat és függvényeket definiál. Az előbbieket tagváltozóknak vagy tulajdonságoknak, az utóbbiakat metódusoknak vagy tagfüggvényeknek is nevezik. Az osztályokból létrehozott példányokat nevezzük objektumoknak, vagy még hangsúlyosabban objektumpéldányoknak. A PHP az objektumokhoz a megszokott változó szintaktikát nyújtja.

Ha egy feladat egy bizonyos bonyolultágot meghalad, elkerülhetetlen, hogy különálló részekre bontsuk. Ez természetesen megoldható a hagyományos függvények segítségével is, ám a közösen kezelt adatokat ilyenkor vagy globális változókban kell átadnunk az egyik függvényből a másikba, vagy különböző állapotváltozókat kell hurcolnunk a kódban paraméterek formájában. Az objektumok lehetővé teszik, hogy az összetartozó adatokat és műveleteket együtt kezeljük, és egy jól definiált felületet biztosítsunk az osztály által megvalósított feladathoz kapcsolódó műveletek elvégzéséhez. Lehetőségünk van néhány alaposztály (adatbáziskezelés, munkamenetek kezelése, stb.) definiálásával a konkrét adatbázisokhoz vagy munkamenet adattárolási módokhoz tartozó megvalósítást különböző fejlesztőkre bízni anélkül, hogy a kész kódok későbbi ütközésétől tartanunk kellene. Érdemes ilyenkor egy központi szerveren tárolni az állományainkat, ellátva valamilyen verziókezelő szoftverrel, így a változásokról minden fejlesztő értesül. Erről részletesebben Bártházi András előadásában esik szó.

A PHP kényelmes, függvény alapú programozását sok fejlesztő nem szereti feladni, pedig az objektum-orientált programozásra átváltani nem is olyan nehéz, mint azt néhányan gondolják.

A gyakorlati bemutató alkalmával egy MySQL adatbázist kezelő osztállyal fogunk megismerkedni közelebbről. Aki használt már MySQL-t, vagy bármilyen adatbázist, tudhatja, hogy a kapcsolódás, és a lekérdezések mind-mind hosszabb kódot igényelnek az eredmények nyilvántartása, ellenőrzése miatt. Ha a programunkban sok ilyen lekérdezés szerepel, könnyen előfordulhat, hogy nehezen igazodunk ki a kódunkon. Az ismétlődő kódok egy kisebb feladat megoldása során sem tesznek jót, amikor azonban egy nagyobb projektben kell valamit megszámlálhatatlan helyen kijavítanunk, az igazán frusztráló lehet. Ekkor térül meg az a látszólagos többletmunka, amit egy osztály elkészítésébe fektettünk, és akkor még nem is említettük az újrafelhasználhatóságot, azaz, hogy egy jól megírt osztály könnyűszerrel felhasználható újabb projektekben is.

Ha az adatbázissal kapcsolatos problémáinkat függvényekkel szeretnénk megoldani, némiképp egyszerűsödik a helyzet. Annak érdekében azonban, hogy programunk minden része elérhesse a kapcsolatot vagy egy korábbi SQL lekérdezés eredményét, globális változókat kell használnunk. Ezek biztonsági problémákat vethetnek fel. Félreértés ne essék, globális változókkal is lehet biztonságos kódot készíteni. Az átláthatóságot azonban semmiképpen sem segítik. Objektumok használatával a globális változók elkerülhetőek, ráadásul a forráskód is rövidíthető, kezelhetőbbé tehető, később egyszerűen módosítható.

A PHP 4 képességeiről

Ezen gyakorlati bemutató célja, hogy bevezetést adjon az objektumok világába azok számára, akik még nem vállalkoztak ennek a hatékony programozási módszernek az elsajátítására. A PHP 4-es nem nevezhető teljes értékű objektum-orientált nyelvnek, számos olyan képességgel nem bír, melyek más népszerű objektum-orientált rendszerekben jelen vannak. Ezeket a hiányosságokat egytől egyik pótolja a PHP 5. Bemutatónk szempontjából mégis elegendő a PHP 4-es képességeit figyelembe venni. Kolman Nándor részletezi előadásában a PHP 5-ös új objektumokhoz kapcsolódó szolgáltatásait.

Az osztály

Az osztályok PHP-ben külön típust képviselnek, definiálva az összetartozó tagváltozókat és metódusokat. Egy objektum mindig egy osztály példánya. Osztályt a `class` kulcsszóval deklarálhatunk.

Az osztály változói

A tulajdonságok a teljes objektum számára elérhetőek, így szükségtelenné válnak a globális változók. A `var $foo` egy foo nevű változót hoz létre az objektumban. Hivatkozni `$this->foo` formátumban lehet rá, a megszokott `$foo`-val szemben, ahol a `$this` az aktuális példányra utal.

A PHP 5 bevezeti a privát és a védett típusú változókat. A privát típusú változót csak abban az objektumban használhatjuk, amelyben definiáltuk, a védett típusút akár kiterjesztett objektumban is elérhetjük.

Konstruktor

A konstruktor az a metódus, ami egy új objektum `new` kulcsszóval történő létrehozásánál automatikusan lefut. A PHP 4-ben egy metódus attól lesz konstruktorrá, hogy a neve megegyezik annak az osztálynak a nevével, ahol definiálták. Ha egy kiterjesztett osztályban nem definiálunk konstruktort, akkor a szülő osztály konstruktora fut le (ha létezik). A PHP 5 egységesen a `__construct()` metódust tekinti konstruktornak.

Destruktor

PHP 4-ben nincs destruktor, ami arra szolgálna, hogy az objektum által lefoglalt erőforrásokat felszabadítsa. Erre

nem túl gyakran lehet szükségünk, hiszen a szkript lefutása után ezt a PHP automatikusan megteszi. Ha mégis szükségünk van a destruktor funkcióhoz hasonló működésre, akkor a `register_shutdown_function()` függvény használatával jelölhetünk meg egy metódust a futás befejezésekor végrehajtandónak. Mivel a metódus a kimenet elküldése után hívódik meg, a böngészőnek üzenetet nem írhatunk már ki. A PHP 5 bevezeti a destruktorokat `__destruct()` néven.

Metódusok

Az objektum változóin a metódusokkal hajthatunk végre műveleteket. A metódus egy többé-kevésbé átlagos PHP függvény, csupán annyi az eltérés, hogy a tagváltozókra `$this->foo` formában kell hivatkozni. Egy egyszerű osztály, mely egy tagváltozójának értékét tudja kiírni konstruktorával:

```
<?php
class phpconf {
  var $text = "PHP Konferencia";
  function phpconf() {
   print $this->text;
  }
}
new phpconf();
?>
```

Ha a szkriptet lefuttatjuk – az előbbiek után nem túl meglepő módon – kiírja: PHP Konferencia.

Öröklődés, avagy metódus cseréje

Előfordulhatnak olyan esetek, amikor egy osztály csak többé-kevésbé felel meg a számunkra. Lehetséges, hogy szeretnénk egy-két metódust másképp megvalósítani, de nem szeretnénk belenyúlni a forráskódjába, szétroncsolva a jól definiált felületet és működést. Ilyen esetekben kiterjeszthetjük osztályunkat, és az úgynevezett leszármazott osztályban felülbírálhatjuk a metódust. A változóink, a konstruktor függvényünk és a többi metódus öröklődik, így szabadon felhasználható a kiterjesztett osztályban. Az öröklődés többszintű is lehet, egy származtatott objektum is alapja lehet kiterjesztésnek.

Lássuk mindezt egy példán keresztül is! Készítünk egy általános konferenciákat kezelni képes osztályt, amely a konferencia nevének és dátumának beállítását teszi lehetővé. Képes információt adni a rendezvényről, azaz a nevét vissza tudja adni. Logikus, hogy a konferencia neve és dátuma azonosítja csak egyértelműen a rendezvényt, ezen be tudjuk mutatni a kiterjesztést is.

```
<?php
class conference {
  var $text;
  var $date;
  function conference($name, $date) {
    $this->name = $name;
    $this->date = $date;
  }
  function getInfo() {
    return $this->text;
  }
}

$phpconf = new conference("PHP Konferencia",
➥ "2004.03.27.");
print $phpconf->getInfo();
?>
```

A következő példa az előző kódot alapul véve az egyértelműbb azonosításra alkalmas mindkét adatot megadja:

```
<?php
class extconf extends conference
{
  function getInfo() {
    return $this->name . ' ' . $this->$date;
  }
}
$ephpconf = new extconf("PHP Konferencia",
➥ "2004.03.27.");
print $ephpconf->getInfo();
?>
```

A kiterjesztett osztály egyszerűen lecseréli a `getInfo()` metódust egy másikra, amelyben nem csak a rendezvény neve, hanem a dátuma is szerepel a visszaadott karaktersorozatban.

Mire fogjuk használni?

A gyakorlati bemutató során objektum-orientált módon készítünk egy MySQL adatbázist kezelő osztályt. A témát azért választottam, mert a PHP MySQL függvényei nagyon jó objektum-szerű tulajdonságokkal rendelkeznek, jól illusztrálható velük az objektumok viselkedése. A PHP-be épített `mysql_fetch_object()` függvény használatával a lekérés eredményét objektumban is kérhetjük, ez segíti az objektum-központú szemlélet kialakítását. Mindemellett az sem elhanyagolható szempont, hogy a fejlesztéseknél legtöbbször valamilyen adatbázist használunk.

Mivel demonstrációs célra készítjük az osztályt, nem támasztjuk a legkomolyabb elvárásokat vele kapcsolatban. Megköveteljük, hogy a lehető legkevesebb bemenő paraméter megadása mellett kezelje a MySQL lekéréseket, rendelkezzen belső hibamegjelenítő mechanizmussal, és minél több lépést magában kezeljen. Az osztály kezelni fogja az adatbázishoz való kapcsolódást, az adatbázis kiválasztását, lekéréseket, azok visszaadását objektumként vagy tömbként, a rekordok számát, valamint egy egyszerű hibamegjelenítőt.

Összefoglalás

Az objektum-orientált programozás megkönnyíti a nagyobb fejlesztési munkákat, mivel egységes illesztőfelületet nyújt a programozóknak. Nem elhanyagolható az sem, hogy egy megfelelően megírt osztály más projektekben is felhasználható, így csak egyszer kell megírni – de jól. Az objektum-orientált programozás rákényszeríti a programozót, hogy átgondolja a feladatot mielőtt nekiállna programozni, ezért csak javasolni tudom az alkalmazását. A PHP 5 számos szolgáltatásának használatakor elkerülhetetlen lesz az objektum szintakszis ismerete, a PEAR kódkönyvtár csomagjainak alkalmazásához pedig már most kell tudnunk objektumokat használni.

Bóna László Márton

A CDM Számítástechnikai Szakképző Iskola multimédia-fejlesztő, valamint a Zrínyi Miklós Nemzetvédelmi Egyetem Bólyai János Katonai Műszaki Főiskolai Kar biztonságtechnikai mérnök szakos hallgatója. Szabadidejében, autodidakta módon tanulta meg a PHP-t, a közösség Ene néven ismeri. A DesignProg.net webdesign, programozási és hálózatbiztonsági portál egyik szülőatyja és fejlesztője, tagja az NJSZT-WFSZ-nek is.

A PHP 5 újdonságai

A PHP 5 sokáig váratott magára, mégis megérte a várakozást, hiszen egy olyan megújított nyelvet kapunk, amely felveszi a versenyt a legmodernebb nyelvek által kínált lehetőségekkel is. A PHP 4-hez képest a legnagyobb előrelépést az objektum-orientált programozás támogatása jelenti.

Osztályok kezelése

A PHP 5 fejlesztői az osztályok és objektumok kezelését teljesen újraimplementálták, így megfelel a legszigorúbb OOP követelményeknek is. Az újraírás során nagy figyelmet fordítottak arra, hogy a PHP 4-gyel futó kód lehetőleg módosítások nélkül végrehajtható legyen az új értelmezővel is.

Referenciák

Az egyik legfontosabb változás, hogy az objektumokat reprezentáló változók már nem az objektum értékét hordozzák, hanem csak egy referenciát jelentenek az adott példányra.

PHP 4-ben amikor egy változó értékét egy másikba másoltuk, vagy egy függvény paramétereként használtuk, típustól függetlenül a teljes érték lemásolódott. Ezért például egy objektum esetén a paraméterezett függvény belsejében történő változtatások nem hatottak az eredeti példányra. A referenciaképző operátorral megkerülhető volt a probléma, a PHP 5-ben azonban az objektumok alapértelmezett referencia alapú kezelésének köszönhetően sokkal könnyebb az életünk.

Láthatóság

A PHP osztály-kezeléséből régóta hiányzik annak lehetősége, hogy a programozó eldönthesse, egy adott metódust vagy tagváltozót honnan lehet elérni. Erre született az a konvenció, mely szerint a csak az objektum által használható metódusok nevét aláhúzás karakterrel kell kezdeni. Sajnos ezt a futtatókörnyezet nem kényszerítette ki.

A PHP 5-ben három új láthatósági direktívát használhatunk metódusokra és tagváltozókra. A változó, vagy függvény definíciója elé írt `private` kulcsszó hatására a tagot csak az objektum belsejéből lehet elérni. Szemben a `protected` direktívával megjelölt tagokkal, amelyeket az osztály leszármazottaiban is használhatjuk. A harmadik és egyben az alapértelmezett direkítva a `public`, amely tudatja az értelmezővel, hogy a tag az osztályon kívülről is használható.

Absztrakció

Az új motor lehetőséget biztosít arra, hogy osztályokat és metódusokat absztraktnak minősítsünk. Akkor használjunk absztrakt metódusokat, ha jelezni szeretnénk, hogy ezek szükségesek az osztály leszármazottjaiban. A csupa absztrakt metódusból álló osztályt kötelező `abstract` kulcsszóval ellátni és ezek egyáltalán nem példányosíthatók.

Interfészek

Az objektumorientált programozás egyik fontos kérdése, hogy egy osztály csak egy, vagy több osztályból is származhasson. A PHP 5-ben továbbra sem megengedett a több ős megadása, e helyett, a sok más nyelvben is használt interfészeket vezették be.

Szigorú osztály típus paraméterek

A PHP-ban a típusok szabadon használhatók egymás helyén is, mivel az értelmező gondoskodik a köztük lévő konverzióról. A PHP 5 lehetőséget ad arra, hogy a függvények paraméterváltozói előtt megadjunk egy osztály típust, ekkor az értelmező futásidőben a függvény meghívásakor ellenőrzi, hogy a változó a megadott osztály vagy leszármazottjának példánya-e, és hibát jelez, amennyiben nem. Ez a lehetőség csak osztály típusokra vonatkozik, beépített típusokra nem.

Objektumok klónozása

Ha objektumokat használunk, előfordul, hogy egy adott példány minden tulajdonsága megfelelően van beállítva ahhoz, hogy sablonként használjuk egy másik objektum létrehozására. A PHP 4 ez esetben az objektum tartalmát egy az egyben lemásolta. Habár ez az esetek nagy részében jó működést eredményez, néha mégis szükség van az új objektum létrehozásakor bizonyos tulajdonságok (pl. egyedi azonosító, erőforrásazonosítók) megváltoztatására.

A másolás segítésére minden objektum tartalmaz egy `__clone()` metódust, amely úgy van defniniálva, hogy tökéletes másolatot hozzon létre az objektumról. Amennyiben nem ez a megfelelő működés a metódus felüldefiniálásával új másolási logikát építhetünk objektumainkba. Az új `clone` kulcsszóval tudunk másolatot készíteni egy példányról.

Egységesített konstrukorok

PHP 4-ben a konstruktorok nevének meg kellett egyeznie az osztály nevével. Az esetek nagyrészében egy leszármazott osztályban meghívjuk a szülő konstruktorát, ezért egy PHP 4 osztály hierarchiában egy osztály áthelyezésekor figyelni kellett a szülő konstruktor nevének átírására. Ezt elfelejtve nehéz volt rájönni a rossz működés okára.

A PHP 5 bevezeti a `__construct()` metódust az osztályok konstruktoraként. Természetesen a kompatibilitás miatt a régi forma is használható, bár új kódoknál nem ajánlott.

Destruktorok

Sok objektum használ erőforrásokat (adatbáziskapcsolat, fájl mutatók stb.), amiket „illik" felszabadítani amikor már nincs rájuk szükség, leggyakrabban az objektum megszűnésekor. A PHP 5 lehetőséget nyújt osztályonként egy `__destruct()` metódus defniálására, amely az objektumhoz tartozó tárterület felszabadítása előtt kerül meghívásra. Hasonlóan a konstruktorhoz, a destruktornál sem hívódik meg automatikusan a szülő osztály destruktora, arról a programozónak kell gondoskodnia.

Statikus tagok

Egy osztály tagjait a `static` direktívával ellátva azok osztályszintre emelkednek, azaz az osztály nevével használhatók anélkül, hogy az osztályból egyetlen példányt is létrehoztunk volna. Ezért a statikusként definiált metódusokban a `$this` nem használható, a statikus tulajdonságok pedig minden példányban közösek.

Konstansok

A PHP 5 osztályokban a `var` kulcsszó mellett lehetőség van a `const` használatára. Az így megjelölt változók osztályszinten konstansok lesznek, azaz az osztály és a konstans nevével lehet rájuk hivatkozni, valamint értékük nem megváltoztatható. Mivel a konstansok az osztályhoz tartoznak, ezért nem foglalnak példányonként külön memóriát.

Konstansként érdemes az osztály működésével kapcsolatos nem változó értékeket definiálni.

Visszaadott objektumok kezelése

PHP 4 esetében nem lehetséges egy függvény által visszaadott objektumra közvetlenül hivatkozni `func()->` valami formában, a visszaadott értéket először egy változóban kell elhelyezni.

A PHP 5-től a visszaadott objektumok közvetlenül kezelhetőek. Érdemes azonban vigyázni, hiszen amennyiben az így visszakapott objektumra egy változó sem hivatkozik, az a függvényhívás és a tag értelmezés után azonnal a szemétgyűjtő kezére jut.

instanceof operátor

A PHP 5-ben bevezetésre kerül az `instanceof` operátor, amely megmondja, hogy a bal oldalon álló objektumreferencia a jobboldali osztály példánya-e. Az operátor akkor is igazat ad vissza, ha a bal operandus az osztály leszármazottjának példánya.

A többszörös öröklés helyett bevezetett interfészek is vizsgálhatók az `instanceof` operátorral. Ez esetben akkor kapunk igaz értéket, ha az objektum osztálya implementálja a jobb oldalon megadott interfészt.

Tulajdonságok beállítása és lekérdezése

PHP szkriptjeikben sokan írtak `getTulajdonsag()` és `setTulajdonsag()` függvényeket egy tulajdonság értékének lekérdezésére és beállítására.

A PHP 5-ös verziója lehetőséget biztosít olyan tulajdonságok használatára is, amelyek úgy használhatók, mint a tagváltozók, azaz értékadások bal oldalán is szerepelhetnek. Csak két függvényt kell az osztályban definiálni: a `__get($property_name)` függvény paraméterként megkapja a tulajdonság nevét, és annak értékével kell visszatérjen. A `__set($property_name, $property_value)` függvénynek pedig az új értéket kell beállítania.

Kivételkezelés

A más nyelvekben már régóta használt kivételkezelés a PHP-ban is bevezetésre kerül. Lényege, hogy a programban előforduló hibák nem a kód futásának azonnali leállításához vezetnek, hanem egy kivételt „dobnak”. Ezeket a kivételeket a programozó speciális `try-catch` blokk segítségével megfoghatja és a hibát kezelheti. A kivétel egy objektum, amelynek osztálya a hiba típusa.

A dobott kivételhez a PHP sorra nézi az egymásba ágyazott blokkokat bentről kifelé egészen addig, amíg olyan `catch` ágat nem talál, amely a kivételnek megfelelő osztály kezelésére íródott. Ezután végrehajta a `catch`-hez tartozó kódrészletet, majd folytatja a végrehajtást a megtalált kivételkezelő blokk lezárása utáni első utasítással. A nem megfogott kivételeket az értelmező kapja el és hibaüzenetet ír a kimenetre.

A kivételkezelést használó kód sokkal megbízhatóbbá és hibatűrőbbé válik, mivel tartalmazza a hibák kezelésére íródott logikát is. Másik előnye, hogy bizonyos hibák globálisan kezelhetők. Például egy adatbázis kapcsolati hibát elég csak a legfelső szinten megfogni, nem kell minden függvényt felkészíteni arra, hogy az adatbázis elérhetetlen lehet.

Az interneten szörfözve sokszor látunk arra példát, hogy az oldalon valami hiba történt. Jobb esetben ezt a hibát elrejtik, de ilyenkor az oldal vagy félig, vagy teljesen nem töltődik be. A kivételek elfogásával a hiba megtörténte esetén speciális tartalom mutatható.

Fontos azonban megjegyezni, hogy a régi „először megvizsgálom az értéket és azután cselekszem” hibaelkerülési módszernek a kivételkezelés mellett még mindig van létjogosultsága, mivel a kivételkezelés költsége egy feltétel vizsgálathoz képes nagy.

Kolman Nándor

A Budapesti Közgazdaságtudományi Egyetem eltévedt hallgatója. Három éve foglalkozik PHP-re épülő webes alkalmazások, portálok és egyedi weboldalak tervezésével és készítésével.

Zenetár a webszerverünkön, avagy XML használata a PHP 5-ben

Bóna László Márton előadásában illetve cikkében a PHP 4-es képességeinek megfelelő bevezetést kaphattak az érdeklődők az objektum-orientált programozásba. Továbbra is az objektumok világában maradunk, azonban áltlépünk a PHP új verziójának használatára. Az ötös változatra való áttérés egyik legfőbb vonzereje a nagy mértékben objektum-orientált kialakítás, valamint az új és izgalmas kiterjesztések, melyek megkönnyítik mindennapi munkánkat. Az objektumok kezelését érintő újdonságokra itt nem térünk ki, hiszen Kolman Nándor alaposan körbejárja a kérdést. Az új változat néhány jól használható beépített szolgáltatásáról lesz szó. Ehhez azonban egy új fogalmat vezetünk be, és ez az XML.

Az XML

Manapság lépten-nyomon hallható ez a mozaikszó, mégis sokan nem tudják, hogy pontosan mit takar ez a név, és mire használható. Először vegyük át az XML nyelvvel kapcsolatos tudnivalókat!

Az XML név egy rövidítés, jelentése: eXtensible Markup Language, azaz magyarul: kiterjeszthető leíró nyelv. A szabványt a W3C keretében kisebb-nagyobb vállalatok delegáltjai fejlesztették ki, és továbbra is a konzorcium kezeli az ajánlást. Ezzel egy gazdasági érdekek és a szoftvergyárak szeszélye felett álló egységes nyelv lett. Az XML születésének egyik fő mozgatórugója a különböző médiumokon terjesztett, struktúrált, elsősorban szöveges adatok egységes formátumának kialakítási igénye volt. Önleíró nyelvnek is nevezik, az adatok formázása helyett összetett adatszerkezetek megadására szolgál, amelyből következik, hogy a leírás adott struktúrát is rögzít.

Az XML formátum első közelítésben hasonlatos a HTML-hez, azonban a leírás módjára néhány szigorúbb megkötés is adott, mint például, hogy egy elemnek – ha másképp nincs jelölve – kell lennie egy záró párjának is, valamint, hogy az elemek tulajdonságainak értékeit mindig idézőjelek közé kell tennünk.

Az XML-t szokás köztes nyelvnek is mondani, mivel az XML valójában csak más nyelvek leírására ad lehetőséget. Ilyen nyelvek például a WDDX, az XHTML vagy az RSS, amelyekben már kötött elem lista van, és csupán egy adott célra szolgálnak. Ahogy a WDDX leírására, vagy más nyelvek definiálására alkalmas, ugyanúgy a saját feladatainkban is felhasználhatjuk. Saját tartalomkezelő rendszerünk konfigurációs állománya is lehet XML nyelvvel leírt adathalmaz. Ekkor – természetesen – csak a nekünk szükséges adatokhoz találunk ki beszédes elem neveket, és csak ezeket az elemeket értelmezzük. Az XML használható például a Macromedia Flash újabb változataival is, így könnyen lehetővé válik olyan Flashes oldalak készítése, amelyek dinamikus tartalmát a PHP szolgáltatja.

Zene, zene, zene

Az XML forrás

Első lépésben készítsünk mindjárt egy XML-ben leírt adathalmazt, amelyet a későbbi példáink során használni fogunk. Az adathalmazunkat egy PHP állományba fogjuk beírni az egyszerűség kedvéért, bár természetesen nyugodtan írhatnánk külön állományba is. Dalainkat néhány jellemzővel fogjuk csupán felruházni.

```
<?php
$xmlData = <<<XML
<?xml version='1.0' standalone='yes'?>
<tracks>
 <track>
  <title>In the End</title>
  <author>Linkin Park</author>
  <album type="album">Hybrid Theory</album>
 </track>
 <track>
  <title>Summer of '69</title>
  <author>Bryan Adams</author>
  <album type="collection">Best of...</album>
 </track>
 <track>
  <title>Katjusha</title>
  <author>Leningrad Cowboys</author>
  <album type="collection">Thank You Very
➥ Many</album>
 </track>
</tracks>
XML;
?>
```

A példánkon jól látszanak a bevezetőben említett sajátosságok, amelyek a HTML nyelvből már ismerősek lehetnek. Ilyenek például, hogy az elemeket „kacsacsőrökkel" jelöljük, illetve, hogy van nyitó és záró párjuk. Az esetleges attribútumokat is a HTML-hez hasonlóan használjuk: a tulajdonság nevét egyenlőségjel követi, majd idézőjelek között az értéke következik. Vegyük észre, hogy az elemek nevei is a nekünk leginkább megfelelően alakíthatók ki, így a számok, a szám címe, előadója, mind-mind egész egyszerűen elnevezhetők, például az angol megfelelőjükkel. Jól látszik a példa adatokon a struktúráltság is, ami nem túl nagy és bonyolult adatmennyiség mellett szabad szemmel is könnyen olvashatóvá teszi az XML dokumentumokat.

Feldolgozás

A PHP 4-es változata is képes bizonyos szintű XML kezelésre, ez azonban sokszor nehézkes és külső kiterjesztések használatát igényli. Az XML dokumentumok két fő feldolgozási módjára is kapunk megoldást a PHP 4-el, a SAX (Simple API for XML) és a DOM (Document Object Model) technológiákra. A SAX azon alapul, hogy az XML adathalmaz különböző elemeinek feldolgozása eseményeket vált ki, és ezekhez az eseményekhez kezelő függvényeket rendelhetünk, míg a DOM a teljes dokumentum feldolgozása után szabványos elérést tesz lehetővé egy jól definiált felület segítségével. A PHP 5-ös új XML

kezelési és feldolgozási szolgálatatásokat nyújt, az újonnan bevezetett objektum-modellre építve. Ezért az új XML-hez kapcsolódó eszközök használatához legalább az objektumok szintaktikáját ismernünk kell. Miben nyújt még többet a PHP 5?

- Nem kell külön XML értelmezőt létrehozni, és azon keresztül feldolgozni az adatainkat, az objektum-orientált kezelésnek köszönhetően a PHP 5 objektumai leveszik a vállunkról a különböző értelmezőkkel való bajlódás terhét.
- Az új XML kezelő kiterjesztések egységesen a libxml2 kódot használják bázisként, amely a PHP 5-ös változatának natív része, így nem kell külön modulként betölteni, és minden PHP 5-öt használó kiszolgálón elérhető lesz. Ez két szempontból is igen előnyös: Egyrészt a libxml2 gyorsabb, mint az előző változatokban használt Sablotron illetve Expat modulok, másrészt a közös alap megteremti a lehetőséget a különböző XML kezelési módok közötti könnyű átjárhatóságra.
- Az egyszerű XML kezelhetőség érdekében nem csak a DOM és SAX feldolgozási módokat támogatja immár a nyelv, hanem egy egyszerűen használható SimpleXML nevű kiterjesztés is elérhetővé vált, amellyel a továbbiakban foglalkozni fogunk.

Vegyünk most egy egyszerű példát: listázzuk ki számainkat. Meglátjuk a gyakorlatban, hogy tényleg milyen egyszerű az XML adathalmaz értelmezése a PHP 5-ben.

```
<?php
include 'tracks.php';
//- A számok listáját tartalmazó állomány betöltése

$xmlObj = simplexml_load_string($xmlData);
        //- Objektum-struktúra felépítése

foreach( $xmlObj->track as $track ) {
        echo $track->title.'<br>';
        //- A szám címének kiírása
}
?>
```

Jól látható, hogy a kapott XML objektum a hagyományos PHP-s eszközökkel kezelhető, a számokon végig tudunk lépkedni, és az objektum szintaktikát ismerve a szám címét is megkaphatjuk. Ezek után lássunk az attibútumok kezelésére is egy példát. Jelenítsük meg most az albumaink címeit, mellette zárójelben az album típusára vonatkozó információt...

```
<?php
include 'tracks.php';
//- A számok listáját tartalmazó állomány betöltése

$xmlObj = simplexml_load_string($xmlData);
        //- Objektum-struktúra felépítése

foreach( $xmlObj->track as $track ) {
        echo $track->album;
        //- Az album címének kiírása
        echo ' ('.$track->album['type'].')<br>';
        //- Az album típusa
}
?>
```

Az XML elemek feldolgozására általában nem ciklikusan van szükségünk. Tipikus, hogy keressük egy XHTML dokumentumban lévő összes linket vagy az oldal címét. Ezt az elemek egyenként történő ellenőrzésével nehéz elvégezni, és nem is túl hatékony. Létezik azonban egy XPath nevű szabvány, amely lehetővé teszi, hogy röviden megjelöljük, hogy mely elemekre vagyunk kíváncsiak a dokumentumból. Az XPath alapvetően a könyvtárstruktúra analógiát használja az egymásba ágyazott XML elemek elérésére, de ezt jóval kiterjesztve számos lehetőséget nyújt. A következő példában egy egyszerű kiválasztót alkalmazunk: a `//author` minden olyan elemre illeszkedik, melynek neve author, bárhol is legyen az a dokumentumban.

```
<?php
include 'tracks.php';
//- A számok listáját tartalmazó állomány betöltése

$xmlObj = simplexml_load_string($xmlData);
        //- Objektum-struktúra felépítése

foreach($xmlObj->xpath('//author') as $author) {
        echo $author.'<br>';
        //- Az előadó kiírása
}
?>
```

Módosítás

Az SimpleXML támogatja az XML adatok módosítását is, melyet a PHP objektum-reprezentációjában végezhetünk el végül előállítva a megváltozott XML dokumentumot. Példánkban most módosítsuk az egyik szám címét:

```
<?php
include 'tracks.php';
//- A számok listáját tartalmazó állomány betöltése

$xmlObj = simplexml_load_string($xmlData);
        //- Objektum-struktúra felépítése

$xmlObj->track[3]->title = 'Ballad Of The
➥ Leningrad Cowboys';
        //- Megváltoztatjuk a szám címét...

echo $xmlObj->asXML();
        //- Az XML adathalmaz kiíratása

?>
```

Végszó

A PHP 5-ben bevezetett új XML kezelő képességek gyorsabb, jelentősen egyszerűbb, és az objektum-orientált szemléletet kihasználó megoldást nyújtanak a struktúrált adatok kezeléséhez kapcsolódó problémáinkra.

Linkek

XML – http://www.w3.org/TR/REC-xml

XPath – http://www.w3.org/TR/xpath

libxml2 – http://www.xmlsoft.org/

Simple XML – http://hu.php.net/manual/en/ref.simplexml.php

Ercsey Balázs

A PHP-s közösség tagjai laze néven ismerik. Öt éve foglalkozik internetes fejlesztésekkel, elsősorban PHP nyelven. Saját tartalomkezelő rendszere két verziót ért meg, most egyéb elfoglaltságok, feladatok miatt nem fejleszti. Jelenleg a netpeople.hu fejlesztési vezetőjeként a napi feladatok mellett a cég saját tartalomkezelőjének, az nPirio-nak a fejlesztésében vesz részt.

PEAR – a PHP gyümölcse

A PEAR (PHP Extension and Application Repository) egy olyan egységes kódgyűjtemény a PHP nyelven programozók részére, mely által a különböző célokat szolgáló kódrészletek nem szétszórva, hanem egységesen, összegyűjtve találhatók meg az interneten.

A többi kódkönyvtárral ellentétben nem csak a projektek kategorizált listájaként szolgál, hanem teljes infrastruktúrális hátteret biztosít az igényeinknek megfelelő kód megtalálására, vagy akár a fejlesztésben történő részvételre is. A PEAR a forráskódokat csomagok formájában kezeli, amelyek száma a folyamatosan növekvő létszámú fejlesztői csoportnak köszönhetően emelkedik. A legkülönfélébb csomagok állnak rendelkezésre a gyorsítótárazástól az adatbáziskezelésen át a HTML sablonokig. A gyűjtemény nem előre meghatározott irányban fejlődik, az egyes csomagok olyan sorrendben kerülnek be a raktárba, amilyen sorrendben elkészítik azokat.

A PHP-t használók között többen inkább kerülik mások által írt programcsomagok alkalmazását, hiszen nem tudhatják, hogy az adott csomag mennyire hibamentes, hibatűrő, biztonságos, illetve gyors. Ráadásul több csomag együttes használata esetén felléphetnek kompatibilitási problémák is. A PEAR csomagok szigorú minőségbiztosítási ellenőrzésen mennek át, valamint a nagyszámú felhasználói tábor miatt állandó tesztelésnek vannak kitéve. Ezért bízhatunk abban, hogy a stabilnak minősített csomagokkal nem lesz túl sok problémánk. A mérleg másik nyelve a gyors és hatékony alkalmazásfejlesztés, amiben PHP terén a PEAR szinte verhetetlen. A projekt minőségét fémjelzi, hogy a PHP csoport saját weboldalán nyújt helyet a gyűjteménynek: http://pear.php.net/.

A PEAR-t bármilyen alkalmazás fejlesztésekor érdemes lehet használni. Mielőtt egy adott funkciót, illetve szolgáltatást saját magunk fejlesztenénk ki, tallózzuk végig a webhelyén elérhető katalógust, hogy mások nem fejlesztették-e ki már előttünk ugyanazt a funkcionalitást.

Csomagok

A legkülönfélébb csomagok a következő címen tallózhatóak: http://pear.php.net/packages.php. Mindegyikük egy-egy kisebb, önálló projekt dedikált felelősökkel és kiadási időszakokkal. Az esetenként megjelölt függőségekől eltekintve a csomagok külön-külön telepíthetőek, nem kell az összeset feltennünk, ha használni szeretnénk egy szolgáltatást. Minden csomaghoz tartozik egy, vagy több vezető (lead), általában több fejlesztő (developers) és néhány egyéb résztvevő (helpers). A csomagoknak különböző kiadásai vannak, időről-időre megjelenik belőlük egy-egy újabb verzió.

PHP Foundation Classes

A PFC a PEAR magja, mindössze néhány csomagot jelent, melyek nélkülözhetetlenek a PEAR működéséhez. Ezek a csomagok az elérhető legjobb stabilitás érdekében minőségre, általánosságra, együttműködési készségre és kompatibilitásra vannak kiélezve. Automatikusan szállításra kerülnek a PHP legfrissebb verzióival.

Telepítő

A telepítőkészlet a PEAR saját – tar.gz tömörítésre alapozott – csomagformátumát használja. Meglévő hálózati kapcsolat esetén közvetlenül a központi nyilvántartásból dolgozik, de használható letöltött csomagok telepítésére is, amennyiben éppen nem rendelkeznénk aktív internetkapcsolattal. Az eszköznek többféle felhasználói felülete van. Legtöbben a parancssori interfészt (CLI) használják, de elterjedt a webes változata is. A PHP „ablakozós" felületére, a GTK-ra is elkészült már a telepítő működő verziója. Természetesen mindhárom felület platformfüggetlen, és mindegyikük képes csomagokat telepíteni, frissíteni illetve eltávolítani. Lehetőség van a helyileg telepített és a központban rendelkezésre álló csomagok listázására, valamint keresni is tudunk a nyilvántartásban.

A PEAR raktárában jelenleg 210 féle csomag található, melyek a parancssori felülettel a következőképpen telepíthetőek szerverünkre:

```
pear install <csomagnev>
```

Az adott csomag használatához ezt követően mindössze include-olni kell azt saját szkriptjeinkben, és máris használhatjuk a benne lévő osztályokat. Ha nem rendelkeznénk a szerveren parancssori hozzáféréssel, akkor az FTP szolgáltatást használva tölthetjük fel a telepítő kódját, majd a böngészőnkből használhatjuk a kényelmes HTML felületet.

Verziószámozás

A csomagok napról napra fejlődnek. Az egyes fejlesztéseket minden alkalommal feltöltik a központi CVS verziókezelő rendszerbe, ahol az érdeklődő felhasználók, illetve más PEAR fejlesztők letölthetik a legfrissebb változatokat. Ezek automatikusan új verziószámot kapnak, éles környezetben történő felhasználásuk azonban nem javasolt. Egy-egy csomag nagyobb fejlesztési szakaszainak végén megjelentetnek egy-egy új kiadást, amelyek a csomag információs lapján (http://pear.php.net/package /<csomagnév>) érhetőek el. A kiadások .tar.gz állományok formájában tölthetők le, melyek tartalmazzák a csomag működéséhez szükséges összes állományt. Függőség esetén a szükséges további csomagokat külön kell beszerezni.

Könyvtárszerkezet

Minden csomagnak ugyanolyan könyvtárszerkezetben kell bekerülnie a CVS rendszerbe. Ez megkönnyíti a tallózást, illetve segít az esetleges hiányosságok felderítésében. A felhasználókat segítő tartalmak meglétének kényszerítése érdekében a csomagok könyvtárain belül a „docs", „examples" és „test" mappáknak kötelező fáj-

lokat tartalmazniuk. Legalább egy mellékelt példaprogramnak lennie kell a csomagban, a dokumentáció nem kötelező. Néhány egyszerű tesztre is szükség van, a csomag működésének ellenőrzésére. A „data" és a „misc" könyvtárak opcionálisak. A telepítés utáni könyvtárstruktúrát a csomagokhoz mellékelt `package.xml` állomány határozza meg.

Központi nyilvántartás

A csomagnyilvántartás a http://pear.php.net/ címen érhető el. Mind felhasználóbarát (HTML), mind „gép-barát" (XML-RPC) felületet biztosít. A HTML felületen tallózhatunk a csomagok között, elolvashatjuk a PEAR kézikönyvet, amely tartalmazza a csomagok dokumentációját is. A fejlesztők saját nevükkel és jelszavukkal léphetnek be, majd feltölthetik a csomagok kiadásait. Az XML-RPC interfészen keresztül információk kérdezhetők le az egyes csomagokról. Le lehet tölteni a frissítések listáját, illetve különböző adminisztrációs szolgáltatásokat lehet igénybe venni. A telepítő ezt a felületet használja szolgáltatásai megvalósításához.

Kódolási szabályok

A csomagok (de különösen a PFC) szigorú kódolási szabványokat követnek. A kód formátuma kötött – pontosan meg van határozva, hogy hova kell szóközt tennünk, mit kell új sorba írni, stb. A kódolási szabályok megfelelően követve lehetővé teszik, hogy több fejlesztő dolgozhasson egy-egy csomagon párhuzamosan, úgy, hogy a napi fejlesztések mellett is átlátható marad a forráskód.

Ezeket a szabályokat a PEAR felhasználóinak nem kell feltétlenül követniük, azonban ha nincs még megszokott szabályrendszerünk kódjaink írására, akkor praktikus megoldás lehet ennek a széles körben elismert rendszernek a használata. Már most több nyílt forrású projekt a PEAR kódolási szabályokat alkalmazza.

Hibakezelés

Minden PEAR csomag ugyanazon hibakezelési módszert használja: a függvényekből egy meghatározott típusú hibaobjektummal térnek vissza. A függvények visszatérési értékét vizsgálva egyszerűen megállapítható, hogy az adott érték hibaobjektum-e. Amennyiben igen, a hibaobjektum függvényeinek segítségével megállapítható a hiba oka. Ellenkező esetben a függvény által visszaadott érték az elvárt sikeres művelet eredménye, így felhasználható az eredeti célokra.

A PEAR hibakezelési módszerét használhatjuk akár a teljes alkalmazásunkban is. Saját függvényeinkből is visszatérhetünk ilyen objektumokkal, így egy egységes hibakezelési rendszert alakíthatunk ki. Az egyes hibák keletkezésekor különböző viselkedés állítható be, de akár globálisan, a teljes alkalmazásra is megadható, hogy mi történjen egy-egy hiba bekövetkezésekor. Ilyen viselkedések lehetnek például a szkript futásának megállítása, a hibaüzenet kiírása és a futás folytatása, vagy egy callback függvény meghívása az adott hibaobjektummal. A callback függvényen belül a hiba tetszőlegesen feldolgozható.

Felhasználók támogatása

A legtöbb nyílt-forrású projekthez hasonlóan a felhasználók támogatása levelezőlistákon keresztül valósul meg. Jelenleg öt levelezőlista létezik: általános, fejlesztői, dokumentációs, webmester, és CVS figyelő lista. A csomagokat használó programozók az általános listán tehetik fel kérdéseiket, a pear-general@lists.php.net címen (angolul). A listák természetesen archiválásra kerülnek, és tetszőlegesen visszakereshetők. Akinek ideje engedi, személyesen is találkozhat a gurukkal az EFNET #PEAR csatornáján.

Fontosabb csomagok

Az alábbiakban a legfontosabb kategóriákat vesszük sorra, kiemelve a széles körben alkalmazott csomagokat. Különösen a PEAR-rel való ismerkedés során lehet hasznos, hogy az alábbi kategóriákra bontott csomaglistát is megtalálhatjuk a http://pear.php.net/packages.php címen.

PEAR

A PEAR csomag a kódgyűjtemény alaposztályát tartalmazza, a legtöbb csomag erre épül. Két fő előnye a beépített hibakezelés és a „destruktorok" használatának lehetősége PHP 4-ben is. Egy, a PEAR osztályból származtatott osztályban (pl. `myClass`) lehetőségünk van a szkript lefutásakor végrehajtódó destruktor függvény definiálására (`function_myClass()`). A destruktor az objektum „törlésekor" ugyan nem hajtódik végre, viszont a teljes szkript lefutása után azonnal automatikusan meghívásra kerül.

A kategóriába tartoznak még a PEAR_Frontend_Gtk, PEAR_Frontend_Web, PEAR_Info és PEAR_PackageFileManager csomagok, melyek a csomagkezelő és a nyilvántartás megfelelő működését hivatottak biztosítani.

Authentication

A kategória csomagjai felhasználók azonosítására és nyomonkövetésére nyújtanak megoldást. Az Auth csomag nevéhez híven beléptetési funkciókat tartalmaz, az Auth_Prefmanager felhasználók beállításait képes tárolni, míg a LiveUser egy egészen összetett jogosultságkezelő rendszer.

Benchmarking

Vannak alkalmazások, amelyekben kiemelten fontos a szkriptek lefutásának sebessége. A kategória jelenleg egyetlen csomagja, a Benchmark a kódunkban felállított ellenőrzőpontok segítségével megmutatja a program egyes részleteinek végrehajtásához szükséges időmennyiséget, így felderíthetjük, hogy mely részek szorulnak még optimalizálásra a sebességet illetően.

Caching

Amennyiben a fent bemutatott Benchmark csomag azt jelzi, hogy egy kódrészletünk nem elég gyorsan fut le, elgondolkodhatunk gyorsítótár alkalmazásán. A gyorsítótárazás ma már jól ismert és széles körben alkalmazott technológia, a Cache és Cache_Lite csomag segítségével saját alkalmazásunkba is egyszerűen beépíthetjük azt. A Cache_Lite egy viszonylag egyszerű csomag, ugyanakkor gyors megoldást biztosít, de kizárólag fájl-konténeret képes használni. A Cache csomag ennél sokkal összetettebb, a fájlokon kívül adatbázist, vagy akár az osztott memóriát is képes igénybe venni gyorsítótárazás céljából.

Configuration

A Config csomag könnyű karbantartási lehetőséget biztosít többféle konfigurációsfájl-formátumhoz. Jelenleg a PHP stílusú ini fájlon kívül XML formátumot is támogat, ismeri az Apache webszerver konfigurációs formátumát, a beállításokat adatbázisban is tudja tárolni, valamint további konténerei által egyéb formátumok kezelésére is képes. Új formátumok egyszerűen adhatók hozzá a meglévő csomaghoz. Egy-egy adott formátumú fájlt betöltve módosíthatjuk a bejegyzéseket, majd tetszőleges formában tárolhatjuk azokat, így a csomag konfigurációs állományok átalakítására is felhasználható.

Database

A PHP-s világ talán legelterjedtebb adatbázis-absztrakciós felületét, a PEAR DB-t ebben a kategóriában találjuk. Az alaprétegre számos további csomag épül. Ilyenek például a DB_DataObject, amely objektumorientált interfészt nyújt az adatbázistáblákhoz; a DB_DataObject_FormBuilder, ami egy erre épülő automatikus űrlapgenerátor, és a DB_Pager, ami rekordok listázásánál a lapozás leprogramozását hivatott megkönnyíteni.

Date and Time

A kategóriában jelenleg két csomag található, mindkettő a dátum-idő típusokkal való munkánkat segíti. A Date csomag képes időzónák kezelésére, és a különböző időzónák közötti konvertálásra. A dátumok belső ábrázolását nem 32-bites időbélyeggel végzi, így az 1970 előtti, illetve a 2038 utáni dátumokat is tökéletesen kezeli. A Calendar csomag is hasonló funkcionalitást nyújt, azonban sokkal általánosabban használható a Date csomagnál. A csomag elkészítését az ösztönözte, hogy az interneten elterjedt egyéb naptárkezelő csomagok általában a HTML kimeneti formátumra vagy egy adott adatbáziskezelőre épültek. Ezzel szemben a Calendar csomag matematikai oldalról közelíti meg a problémát, így akár HTML, WML, SOAP, XML-RPC, ASCII, stb. formátumú kimenet is előállítható belőle.

File Formats

A PEAR gyűjteményébe folyamatosan kerülnek be újabb és újabb fájlformátumok kezelésére alkalmas csomagok. Használatukkal egy-egy feladat során megtakaríthatjuk a formátumok belső szerkezetének megismerésére fordított energiát. Például az MP3_ID csomag segítségével az mp3 fájlok „ID3-tag”-jét tudjuk beolvasni, a memóriában tetszőlegesen módosítani, majd igény esetén az állományba visszaírni, mindössze néhány sornyi kód segítségével.

File System

Ebben a kategóriában többnyire keresztplatformos csomagokat találunk. A File_Find csomag segítségével egy adott állományra kereshetünk a fájlrendszerben, szabályos kifejezések megadásával. A File_Passwd csomag különböző jelszó-fájlok karbantartására képes, a File_Htaccess csomag segítségével az Apache könyvtáranként megadható konfigurációs állományait kezelhetjük. A File_SearchReplace csomaggal intelligens cseréket hajthatunk végre egyszerre több fájlban. A kategóriába kerültek az archívumokat kezelni képes csomagok is, jelenleg az Archive_Tar, és az Archive_Zip csomagokat találjuk meg itt, amelyek közül az Archive_Tar-t maga a PEAR is használja, a telepítendő csomagok kibontására.

HTML

A HTML kategória kevésbé összetett csomagjai különböző HTML elemek (táblázatok – HTML_Table, formok – HTML_Form stb.) generálását teszik egyszerűvé. Néhány csomag (pl. HTML_Progress, HTML_Treemenu) kliensoldali JavaScript kódot is generál.

A kategóriában külön csoportot képez az az öt sablonrendszer, amelyeket a többi HTML csomaggal együtt is használhatunk. Kiemelendő még a HTML_QuickForm, amely a lehető legnagyobb részletességgel segít az űrlapgenerálásban. A fentiekben már említett DB_DataObject_FormBuilder csomag is a HTML_QuickForm osztályt használja űrlapok generálására, adatbázisos feldolgozásra.

HTTP

A HTTP protokollal kapcsolatos csomagok ebben a kategóriában kaptak helyet. A HTTP_Upload űrlapokon keresztül feltöltött fájlok objektumorientált kezelésére szolgál, a HTTP_Request segítségével egyszerű módon fogalmazhatunk meg HTTP kéréseket.

Images

Az Images kategória csomagjaival képfájlokon végezhetünk műveleteket. A csomagok közül hármat emelnék ki: az Image_Barcode csomag vonalkódok előállítására képes, az Image_Text segítségével feliratokat generálhatunk meglévő képfájljainkra, míg az Image_Transform csomaggal különböző átalakításokat (méretezés, forgatás stb.) hajthatunk végre a képeken.

Internationalisation

A nemzetköziesítés egyre fontosabb szempont napjaink webes alkalmazásainál. A Translation és a Translation2 csomagok többnyelvű alkalmazások készítésében nyújtanak segítséget. A lefordított szövegrészeket adatbázisból kérdezik le oldalanként. Az oldalankénti lekérdezés az adatbázis terhelésének csökkentését és a sztringekhez való gyors hozzáférést segíti elő.

Logging

A Log csomag egy általánosított naplózó keretrendszer. Többféle kimenete (konzol, fájl, rendszernapló (unix), eseménynapló (windows), SQL, sqlite, email, mcal) lehetővé teszi, hogy szinte bármilyen rendszerben használhassuk.

Mail

A Mail_Mime csomag segítségével több részből álló üzeneteket hozhatunk létre, Mail_MimeDecode osztályának köszönhetően pedig dekódolhatjuk is azokat. A Mail_Queue csomag tömegesen kiküldendő levelek várósorba állítására nyújt megoldást, ahonnan meghatározott időközönként, a konfigurációnak megfelelő részletekben kerülnek majd továbbításra a levelek. A csomag rendkívül hasznos például hírlevelek fejlesztésénél.

Networking

A kategória a hálózatkezeléssel kapcsolatos csomagokat gyűjti össze. Az egyszerű, hálózati kapcsolatot nem igénylő csomagoktól kezdve (pl. IP cím-ellenőrző, URL-felbontó) az egészen összetett, élő kapcsolatot igénylő kódokig már több, mint harmincféle csomagot tartalmaz. A Net_FTP csomag által objektumorientált interfészen keresztül férhetünk hozzá a PHP beépített FTP-függvényeihez, a Net_Geo csomag egy IP cím fizikai helyzetét

próbálja meghatározni a világban, a Net_IRC csomag segítségével IRC-szerverekkel kommunikálhatunk, a Net_Ping csomag és a Net_Portscan csomag akár hibák felderítésére is alkalmas lehet, a Net_POP3 és Net_SMTP csomagok pedig a levelekkel kapcsolatos teendőink leprogramozásában nyújtanak segítséget.

PHP

A PHP kategóriába került a Validate csomag, amely számok, karaktersorozatok, email címek, bankkártyaszámok, és sok egyéb adat ellenőrzésére képes. Egyidejűleg több értéket is átadhatunk neki ellenőrzésre, így általában hatékony eszköznek bizonyul. A PHPUnit csomag által úgynevezett „unit teszteket" futtathatunk az elkészült kódunkon. Már készül a PHP_Requires csomag, amely egy tetszőleges szkriptről meg tudja majd mondani, hogy futtatásához minimálisan milyen PHP verzió szükséges, illetve hogy mit kellene módosítanunk a kódon, hogy korábbi verziókkal is kompatibilis legyen.

További kategóriák

A fentieken kívül további tizenöt kategóriában léteznek PEAR csomagok: Console, Encryption, Gtk Components, Math, Numbers, Payment, Processing, Science, Streams, Structures, System, Text, Tools and Utilities, Web services és XML.

Részvétel a fejlesztésben

Bárki részt vehet a PEAR fejlesztésében, akár bekapcsolódva a meglévő csomagok javításába, akár teljesen új csomag hozzáadásával, vagy dokumentáció írással. Egy letöltött csomag továbbfejlesztése esetén a bővített verzió visszajuttatása a PEAR csoporthoz azzal az előnnyel jár, hogy megfelelő működés és színvonalas programozás esetén jó eséllyel az új kód is bekerül a csomagba. Ebben az esetben továbbra is rendszeresen frissíthetjük csomagunkat a központból, anélkül, hogy aggódnunk kellene kifejlesztett funkcióink elvesztése miatt. Egy általunk készített új csomag akkor kerülhet be a központi rendszerbe, ha nem valósítja meg már jelen lévő csomagok funkcióit, illetve a közösség az ötletet megszavazza, valamint a kód minőségét elfogadja. Az új csomagot először a PEAR fejlesztői levelezőlistáján kell bemutatni, majd amennyiben a csomag elnyeri a többség tetszését, egy webes felületen keresztül kell előterjesztenünk, ahol a PEAR csoport tagjai leadhatják szavazatukat rá. A dokumentáció területén mindig igény van segítségre, a magyar nyelvű fordítás sem teljes.

Az előadásról

A PEAR rejtelmei iránt érdeklődők egy előadás és két gyakorlati bemutató keretében tudhatnak meg többet a rendszerről. Az előadás átfogó bevezetést nyújt a PEAR használatába, míg a bemutatókon a gyűjtemény csomagjainak használatával kapcsolatban konkrét példákat láthatnak az érdeklődők.

Mocsnik Norbert

Tizenkét éve, egészen fiatalon került kapcsolatba az informatikával. Nyolc éve programozik, négy éve fejleszt PHP nyelven. Szabadúszó webfejlesztőként a szoros határidők betartása, és az ügyfelek elégedettségének fokozása érdekében egyre nagyobb igényt érzett előre elkészített, minőségi csomagok iránt. Így került kapcsolatba 2003 januárjában a PEAR-rel, és azóta többféle módon is részt vesz a nyílt forrású projektben. A hibajelentéseken és folt-készítéseken, levelezőlistán keresztüli támogatáson kívül a PEAR kézikönyv egyetlen magyar fordítója. Előszeretettel használja a különböző csomagokat saját projektjeiben is, ezáltal megbízóinak magas színvonalú, biztonságos alkalmazásokat fejleszt, a legrövidebb határidők mellett. Mindig nyitott új, alvállalkozói rendszerben végzett megbízásokra, szívesen dolgozik együtt más fejlesztőkkel, és tervezőkkel. Bővebb információ saját weboldalán: http://norbert.mocsnik.hu/

Általános űrlapkezelő

Az ügyviteli alkalmazások mindegyikében szükséges a törzsadatok karbantartása. Ennek bevált receptje, ami mit sem változott hosszú évek során, a következő: Készítsünk egy táblázatot, amely felsorolja az adatrekordokat, esetleg szerkeszthető formában is. A táblázat egy sorát kiválasztva megjelenik egy űrlap a választott rekord adatait tartalmazó szerkesztőmezőkkel. Ez a megközelítés a PHP-ben készült alkalmazásoktól sem áll távol.
A következőkben egy ilyen, általános törzsadat szerkesztő felületet fogunk készíteni. Először felállítjuk a követelményrendszert, majd megtervezzük a szükséges adatbázistáblákat, és PHP osztályokat, valamint ezek használatáról is szót ejtünk.

Példa

A fejlesztést egy konkrét példán keresztül követjük végig. Mivel a példa csak demonstrációs célokat szolgál, ezért elég, ha a készülő rendszernek csak az egyik törzsadatát emeljük ki. További egyszerűsítés céljából, a táblából eltávolítottuk a szempontunkból nem releváns mezőket is.

Tegyük fel, hogy egy adathordozók nyilvántartására szolgáló alkalmazást szeretnénk kifejleszteni. Vizsgáljuk meg a rendszer hordozók tárolására szolgáló tábláját:

hordozo

hordozo_id	autoincrement	not null	elsődleges kulcs
hordozo_cime	varchar(60)		
hordozo_tartalma	text		
hordozo_tipus_id	int(11)		idegen kulcs (hordozo_tipus.hordozo_tipus _id)

A tábla egy idegen kulcssal kapcsolódik egy másik fontos táblához, amelyben a hordozók típusait tároljuk.

hordozo_tipus

hordozo_tipus_id	int(11)
hordozo_tipus_nev	varchar(20)

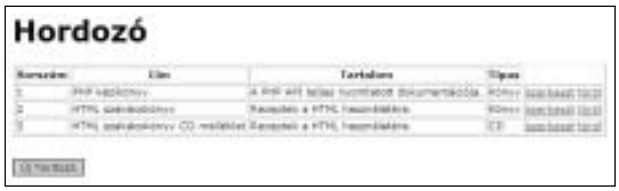
A tervezés folyamán mindig a példából indulunk ki, de megpróbálunk attól elvonatkoztatni, általánosítani. Ezzel kívánjuk elérni, hogy az általunk készített kód más törzsadatok szerkesztésére, esetleg más alkalmazásmodulok alapjaként is szolgáljon. A könnyebb áttekinthetőség kedvéért a továbbfejlesztési lehetőségeket a cikk végén külön bekezdésben emeltük ki.

Tervezés

Képernyőtervek

Több szoftverfejlesztési módszertan javasolja, hogy készítsük el a képernyőterveket, és annak alapján kezdjünk el tervezni. Most mi is ezt a metodológiát követjük. Kiindulópontnak két képernyőt terveztünk, amelyekben a jobb vizualizáció érdekében adatokat is elhelyeztünk.

Az első képernyő egy táblázatot tartalmaz, amelynek sorai a `hordozo` tábla rekordjainak feleltethetők meg. Minden sorban található két link, amelyek a rekord szerkesztésére, illetve törlésére használhatók. Az „új hordozó” gomb segítségével új rekordot szúrhatunk az adattáblába.



A második képernyő a `hordozo` tábla egy rekordjának űrlapnézete, kitöltött szerkesztőmezőkkel. Ugyanezt a formát használjuk fel új hordozó beszúrása esetén is, ilyenkor az űrlap mezői üresen jelennek meg.

Követelmények

A képernyőtervek alapján pontokba szedjük azokat a jellemzőket, követelményeket, amelyeket a fejlesztés során figyelembe veszünk.

Táblázatos forma:

- A képernyőnek van egy címe.
- A táblázat egy sora a tábla egy rekordját jeleníti meg.
- A táblázatnak van egy fejléce, amelyben a mezőkhöz rendelt címke jelenik meg. A címke különbözik az adattábla mezőjének nevétől.
- A rekordok szerkesztésére egy link szolgál, ami behozza a rekordot űrlapformában.
- A rekord a „törlés” link megnyomásával törölhető az adattáblából.
- A „Típus” mezőhöz tartozó értékek egy másik táblából jönnek, amely idegen kulcssal kapcsolódik a táblához.

Űrlap:

- A sorszám mező nem szerkeszthető.
- Többféle szerkesztőmező típus használható (text, textarea, select).
- Minden mezőnek van egy címkéje.
- A típus lenyíló menü egy másik idegen kulcssal kapcsolódó táblából jelenít meg adatokat.
- Az űrlapforma egyaránt képes egy új rekord beillesztésére és egy létező szerkesztésére.

Entitások

A felsorolt jellemzők alapján két entitás körvonalazódik:

Űrlap

Az „űrlap” entitás foglalja magába a törzsadat szerkesztéséhez szükséges általános adatokat. (Lásd a túloldali, első táblázatot.)

Tulajdonság

A „tulajdonság” entitás felel meg az adattábla mezőinek. (Lásd a túloldali, második táblázatot.)

Első táblázat:

Adat	Magyarázat	Példa
Cím	Az űrlap és a táblázat felett megjelenő cím	Hordozó
Egyedi azonosító	Az űrlap azonosítására szolgál.	
Select (SQL)	A táblázat adatainak előállításához szükséges SQL.	select * from hordozo
Update (SQL)	A módosító űrlap mentéséhez szükséges SQL.	update hordozo …
Delete (SQL)	Egy rekord törléséhez szükséges SQL.	delete from hordozo …
Elsődleges kulcs	A rekordok egyértelmű azonosításához használt mező	hordozo_id

Második táblázat:

Adat	Magyarázat	Példa
Címke	Táblázat fejlécében, valamint az űrlapon a mező előtt megjelenő címke.	Sorszám
Egyedi azonosító	A tulajdonság azonosítására szolgál.	
Szerkesztő típus	Az űrlapon megjelenő szerkesztő típusa.	textarea
Paraméterek	A szerkesztő előállítására szolgáló egyéb paraméterek.	width=100px
Szerkeszthető?	Megmutatja, hogy a mezőben található érték szerkeszthető-e.	A hordozo_id nem szerkeszthető.

Entitások tárolása és használata

A fenti entitásoknak két reprezentációját fogjuk megvalósítani.

Az egyik az entitások tárolására szolgál egy háttértárolón. A javasolt megoldás két adattábla létrehozása: űrlap és tulajdonság. A tulajdonság táblában helyezzünk el az űrlap táblára mutató idegen kulcsot. Felfigyelhetünk arra, hogy a két táblánkat a most készülő szerkesztő felülettel jól tudjuk majd szerkeszteni, azaz egyben elkészítjük a karbantartó felületüket is.

Az entitásainkat a PHP kódban reprezentálnunk kell egy-egy osztállyal. A `TUrlap` osztály lesz felelős az űrlap jellemzőkért, míg a `TTulajdonság` a mezők jellemzőiért. Az `TUrlap` egy példányának létrehozásánál meg kell adnunk egy űrlapazonosítót, ez alapján tölti ki saját tagváltozóit az űrlap táblából, valamint hozza létre az űrlaphoz tartozó `TTulajdonsag` objektum példányokat is a tulajdonság tábla adatait forrásul használva.

Ha vetünk egy pillantást követelménylistánkra, látszik, hogy több típusú szerkesztőt kell egy `TTulajdonsag` osztállyal leírni. Ennek a problémának feloldására a `TTulajdonság` osztályt absztraktnak bélyegezzük, és csak az alapvető, minden szerkesztőre jellemző funkciókat építjük bele. Az egyes szerkesztő típusokhoz egy-egy osztályt származtatunk a `TTulajdonsag`-ból, amely megvalósítja a szerkesztőhöz tartozó egyedi logikát.

A megjelenítendő kód előállítása

Most, hogy a két PHP osztály készen áll, már csak a folyamatot, vezérlő osztályt kell elkészítenünk. Ennek konkrét formája függ attól, hogy a szerkesztő felületet milyen keretrendszerbe fogjuk beépíteni. Most ezt figyelmen kívül hagyva csak az általános vezérlésre koncentrálunk.

A vezérlő csak az akciót azonosítja, a végrehajtás azonban a `TUrlap` osztály feladata. Ezért képessé kell tennünk a fent említett akciók elvégzésére.

Amennyiben a vezérlő nem kap akciót, utasítja a `TUrlap`-ot, hogy készítse el a táblázatos formát.

A vezérlő egyes funkcióihoz rendeljünk akciókat:

HTML kód legyártása	ez az alapműködés nincs akció
Új rekord beszúrása az adattáblába	„beszur” akció
Rekord törlése az adattáblából	„torol” akció
Rekord szerkesztéseinek mentése	„modosit” akció

A vezérlő „beszur” akció esetén megvizsgálja, hogy érkeztek-e adatok a kérésben. Amennyiben igen, a `TUrlap` segítségével elhelyezi azokat az adattábla egy új rekordjában, különben az üres űrlap kódját kéri az objektumtól.

A `TUrlap` törli a paraméterként kapott rekordot az adattáblából, ha a vezérlő „torol” akciót azonosít.

A „modosit” akció hasonlóképpen működik a „beszur”-hoz. Ha csak a rekord azonosítóját kapja paraméterként, akkor a kitöltött űrlapot mutatja, különben módosítja a rekordot a táblában.

Lehetőségek

A következőkben felsorolunk néhány olyan lehetőséget, amivel az elkészült szerkesztő felületet tovább fejleszthetjük:

- Az űrlapforma később felhasználható egyéb (törzsadatok szerkesztésén kívüli) alkalmazás űrlapok készítésére és feldolgozására.
- Az egyes tulajdonságokhoz ellenőrzések kapcsolhatók.
- A select típusú tulajdonságokból könnyen készíthetők szűrők, amivel a táblázatos formában megjelenő adatok halmazát lehet szűkíteni.
- Amennyiben az adattábla sok rekordot tartalmaz, a táblázatos forma navigálhatóvá tehető.
- A táblázatos formában az egyes tulajdonságok alapján rendezés valósítható meg.
- Az egyes tulajdonság címkék mellé ellipszis gomb (…) helyezhető el, amelyre kattintva az űrlapkezelő egy másik törzsadat táblára ugrik.

Kolman Nándor

A Budapesti Közgazdaságtudományi Egyetem eltévedt hallgatója. Három éve foglalkozik PHP-re épülő webes alkalmazások, portálok és egyedi weboldalak tervezésével és készítésével.

PHP Chemotox

„Mihelyst elkezdtünk programozni, meglepődve tapasztaltuk, hogy ez nem is olyan egyszerű, mint amilyennek először gondoltuk volna. Fel kellett fedeznünk a hibakeresést. Élesen él bennem az a pillanat, amikor rádöbbentem, hogy attól kezdve életem nagy részét azzal fogom tölteni, hogy hibákat keresek a saját programjaimban." — 1949, Maurice Vincent Wilkes, a mikro-programozás atyjaként ismert, angol „számítástechnikai úttörő". (http://en.wikipedia.org/wiki/Maurice_Vincent_Wilkes)

Mindegyikünk megtapasztalhatta már ezt az érzést. Maurice Wilkes szavai a mai napig aktuálisak, és könyörtelenül emlékeztetnek arra a tényre, hogy nincs program hiba nélkül. Már itt érdemes megállni egy pillanatra, hogy meghatározzuk, mit is értünk bug-on – magyarul programhibán, hibán –, milyen típusaival találkozhatunk PHP környezetben, és ami a legfontosabb, mit tehetünk ellene?

A hiba bennünk van

„A hiba a program nem kívánt viselkedése, helytelen reakciója, a kívánttól eltérő kimenet produkálása valamilyen bemenet és belső állapot kombinációjára." Sajnos, ennyi év tapasztalata után sem tudunk ennél jobb választ adni az első kérdésre. Ennek az oka az, hogy a hibák, amint az azokat „hordozó" algoritmusok megalkotása is, manapság csakis emberi tényezőkön múlik, azaz rajtunk.

A cikket szándékosan nem a jól ismert, zárlatot okozó cserebogár történetével kezdtem, ahonnan a jelenség és az egész témakör a nevét kapta. Dijkstra meg is jegyezte, hogy az angol névadás felelőtlen, és azt sugallja, hogy a programozó nem is hibáztatható. Bár a programhibák a legritkább esetben vezethetők vissza hardverhibákra, mégis könnyebb először egy fantomlényt hibáztatni, mint rögtön saját tökéletlenségünkkel szembesülni, így az anekdota még ma is tovább él.

A fellépő hibákat hatásuk és megjelenésük helye szerint is csoportosíthatjuk:

- A futtató környezet (webszerver, PHP) leállását okozó hiba. Ezzel leginkább még fejlesztési stádiumban levő kiterjesztések (bővítmények) használatakor találkozhatunk. Ebben az esetben még az is valószínű, hogy nem a saját szkriptünkben van a hiba, hanem olyan extrém értékekkel végeztünk műveleteket, amire a fejlesztőknek „még nem volt ideje gondolni".
- PHP értelmezési hiba. Ez már sokkal ismerősebben hangzik. Ilyenkor nincs más dolgunk, mint megértetni az értelmezővel a kódunkat. Mivel az nem fog új nyelvi elemeket megtanulni, kénytelenek vagyunk saját magunk az elgépelések nyomába eredni, és valami mással próbálkozni.
- PHP futási hibák. Számos egyéb mellett ide tartoznak azok is, amelyek a szoftverkörnyezet kiesése miatt lépnek fel, és bizonyos értelemben nem tekinthetők „helytelen reakciónak". Szerencsére a hibaüzenet révén van mibe kapaszkodni, és lehetőségünk van megkeresni a zavar okát.
- Végül, de nem utolsó sorban, amikor a futtató környezet nem észlel hibát, de valami egészen mást látunk végeredményként, mint amit kellene. Ez az igazi logikai hibák szintje, amelyeket a legnehezebb detektálni, ez a „hibakeresés művészetének bölcsője".

A gyakorlati bemutató remélhetőleg ez utóbbi két csoportba tartozó hibák hatékony felderítéséhez nyújt segítséget, míg ez a cikk tippekkel szolgál az elkerülésükhöz, és hibamentesebb PHP programok írásához.

Előtte gondolkodjunk!

A fenti hibadefiníció elsőre nem sok hasznosat mutat, mint a matematikai definíciók általában, de jobban elidőzve felette mégis rengeteget meríthetünk belőle. Mindennél többet ér, ha alaposan megtervezzük az algoritmust, és specifikáljuk a kívánt működést és annak feltételeit! Ennek miként tjére a többi cikkben részletesebb utalásokat találunk.

A tervezés során érdemes részletesen kiismerni a program környezetét is, és feltérképezni, hogy milyen értékeket kaphat bemenetként.

Az első hibát itt ejthetjük, ha nem különböztetjük meg a lehetséges bemenet fogalmát az érvénytelenekétől, amelyek ugyanúgy felléphetnek, és ezért le is kell kezelni azokat. Ebből a két halmazból már az elején alakítsunk ki úgynevezett tesztcsomagokat, amelyekkel egyszerűen ellenőrizhető egy-egy kisebb funkció – függvény vagy osztály – helyessége. Ezt a munkát hivatott megkönnyíteni, és összefogni a PEAR PHPUnit csomagja.

A „védelem" másik eszköze a C nyelvből átvett, de igen kevéssé kiaknázott `assert()` függvény. Ennek lényege, hogy a programfejlesztés során bármilyen feltételezésünket logikai kifejezések (kijelentések) formájában elhelyezzük a kódunkban:

```
function get_input ($nev, $forras='GPCSE') {
  $forras = strtoupper($forras);

  assert ('strspn($forras, "GPCSE") ==
➥ strlen($forras)');

  for ($i = strlen($forras); $i--;) {
    switch ($forras{$i}) {
      case 'P':
       if (isset($_POST[$nev]))
         return $_POST[$nev];
      ...
    }
  }
}
```

Ha a feltétel hamis, akkor vagy figyelmeztetést kapunk (`assert.warning`) vagy megszakad a futás (`assert.bail`) a `php.ini` beállításoktól függően.

Ezt a technikát főleg alapvető típus- és paraméter-ellenőrzésekhez, szigorú programozási szabályok kikényszerítéséhez – interfészek, keretrendszer-komponensek kialakításakor – vehetjük igénybe. Némi körültekintéssel felhasz-

nálható figyelmeztetésként a még ki nem dolgozott területek jelölésére. Alapszabály azonban, hogy az `assert()`-ben elhelyezett kódnak semmilyen mellékhatása nem lehet az őt követő kódra, ha mégis lenne, azt tilos kihasználni.

Az `assert()` használatának további előnye, hogy egyetlen `php.ini` beállítás (`assert.active`) megváltoztatásával minden ilyen kiértékelés kikapcsolható, mintha bele sem írtuk volna a forrásba, egyszerűvé téve ezzel az éles verziók üzembe helyezését. A programnak mindkét esetben ugyanúgy kell viselkednie, ez a legfőbb oka, hogy nem használható általános hibakezelőként.

Az érdemi munkát végző programrészt meg kell védeni az érvénytelen bemeneti paraméterek vagy változóértékek elől – ez az általános hibakezelés. Ez bizony sok esetben nagy körültekintést, és rengeteg gépies munkát igényel. Az alábbi kódrészlet minden PHP programozó számára ismerős lehet, még ha más adatbázis-kezelőt használ is:

```
<?php
if (!($conn = pg_pconnect("dbname=chemotox"))) {
  exit; // Hiba: nincs adatbázis kapcsolat
}
if (!($result = pg_query($conn, "SELECT * FROM
➥ bugs")) ) {
  exit; // Hiba: rossz lekérdezés
}
if (($count = pg_num_rows($result)) == -1) {
  exit; // Hiba az adatbázis-szerveren belül
}
for ($i = 0; $i < $count; $i++) {
  if ($row = pg_fetch_row($result, $i))
    echo join(' | ', $row), "\n";
}
?>
```

Remélem, minden olvasóban viszolygást kelt ez az aprólékosságig kidolgozott példa. Minden olyan függvény visszatérési értéke egy előzetes szűrésen esik át, amely hibával térhet vissza. Megnyugodni mégsem tudunk, mert a kód túl terjengős lett, és – ami még fontosabb(!) – könnyen szem elől veszthetjük benne a lényeget, hogy mit is csinál a program tulajdonképpen. Igencsak megkeserítheti az életünket, ha ilyen elaprózott kódban kell hibát keresni.

A példában szemet szúrhat még valami. Az ellenőrzéseket ugyan mind elvégeztük, de túl sok hasznosat nem tudtunk velük kezdeni (exit). Ez nem véletlen, ezek a műveletek szorosan egymásra épülnek, ha valamelyik sikertelen, akkor a láncot meg kell szakítani. Az ehhez hasonló „egybe zárandó" (atomi) műveletek képzésére felhasználhatjuk a feltételes kifejezésekben használt logikai operátorokat:

```
<?php
// hozzáadunk egy kérdést a BARKOCHBA fájlhoz
$siker  = (($f = fopen(BARKOCHBA, 'a'))
            &&  fwrite($f, $kerdes)
            &&  fclose($f));
?>
```

Szerencsénkre kedvenc PHP-nk legtöbb beépített függvény sikertelenségéről külön hibaüzenetet is küld, amit lehetőségünk van saját, PHP-ban írt függvénnyel lekezelni. Ez is segít elkerülni a fenti kód-dzsungelek kialakulását.

```
<?php // hiba.php
function my_exit () { exit(1); }
set_error_handler('my_exit');
// csak saját függvény lehet!
?>
```

```
<?php
require 'hiba.php';
$conn = pg_pconnect("dbname=chemotox");
$result = pg_query($conn, "SELECT * FROM bugs");
for ($i = 0; $row = pg_fetch_row($result, $i);
➥ $i++ ) {
  echo join(' | ', $row ), "\n";
}
?>
```

Már ez a rövid példa is rávilágít a külön terjedő hibaüzenetek kézzel fogható előnyére, amelynek továbbfejlesztését jelenti a PHP 5 kivételkezelése. Egyfelől a hibák nem (csak) a normál adattérben kerülnek átadásra a rendes visszatérési értékekkel azonos formában, hanem egy dedikált, csak a hibáknak fenntartott csatornán keresztül is. Így kódunkban is különválaszthatjuk végre a hibakezelést a hasznos funkciókat végző részektől.

A PHP beépített hibakezelője elég kezdetleges, de sajátunk képességeinek csak a képzeletünk és az észszerűség szab határt. Ízelítőül közkedvelt jellemzők létező megvalósításokból: a sikertelen kódrészlet kiíratása, függvényhívási lánc és a pillanatnyi változók listázása, automatikus értesítés email-ben (SMS-ben), éles szervereken hibaoldal generálása vagy gyorsítótárazott tartalom megjelenítése a félkész lap helyett, stb. Az assert-ek kezelését is magunkhoz ragadhatjuk az `assert_options()` függvénnyel. A saját hibakezelő és a logikai kijelentések (`assert()`-ek) jól kiegészítik egymást: az előbbivel a PHP által detektált futási hibákat deríthetjük fel, míg az utóbbi a tervezés során kialakított feltételek teljesülését ellenőrzi.

Nem tanácsos önmagában az `error_reporting` szint csökkentése! Saját hibakezelővel a hiba fellépésekor minden elérhető információt meg lehet szerezni a hiba felbukkanásáról, amivel jelentősen lerövidíthető a hibakeresés első fázisa: az adatgyűjtés. A hibák némelyike csak ritkán – bizonyos feltételek esetén – következik be, és ezek reprodukálása rendkívül időigényes és fárasztó feladat.

„Na, akkor gondolkodjunk!"

Nem kell elkeseredni, ha minden előző erőfeszítésünk ellenére maradnak hibák a programjainkban. Emlékezzünk arra, hogy nincs program hiba nélkül („No feature, no bug"). Ezennel elérkeztünk a programozás legkreatívabb és legintellektuálisabb kihívásához: az igazi hibakereséshez. Ez egyben a legidegesítőbb is tud lenni, azaz, a folyamat legtöbbször nem is annyira technikai, mint inkább pszichológiai tényezőkön múlik, de ennek tárgyalása nem ide tartozik. A pszichológiai hadviselésen kívül a legfontosabb megszívlelendő szabályok:

Lehetőleg szüntessünk meg minden futási hibát, mielőtt nekiállunk a logikai hibák feltárásának! Egyfelől könnyebb egy "szintaktikailag korrekt" algoritmusból kiindulni, másfelől még az is megeshet, hogy ezek a futási hibák a kiváltói az üldözött bogárnak.

Webes környezetben a bemenetek egészen a böngészőig vezethetők vissza. Ha valami nagyon érthetetlent tapasztalunk, érdemes futó pillantást vetni a felhasználótól elküldött adatok alakjára, mert egy rosszul megírt JavaScript is jól összekuszálhatja azokat, amit a PHP szkriptünk hiába próbál a legnagyobb jóindulattal tolerálni. A böngészők és a szerver gyorsítótáráról sem szabad elfeledkezni.

Mindig ügyeljünk arra, hogy ugyanolyan verziójú programokat használjunk a tesztelés során, mint amilyen környezetben a hiba előfordult. Bármekkora körültekintéssel is tervezzük meg programunkat, a futtató környezet jövőbeni változásaira nem tudjuk felkészíteni. Szerencsénkre a migráció kérdése és a kompatibilitás egyre nagyobb figyelmet kap a PHP fejlesztésében is, de vannak még emlékeim az `allow_call_time_pass_reference` vagy a munkamenet-kezelést érintő változásokról.

Ha csak egy mód van rá, használjunk valamilyen hibakereső (nyomkövető) programot, amivel a forrás piszkálása nélkül vizsgálgathatjuk programunk viselkedését. Futás közben bármelyik változó értékét lekérdezhetjük, sőt meg is változtathatjuk, ezzel akár a futást egy másik ágra terelve. Már folynak a fejlesztések, hogy ugyanígy a kódunkat is átmenetileg – futás közben – átalakíthassuk, ha éppen valami szikra beütött volna.

Vigyázzunk, bármilyen egyszerű és ártalmatlan utasítást (`echo()`, `var_dump()`, `print_r()`) is adunk a programhoz, ezzel végső soron megváltoztatjuk az eredeti szituációt, és a HTTP átirányítások, sütiküldések nem működnek esetleg ezután. Azt a lelkiismereti gátat pedig, hogy ne javítgassunk tovább itt-ott, ettől kezdve nagyon könnyű lesz átlépni. Ha mindenképpen írásos nyomát akarjuk látni valaminek, használjuk inkább az `error_log()` függvényt!

Oszd meg és uralkodj! – a szisztematikus hibakeresés aranyszabálya. Ha semmilyen fogódzót nem sikerült találni, ez az egyedüli hatékony módszer. De mit érdemes felosztani?

A programsorok közül meg kell határozni azokat, amelyek a hibás kimenet előállításában érintettek. Ennek szükséges feltétele azonban, hogy az adott programsor lefusson. Ennek kiderítése erősen szétágazó és sokat kommunikáló szkriptekben nem mindig egyszerű – eltekintve a naplózástól. A mai nyomkövető programok már képesek ennek a rögzítésére is (függvényhívási lánc), így egy-egy teszt futtatása után könnyebben képet alkothatunk a mélyben lejátszódó folyamatokról. PHP-ben pedig blokkonként külön-külön is meg tudjuk mondani, melyik sor futott le:

```
<?php // line_profiler.php
function line_profile($return = FALSE) {
  static $lines;

  if ($return) {
    $tmp = $lines; $lines = array();
    return ($tmp);
  }
  $lines[] = array_pop(debug_backtrace());
  // PHP 4.3.0
}
register_tick_function('line_profile');
?>
```

Ezt a függvényt a `declare` vezérlő szerkezettel tudjuk működésbe hozni:

```
<?php
include 'line_profiler.php';

declare (ticks = 1) {
  ... // a kivizsgálandó kód helye
}

print_r(line_profile(TRUE));
// a lefutott sorok listája
?>
```

A `declare` által körbefogott részben minden utasítás után lefut a függvényünk, ami megjegyzi az aktuális sor számát. A blokk végén egyszerűen kiíratjuk, hogy melyik sorra került rá a vezérlés. (Sajnos a függvényhívási lánc nyomon követése már nem ilyen könnyű.)

Meglepő, de olyan elsőre irreleváns eszközök is, mint a verziókezelők segíthetnek a hiba felderítésében. A különböző verziók tesztelésével kideríthető, melyik változtatással jelent meg a hiba a programban. Ezzel gyorsan lokalizálható a hibás kódrész, amit ezután az alábbiak szerint alapos vizsgálatnak vethetünk alá. Természetesen, ha nincs verziókezelőnk, akkor érdemes a már letesztelt régebbi verziókat más módon megőrizni, hogy ezt a módszert is bevethessük.

A bemenő adatok kombinációjából érdemes megkeresni azokat, amelyek a lehető legkisebb mértékben térnek el a hibát előidéző bemeneti paraméterektől, de még jó eredményt adnak. Nyilvánvalónak tűnik, hogy a hibát e kettő különbsége, és az ezt feldolgozó programsorok okozzák.

A hiba oka, mindig a végrehajtandó utasítások sorában és a változók értékeiben van elrejtve, még ha elsőre nem is tűnik következetesnek a hibák fellépése. Ha azt vesszük észre, hogy a vizsgált értékektől függetlenül véletlenszerűen lépnek fel a hibák, akkor két eset lehetséges: figyelmen kívül hagytunk néhány „mégis csak fontos" változót (`global` és `static` kulcsszó függvényekben!), vagy egyéb rendszer-anomáliák léptek fel (pl. fájlrendszer, adatbáziskiszolgáló kiesése). Sajnos, az előbbi a gyakoribb, mivel a PHP az utóbbiról legtöbbször hibajelzést küld.

Előbb mindig igazoljuk a gyanúnkat és megérzéseinket! Gyakorlott fejlesztők legtöbbször már a tünetekből sejtik, hogy mi idézte elő a hibát. A gyors felfedezés keltette örömünk csillapodtával higgadtan gondoljuk végig, hogy tényleg ez lehetett az ok, az egyetlen és kizárólagos ok, vagy még tovább kell kutatnunk.

Lassan járj, tovább érsz! A kód egyetlen részéről sem szabad feltételezni a helyes működést, amíg meg nem győződtünk az ellenkezőjéről. Minden egyes egyszerűnek tűnő utasítást vagy annak hatását vizsgáljuk meg! Ki ne járt volna már úgy, hogy egy feltételben == helyett = -t írt volna? Egy karakter is végzetesen félreviheti a feldolgozást, és ha nem figyelünk, akkor a hibakeresést is.

Tartózkodjunk az átfogó javítástól, ami ugyancsak a felfedezés pillanataiban törhet ránk, mert ezzel több ökölszabályt is megsértünk egyszerre: elmulasztjuk a tervezést, és egyszersmind semmibe vesszük az előző eredményeit, átfogóbb tesztelés nélkül új funkciót építünk a programunkba, stb. Gondoljunk arra a nem túl hízelgő érvre, hogy az előző hiba is egy összetett kód (vagy többszörös módosítás) révén került a rendszerbe, miért lenne a mostani kivétel?

Az előadás ezeknek a szabályoknak a gyakorlati alkalmazásáról és az ehhez felhasználható eszközök bemutatásáról fog szólni.

Papp Győző

Öt éve, az egyetemi tanulmányai során a hagyományos asztali programok után újat keresve csöppent bele az internet világába és a dinamikus weboldalak készítésébe. Ekkor ismerkedett meg a PHP nyelvvel, amely azóta a hobbija és munkaeszköze egyszerre. Az első és második konferencia compo-felelőse.

Web-szabványok hazai alkalmazásának statisztikai elemzése

Az interneten publikálható dokumentumok létrehozásának feltételei jól körülhatároltak, szabatos leírások segítik az eligazodást (http://www.w3.org/). Egy honlapot készítő programozó feladata a megfelelő szabványok kiválasztása, amelyek meghatározzák, hogy milyen technikát, utasításokat és paramétereket alkalmazhat a dokumentumok kialakításához. A következő szabványok állnak jelenleg rendelkezésre:

Szabvány	Megjelenés	URL
HTML 3.2	1996	http://www.w3.org/TR/REC-html32
CSS 1	1996	http://www.w3.org/TR/CSS1
HTML 4.0	1997	http://www.w3.org/TR/REC-html40
CSS 2	1998	http://www.w3.org/TR/CSS2
HTML 4.01	1999	http://www.w3.org/TR/html401
ECMAScript	1997-1999	http://www.ecma-international.org
XHTML 1.0	2000	http://www.w3.org/TR/xhtml1
ISO HTML	2000	http://www.cs.tcd.ie/15445/15445.HTML
XHTML 1.1	2001	http://www.w3.org/TR/xhtml11
DOM L1-L3	1998-2004	http://www.w3.org/DOM/DOMTR

Alkalmazásuk során a legfontosabb szempont, hogy összekeverésük nélkül tartsuk be őket. Habár ez szakmai minőségbiztosítási kérdés, az ügyfelek gépe előtt ülve használhatóságot, élvezhetőséget és kompatibilitást jelent.

Az előadásban bemutatott felmérés célja a honlapok készítése során alkalmazott szabványok számbavétele és az elkövetett hibák elemzése, lehetséges okainak feltárása.

Módszerek

Az elemzés során öt fő területről választottam honlapokat. A nagy számú kínálat miatt csak azokat vizsgáltam meg, amelyek bevallottan sok látogatót szeretnének kiszolgálni, vagy piaci helyzetüknél fogva meghatározzák a honlapok fejlesztésének irányát, és formálják a látogatók ízlését, böngészési szokásait:

- Távközlési vállalatok
- Közhasznú kormányzati intézmények
- Elektronikus média piacvezetői
- Honlapok tervezéséhez használható szoftverek gyártói
- Honlapok tervezését kínáló piacvezető vállalatok

A vizsgálatot minden esetben a W3C ingyenes internetes szolgáltatásával végeztem (http://validator.w3.org/), böngészőnek a Netscape 7.1-et használtam UHU-Linux környezetben. Sok esetben nem lehetett eldönteni, hogy az oldal melyik szabvány szerint készült, ilyenkor a „HTML 4.01 Transitional”-t feltételeztem. Az előadás megemlíti a munka során lelt egyéb hibákat is: rosszul beállított webszerver, trükkösen megírt keretek és felugró ablakok, „Az ön böngészője nem támogatott!” stb.

Eredmények

Összességében a hibák legnagyobb részét a figyelmetlenség okozza: hiányzó paraméterek (elsősorban a képek alternatív szövege), helytelen elem (entitás) hivatkozások, a szabványban nem létező paraméterek használata. Ezeken kívül jelentős számban fordultak elő a forráskód „nyelvtani” hibái: hatáskörök tévesztése, lezárások elmulasztása.

Szerkesztési hibák százalékos megoszlásuk alapján:

Hiba típusa	Százalék
Nem létezik	15,75
Hiányzó paraméter	34,28
Érvénytelen adat	3,57
Hiányos szerkezet	2,57
Rossz helyen van	8,83
Elem hivatkozás	15,51
Ismeretlen elem	3,14
Nincs lezárva	8,03
Szöveg hiba	2,13

Az összehasonlíthatóság miatt nem figyelhetjük csupán a hibák százalékos eloszlását. Az is fontos, hogy mekkora kódban mennyi hibát követnek el.

Az 1K HTML kódra jutó hibák száma szektoronként:

Szektor	Hibák száma
Webdesign	10,63
Távközlés	4,5
Kormányzat	9,13
Média	6,64
Szoftvergyártó	1,33

A második táblázatban látható, hogy a honlapok készítéséhez használható szoftverek gyártói közt a legkisebb a hibák átlaga, míg a honlapok tervezőinek és kivitelezőinek saját oldalai a lista másik végét zárják. Ki kell emelni a szoftvergyártókat, ahol a hibátlan lapok száma elérte a 29%-ot, ugyanez az érték a kormányzati lapoknál 5%, a média vizsgált képviselőinél 0%.

Az elemzés során meglátogatott oldalak 10%-a alkalmazott Flash animációt a lap szerves részeként (nem csak díszítéshez), 8%-a hibaüzenettel fogadott, 6%-ban találtam alapfokú HTML szerkesztési hibát.

dr. Baranyai László

A LAAZ Bt. családi vállalkozás üzletvezetője, főállásban egyetemi oktató. A cég fő profilja egyedi igényű tudományos szoftverek programozása adatgyűjtés és feldolgozás céljára, valamint algoritmusok fejlesztése. Ezen túl részt vállalnak GNU GPL projektekben is. Az akadémiai szférában több konferencia és intézményi honlapot készítettek (ilyen az Alma Máter BKÁE Élelmiszertudományi Kar lapja). A European Society of Agricultural Engineers 2002-es konferenciájának CD-ROM anyagához a LAAZ Bt. készítette a szöveges adatbázist és a felhasználói felületet.

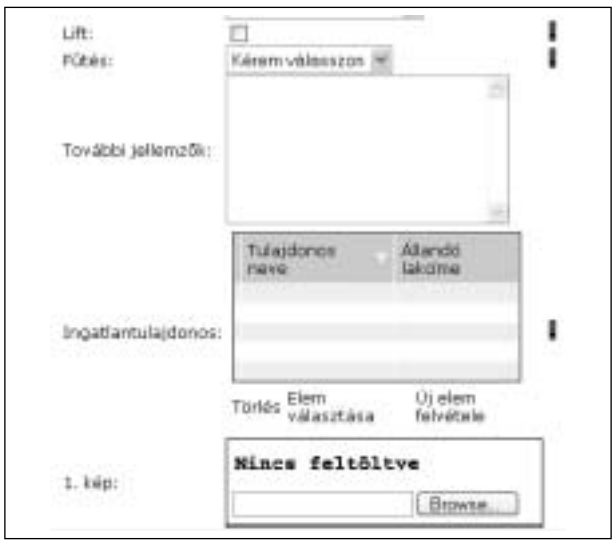
Gomba for PHP

Jelenleg több keretprogram is rendelkezésre áll webalkalmazások fejlesztésére. Szoftvertechnológiai hiányosságaik azonban használatukat korlátolttá, nehézkessé tehetik nagy rendszerek fejlesztése során. A Gomba for PHP egy objektum-orientált, eseményvezérelt környezet, melynek magja felügyeli az alkalmazások futását, és komplex szolgáltatásaival, integrált komponenseivel nagymértékben könnyíti az alkalmazások fejlesztését. Az előadásban részletesen tárgyaljuk a rendszer eseménymodelljét. A szolgáltatásokat mind fejlesztői, mind felhasználói szintről megvizsgáljuk. A fejlesztés menetét gyakorlati példákon mutatjuk be.

Bevezető

A webalkalmazások alapvetően eseményvezérelt alkalmazások. Esemény lehet például egy űrlap tartalmának elküldése vagy egy link aktiválása. A legtöbb PHP alapú web fejlesztési keretrendszerben az események kezelésére nincs semmiféle támogatás, így az manuálisan, if és case szerkezetek segítségével történik. Az események kezelésének egyik legnagyobb nehézsége, hogy az ablakozós alkalmazásoknál megszokott eseménymodell a HTTP protokoll sajátosságai és a hálózati késleltetések miatt hatékonyan nem használható. A Gomba for PHP rendszer legjelentősebb eredménye egy web környezetekben használható eseménymodell, mely megteremtette az ablakozós alkalmazásoknál megszokott objektum-orientált komponens-alapú szoftverfejlesztés lehetőségét.

Eseménymodell



1. ábra: A CAutoForm komponens a gyakorlatban

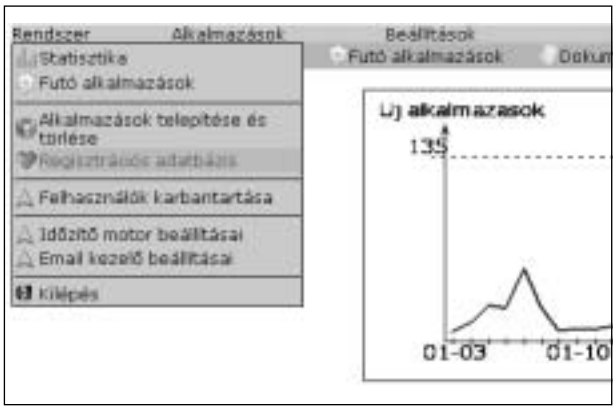
A webalkalmazások felhasználói felületeként használt böngésző szakaszos kapcsolatban áll a webszerverrel. Ezen túl a böngészőben több esemény is történhet, melyről a szerver kizárólag a következő kapcsolatfelvételkor értesülhet. A web környezetekben használható eseménymodellt tehát fel kell készíteni több – esetleg sorrendileg nem ismert – felhasználói akció kezelésére. A Gomba for PHP rendszer eseménykezelő osztálya az ún. eseménymátrix. Az eseménymátrixban megadhatjuk, hogy bizonyos változóknak milyen értéket kell felvenniük ahhoz, hogy a megadott eseménykezelő függvény meghívódjon. Jellemző gyakorlati példa egy űrlap tartalmának ellenőrzése. Példánkban az űrlap tartalmazzon egy kötelezően kitöltendő mezőt és egy gombot, mely az űrlap elküldésére alkalmas. Egy eseménykezelést nem támogató rendszerben ellenőriznünk kell, hogy a mező ki van-e töltve, majd hogy a gomb meg lett-e nyomva. Az ellenőrzéseket végrehajtó szerkezet magjában megtörténhet a tényleges eseménykezelés. Az eseménymátrix használatával azonban egyszerűen kiköthetjük, hogy egy bizonyos függvény, mely a tényleges eseménykezelésre alkalmas, akkor hívódjon meg, ha a gomb meg lett nyomva és az űrlap mezője ki lett töltve. A Gomba for PHP rendszer tartalmaz egy CAutoForm komponenst, mely a fenti feladat ellátására alkalmas (1. ábra).

Keretrendszer és szolgáltatásai

A Gomba for PHP rendszer magja biztosítja az alkalmazás futásának vezérlését és adatainak tárolását a létrehozástól egészen az alkalmazás megszűntéig. A rendszer alkalmazásainak nem szükséges session változót használni fontos információk tárolásához, mint például az alkalmazás belső állapota. A keretrendszer biztosítja, hogy a környezet `CComponent` alaposztályából származó osztályok tagváltozóik értékét a következő webszerverhez érkezett kérésig megtartják. Ezt a rendszer magja a teljes alkalmazás adatbázisba mentésével éri el, a `serialize()` és `unserialize()` PHP függvények alkalmazásával.

A fejlesztőrendszer több komponenst tartalmaz, melyeket a következő kategóriákba sorolhatunk: GUI komponensek (ablak, iframe, űrlap, stb.), Rendszer szintű szolgáltatások (pl. időzítő, virtuális fájlrendszer, registry, naplózó, stb.), Egyéb szolgáltatások (pl. dokumentum kezelő, email, egyéb hálózati szolgáltatások). A GUI komponensek két részre oszthatóak: alapelemek és összetett elemek. Az alapelemek a HTML nyelvhez és tagjaihoz kapcsolódnak (form, select, text, stb.), míg az összetett elemek egy-egy komplex funkció ellátására hivatottak (űrlap, fanézet, statikus keret, grafikon, stb.). A rendszer magja képes több GUI motor kezelésére. A GUI motorok az X11-ben használt widget set-ekhez (GTK, QT) hasonlíthatóak, tehát ha nem elég, vagy nem tetszik az alapértelmezett motor konfigurálhatósága, megjelenése, új motor készíthető és akár párhuzamosan használható a többivel. A rendszer több

szolgáltatással rendelkezik, melyek közül az egyik legfontosabb az a virtuális fájlrendszer, mely képes tárolni a szerverre feltöltött állományokat. A fájlrendszer azért virtuális, mert szerkezete adatbázisban tárolt, a feltöltött fájlok fizikailag egy helyen találhatóak.



2. ábra: A Gomba for PHP adminisztrációs felülete

A Gomba for PHP rendszer tartalmaz egy adminisztrációs felületet, melyen a rendszert és telepített alkalmazásait érintő karbantartási feladatok elláthatóak (2. ábra). A felület talán legfontosabb szolgáltatása a programcsomagok kezelése. A rendszer lehetőséget nyújt GNU/Linux rendszerekben megszokott módon programcsomagok telepítésére, frissítésére és törlésére függőségek és egyéb integritást megőrző információk figyelembe vételével.

Alkalmazások

A Gomba for PHP rendszer alkalmazásai hierarchiába szervezett komponenspéldányokból épülnek. A hierarchiát jól meghatározott gyermek-szülő viszony jellemzi, például egy alkalmazás komponensnek lehet egy ablak gyermeke, és az ablaknak lehet egy „Hello World" tartalmú szöveg gyermeke. A mag ezen hierarchia elmentésére képes a kliens kérésére létrehozott válasz elküldése után, valamint ezt a hierarchiát állítja vissza a kliens következő kérésénél. Minden kérés alkalmával a komponensek `Process()` függvénye hívódik meg, melyben lehetőség nyílik az események kezelésére.

Hivatkozások

- PHP dokumentáció: http://www.php.net/
- MySQL dokumentáció: http://www.mysql.com/
- Apache webszerver dokumentáció: http://www.apache.org/
- Gomba for PHP dokumentáció: http://www.agmen-software.com/gomba-for-php/

„Hello World" alkalmazás

```
<?php
include_once("gomba/Gomba.php");

class CMyWebApp extends CWebApp {
 function CMyWebApp() {
  $window = new CWebWindow('fw', 1);
  ➥ // Ablak létrehozása, melyben a Hello World
    // szöveg megjelenik...
  $hello = new CWebText('Hello World!');
  ➥ // Hello World szöveg létrehozása
  $window->AddChild($hello);
  ➥ // Szöveg hozzáadása az ablakhoz
  $this->AddChild($window);
  ➥ // Az ablak hozzáadása az alkalmazáshoz
 }
}
GRunApplication('CMyWebApp');
?>
```

Példa eseménykezelésre

```
<?php
include_once("gomba/Gomba.php");

class CAutoForm1App extends CWebApp {
 var $m_eventmatrix;
 var $m_status;
 var $m_autoForm;

 function CAutoForm1App() {
  $window = new CWebWindow('fw', 1);
  ➥ // Ablak létrehozása
  $manager = new CManager();
  ➥ // Tartalom menedzser létrehozása
  $manager->SetLanguage('hu');
  ➥ // Magyar nyelv beállítása az automatikusan
  ➥ //detektált nyelv helyett

  $this->m_autoForm = new CAutoForm('af', array(
     'manager'=>&$manager,
     'fields' => array(
      array('Text', 'text', 'must'),
      array('Number', 'number', 'must', 100),
      array('Select', 'select', 'must',
      ➥ array('elso','masodik')),
      array('Check', 'check', 'normal')
     )
    )
   );

  $this->m_status =
            ➥ new CWebText('form is not ready');

  $window->AddChild($this->m_autoForm);
  ➥ // Űrlap hozzáadása az ablakhoz
  $window->AddChild($this->m_status);
  ➥ // Szöveg hozzáadása az ablakhoz
  $this->AddChild($window);
  ➥ // Ablak hozzáadása az alkalmazáshoz

  // Eseménymátrix létrehozása
  $formDp =&
➥ $this->m_autoForm->GetDataDispatcher('ready');
  $this->m_eventmatrix = new CEventMatrix();
  $this->m_eventmatrix->
➥ AddDimension('formReady', $formDp, 'ready');
  $this->m_eventmatrix->AddCallBack('OnFormReady',
➥ array('formReady' => 'true'));
 }

 function Process() {
  CComponent::Process();
  $this->m_eventmatrix->HandleEvents($this);
 }

 function OnFormReady() {
  $this->m_status->SetText('form ready.
➥ Text is \'' . $this->m_autoForm->
➥ GetValueByName('Text') . '\'');
 }
}
GRunApplication('CAutoForm1App');
?>
```

Nováki Szilárd

Az AGMEN-Szoftver Bt. programtervezője. Vezetésével két éve kezdődött el a Gomba for PHP projekt, amelynek lehetőségeit kihasználva átfogó hálózati ügyviteli rendszereket alakított ki, melyek a webes megjelenés mellett biztosítják a teljes elektronikus ügyvitelt.

Elterjedt technológiákra építő webes fejlesztőrendszer

A fejlesztőrendszer feladata

Az Internet használatának terjedésével a weben található információk és publikálásuk minősége egyre fontosabb szempont a cégek számára. Az elmúlt években megfigyelhető előrelépés, hogy a közepes-, sőt a kisebb vállalkozások is dinamikus weboldalakat készíttetnek. Azonban nem engedhetik meg maguknak a nagy méretű webalkalmazások írásához készített fejlesztőeszközök használatát, az azokhoz szükséges szakértelemmel nem rendelkeznek. Általában az egyszerűen használható szkript nyelvekkel fejlesztenek, inkább ad-hoc jelleggel, tervezés nélkül, időről–időre bővítgetve alkalmazásukat. Ez átláthatatlan kódhoz, bonyolult továbbfejlesztéshez vezet, és nem hatékony.

Mint a kutatásokból kiderül, amennyiben a fejlesztői csoportok teljesen figyelmen kívül hagyják a szoftver tervezésének feladatát, a következmények sokszorosan megbosszulják magukat rajtuk vagy utódaikon a szoftver karbantartása vagy továbbfejlesztése során. Számukra egy olyan keretrendszert kell tehát kidolgozni valamely népszerű szkript nyelven, amely a gyakran felmerülő specifikus problémákra választ ad, gyorsítja és struktúrálttá teszi a fejlesztést, valamint hatékony alkalmazások létrehozását teszi lehetővé. A keretrendszerre fejlesztőeszköz, vagyis kódgenerátor építhető, amellyel a fejlesztési munka nagyban felgyorsítható. Az elkészült fejlesztői környezet (az angol speed, azaz sebesség szóra hajazva) a cPEED nevet kapta. Jelenleg a harmadik verziójánál tart.

Meg kell tervezni ezen szkript-alapú webalkalmazások felépítését. Kiindulásként a hagyományos üzleti alkalmazások világából ismert struktúrák és ötletek alkalmazhatóak, azonban figyelembe kell venni a web felépítéséből és az elterjedt technológiák korlátaiból adódó feltételeket is. Az architektúra megtervezésénél a fejlesztői szerepkörök szétválasztása és az alkalmazás többnyelvű környezetben való felhasználása kitüntetett szerepet kell kapjon.

A keretrendszernek magasabb szinten meg kell felelni a kód nagyobb mennyiségéből következően nem egyszerű átláthatóság biztosításának, az alkalmazás vagy részei újrafelhasználásának, illetve az új környezetre való egyszerű konfigurálhatóság követelményeinek. Alacsonyabb szinten technológiai problémákat kell áthidalnia. A lényegesebbek: a kód és megjelenítés szétválasztása, az üzleti logika funkcionális egységeinek elkülönítése, a munkamenetek folyamatának biztosítása, a felhasználók azonosításának, bejelentkezési folyamatának egységesítése, jogosultságaik kezelése, az űrlapkezelés egyszerűvé tétele, az adatbázis-alapú információkezelés folyamatának szabványosítása és gyorsítása, fájl erőforrások és meta-adataik kezelésének egyszerűsítése.

A kódgenerátor elsődleges feladata a fejlesztés felgyorsítása az ismétlődő feladatok elvégzésével, valamint a keretrendszer használatából adódó többletfeladatok átvállalásával. Ez magában hordja az alkalmazás előkészítésének, a keretrendszer telepítésének feladatait, valamint a különböző típusú fájlok kiindulási vázának elkészítését is.

Az ismétlődő feladatokhoz tartozó (PHP- és HTML) kódot a fejlesztő által megadott paraméterek alapján állítja elő. A funkciók összefogását egy integrált fejlesztőrendszer végzi, amely jelen esetben a szerveren, önálló webalkalmazás formájában fut. A fontosabb feladatok: az adatbázist felhasználó üzleti logika függvényeinek előállítása, az adatok megjelenítéséhez tartozó összetett elemek (adatrácsok) elkészítése, nyelvfüggetlen szövegrészek beillesztése, űrlapok, űrlap-elemek felvitele, azok ellenőrzése, a felhasználó utasításait kezelő funkciók létrehozása és kezelése.

A webalkalmazás felépítése

A tervezett webalkalmazások felépítése a jól ismert több rétegű modellt követi. Az architektúra fontos rész, mivel itt történik a fejlesztői szerepeknek (pl. fejlesztő, grafikus, HTML-tervező, fordító) megfelelő alkalmazás-részek szétválasztása. A megértéshez induljunk ki a kérés kiszolgálása során lejátszódó folyamatokból!

A HTTP protokollal lekért tulajdonképpeni oldalakat nevezzük PHP oldalaknak. Ezeket kicsit nehéz meghatározni a hagyományos alkalmazások rétegei alapján, ugyanis szerepük leginkább közvetíteni a megjelenítés illetve az üzleti logika között, tehát a két réteg határán helyezkednek el. Ezért a megjelenítési logika névvel illetem őket. Ezeknek a PHP fájloknak jól meghatározott struktúrájuk van. A felhasználói oldalról érkező inputok (űrlapok, adattáblák) ellenőrzésén, a jogosultság-ellenőrzésen, illetve a nyelvfüggő elemek behelyettesítésén túl nem végeznek más tevékenységet, minthogy az elvárt funkcionalitáshoz a megfelelő modul valamely függvényét az aktuális paraméterekkel meghívják.

Ezek a modulok kétfélék lehetnek: a keretrendszerhez tartozó, általános, webalkalmazáshoz kötődő tevékenységet végző modul (ilyenek: session, language, log, forms, filebroker, stb.), vagy a konkrét alkalmazás üzleti logikáját megvalósító modulok. Ezek az állományok szintén PHP szkriptek, azonban nem kötődnek konkrét oldalakhoz. Egy modul tartalmaz egy objektumot, amely használatához az adott állományt be kell illeszteni (`require`) az oldalhoz tartozó szkript fájlba. Az objektum függvényei egy-egy jól meghatározott funkcionalitást képviselnek. A modulok végezhetnek adatbázis-hozzáférést, vagy egyéb fájlokkal műveleteket. A modulok és adatbázis között az elterjedt ADODB adatbázis-függetlenségi réteg helyezkedik el.

Miután a webalkalmazás elvégezte a tennivalóit, valamilyen választ kell küldenie, méghozzá HTML nyelven. Minden PHP oldalhoz tartozik egy (néha több) HTML sablon állomány, amelyet a grafikus készít el. A webalkalmazás ezeket küldi válaszként. Természetesen, mivel dinamikus

weboldalakról van szó, a válasz nem lehet mindig ugyanaz az oldal. A sablonokban speciális mezők vannak, amelyek helyére a dinamikus tartalom kerül az ún. template parser felhasználásával. A kódgenerátor alapvetően a kvázi-szabvány Smarty sablonrendszerrel működik együtt, azonban más parserekkel is összeköthető. A gyorsítótárazást a Turck MMCache végzi.

A fejlesztőrendszer további elemei

Mint azt az architektúrát leíró részben láttuk, a modulok az üzleti logika megvalósításáért felelősek. A funkcionalitást a modulok osztályainak tagfüggvényei valósítják meg, így ezeknek a függvényeknek logikusan jól szeparálható műveleteket kell végrehajtani. Modulokból két típust különböztetünk meg: a keretrendszerhez tartozó alap modulokat, illetve egyéb, alkalmazás-specifikus modulokat. A kettő között felépítésben és használatban semmiféle különbség nincsen. A megkülönböztetést az indokolja, hogy az alapmodulok olyan szolgáltatásokat nyújtanak, amelyeket minden egyes oldalon fel kell használnunk (munkamenetkezelés, nyelvi műveletek, naplózás, stb.), vagy pedig általános műveleteket, amelyeket szintén nagyon gyakran használunk (pl. űrlapok, strukturált fájl-tárak kezelése, web szolgáltatások), és emiatt a keretrendszer szerves részét képezik. Az efféle funkcionalitások kihasználására kódgenerátor támogatás is készült, és a megjelenítési logika szkriptjeinek struktúrájában is meghatározott helyük van.

Az előzőekben egy jól definiált szabályok szerint felépített, körülhatárolt elemeket és szerkezeteket tartalmazó architektúra képe bontakozott ki. Ez a strukturáltság azonban nem öncélú. Tervezésekor ugyanis azt tartottam szem előtt, hogy megfelelő eszközzel – egy kódgenerátorral – a webalkalmazások során előkerülő elemeket automatikusan elkészíttethessem.

Maga a kódgenerátor egy webalkalmazás, amely a cPEED architektúrára épül. Éppen azt használja ki, hogy a keretrendszer miatt a gyakori feladatok megoldása egyértelművé válik, tehát egyes paramétereken kívül egységes lesz. Ezeket a paramétereket kérik be a webes varázslók.

A keretrendszer használatából adódóan bizonyos többletmunka is származik. Ilyen például a fájlok struktúrájának felépítése, az azonosításhoz, munkamenethez szükséges kódrészletek, vagy a nyelvfüggetlenséghez az összes kiírandó szöveg erőforrásfájlokba gyűjtése és feldolgozása. A kódgenerátor elsődleges feladata ezeknek az elrejtése.

Az első varázsló egy teljesen új alkalmazás készítését kezdi meg a rendszeren. Ugyancsak alkalmas arra, hogy egy már kész cPEED alkalmazást telepítsünk.

A webalkalmazások készítésének legjelentősebb gyorsítását a „megjelenítési logika" névvel illetett rétegben lehet elérni. Azonban ahhoz, hogy ezekbe az oldalakba helyesen és áttekinthetően vigyük be a funkcionalitást, gondoskodnunk kell a megfelelő szerkezetről, valamint a nyelvi és sablon állományok létrehozásáról is. Ennek a folyamatnak a megkönnyítésére (kiváltására) készítettem egy varázslót, amely a megadott adatok alapján a megjelenítési logika fájljainak alapját készíti el.

Az oldal létrehozása után a megjelenítési logikához tartozó generátorelemeket tartalmazó oldalra jutunk, ahol a varázslók elindításán kívül az oldalon található cPEED objektumokat (pl. Form) választhatjuk ki további szerkesztésre.

Az alkalmazások üzleti logikájának megvalósítása a modulokban történik. A modul fájlok vázának legenerálásához a varázslónak meg kell adni, hogy mely táblákkal fog az adott modul dolgozni. Ezek neve nem lesz bedrótozva a modulba, hanem a tényleges nevüket a modul osztály konstruktorában kell megadni, a konkrét adatok a konfigurációs állományban szerepelnek. Egyéb konstruktor paramétereket is megadhatunk a létrehozandó modulhoz.

A modulok négy fő funkciója az alap adatbázis-függvényekre épül, így ilyenek készítéséhez a kódgenerátor támogatást nyújt. Az adatbázis lekérdezések kódját tartalmazó függvények úgy készülnek, hogy közvetlenül kapcsolhatóak legyenek egy megjelenítési elem, az adatrácsok bemenetére. Így lapozható, rendezhető, szűrhető listákat egyszerűen, csupán kattintgatásokkal létre tudunk hozni.

További függvényeket generáltathatunk adatok beillesztésére, felülírására, illetve törlésére. Mindezek a függvények a relációs adatmodellhez kapcsolódnak.

A fejlesztőeszköz által biztosított varázslók közül még egy csoportot kiemelnék: ezek a HTML űrlapok készítését végzik. Az adott oldalon található űrlapokat bármikor szerkeszthetjük, a listában nemcsak a megszokott HTML űrlap elemek szerepelnek, hanem újabb, összetettebb változatok is (pl. checkbox-tömb). Természetesen az egyes beviteli mezők szerkesztése is varázslókkal történik (pl. menük, radiogomb-csoportok elemeinek feltöltése, ezek is történhetnek adatforrás modul-függvényekhez való kötéssel). A mezőkhöz az érvényességet vizsgáló validátor függvényeket adhatunk, melyek meghívása szerver oldalon történik. Az elemekhez, pl. nyomógombokhoz, linkekhez parancsokat köthetünk, melyekhez az űrlapnak megfelelő objektumban eseménykezelő-függvények tartoznak.

Összefoglalás

A kitűzött cél átlátható struktúra készítése a közepes- és nagyméretű webalkalmazások számára, valamint a fejlesztési idő lecsökkentése volt. A webes környezetből származó nehézségek és fogyatékosságok elfedése és az ismétlődő munkák átvállalása mellett a keretrendszer használatával a webalkalmazások fejlesztése jóval kényelmesebbé is vált, ezzel is segítve, hogy a programozó a konkrét problémák megoldásával tudjon foglalkozni.

Mindezek mellett a cPEED nyílt a további fejlesztésekre, illetve egyéb rendszerekkel (pl. ASP.NET) történő együttműködésre, valamint webszolgáltások fejlesztésére. Ezek a területek további kutatások tárgyát képezik a Budapesti Műszaki és Gazdaságtudományi Egyetemen.

Forstner Bertalan

A Budapesti Műszaki Egyetemen szerzett mérnök informatikusi diplomát, jelenleg ugyanott PhD hallgató. Kutatási területei közé tartozik a webes keretrendszerek és hatékony webfejlesztés vizsgálata, a szemantikus peer-to-peer információ-visszakeresés, valamint szoftverfejlesztés Symbian platformra. PHP fejlesztéssel 1998 óta foglalkozik, a cPEED keretrendszert készítő csapattal számos nemzetközi és hazai projektben vett részt.

Így készült a Weblabor

A Weblabor megújítását évek óta terveztük, sok próbálkozás után végül idén jutottunk el odáig, hogy egy új megjelenéssel és egy több lehetőséget kínáló motorral el tudtuk indítani az új verziót. Még sokat kell rajta finomítani, de úgy gondoljuk, hogy elégedettek lehetünk az eredményekkel.

Ebben a cikkben, illetve előadásban pár, a fejlesztés során használt eszközt fogok bemutatni, valamint röviden azt a döntési folyamatot, mely a használatukhoz vezetett. Nem titkolt célom, hogy másoknak is kedvet adjak ezeknek az eszközöknek a használatához. A bemutatandó programok platformfüggetlen, ingyenes, szabad forrású megoldást nyújtanak, melyet az is tükröz, hogy a fejlesztés is több platformon zajlott: nagyrészt UHU-Linux, kisebb részt Microsoft Windows rendszeren.

Programok, eszközök

Az alábbiakban a következő alkalmazásokról esik szó, melyeket valamilyen formában a Weblabor felépítésénél hasznosítottunk:

- Subversion verziókezelő rendszer
- Eclipse fejlesztői környezet
- Firefox webböngésző

Ezeken kívül természetesen további kisebb-nagyobb programokat is felhasználtunk, ezeket nem mutatom be, részben helyszűke miatt, részben azért, mert ezek nem képezték annyira szorosan részét a Weblabor fejlesztésének. Az előadásban ezen a három eszközön kívül az mnoGoSearch keresőszerver is bemutatásra kerül.

Subversion verziókezelő rendszer

A tavalyi konferencia weblapjának készítése közben, valamint egyéb projektjeinkben végzett munkáink során megtapasztaltuk, hogy ha hatékonyan együtt szeretnénk dolgozni, akkor mindenképpen elengedhetetlen ehhez valamilyen segédeszköz. Az FTP-s belépéssel és módosításokkal több kérdés és probléma is jelentkezett: ki min dolgozott; mikor írtunk felül valamint; nem adtunk jogot más felhasználóknak az új állományokra; nem mentettük el a korábbi működő verziót; stb. Úgy láttuk, hogy az egyetlen hatékony megoldás a problémáinkra csak és kizárólag egy verziókezelő rendszer lehet. Röviden szólva nagyon bevált a dolog, de hogy ne ugorjak hirtelen a végére, lássuk, hogy milyen további lépéseket tettünk, és végül miért kötöttünk ki a Subversion mellett!

Aki a szabad forráskódú környezetekben járatos, annak a verziókezelő rendszer szóösszetételre rögtön a CVS jut eszébe, és ezzel mi sem voltunk másként. Az egyszerű telepítés után azonban nagyobb feladatnak bizonyult az igényeink szerinti beállítás. Mivel a szerveremen több repository-t (több projekt számára kialakított helyet) is be szerettem volna állítani, ezért fontos szempont volt a jogosultságkezelés. Úgy tűnt, hogy a megoldást a CVS műveletek előtt és után lefutó wrapperek hozhatják, ezekkel azonban csak azt tudtuk megoldani, hogy csak bizonyos emberek írhassanak adott állományokat. Hiányzott azonban a különböző fájlok olvasásának védelme, azaz ha valakinek volt egy felhasználója, akkor az összes projektet olvasni tudta. Ráadásul nem is tűnt lehetségesnek minden CVS parancshoz illesztett wrappert készíteni. A keresgélés végeredményeként arra jutottam, hogy két megoldás lehetséges: vagy kijavítjuk a CVS forráskódját és a számunkra szükséges megoldásokat megírjuk bele, vagy pedig egy másik lehetőség után nézünk. Az előbbi út a CVS belső világának kellő ismerete, és a saját javításaink karbantartásához szükséges idő hiányában végül nem volt reális lehetőség, ráadásul éppen akkor több biztonsági rést is találtak, ennek következtében gyakoribbak voltak a verzióváltások is.

A Subversion már a honlapja (http://subversion.tigris.org/) alapján is rendkívül bíztató tulajdonságokkal bírt, és tapasztalatból mondhatom, ezek a gyakorlatban is működnek: több lehetőség van különböző fájlműveletek végrehajtására, a biztonság és annak szabályozása is jobban átgondolt, ráadásul nyílt és már bevált szabványokon alapul: HTTP(S) alapú WebDAV/DeltaV protokollal kommunikál.

A telepítés után gyakorlatilag nem volt vele gondom, itt is csak a beállítások maradtak hátra. Mivel a kommunikációhoz egy Apache 2 szerver biztosítja az alapkövet (bár különálló szerverként is lehet telepíteni), ezért a jelszavas védelem semmiben sem különbözött a máshol megszokott könyvtárakra történő hozzáférési beállításoktól. További indok volt a verziókezelő rendszer használata mellett, hogy információt kapjanak a projekttagok a változásokról. Erre a megoldást végül az UHU-Linux Subversion rendszeren működő fejlesztéséhez használt, egy kicsit hozzánk illesztett szkript biztosította: a fejlesztők minden változtatásról kapnak egy levelet, mely tartalmazza a hozzá kapcsolt megjegyzést, valamint a módosítások teljes listáját is. Már csak egyvalami volt hátra: a Subversionbe kerülő változtatásokat a projekt honlapjára is egyből el kell juttatni. Erre végül azt a megoldást választottuk, hogy minden commit során a szkriptünk átmásolja a webszerverre az új verziót. Ennek több előnye is van, az egyik például, hogy eggyel több helyen megvannak a fájlok, gyakorlatilag biztonsági mentésként.

A Subversion rendszerhez több jó kliens is rendelkezésre áll. Windows alatt a lehető legjobb megoldást a TortoiseSVN (http://tortoisesvn.tigris.org/) nyújtja, mely beépül a Windows Explorerébe, és ezen keresztül az olyan fejlesztőeszközökbe is, melyek az Explorert használják. Linuxos megoldásként a legegyszerűbb az eredeti parancssoros Subversion klienst használni, hiszen minden lehetőséget megad, ami rendelkezésre áll, és (legalábbis számomra) sokkal kézenfekvőbb megoldást is nyújt. Amennyiben valakinek nem válnának be ezek a megoldások, elég széles a választék további kliensekből (http://subversion.tigris.org/project_links.html).

A rendszert azóta is megelégedetten használjuk, sőt annyira jól bevált hogy többek között a konferencia honlapját is így tartjuk karban.

Eclipse fejlesztői környezet

A projekt beindulásakor éppen Microsoft Windows alatt dolgoztam (az Internet Explorer szükségessége miatt), így ezen a rendszeren működő fejlesztői környezet után kellett néznem. Mivel a Weblabor alapja UTF-8 kódolásra épül, ezért eléggé fontos szempont volt ennek a kódolásnak a támogatása. Meglepetésemre, nem sok ilyen szerkesztőt találtam, a legtöbb híres, színes-szagos megoldás elbukott rajta (legalább tíz bíztatót próbáltam ki). Egyéb követelmények megítélése után a keresés végére a jEdit és az Eclipse maradt, s bár mindkettő viszonylag jól támogatja a PHP alapú fejlesztést, nekem az utóbbi vált be jobban.

Az Eclipse (http://www.eclipse.org/) célja egy általános, vállalati szintű platform kialakítása, melyet egy kis Java programozói tudással bárki kiegészíthet. Rengeteg plugin (http://eclipse-plugins.info/, http://www.eclipsepluginscentral.com/) érhető el hozzá, ha jobban keresünk, számos a PHP-hez és a webes fejlesztéshez kapcsolódó plugint találhatunk. Bemutatandó az Eclipse lehetőségeit, ezek közül lássunk néhányat. Az általam egyik legjobbnak tartott plugin, az EclipseColorer (http://colorer.sourceforge.net /eclipseplugin.html), mely a kódjaink színezését vállalja magára. Rengeteg nyelvet ismer és támogat nagyon jól, illetve köszönhetően az XML alapú színezést leíró megoldásának, akár saját nyelveinket is megismertethetjük vele. Érdekesség, hogy a PHP kóddal egy fájlban levő HTML, CSS és Javascript részeket is képes színezni, a leíró nyelve alkalmas arra, hogy egy adott területre egy másik kódszínezést használjon. Lehetőséget ad a nyitó/záró elemek (zárójel, idézőjel, HTML elemek) kijelzésére (egy adott elemnek láthatóvá teszi a párját is), illetve gyorsbillentyűkkel a kettő aktuális között is ugrálhatunk. Az éppen szerkesztett állomány HTML formátumba is exportálható, így dokumentációk is kényelmesen készíthetőek segítségével. A végére hagytam egy további kellemes tulajdonságát: az Eclipse által biztosított kivonat nézetet mutató dobozba is képes dolgozni, azaz a legtöbb nyelvnél megjelenik ebben a dobozban a szerkesztett állomány tömör, vázlatos nézete.

Az Eclipse lehetőségeit lezárandó, az egyik PHP szerkesztést legjobban támogató megoldást, a TruStudio/PHP-t (http://www.xored.com/features.php) mutatnám be. Ez a plugin is képes kódszínezésre, s bár sokkal kevesebb 'nyelvet beszél' (XML, HTML, PHP), jelentősen többet tud PHP esetén, így megéri használni (különösen, hogy működőképes a Colorer pluginnel párhuzamosan). Emellett azonban még számos, nyelvi szerkesztőktől elvárható megoldással rendelkezik: például ha egy PHP függvény felé állunk az egérkurzorral, megjeleníti a függvény paraméterezését és rövid leírását. Nagyon részletes kivonatot készít, a kódunkban található utasításokat részben értelmezi. Beépített megoldással rendelkezik debuggolásra, helyi, vagy akár távoli gépen is, illetve mindeközben lehetőséget biztosít a változók megjelenítésére, sőt, akár megváltoztatására is. Mindent összevetve bizton állíthatom, hogy PHP fejlesztőknek nagyon jól használható pluginről van szó.

A bemutatott pluginek alapján talán látható, hogy nem egy egyszerű, kis otthon készült szerkesztőről, hanem egy komoly fejlesztői környezetről van szó, melynek viszonylag könnyű programozhatósága az egyik legnagyobb előnye, s ennek kapcsán az, hogy sok olyan plugint találni hozzá, mely számunkra is hasznos megoldást nyújt.

Firefox webböngésző

A bemutatott programok közül talán ez az, ami a legismertebb mindenki számára, így csak pár érdekes kiterjesztésére hívnám fel a figyelmet. A Firefox a szabványokat nagyon jól támogató, az általános böngészést sok funkciójával megkönnyítő, HTML és JavaScript programozással nagyon könnyen kiegészíthető Mozilla alapú böngésző (korábban Phoenix, majd Firebird névvel). A webfejlesztők számára nyújtott rengeteg lehetősége miatt, ha lehet ilyet mondani: nyomatékosan javaslom, hogy mindenki ezt a böngészőt használja a fejlesztés során (persze a végeredményt más böngészőben is le kell ellenőrizni).

Az alapfunkciói között a fülek segítségével történő böngészés, beépített Javascript konzol, felugró ablakok blokkolása szerepel, de különböző kiterjesztések segítségével hatékonyan lehet szűrni a reklámokat, meg lehet tanítani az egér gesztusokra, stb. A fejlesztők napi munkáját nagyban segítő két kiterjesztést mutatnék be.

Az egyik a Web Developer (http://chrispederick.myacen.com/work/firefox/webdeveloper/), mely egy új eszköztárat tesz elérhetővé számunkra. Az eszköztár tíz részre van osztva: Disable, Forms, Images, Information, Miscellaneous, Outline, Resize, Validation, View Source, Options, melyek mindegyike 1-8 további menüpontot rejt. A rengeteg lehetőséget felesleges felsorolni, javaslom mindenkinek, hogy próbálja ki saját maga! Érdekességként pár funkció: jelentés az adott oldal űrlapjairól, hibás képhivatkozásokról, az adott oldal sütijeiről, továbbá az ablak átméretezése különböző felbontásokra, ellenőrzés sebesség és szabványok szempontjából, elemek kiemelése különböző tulajdonságok szerint, stb. Valóban nagyon sok a mindennapi munkában jól használható funkcióval bír.

Egy másik alapvető fontosságú kiterjesztés az EditCSS. Ez nem tud annyi mindent, mint az előbbi eszköztár, viszont amit tud, azt nagyon jól tudja: az adott oldalhoz rendelt stíluslapot, illetve stíluslapokat jeleníti meg a bal oldali sávban. Mindez persze nem lenne forradalmi funkció, ha nem lenne lehetőségünk az élőben történő szerkesztésre a változások azonnali követésével. A tapasztalatok szerint ez nagyon gyors CSS fejlesztést tesz lehetővé, hiszen gyorsan tudunk kipróbálni több megoldást is, mentés és az oldal újratöltése nélkül is.

Úgy gondolom, hogy ez a két leghasznosabb általános webfejlesztői célú kiterjesztés, illetve miután kipróbálta az ember ezeket, nehéz nélkülük élni. Bár hasonló megoldások előbb-utóbb más böngészők alá is megjelennek majd, egyelőre egyszerűen nem látok alternatívát a webfejlesztők számára: aki kimarad (nem használ Firefox-ot), az lemarad.

Bártházi András

A Wish Internet Consulting ügyvezető igazgatója. Cégének vezetőjeként ars poeticája, hogy a legfrissebb technológiák segítségével, az ügyfelek igényeit messzemenőkig kielégítő megoldásokat nyújtsanak, lehetőségek szerint alternatívákat kínálva. Több nagyobb portál elkészítése fűződik nevéhez, többek között a www.euroastra.hu, vagy a folyamatosan frissülő www.uhulinux.hu, legutóbb az egyik kormányzati weblap megjelenésének szabványokon alapuló megújításában vett részt. Cége hosztolja a www.weblabor.hu oldalt (melynek egyik szerkesztője) és a kapcsolódó levelezőlistákat. A konferencia egyik szervezője.

PHP a grid technológiában – egyszerűen és célratörően

Bevezető

A grid technológia eredete az 1998-as évekig nyúlik vissza, noha az előzményei a párhuzamos és elosztott rendszerek területén már korábban is megtalálhatóak voltak. Mára világossá vált, hogy a nagy teljesítményű adatfeldolgozást igénylő tudományos és műszaki számításoknak új informatikai infrastruktúrára van szüksége. Ezt az infrastruktúrát megvalósító szoftver és hardver megoldásokat fogjuk a továbbiakban grid technológia néven illetni. Összefoglaljuk a hazai grid infrastruktúra fejlesztés egy prominens képviselőjét az NIIFI ClusterGrid Infrastruktúrát, és azt, hogy az ehhez kapcsolódó szoftver fejlesztésekben miként használtuk sikeresen a Web és a PHP technológia nyújtotta lehetőségeket.

NIIFI ClusterGrid Infrastruktúra

Magyarországon 2002 júliusában merült fel, hogy az Oktatási Minisztérium által a felsőoktatási intézmények számára PC laborok (34 labor kb. 2000 gép) beszerzésére kiírt pályázatát szerencsésen lehetne kombinálni a hazai grid-szerű infrastruktúra hardver alapjainak megteremtésével. Az elképzelés szerint az egyetemeknek, főiskoláknak szállított PC laborok napközben szokásos feladatukat látják el, de éjszaka és hétvégenként egy egységes rendszerbe kapcsolódnak, és tudományos műszaki számításokban vesznek részt.

Az elmúlt másfél évben elsősorban hálózati és operációs rendszer szinten történtek intenzív fejlesztések, s már a második generációnál tartunk. Hálózati szinten messzemenően támaszkodtunk az akadémiai hálózat gerincének (HBONE) lehetőségeire. Így figyelembe véve a biztonsági és megbízhatósági követelményeket egy teljesen szeparált MPLS VPN rendszer került kialakításra az akadémiai gerinchálózaton belül. A felhasználók néhány jól védett belépési ponton keresztül „érintkeznek" a rendszerrel publikus hálózaton keresztül.

Operációs rendszer szinten egy Linux alapú vékony-kliens megoldás mellett döntöttünk és minden PC labort egy önálló klaszterbe szerveztünk egy vagy több központi szerverrel.

A ClusterGrid egy országos összefogáson alapuló rendszer, melyben szinte minden bekapcsolt egyetem és főiskola tevékenyen részt vesz a hálózati, szoftver fejlesztésekben vagy a tesztelésben és az üzemeltetésben.

ClusterGrid Bróker

Az első generációs Condor alapú ClusterGrid rendszer számos ok miatt alkalmatlannak bizonyult a feladatra. Ekkor döntöttünk amellett, hogy saját feladatütemezőt készítünk, mely figyelembe veszi a ClusterGrid rendszer sajátosságait, de egyben prototípusa is egy olyan erőforrás-ütemezőnek, amit a ClusterGrid rendszer keretein kívül is működőképesnek gondolunk.

PHP és web technológia – miért?

Az erőforrás-ütemező elkészítésekor a web technológia mellett tettük le voksunkat. Ennek a döntésnek a magyarázata több okra vezethető vissza. Egyfelől figyelembe kellett vennünk, hogy jelenleg a grid technológia területén nincsenek széles körben elterjedt szabványok, továbbá, hogy a jelenleg rendelkezésre álló szoftver eszközök (elsősorban a Globus, a Condor és egyéb hasonló rendszerek) éles környezetben használhatatlannak bizonyultak, mind biztonsági, mind megbízhatósági, mind üzemeltethetőségi szempontból. Továbbá a grid technológia egy igen gyorsan változó kutatási, fejlesztési terület, mind a felhasználók, mind az üzemeltetők oldaláról. Ezért gyorsan kell tudnunk reagálni a változásokra. Világossá vált, hogy a web technológiában számtalan a grid rendszereken belül is felmerülő kérdésre már kiforrott, hatékony és megbízható megoldás található.

Végül, de nem utolsó sorban a PHP technológia hatékony „ragasztó" szerepet tud betölteni a különféle – akár operációs rendszereken átívelő – komponensek között is.

Architektúra

A ClusterGrid rendszerben egy feladatot egy könyvtár reprezentál, melynek meghatározott struktúrával kell rendelkeznie. Ez a struktúra gyakorlatilag megegyezik a FHS-ben rögzített alapelvekkel. Emellet a könyvtár struktúrának tartalmaznia kell egy ún. submit állományt, ami a feladat specifikációját tartalmazza. Ez magában foglalja a futtatandó bináris állomány nevét, a szükséges paramétereket, stb.

A rendszer szolgáltatásait különféle komponensek hatékony együttműködése valósítja meg. A kommunikációs alrendszer az átvitelt HTTPS protokoll felett valósítja meg POST üzenetváltásokkal. Kliens és szerver oldali azonosítást egyaránt végez a ClusterGrid rendszer saját CA-ja által kibocsátott tanúsítványok alapján.

A rendszerben található erőforrásokkal kapcsolatos adatok begyűjtését, tárolását az információs rendszer végzi. A feladatokkal végezhető műveletek egy az egyben tranzakciókra képeződnek le és az erőforrások és az ütemezők bizonyos események bekövetkezésekor notificationokat küldenek egymásnak. Az ütemező alrendszer

feladatok erőforrásokhoz rendelését végzi, valamint figyelemmel kíséri a feladatok állapotváltozásait is.

A rendszer egyik központi eleme az a futtató környezet, ami a helyi ütemezőn keresztül biztosítja a feladatok biztonságos végrehajtását, oly módon, hogy a felhasználóknak nem enged közvetlen hozzáférést az erőforrásokhoz és minden feladatot önálló dinamikus rendszerazonosítóval (UID/GID) lát el. A dinamikus azonosító kiosztást a Grid Underground projekt keretében létrehozott libnss-dynmap és IDRegister szoftverek segítségével valósítottuk meg.

Ehhez járulnak még az egyéb, a hibakezelést, naplózást, adatbázis-hozzáférést biztosító objektumok. A komponensek relációs adatbázisban tárolnak minden perzisztens adatot állapotukról.

Összefoglalás

Összefoglalva elmondhatjuk, hogy a ClusterGrid Infrastruktúra feladatütemező rendszerének megvalósításakor a web és PHP technológia nyújtotta keretek kielégítőnek és hatékonynak bizonyultak. A fejlesztés folyamatosan zajlik, új ötletek merülnek fel, amiket gyorsan tudunk megvalósítani a kialakított keretek között. A jelenlegi rendszer legnagyobb erénye meglátásunk szerint a kicsi, áttekinthető és egyszerű szerkezete.

Linkek

Condor – http: //www.cs.wisc.edu/condor/

Globus – http: //www.globus.org/

FHS – http: //www.pathname.com/fhs/

Grid Underground – http: //gug.sf.net/

Szalai Ferenc

Fizikus PhD hallgató az MTA Szilárdtestfizikai es Optikai kutatóintézetében valamint a NIIF Iroda tudományos munkatársa. Mindkét intézetben a nagy számítás igényű tudományos és műszaki problémák megoldásával foglalkozik grid, klaszter, web és egyéb elosztott technológiak segítségével.

Segítség! Felnőttem!

avagy nagy terhelhetőségű, magas rendelkezésreállású rendszerek építési és üzemeltetési útmutatója.

Tipikus web-kiszolgáló rendszerek

Megszületett

Ezen cikk célja, hogy rövid, velős áttekintést adjon a webkiszolgálók működéséről, valamint rávilágítson azokra az egyedi sajátosságokra melyek ezen típusú kiszolgálókra érvényesek. Kiemeljük azon pontokat, amelyeken egy webkiszolgáló szerver nagymértékben különbözik más kiszolgálóktól, pl. egy levelezéskiszolgálótól vagy fájlszervertől.

Fontos látni, hogy a webkiszolgálók fejlődése erősen összefügg az internet és az azt működtető technológiák fejlődésének történetével.

Kezdetben igen nagy mennyiségű szöveg állt rendelkezésre, míg multimédiás anyagok egyátalán nem léteztek. Talán emlékeznek még erre az FTP-Mosaic böngésző nevével fémjelezhető időszakra. Tartalom gyakorlatilag csak angol nyelven létezett, és egy oldal tipikus mérete 1 és 20Kbyte között mozgott. Egy átlagos email mérete a néhány ezer bájtos méretet nem haladta meg és a sávszélesség (vagy inkább „sávszűkség") jobb kihasználását figyelembevéve nem küldtek egymásnak felesleges adatokat. Később megjelentek a képek, animációk (gif89a) ezek mérete 4 és 30Kbyte között mozgott. Egy átlagos felhasználó 9600 – 48Kbit/sec sebességgel érhette el a hálózatot.

Már a kezdetek kezdetén megjelentek a célhardveres megoldások. Példaként a sokak által ismert SUN COBALT-ot hoznám fel, – ami természetesen nem azonos az ismert phpBB, sablonozó megoldásával – mely kifejezetten webkiszolgáló folyamatokra kifejlesztett céleszköz, bár architektúrájában és felépítésében alig tér el egy hagyományos SUN eszköztől, erőforrásait tekintve viszont messze alul marad egy, a napokban használatos PC-vel szemben.

Később az RFC 1866, 2854 HTML 2.0 illetve a RFC 1341 MIME adta lehetőségek meghozták a multimédiás tartalmak korát az internet világába. Ettől a pontól kezdve a fejlődés új fordulatot vett.

Egy tojást egy körtével?

Ahhoz, hogy megértsük egy webkiszolgáló lelkivilágát, fontos látni a működésbeli különbségeket. Bár egy webkiszolgáló működése erős hasonlóságot mutat egy fájlszerver működéséhez, mégis vannak alapvető különbségek.

Egy fájlszerveren azonos időben viszonylag kevés számú – átlagosan 50-150 – felhasználó veszi igénybe a szolgáltatásokat és általában közepes és nagy méretű fájlokat töltenek le a szerverről. Ennek megfelelően egy fájlszerver erőforrásigénye inkább az IO alrendszert terheli.

Egy webszerver – szerencsés esetben – 150-1500 kérést fogad el és szolgál ki azonos időben és a fájlméretek jellemzően a néhány 10Kbyte és a néhány 100Kbyte között mozognak. Természetesen ez erőssen függ a kiszolgált tartalomtól. Szélsőséges esetben egy kiadott fájl mérete akár több száz megabájt is lehet, ám ilyen esetek szerencsére ritkák. Egy dolog szinte majdnem biztos, hogy a kiszolgált tartalom jól bufferelhető, ezért megfelelő mennyiségű memória felhasználásával csökkenthető az IO alrendszerre nehezedő nyomás.

Egy levelező szerver esetében a tulajdonságok függenek a felhasznált rendszer típusától, (MailBox vagy MailDir) ezért

- A MailBox jellegű kiszolgáló folyamatok esetében minden POP3 bejelentkezésre végig kell olvasni az adott felhasználó fiók-fájlját, ami a fálj méretétől függően vesz igénybe IO erőforrást, azaz csak igen nagy MailBoxok esetében jelent problémát.
- MailDir rendszerű kiszolgálók esetében a tipikus fájlméret bár csupán csak néhány Kbyte, ám ilyen IO műveletből igen nagy számút kell végrehajtani adott idő alatt és a forgalom nem bufferelhető, ezért igen masszív IO alrendszert igényelnek. (UW3 SCSI)

Fontos látni, hogy a fenti különbségeken túl a webkiszolgálás folyamata más erőforrásokat is igényel a szerver oldalon, hiszen még meg sem említettük a CGI programok futtatásának többletterheit. Egy rosszul megírt PHP alkalmazás akár csekély terhelés hatására is felemésztheti a rendszer erőforrásainak túlnyomó többségét.

All-in-one megoldások

Egy biztos, hogy amíg tíz évvel ezelőtt a tartalom nem volt más, mint Times New Roman betűvel szedett, feketefehér, vagy éppen mozgó gif képekkel tarkított oldalak sokasága, addig mára a PHP vagy Java alapú összetett, adatbázist használó rendszerek a jellemzőek.

Ennek megfelelően míg korábban az egy hoszt sok domain, sok honlap volt a trend, addig ma egy hoszton átlag tíz domain fut. Persze régebben is voltak, ma is vannak kivételek, de ezek csak erősítik a szabályt.

All-in-one megoldásnak tekintem azokat a kiszolgálókat melyeken a

- HTTP kiszolgálás folyamata
- a szerver oldali programok futtatása (CGI, Fast-CGI, SSI, PHP)
- az adatbázis kezelése és tárolása
- a DNS szerver
- levelező szerver

egy gépen, egyetlen hoszton fut.

Jelenleg a Szerverhotelben 1200 darab webkiszolgálót üzemeltetünk, ezek több mint 80%-a egygépes rendszer, mi több, – eltekintve a törpe minoritástól – ezek nagy része Debian Linux vagy más OpenSource alapú rendszer.

Web-kiszolgálo folyamatok skálázhatósága

Web-kiszolgáló rendszerek felépítése

Eltekintve néhány egyedi megoldástól, egy átlagos all-in-one (AIO) rendszer a következő elemekből áll:

Web-szerver

- Apache – 0-3000 egyidejű session
- Caudium – 1500-n egyidejű session
- Tomcat – Java alapú rendszerek kiszolgálására
- Vegyes üzemű rendszerek (pl. Apache+Tomcat, Caudium-Tomcat)

Adatbázis szerver (RDBMS)

- MySQL
- PostgreSQL
- DB2

MTA

- Sendmail
- Exim
- Postfix

POP3/IMAP

- Ipop3d

NAME szerver

- Bind

Az így felépített rendszerek gyenge pontja, hogy amennyiben valamely alrendszer terhelése növekszik, úgy a megnövekedett igények kiszolgálásához szükséges erőforrásokat a többi alrendszertől veszi el.

Közelebbről ez azt jelenti, hogy ha a terhelés a webszerveren esik (PHP-pharse), úgy az RDBMS rovására igyekszik a szükségleteit kielégíteni és viszont. Ennek hatására egyik alrendszer sem fog optimálisan működni.

Ilyen jellegű problémákat az alrendszerek szétválasztásával lehet megoldani. Tipikus a webszerver – adatbázisszerver szétválasztása. Olcsó, ám hatékony megoldás, hogy a webkiszolgáló gépbe több interfészt beépítve egy belső lábon keresztül kommunikál az adatbázisszerverrel.

A Terhelés fogalma

Terhelésnek nevezzük azt az üzemszerű állapotot amikor a kiszolgálóra kapcsolódó felhasználók onnan adatokat töltenek le, ezzel terhelve a rendszer rendelkezésre álló erőforrásait:

- Memória
- CPU
- PCI busz
- Merevlemez IO

A Terhelés természete

„A terhelés az egyik legkevésbé kiszámítható dolog a szerver életében”.

Ennek okát abban kereshetjük, hogy kizárólag az üzemeltető szándékán kívül álló okok befolyásolják a terhelés mértékét és természetét. Jelesül a rendszer látogatottsága, az egyidőben a szolgáltatásokat igénybevevő kapcsolatok száma, bár függ az üzemeltető szándékától és a tartalomtól, de lényegében az internet felhasználói döntik el, hogy mikor és melyik rendszert látogatják. Persze itt is akadnak eltérések, hiszen van olyan – rosszul tervezett és implementált – rendszer, mely néhány száz, míg más rendszerek melyek akár több ezer szimultán felhasználó kéréseit szolgálják ki, azonos peremfeltételek mellett.

A terhelés mindig egy statikus és egy dinamikus részből áll, a dinamikus rész (a fenti okoknál fogva) nem számítható ki előre, ezért a rendszereket mindig a statikus-terhelési adatok alapján méretezzük!

Tervezés, méretezés

Milyen adatokra van szükségünk?

Fontos előre látni:

- a tervezett terhelést, a várható egyidejű kérések számát
- a szükséges tárolókapacitást (fájlrendszer méretezéséhez)
- Milyen szolgáltatásokat fogunk használni a rendszeren
  - Ezek közül melyiken milyen forgalom várható
  - Mely rendszereket kell redundánssá tenni

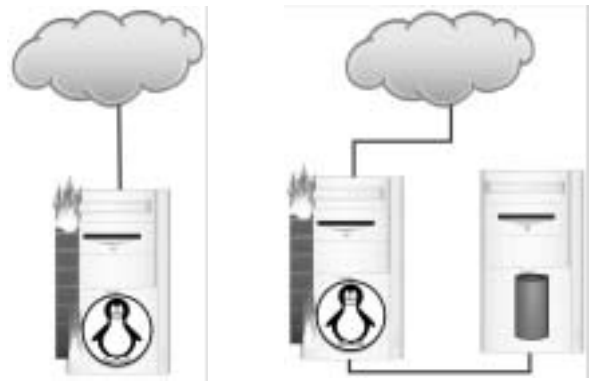
Hogyan skálázzuk rendszerünket?

Elméletek, topologiák miértek és hogyanok

Egygépes rendszer

Mint már említettem jelenleg az ilyen rendszer a legelterjedtebb. Ennek oka, hogy az üzemeltetés rendkívül gazdaságos, hiszen egyetlen szerverről van szó. Természetesen az ilyen rendszerek a legkevésbé terheléstűrőek. Skálázhatóságuk kimerül az erőforrások fokozatos bővítésében. Amennyiben ez az egygépes rendszer nem egy nagyobb Blade része, úgy pl. a CPU és memória erőforrások bővitése csak leállással oldható meg.

Ahogy a terhelés növekszik és az erőforrások bővíthetősége kimeríti a rendelkezésre álló lehetőségeket, úgy célszerű szétválasztással növelni a hatékonyságot.



Front-end/RDBMS

Természetesen az adatbázis szerver leválasztása a legcélszerűbb és a legkönnyebben kivitelezhető megoldás. Az ábrán látható megoldásban a webszerver egy újabb interfésszel gazdagodott és egy patch-kábellel csatlakozik az adatbázisszerverhez. Nem failover, nem elegáns, viszont igen gazdaságos megoldás.

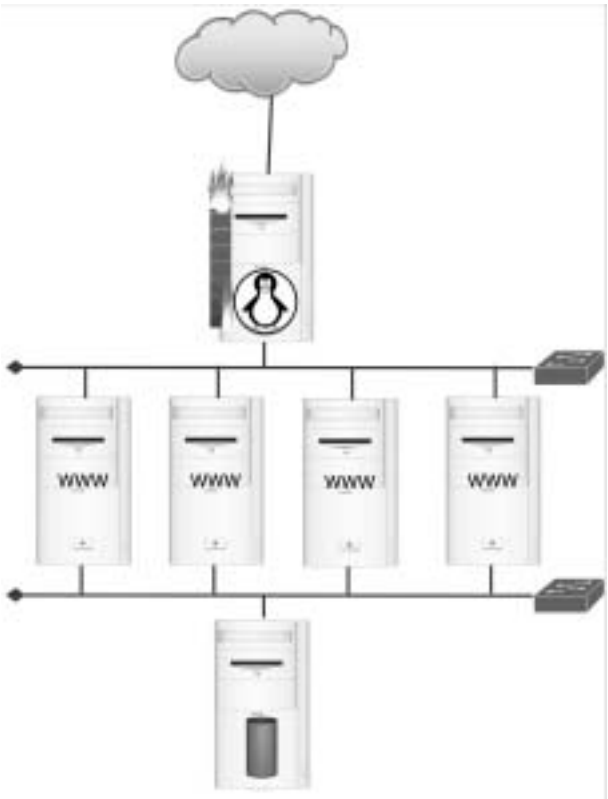
Ám a gazdaságos megoldások is csak korlátozott mértékű skálázhatóságot tesznek lehetővé. Ha a tényleges terhelés a webszerveren csapódik le, úgy kézenfekvő, hogy a frontendek számának növelésével lépjen tovább.

DNS- load-balancing (Round Robin DNS)

A frontendek számának növelése magával hozza azt a problémát, hogy mi módon irányítsuk a HTTP kérést egyik vagy másik szerver felé. A következő megoldás lényege, hogy egy egyszerű DNS szerver beállítással elérjük, hogy egy A rekordhoz több IP tartozzon a zónában.

```
IN   A   www   195.70.37.15
IN   A   www   195.70.37.16
IN   A   www   195.70.37.17
IN   A   www   195.70.37.18
```

A fenti beállítás hatására a DNS szerver az első kérésre az első alias-hoz tartozó címet, míg a másodikhoz a másodikat -és így tovább- társítva adja a felhasználó böngészőjének. A terhelés egyenletesen osztható szét a frontendek között. Olcsó, hatékony, volt idő amikor a Google is ezt a módszert választotta a terhelés szétosztására.



Ez természetesen feltételezi, hogy

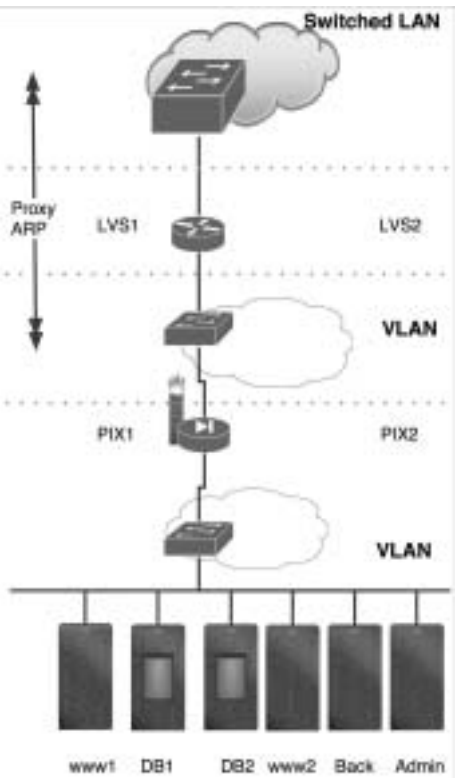
- rendelkezünk kellő mennyiségű IP felett,
- a frontendek azonos teherbírásúak,
- és mindegyik működik.

Mi van, ha mégsem?

Abban az esetben, ha valamelyik frontend nem tudja kiszolgálni a kérést, a felhasználó időtúllépés után „404 az oldal nem található" hibaüzenetet kap a kért oldal helyett, ám frissítéskor egy másik, még működő frontend címét kapva ismét elérhető az oldal számára.

LBS/LVS

Sokan sokféleképpen tévesztik össze ezeket a fogalmakat. Ennek a szakasznak a célja, hogy magyarázatot adjon az alapvető különbségekre, valamint egy gyakorlati megvalósítás keretein keresztül ismertesse a rendszer elvi működését, illetve azokat a komponenseket, melyek a működést szavatolják.



Nem célunk általános telepítési útmutatót adni, inkább gondolatokat ébreszteni és utat mutatni egy rendszer megvalósítására.

Load Balancig Server. Elegáns, szofisztikált megoldás, melynek lényege, hogy a router (linux box) a külső lábán egy IP címen fogadja a bejövő kéréseket és DNAT-tal címzi meg a belső lábon, (nem feltétlenül) publikus címen lévő frontendeket.

A gyakorlatban az LBS megvalósítását a Linux Virtual Server kernel kiegészítéssel valósíthatjuk meg. A jól konfigurált kernelen kívül szükség lesz még a hidden és az ipvs kernel patch-ekre, valamint az ipvsadm nevű csomagra is. Ez utóbbiból a Debian Woody csomagot célszerű választani kompatibilitási okokból.

A telepítés és konfigurálás után az ipvsadm paranccsal lehet beállítani, hogy melyik külső címre érkező kéréseket

melyik belső cím(ek)re lehet továbbítani és ezeket milyen prioritással, súlyozással tegye a router.

A hangsúly itt a súlyozáson van, hiszen így mód van eltérő terhelhetőségű frontendek üzemeltetésére is. A router nyilvántartja, hogy melyik frontendre mennyi hívást irányított át, valamint számolja az aktív TCP kapcsolatokat is.

Az LVS router ily módon intelligensen meg tudja oldani, hogy a kéréseket a kevésbé terhelt (kevesebb kiadott és kevesebb aktív kapcsolat) frontend felé továbbítsa.

Mi van, ha mégsem?

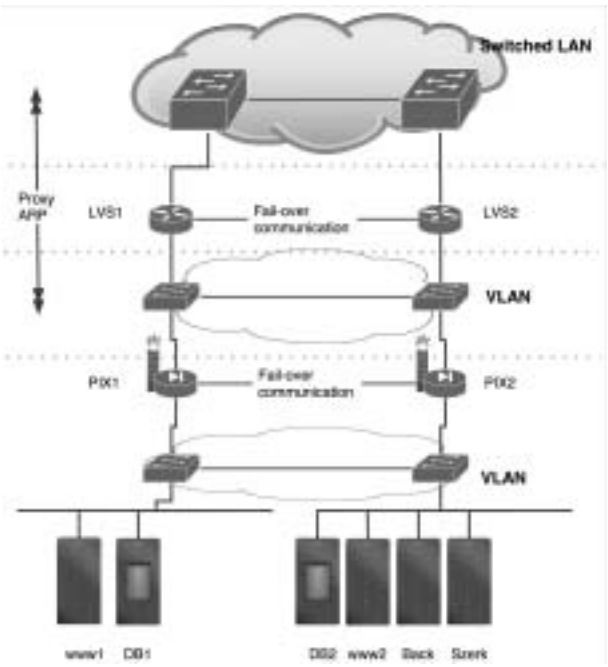
Az előbbi kérdést célszerű ebben az esetben is feltenni. Ha egy frontend meghibásodik az alaprendszer pontosan ugyanúgy viselkedik, mint az RRDNS esetében. A kérés kiszolgálása elakad, időtúllépés után a felhasználó a 404 hibaoldal tartalmával ismerkedhet meg.

Amennyiben a fentieket kiegészítjük egy monitoring megoldással (pl. MON, vagy NAGIOS), akkor nemcsak a frontendek telemetria adataihoz jutunk hozzá, hanem probléma esetén (pl. egy frontend kiesése) mód van arra, hogy az adott eszközt kivegyük az ipvs táblából, pontosan olyan egyszerűen mint egy tűzfal-szabály esetében. Ezen a módon megvalósítható, hogy csak korlátozott számú kapcsolatvesztéssel automatikusan átirányítsa a kiesett gépre eső terhelést.

Természetesen, ha maga az LVS hibásodik meg, akkor az egész rendszer elérhetetlenné válik, de ez elkerülhetetlen egy olyan rendszer esetében melynek feladata „csak" a terhelés elosztása!

Építsünk hibatűrő rendszert!

Értelemszerűen, ha az előző vázlatot kiegészítjük egy második, tartalék rendszerrel, akkor a feladat megoldható!



Ebben a megoldásban az aktív eszközöket és a tűzfalat is megkettőztük. Ha a hardver eszközök ezt lehetővé teszik, akkor van mód a failover kommunkációra is az eszközeink között. Ez minőségi aktív eszközök esetében spaning tree beállításával oldható meg, míg LVS routerek esetében a heartbeat alkalmazásával.

Ekkor az eszközök a hálózaton (TCP) és opcionálisan soros vonalon (erősen javasolt!) figyelik egymás tevékenységét. Ha valamelyik rendszer leáll, arról értesül a tartalék rendszer, felveszi az eredeti eszköz IP címét és funkcióját. Ha az eredeti eszköz ismét működőképes lesz, a kommunikációs csatornán keresztül értesül a másik eszköz jelenlétéről és „tartalék" állapotot vesz fel.

HA-Cluster

High availability, azaz magas rendelkezésreállás. Álmaink rendszere nem áll le. A hibákat automatikusan javítja, és dinamikusan reagál a terhelés változásaira. Általánosságban elmondhatjuk, hogy az előbbiekben vázolt rendszer alkalmas magas rendelkezésreállású, nagy terhelhetőségű webkiszolgáló folyamatok megvalósítására, ám nem alkalmas számításigényes feladatokra, mert:

- a maximálisan rendelkezésre álló erőforrás egy gép erőforrása,
- nincs mód részfeladatok egyszerre több gépen történő végrehajtására,
- nehezen konfigurálható, hiszen nincs közös fájlrendszer.

Ennél jobb megoldásnak látszik, ha olyan rendszert üzemeltetünk, melyre igazak az alábbiak:

- Rendelkezik a Computing Clusterre jellemző Thread átdobás képességével (OpenMosix).
- Rendelkezik a Socket Migration képességével.
- A rendszer elemei közös fájlrendszert használnak (SAN, Coda) így az adatokat csak egy helyen kell tárolni.
- A rendszer elemei nincsenek feladatra dedikálva, azaz nem specializálódnak és szükség szerint intelligensen átkonfigurálják magukat egy-egy feladat végrehajtására.

Tekintettel arra, hogy a konferencián szó lesz a ClusterGrid-ről, így most nem térek ki a megvalósítás miként jére, szándékom csupán útmutatás volt.

Üzemeltetési kérdések

Az üzemeltetés követelményei

A szerver, még ha hasonló alkatrészekből („szegény ember" szervere) is épül fel, mint egy munkaállomás, mégis teljesen más üzemeltetési körülményeket kíván meg a biztonságos működéshez, mint asztali társa. Ennek oka alapvetően a folyamatos üzemből adódó, az alkatrészekre nehezedő terheléskülönbségben kereshető.

Hol üzemeltessünk?

Évekkel ezelőtt a vállalatvezetők többsége meg volt győződve arról, hogy a vállalati webszervernek közvetlenül a fáljszerver és a nyomtatószerver mellett, – zömmel a takarítószer-tároló helységből átalakított – szerverszobában

van a helye. Ezért sok helyen még mind a mai napig, – jó esetben egy bérelt vonal végén, – a levezőszerver fogalmával konkurálva végzik feladatukat ezek az eszközök.

Nem meglepő tehát, hogy a

- vállalat mérete, illetve a
- vállalat kommunikációjának növekedése folytán,
- vagy éppen a honlap látogatottsága miatt

több és több szakemberben érik meg az elhatározás, hogy a gondosan felépített rendszerüket a feladatnak megfelelő környezetben tárolja, üzemeltesse, pedig az a környezet ami egy ember számára megfelelő, egy szervernek maga a pokol (por, légmozgás, hőmérséklet, páratartalom).

A Data centerrel szemben támasztott követelmények

A fenti problémakörre megoldásként a Szerverhotelt hoznám fel, mint egy olyan helyszínt, ahol a szerverek biztonságos és tartós üzemeltetéséhez elengedhetetlenül szükséges feltételek,

- mint a széles (és egyre szélesebb)-sávú internetelérés,
- az automatikus tűzjelző és tűzoltó-rendszer,
- a szünetmentes védett áramellátás,
- a vagyonvédelem,
- a pormentes és szabályozott páratartalmú és ellenőrzött hőmérsékletű klimatikus viszonyok
- ipari-szinten biztosítottak csakúgy, mint a 24 órás operátori felügyelet.

Elegendő biztonság elve

Biztonságtechnikai alapszabály: egy rendszer védelmére optimálisan annyi erőforrást érdemes áldozni, hogy a biztonsági rendszer kompromittálásához szükséges energia arányban álljon a megszerezhető birtok értékével, ugyanakkor ne akadályozza felesleges mértékben a rendszerhez töténő üzemszerű hozzáférést.

Kovács Zsolt

Az Interware Szerverhotel igazgatója. E munka mellett a mai napig több, saját rendszert (webkiszolgáló rendszereket, intra- és extraneteket) is üzemeltet, ma már „csak" hobbiból. Sajátjának vallja a rendszerintegrátor, a rendszergazda, és a fejlesztő látásmódját is.