Merre tovább, melyik úton?

Nézzünk körbe a webes világban: milyen módszerek és eszközök vannak, melyek a legígéretesebbek?

Amerikai testvérlapunk elérkezett a webes témákkal foglalkozó rovatának ötvenedik cikkéhez. Ez alkalommal nem egy témakör boncolgatására, hanem az internetes fejlődési irányvonalak átfogó elemzésére vállalkoztak, amit merész jövendöléssel is kiegészítettek.
A Kovácsműhely (At the forge) 1996 eleje óta jelenik meg rendszeresen a *Linux Journal*-ban. Az utóbbi néhány évben ezeken az oldalakon számos webbel kapcsolatos módszert, eljárást és alkalmazást ismerhettek meg az olvasók, ezek között minden megtalálható az egyszerű CGI programoktól kezdve a mod_perlt használó adatbázis-kezelő alkalmazásokig. Következzék most néhány gondolat a webes alkalmazások jövőjéről.
Egyrészt a dolgok soha nem álltak ilyen jól a webes alkalmazások fejlesztői számára: az eszközök még mindig igen gyors ütemben fejlődnek, és az összetett alkalmazások készítése egyre egyszerűbbé válik. Másrészt viszont a beágyazott programnyelvek, alkalmazáskiszolgálók és adatbázis-csatolók területe igencsak zsúfolt, így nehéz eldönteni, hogy nekünk pontosan mire van szükségünk.
Írásomban megkísérlem a webes módszerek és alkalmazások jövőjét megjósolni, ezért a szövegből következtetni lehet arra is, hogy milyen témaköröket érintünk a Kovácsműhely rovat későbbi számaiban. Az e havi cikkben ismertetett irányvonalakat követi saját tanácsadó cégem is, tehát az olvasó kitalálhatja, hogy az elkövetkező évben (vagy akár később) milyen javaslatokat teszek majd. Mivel mind a *Linux Journal*, mind a cégem elsősorban a linuxos kiszolgálókkal foglalkozik, ezért a témakörön belül főként a Linuxszal és az ingyenes programokkal kapcsolatos részterületekre összpontosítok.

Egy kis visszatekintés

A webes alkalmazások fejlesztésének története a Web hőskorában kezdődött. Azóta, hogy az első dinamikus oldalt letöltötték (ez még jóval a CGI megjelenése előtt történt, az Internet Explorer, a Netscape és az Apache pedig még csak gondolatban létezett), a programozók egyre nagyobb számban készítenek összetett alkalmazásokat az Internetre.
A CGI, vagyis a „Common Gateway Interface” megérkezésére sem kellett sokáig várni. A CGI eredetileg azért készült, hogy a nem webes alkalmazásokat webes felhasználói felülettel lehessen ellátni. A CGI-nek hála, létrejöhettek az első hordozható kiszolgálóoldali programok. A legtöbb webes alkalmazást még ma is a CGI segítségével írják, és ezt főleg egyszerűségének és teljesen felületfüggetlen kialakításának köszönheti. Az sem elhanyagolható tény, hogy a tárhelyszolgáltatók minden további nélkül engedélyezhetik felhasználóik számára a CGI programok futtatását, hiszen azok (kellő körültekintés mellett) nem veszélyeztetik a kiszolgáló biztonságát. CGI programot bármilyen kiszolgálóhoz és operációs rendszerhez készíthetünk, tetszőleges nyelven; majdnem biztos, hogy működni is fog. Azonban a CGI-nek is akad néhány hátrányos tulajdonsága. Például a kiszolgálónak minden olyan kérelemhez új folyamatot kell indítania, amely valamilyen CGI-programra irányul. Más szóval: egy percenként 60 új látogatót fogadó honlap kiszolgálója minden másodpercben új folyamatot indít.
Ez még önmagában nem is olyan aggasztó, hiszen egy egyszerű linuxos gép képes akár tizedmásodpercenként új folyamatot indítani, ugyebár? Az egyes folyamatok mérete és indulási sebessége azonban már fontosabb szerepet játszik.
A Perl, mely évek óta a kedvenc programnyelvem, bebizonyította, hogy segítségével hatékony CGI-programokat írhatunk. A CGI.pm modul rengeteg szolgáltatást bocsát rendelkezésünkre, ezek segítségével a CGI-vel kapcsolatos álmainkat tökéletesen megvalósíthatjuk (sőt, olyanokat is, melyeket valószínűleg soha nem használnánk ki). A Perl emellett a mintakeresésre is igen jól alkalmazható, valamint a népszerű internetes protokollokat és szabványokat támogató modulokat is megtaláljuk benne. A DBI (adatbázis-csatoló) modult is rengetegen használják. Segítségével a dinamikusan létrejövő oldalakba SQL-lekérdezések kimenetét igényeink szerint illeszthetjük be.
Bármennyire is tömör, rugalmas és biztonságos a Perl, a CGI-szabvány eredeti rendeltetésénél fogva alkalmatlan nagy mennyiségű dinamikus oldal egyidejű létrehozására. Minden Perlben írt CGI-program futtatásához új folyamatot kell indítani, a Perlt, majd a programot be kell tölteni a memóriába, a programot le kell fordítani a Perl belső kódjára, és végül a Perl futtatómotorjával értelmezni kell. Így nem csoda, hogy elég néhány tucat egyszerre futó CGI-program, és egy átlagos kiszolgáló máris megadja magát.
Ennek ellenére a CGI sikeres, mert egyszerű. Semmilyen más API segítségével nem írhatunk meg egy „Szia világ!” típusú programot ilyen könnyedén:

```
#!/usr/bin/perl -wT
use strict;
use CGI;

my $query = new CGI;
print $query->header("text/html");
print $query->start_html;
print "<P>Szia, világ!</P>\n";
print $query->end_html;*
```

Beágyazott nyelvek

Az utóbbi pár évben a webes programfejlesztés önálló szakterületté nőtte ki magát: a programozónak a rendszer, a hálózat és az adatbázis üzemeltetésével is tisztában kell lennie, emellett a megfelelő programozói módszereket és biztonsági szempontokat is fejben kell tartania.

A webes fejlesztéseknek jelenleg három olyan iránya van, melyek alapjaiban változtathatják meg a felhasználók és a fejlesztők helyzetét. A három irányvonal együttes használatát „alkalmazáskiszolgálónak” nevezzük.
Az első irányvonal inkább a régi elvek módosítását jelenti: a programozók egyre inkább eltávolodnak az egyszerű CGI-programoktól – az alkalmazásokat a kiszolgálóba vagy más környezetbe építik. És miért használjuk a CGI-programokat dinamikus oldalak készítésére? Mert a webkiszolgáló önállóan nem képes egyéni HTML-oldalak előállítására. Elméletileg lehetséges volna, ehhez azonban egy új Apache modult kellene írnunk C nyelven, majd azt a rendszerbe fordítani – ez azonban a legtöbb esetben óriási feladat lenne, és az időtakarékosság itt elsődleges szempont.
Szerencsére, létezik egy közbülső megoldás a két változat (csak kiszolgálóoldali, illetve csak külső programok használata) között. Mi lenne, ha a kiszolgálót egy egész programnyelvvel bővítenénk, és az általunk igényelt szolgáltatásokat ennek segítségével valósítanánk meg? Ez a módszer az utólagos bővítéseket és javításokat is nagymértékben leegyszerűsíti, és így a kiszolgáló újrafordítását és újraindítását is megúszhatjuk.
Ez a lényege a mod_perl használatának, mely az Apache kiszolgálót a Perl nyelvvel bővíti. Segítségével Perl nyelven elérhetjük az Apache legbelső szerkezetét és értékeit, így a lekérdezés tárgyául szolgáló oldallal, fájllal bármit megtehetünk. A mod_perl bármire képes az Apache-ban, amit máskülönben egy C nyelvű modullal hajtanánk végre, az egyéni válaszfejlécektől kezdve az azonosítás módjának megváltoztatásáig.
A CGI-programokkal ellentétben, ahol a Perl a programot egyszer fordítja le, egyszer futtatja, majd kilép, a mod_perl a program lefordított változatát a gyorstárba helyezi, majd annyiszor futtatja, ahányszor ez szükséges. Ne felejtsük el, hogy ez a módszer néha igencsak megnöveli a rendszer memóriaigényét, a programozóknak pedig rendkívül körültekintően kell eljárniuk.
Régebben a mod_perl volt az egyetlen olyan modul, mely egy beágyazott programnyelvvel bővítette az Apache-t, manapság azonban egyre több ilyen elem kerül rivaldafénybe. Az egyik ilyen a mod_snake, mely ugyanazt végzi a Python programok esetében, mint a mod_perl a Perlnél, azaz segítségével egyéni Apache-kezelőket írhatunk Pythonban. Még egy mod_tcl is létezik, mellyel az Apache-t beágyazott Tcl-lel bővíti, de jómagam még nem hallottam olyan honlapról, ahol ezt használnák.
Egy másik nyílt forráskódú webkiszolgálónak, az AOLServernek már régóta van beépített Tcl-támogatása. Így a dinamikus tartalmat Tcl-eljárásokkal is elkészíthetjük anélkül, hogy ehhez CGI-programokhoz kellene nyúlnunk. Aki a Tcl helyett a Pythont szeretné megismerni, annak figyelmébe ajánlom a PyWX (Python Web Extensions, Python webes kiegészítő) nemrég megjelent bétaváltozatát. A PyWX az AOLServer által támogatott Tcl szolgáltatások mindegyikét megvalósítja Python nyelven. Ezzel együtt jár az is, hogy a PyWX az AOLServer számára elérhető Tcl-kódokkal nem képes együttműködni, ennek ellenére számos lehetőség kihasználását nagymértékben megkönnyíti – elég csak a Webről letölthető rengeteg Python-modulra gondolnunk.

A kód és a HTML keverése

A második fő irányvonal, hogy a dinamikus kódot a HTML forráskódba ágyazzuk be. A Microsoft Active Server Pages (ASP) nevű rendszere talán a legjobb példa erre, de ezek mellett jó néhány más hasonló jellegű rendszert használhatunk. Linux alatt például a Java Server Pages (JSP), a HTML::Mason (ez a mod_perllel működik), a PHP és az ADP közül választhatunk.
Javával a rendszer első megjelenése óta foglalkozom, és régóta tudom, hogy ez a nyelv valóban megér egy kis odafigyelést. Mint sok más szakembert, engem is az appletek megjelenése taszított el tőle, ezek ugyanis lassúak, nem biztonságosak és hibásak voltak.
Az utóbbi években azonban a kiszolgálóoldali Java igencsak megerősödött. Minden Java „servlet” (kiszolgálóoldali programka) egy osztály, mely egy Java Virtuális Gépen (JVM) belül fut.
A servletek a dinamikus tartalomkészítés minden területén használhatók – a JDBC segítségével elérhetjük az adatbázisokat, HTTP-fejléceket húzhatunk le és módosíthatunk, és a válaszok tartalma a felhasználó igényei szerint módosítható.
A JSP oldalak segítségével könnyebben dolgozhatunk a servletekkel, feltéve, hogy a <% és %> tagok közötti rész kivételével minden szabványos, HTML nyelven íródik. Amikor a böngészőből egy JSP oldalt kérünk le, a JSP azonnal egy Java servletbe, majd ezt követően egy Java .class fájlba fordítódik. Ez a .class fájl töltődik be a servlet motorba, itt lefut és a „közelben” marad, egy későbbi meghívás esetére. A JSP-k és a servletek a Java Bean objektumokat is használhatják, ezekkel például a felhasználói szokásokat modellezhetjük. Ilyen elemek vezérlik a legtöbb mai háromrétegű webalkalmazás üzleti logikáját.
A mod_perl segítségével hatékonyan készíthetünk Apache-kezelőket, de néha túl alacsony szinten kell dolgoznunk miatta. Ennek kiküszöbölésére létezik néhány modul, amelyekkel a Perl kódot a HTML fájlba illeszthetjük. A HTML::Mason, mellyel a rovat tavalyi számaiban már foglalkoztam, a személyes kedvencem, és ezt egyszerű formai követelményeinek és az egymásba építhető sablonoknak köszönheti. Az idei YAPC::Europe találkozón Londonban láttam a Template Toolkit (Sablonkezelő Eszköztár) bemutatóját, mely a HTML::Masonra hasonlít, de bővítményeket is használhatunk benne. Míg a Java és a Perl főleg a kiszolgálóoldali webes programozási feladatok elvégzésére alkalmas, a PHP-t kifejezetten a dinamikus tartalomkészítésre fejlesztették ki. A PHP rengeteg szolgáltatással rendelkezik, ezeket sokféle fájltípushoz, adatbázishoz és internetes szabványhoz felhasználhatjuk. A PHP legfrissebb változatai Java-objektumokkal is képesek dolgozni, sőt, a CORBA csatolójára sem kell már sokáig várni. A PHP egyetlen hátránya, hogy nem bővíthető: ha későn jövünk rá, hogy például PDF fájlokkal is szeretnénk dolgozni, akkor ezt csak a PHP újrafordításával érhetjük el.
Az AOLServer felhasználói egy másik hasonló rendszerrel büszkélkedhetnek, melynek neve ADP (AOLServer Dynamic Pages) az ADP oldalak lehetővé teszik, hogy a Tcl-kódot a HTML fájlba illesszük be, ahol a Tcl az AOLServer által nyújtott szolgáltatások bármelyikét elérheti. Készíthetünk tehát olyan ADP oldalt, mely rekordokat olvas ki egy adatbázisból, egy másik kiszolgálóról letöltött HTML oldalt értelmezhetünk, vagy a felhasználói adatbevitel alapján számításokat végezhetünk.

Kapcsolatok az adatbázissal

A kiszolgálóoldali programozás világában a harmadik új irány az állandó adatbázis-kapcsolatok létrehozása. Az adatbáziskiszolgálókat eredetileg felhasználónként napi egy-két kapcsolatra készítették, és nem percenként vagy másodpercenként egyre. Gondoljuk csak el: ha egy CGI-programmal másodpercenként kapcsolódunk az adatbázishoz, akkor az eredeti terhelés 86 ezerszeresével dolgoztatjuk azt a szerencsétlen kapcsolatot! Már ha ez nem jelent különösebb gondot, azonban rengeteg olyan adatbázis létezik, ahol egyetlen kapcsolat is túl sokba kerül.
Az egyik megoldás lehet, hogy a kiszolgáló indulásakor létrehozunk egyetlen kapcsolatot, és ezt használjuk újra és újra, amikor egy programnak az adatbázisra van szüksége. Nagyjából ezt teszi a Perl,

Apache, mod_perl hármas együttes használata mellett alkalmazható Apache::DBI modul. Amikor a `$dbh->disconnect` segítségével leválunk az adatbázisról, az Apache::DBI csendben figyelmen kívül hagyja ezt, és életben tartja a kapcsolatot. A `DBI->connect` hívásakor az Apache::DBI beolvassa a kapcsolati karakterláncot, és az új kapcsolat indítása előtt megpróbál egy már létezőt fölhasználni. Mivel egy Apache-folyamat egyszerre csak egy HTTP-kérelmet szolgál ki, ezért folyamatonként egyetlen adatbázis-kapcsolatra van szükségünk. Az állandó kapcsolódást–leválasztást így megúszhatjuk.

Mindemellett ez azt is jelenti, hogy minden alfolyamatnak külön kell elérnie az adatbázist, ami egy erősen terhelt kiszolgáló esetén több száz vagy ezer egyidejű adatbázis-kapcsolatot jelent. Az AOLServer az adatbázis-kapcsolatok számát úgy csökkenti, hogy nem több folyamatot, hanem egyetlen, több szálat futtató folyamatot használ. Mivel a szálak ugyanabban a folyamatban kapnak helyet, így adatokat is képesek megosztani egymás között.

Az AOLServer ezt úgy használja ki, hogy a kapcsolatokból kis gyűjteményt hoz létre, és ha egy bejövő HTTP-kérelem igényt tart rá, akkor az egyiket kiválasztja, és ezen keresztül érhetjük el az adatbázist. A kapcsolatok igény szerint megoszthatók és nem kötődnek egy-egy szálhoz, így a kiszolgálónak jóval kevesebb kapcsolatot kell nyitnia az adatbázis felé.

A Java servletekkel és JSP oldalakkal való munka más megközelítést jelent. A Jakarta-Tomcat servlet/JSP rendszer általában a webkiszolgálón kívül található, tehát mindig a Tomcat folyamat részeként fut, az Apache alfolyamatok számától függetlenül. A Tomcat folyamaton belül tetszőleges számú servlet szál futhat egyidejűleg.

A servletek és JSP-k általában a JDBC segítségével érik el az adatbázist, és ugyanez igaz a Java Beanekre is (ezekkel a JSP-k és servletek magas szintű „gondolkodásra” képesek). A JDB azonban nem képes kapcsolatgyűjtemények kezelésére (connection pooling).

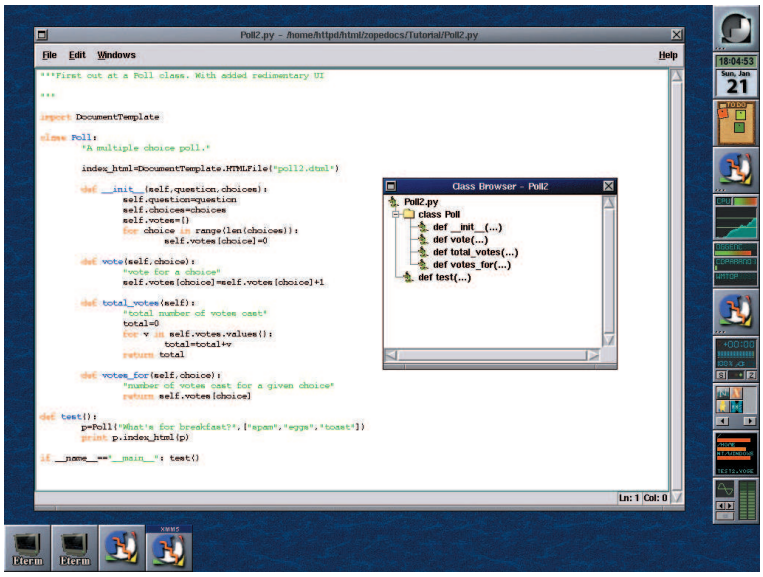
A JDBC 2.0 lehetővé teszi ezt is, de a művelet nem teljesen önműködő, és jelenleg kevés JDBC 2.0 meghajtó van.

Más nyelvek ezt teljesen másképpen oldják meg. Például a PHP adatbázis-meghajtói lehetővé teszik az állandó adatbázis-kapcsolatokat, de ezt a programozónak kérnie kell. Magyarul a `pg_connect` segítségével kapcsolódhatunk a PostgreSQL adatbázishoz, vagy állandó kapcsolatot is létrehozhatunk a `pg_pconnect` használatával. A két különböző elérési módszer megvalósítása az adatbázis programozójának feladata, a PHP programozónak pedig a kapott szolgáltatásokat kell a lehető legkedvezőbben kihasználnia. Mindezek közül az AOLServer állandó, gyűjteményes kapcsolatot biztosító módszere a legelegánsabb, hiszen bármilyen nyelvvel használhatjuk (bár általában ez a Tcl marad), és különösen jól méretezhető. A mod_perl Apache::DBI-je a Perl-programozók számára nagyszerű megoldás, mivel az önálló Perl-programokat és -modulokat nem kell módosítanunk azért, hogy az állandó kapcsolatokat kihasználhassuk. Az, hogy az Apache::DBI csak állandó, és nem gyűjteményes kapcsolatot ad, az Apache többfolyamatos felépítéséből következik. Az Apache 2.0, mely nagymértékben fog emlékeztetni az AOLServerre, valószínűleg már támogatni fogja a többszálas futtatást.

A JDBC gyűjteménykezelése (pooling) jó, még akkor is, ha már mindenki teljesen lemondott róla, és elkezdte saját osztályait írni. Azonban csak Java servletekkel működik, és nem segít az olyan kiszolgálóknál sem, melyeknek gyűjteményes kapcsolatokra van szükségük több szolgáltatás egyidejű működtetéséhez (mod_perl és JSP). A PHP rendszere talán a legkezdetlegesebb, hiszen nem bocsát rendelkezésünkre egységes felületet az adatbázis elérésére, az adatbázis-meghajtók nem kezelhetik önműködően a gyűjteményes kapcsolatokat, és a programok számára sem enged utat e kapcsolatok kihasználására. Az állandó kapcsolat viszont nagyon jól működik, és már ez önmagában is jelentős sebességnövekedést jelenthet.

Merre tartunk?

Bár az „alkalmazáskiszolgáló” kifejezést kétértelműsége miatt nem szeretem, tisztán látható, hogy a webes alkalmazások ebbe az irányba tartanak. Az alkalmazásokat már nem önálló programokként, hanem az alkalmazáskiszolgáló által rendelkezésünkre bocsátott objektumok és modulok egy csoportjaként kell elképzelnünk, melybe az általunk írt program tökéletesen illeszkedik. Sok esetben a lehető legkevesebb munkával is előállíthatunk bonyolult programokat, egyszerűen azért, mert a feladat nehezebbik részét mások már elvégezték helyettünk. Természetesen ez azt is jelenti, hogy az operációs rendszerre is másként kell tekintenünk, ugyanis valójában már csak az alkalmazáskiszolgáló legalsó rétege, s ez utóbbi a rendszer igazán lényeges eleme. Az ügyféloldali programok íróinak azt kell eldönteniük, hogy Windows, Unix vagy Macintosh rendszerre írják programjukat, a webes alkalmazások fejlesztőinek pedig az alkalmazáskiszolgálók közül kell választaniuk. Csakúgy, mint az operációs rendszerek esetében, az egyik alkalmazáskiszolgálóról a másikra történő áttérés is nehéz feladat. Ez, sajnos azt is magával hozza, hogy ha egy lassú, nehezen módosítható kiszolgáló mellett döntünk, akkor a jövőben lehet, nagyon komoly gondjaink lehetnek. Még az ugyanazon nyelvet és szabványokat használó alkalmazáskiszolgálók (például az Enhydra és az ATG Dynamo) is különböző objektumokkal és szolgáltatásokkal dolgoznak, így az áttérés eléggé fáradságos dolog.



A Zope kezelőfelülete

Az ingyenes programok hozzám hasonló híveinek ez azt is jelenti, hogy a nyílt forrású alkalmazáskiszolgálók legalább olyan fontosak, mint a nyílt forrású operációs rendszerek. Szerencsére az előbbi csoportból is számos változatot letölthetünk az Internetről. Működésükben és szolgáltatásaikban nagymértékben különböznek, de el kell ismernem, hogy a most ismertetendő módszerekhez még nem sokszor volt szerencsém. Remélem, hogy az elkövetkezendő hónapokban többet foglalkozhatok személyesen is ezekkel.

Talán a legismertebb alkalmazáskiszolgáló a Zope, ez sok részből épül fel, és kevesen ismerik elég jól. A Zope egy objektum-adatbázis, egy sablonozó rendszer és egy alapszintű tartalomkezelő rendszer. A Zope-pal még nem volt lehetőségem komolyabban foglalkozni, de a hallottak alapján egy rendkívül hatékony rendszer képe körvona-

lazódik előttem, amit főleg akkor használhatunk ki teljes mértékben, ha az igényelt szolgáltatást megvalósító modul már elérhető az Interneten.
Egy másik, gyakran emlegetett alkalmazáskiszolgáló rendszer az ArsDigita Content System, ennek fejlesztését nagyrészt az ArsDigita tanácsadó vállalat vezeti, s a GNU felhasználói engedély feltételei mellett használható. A rendszer egyik hátránya, hogy csak az Oracle adatbázis-kezelőt támogatja, ez egyébként egy kifogástalan rendszer, de elég drága és forráskódja sem nyílt. Erre is született már megoldás: az OpenARS nevű önkéntes fejlesztés a PostgreSQL adatbázissal igyekszik ötvözni a rendszert. Az OpenARS még nem készült el teljesen, de már most is hatalmas mennyiségű szolgáltatást találunk benne, melyek száma és hatékonysága minden bizonnyal tovább nő a jövőben.
Az XML jó néhány éve a legizgalmasabb viták tárgya a webes közösségben, de csak az utóbbi fél évben lehettünk szemtanúi egyre nagyobb mértékű elterjedésének. Az XML az adatokat a megjelenésüktől teljesen függetlenül írja le.
Az Enhydra egy Java-alapú alkalmazáskiszolgáló, mely sok tekintetben a Zope-ra hasonlít, de XML, Java servletek, JSP és Enterprise Java Beans használatát teszi lehetővé. Az Enhydra meglehetősen összetettnek tűnik, de legalább megbízható keretet teremt alkalmazásainknak.
Ha az XML-lel kívánunk dolgozni, akkor a Cocoon és AxKit csomagoknak is szenteljünk figyelmet. Az Apache Software Foundation által támogatott Cocoon most XML-adatok számára készít egy Java-alapú kiszolgálót. Az AxKit XML-alapú tartalomkészítést tesz lehetővé Perl nyelven, és így a programokat a tartalomtól, illetve a tartalmat a látványtól az XML, az XSL és a Perllel együtt az XSTL segítségével különíti el.
Végül az Oracle fejlesztését említeném meg, ennek neve Internet Application Server (IAS). Az IAS egy Apache-modul, mely egy Java futtatórendszerrel, az Enterprise Java Beansszel, JSP oldalakkal, a JDBC-csatolóval és az Oracle használatával végzi feladatát. Cikkem írásának időpontjában a rendszer még új és nem próbálták ki széles körben. Az Oracle természetesen ehhez sem ad forráskódot. Ennek ellenére az IAS futtatható Linux alatt és minden bizonnyal az Oracle-felhasználók kedvence lesz.

És merre tartok én?

Egészen idáig tanácsadói munkám legnagyobb része a Perlhez kötődött, ezt még mindig egy hatékony webes programnyelvnek tartom. Aki kérdezi, annak eddig azt mondtam, hogy nyolcvan százalékban Perllel, húsz százalékban pedig más programnyelvekkel (Java, Tcl, Python, C) foglalkoztam.
Az utóbbi időkben a webes programozói környezetek területén bekövetkezett jelentős változások hatására (ezt csak fokozta a szakma elmozdulása az alkalmazáskiszolgálók irányába) csapatommal együtt úgy gondoltuk, hogy nálunk is irányváltásra van szükség. Sok esetben még mindig a Perlt választanám, főleg akkor, ha a mod_perlt és a HTML::Masont is használhatom. Munkáinkhoz azonban egyre növekvő arányban alkalmazzuk a Java servleteket és JSP oldalakat, különösen a Tomcat servlet/JSP motorral és a PostgreSQL adatbázissal. A mod_perlről gyűjtött tapasztalatunk az AxKit felé terel bennünket, a servletek pedig egyre jobban ösztönöznek az Enhydra felfedezésére.
Az ACS-t már a korai időktől kezdve használjuk bizonyos nagyobb terjedelmű munkákhoz, javarészt a csomaghoz járó, már működő alkalmazások igen nagy száma következtében. Emellett az a tény, hogy az ACS ingyenes és Linux alatt is használható, nagyon egyszerűvé teszi a vele végzett munkát, hiszen az önkéntes fejlesztőközösség képzettségéhez és lelkesedéséhez nem férhet kétség, valamint a szolgáltatások, a leírás, az ellenőrzés és a hibajavítások miatt sem kell aggódnunk.
Összefoglalva: már most is számos módszer közül választhatunk, ezek némelyike csak az utóbbi időkben fejlődött ki. A rovat ötvenedik cikkének befejezéséhez közeledve és a jövőbe tekintve a webes fejlesztők számára hatalmas lehetőségeket látok. Különösen azok lesznek sikeresek, akik hisznek az ingyenes programok erejében és Linuxot használnak. Az eljövendő néhány év bizonyára rendkívül izgalmas lesz, és remélem, hogy néhány hónapon belül olvasóimmal is megoszthatom tapasztalataimat, és a kipróbált alkalmazásokkal használható példaprogramokat is.

További érdekességek

Az Apache Software Foundation, mely az Apache webkiszolgáló és a Jakarta-Tomcat, a Jakarta-Oro (regexp), valamint a Cocoon (Java-XML) projekteket támogatja:
➲http://www.apache.org/

A mod_perl honlapja
➲http://perl.apache.org/.
Itt mindenképpen olvassuk el *Stas Bekman* ismertetését a modulról, ez a programkörnyezetről összegyűlt adatokat is tartalmazza.

Az Oracle IAS
➲http://www.oracle.com/

Az AOLServer
➲http://www.aolserver.com/

Az AOLServert Python-értelmezővel bővítő PyWX
➲http://pywx.idyll.org/

Template Toolkit
➲http://www.template-toolkit.org/

HTML::Mason
➲http://www.masonhq.com/

Enhydra
➲http://www.enhydra.org/

Zope
➲http://www.zope.org/

AxKit
➲http://www.axkit.org/

ArsDigita
➲http://www.arsdigita.com/

Az OpenACS projekt
➲http://www.openacs.org/

Reuven M. Lerner (reuven@lerner.co.il) cége internetes tanácsadást vállal, székhelyük Modi'in-ben, Izraelben van. A *Core Perl* című könyv szerzője, mely a Prentice Hall kiadónál jelenik meg. Szeretettel vár mindenkit az ATF honlapján ➲http://www.lerner.co.il/atf/.