tech.net

working with windows

Módszertanok kora

IV. / 05. szám
2003. május
1344 Ft

ISSN 15865185 05
9 771586 518005

Szerkesztőség:
Főszerkesztő: **Fóti Marcell**
marcellf@netacademia.net
Főszerkesztő-helyettes: **Fülöp Miklós**
mick@netacademia.net
A szerkesztőség címe:
1062 Budapest, Andrássy út 62.
Telefon: 472-1214
technet@netacademia.net
Nyilvános levelezési lista:
tech.net@technetklub.hu

Kiadja és terjeszti a
NetAcademia Kft.
Terjesztési, előfizetési információ:
Telefon: 472-1214
terjesztes@netacademia.net
Megjelenik havonta, ára 1.344 Ft

Előfizethető megrendelőlevélben a szerkesztőségnél:
1062 Budapest, Andrássy út 62.
Fax: 472-1215
http://technet.netacademia.net/subs

Hirdetésfelvétel: **Szívós Éva**
Telefon: 472-1214
Fax: 472-1215
info@netacademia.net

Nyomdai előkészítés:
Ars Luna Bt.
Vezető: Dobák Ildikó
Címlapgrafika: **Fülöp József**

Nyomda:
AduPrint Kiadó és Nyomda Kft.
1061 Budapest,
Paulay Ede utca 55.
Felelős vezető: **Tóth Béláné**

ISSN 1586-5185

Módszertanok kora

Tervezési minták a fejlesztésben

Sokan úgy gondolják, hogy az elemi programozási ismeretek *(ciklus, változó stb.)* birtokában a szoftverkészítés nem ördöngösség. Csak leírjuk, amit akarunk, és működik. Ha mégsem, hát teszünk bele néhány if-et a kivételes esetek lekezelésére. Ezt a fejlesztési „módszertant" buherálásnak hívják, ami garantáltan hibás programokat eredményez. Kell ez nekünk?

Kellhet. Jó zsíros karbantartási szerződéseket lehet kötni a buherált programokra. De gondoljuk végig egy pillanatra a másik utat: mi lenne, ha szoftverfejlesztéseink során nem egyik zsákutcából a másikba, egyik if-ből a másikba támolyognánk? Ha nem írnánk meg olyan kódot, amit már mások megírtak? Ha felhasználnánk jeles elődeink eredményeit?
Egy éve egy utazás kapcsán találkoztam két autótervező mérnökkel. A fiatalok egy vidéki cégnél dolgoztak, német megrendelésre. A beszélgetés fonalát régi rögeszmémmel indítottam, nevezetesen hogy a szoftverek milyen vackok, bezzeg az autógyártás! S a minőség, a precizitás csúcsa, a német autó. Hogy meg van tervezve, mekkora csoda egy-egy autó! S milyen felemelő érzés lehet alkotó módon hozzájárulni egy remekműhöz!
Hihetetlen választ kaptam! Még hogy az autók jól meg lennének tervezve? Mit gondolok én, miért nem próbálható ki a szakmai seregszemléken a prototípus? Mert a négyhengeres motorból egy fél hengernyit le kellett vágni flexszel, hogy beférjen a karosszériába!
Eh, mondom, az szörnyű vakvéletlen. Erre sorolják, hogy melyik autóban fut derékszögben megtörve az ékszíj, mert teletervezték alkatrésszel a futási útvonalát, így kerülőúton kell haladnia. Egy kocsit teljesen újra kellett tervezni, mert – szintén tervezési baki miatt – az olajszint pálcának nem maradt helye a kihúzásra: odakerült egy másik alkatrész. Mire az olajpálca végre kiszabadult, az egész elrendezés felborult. Vagy gondoljunk a visszahívott kocsikra, melyek egy piciri alkatrész hibás tervezése folytán borulékonnyá váltak. Vagy arra az olasz autóra, amiben csak a motor kiszerelése árán lehet izzót cserélni satöbbi. S hogy mindez miért? Mert a tervezést alávetik mind a dögös dizájnnak, mind az időnek.
Először a karosszéria készül el, művész urak tervezik. Ebből előáll egy drótváz. A lényeg *(motor, akku stb.)* tervezőinek az a feladata, hogy a látványtervezők által megálmodott drótvázon belül maradjanak, mert ha nem, akkor zaporozsec születik *(kívül lesz a térképtartó)*. A hely és az idő iszonyú kevés, ezért egyszerre mintegy négyszázan esnek neki a motortér elfoglalásának, s aki fürgébb, szabadabban tervezi be a saját alkatrészét a térbe. Aki lassabban ocsúdik, sarokba szorul: teheti az alkatrészét a motorblokk alá *(ekkor beszélünk újratervezett modellről :-)*. Bukta akkor van, ha a lefoglalt területeket nem publikálják a közös adatbázisban, magyarul nem csekkelnek in. Minden tervező kötelessége, hogy a munkanap végén becsekkolja a közösbe, mit alkotott, milyen erőforrásokat pusztított. Ha elmulasztja, lehet, hogy később már nem tudja betenni a közösbe, mert nem fér.
Ennyiből is látható, hogy az autógyártás kísértetiesen hasonlít a szoftvergyártásra. Modulok, check in a nap végén, s másnap minden kezdődik elölről. Egy igen lényeges különbséget azonban rögön megemlítenék: míg általában az autók tervezését hozzáértő mérnökök végzik, addig sok szoftvert nem tervez meg senki. Csak írjuk, írjuk, írjuk. A következő különbség az alkatrészek felhasználásában mutatkozik: egy-egy autógyár a költségek csökkentése érdekében bizonyos alkatrészeket olyan jól megtervez, hogy változtatás nélkül felhasználhatók a különböző modellekben. És ne csak iciri-piciri alátétekre gondoljunk: egy-egy jól tervezett alváz elcipel 3-4 modellt! Évekig nem nyúlnak hozzá. Minek, hisz bizonyítottan jó!

OOP

Vessük ezt össze az objektumorientált programozással: elvileg adott volna az objektumok újrahasznosításának lehetősége, ha azt az átkozott objektumot megtervezték volna. De mivel ez nem történt meg, minden egyes felhasználáskor meg kell a kódot fűszerezni egy kevés if-fel. *(És máris kész a 4.2.45.132 verzió, valamint a DLL Hell.)* Mi lenne, ha az alvázakat is csak if-ek árán lehetne újrahasznosítani *(még 1-2 furat ide, egy fülecskét hegesztünk oda...)*?

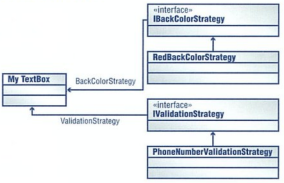
...folytatás a 4. oldalon!

2003. 05. szám

Design Patterns bevezetés

Szoftvertervezési kérdésekről beszélgetve előbb-utóbb beleütközünk a Design Patterns-ekbe. A követelmények feltérképezése után egy nagy szakadék tátong az emberi nyelven leírt követelmények és az azt megvalósító kód között. A szakadékot a tervezési folyamatnak kell*(ene)* áthidalnia. Az építészek vagy a gépészek évekig tanulnak rendszereket tervezni. Nem kellene ennek így lenni az informatikában is? Elő hát a Tervezési mintákkal, elő a Design Patternekkel!

5. oldal



Microsoft Solutions Framework

Avagy ki látott már sikeres, határidőre befejezett, költségkereten belül maradó szoftverfejlesztési projectet?

Ha egy fejlesztőcsapat fejébe veszi, hogy következő projektjét már nem bukja el, érdemes megismerkednie az MSF keretrendszerrel. Ez ugyanis nem mást tartalmaz, mint a világ legnagyobb szoftvergyárában korábban előfordult bukásokból levont tanulságokat, és a „kihívások" elkerülésének lehetőségeit.

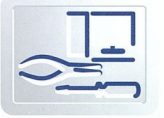
9. oldal

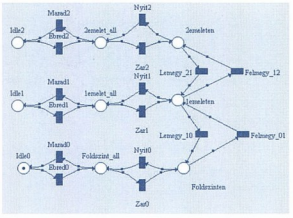
Portál Para[digma]

III. rész – the X files

Sorozatunk első két részében sokat emlegettük a különféle x-szel kezdődő fájlokat, elsősorban az XML-t és az XSLT-t. Arról azonban kevés szó esett, hogy pontosan milyen szerepet is kapnak az X-fájlok egy portál életében, illetve hogyan kell őket alkalmazni egy többrétegű korszerű portálmegoldás fejlesztése közben. Az igazság ezúttal ideát van.

12. oldal





Petri-hálók modellezése és analízise

A korábbiakban bemutattam a Petri-hálók működését. Most egy kicsit gyakorlatiasabb oldaláról igyekszem megközelíteni a dolgot, így egy olyan eszközt mutatok be, amely modellezési- és különböző analíziscélokra egyaránt alkalmas.

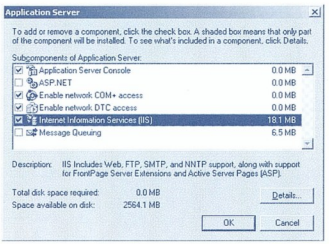
17. oldal

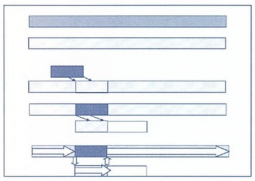
Internet Information Services 6.0

II. rész: A Windows 2003 Server webkiszolgálójának újdonságai

Az előző részben megismertük az IIS 6.0 webkiszolgálójának új belső felépítését. Most tovább ismerkedünk a webkiszolgálóval és terítékre kerül az FTP szolgáltatás is.

22. oldal





BrightStor ARCserve v9 for Windows

Foglalt állományok és alkalmazásszerverek mentése

Az adatmentés teljességének feltétele a mentendő adatok állományszintű változatlansága. Ez a feltétel feloldandó érdekütközéshez vezet a gazdaalkalmazás és mentőmegoldás között.

26. oldal

A DHCP rejtett szépségei

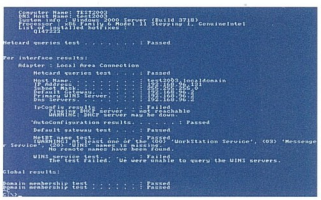
1. rész

Szinte minden TCP/IP hálózat rendelkezik DHCP szolgáltatással. Ez az alkalmazás háttérbe húzódik, a felhasználók sohasem találkoznak vele, és ha jól működik, a rendszergazdák is csak hébe-hóba vetnek rá pillantást. Itt is érvényes az ókori bölcsesség: ami nem látható, az fontosabb, mint ami látható.

30. oldal

Parancssori környezet és eszközök

Halál a GUI-ra

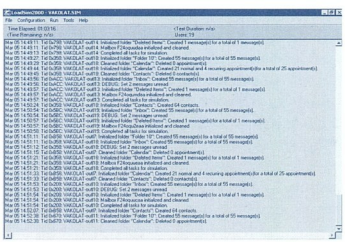


Sok UNIX-os szakember nézte le a Windows kiszolgálókat, mert csak grafikus környezetből lehetett üzemeltetni őket. Megpróbálom bebizonyítani, hogy az új generációs Windows kiszolgálókat immáron kizárólag parancssorból, manuálisan *(és automatizáltan is)* lehet mindenre kiterjedően adminisztrálni, konfigurálni és felügyelni. A cikkben – ismétlésképpen – némi DOS-ismeretet is megemlítek azon fiatalok kedvéért, akik akkor még nem éltek…

33. oldal

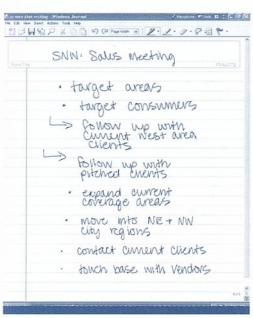
Communicare necesse est!

Használjuk erre a legmegfelelőbb eszközt!



A Microsoft Exchange a maga 110 millió felhasználójával magasan napjaink vezető üzenetkezelő és csoportmunka platformja. És hamarosan itt az Exchange 2003 *(Titanium)* az új verzió! Bátorfi Zsolt bevezető cikke után vizsgáljuk meg közelebbről, hogy mit rejtenek a csillogó korongok. Áttérjünk? Várjunk? Jobb lesz? Próbáljuk ki! Nézzük meg a gyakorlatban, hogy mi lakik a hangzatos név mögött!

37. oldal



A .Net és a Tablet PC

Miért fontos, hogy a Tablet PC a .Net keretrendszerre épül és a felhasználó számára milyen haszonnal jár a beépített .NET infrastruktúra? Hogyan illeszkedik a táblás modell a .NET stratégiába? A Microsoft PressPass kérdéseire Vic Gundotra, a Platform Strategy igazgatója és Alex Gounares, a Tablet PC vezető tervezője válaszolt.

43. oldal

...folytatás az 1. oldalról

És vajon mi lenne, ha az objektumorientált programozást arra *(is)* használnánk, amire való? Ha jól megtervezett, módosítás nélkül újrahasznosítható objektumok kerülnének ki az objektumgyárakból? Megmondom: stabilabb kód, kevesebb biztonsági rés, kevesebb munka. Akkor miért nem így fejlesztünk? Mert senki sem tudja, mi fán teremnek a tervezési minták. *(Egy éve még én sem tudtam...)*

Design Patterns

A tervezési minták, vagy más néven sablonok arra szolgálnak, hogy ha lehet választani, inkább ezerszer felhasznált, rongyosra tesztelt megoldásokat keressünk ahelyett, hogy nekiállnánk a spanyol viasz feltalálásának. Autós példával élve: ha belülről állítható visszapillantó tükörre van szükség a tervezés folyamán, akkor vegyünk le a polcról egy bevált tervet ahelyett, hogy elkezdenénk gondolkozni azon, hogyan lesz a kvarchomokból üveg, az üvegből tükör stb. A pattern szó mintát jelent, melyet összekapcsolhatunk a mintafelismeréssel. Hiába vannak ugyanis kiváló sablonjaink, ha adott helyzetben nem ismerjük fel, hogy melyik illene a problémára.

Az objektumorientált programozást elvileg abból a célból *(is)* alkották nekünk, hogy kész objektumokból építkezhessünk. A napi valóság azonban azt mutatja, hogy szinte nincs olyan objektum, melyet érintetlenül újrafelhasználnák készítőik. Egy kevés IF mindig kell bele! Különösen sok IF megy bele, ha olyanra sikeredett, amely csak egy adott helyzetben használható. A világ változik, az objektumot *(tükröt)* legközelebb mikrobuszra kell felszerelni. Ennek a követelménynek csak a jól megtervezett alkatrészek, objektumok felelnek meg változtatás nélkül.

A spanyol viasz

Az egyik leggyakoribb jelenség, amikor a programozó nem ismeri fel, hogy olyan problémán küzd, amit előtte ezermillióan megoldottak már. Ki jól, ki rosszul, de az ezermillió közt legalább száz jó megoldás van. Hősünk azonban fabrikál egy ezermillió-egyedik megoldást, amely alig hibás, és csak négy teljes napja ment rá. Ilyen az a tipikus probléma, hogy egy alkalmazásból / objektumból / végrehajtási szálból / tcp csatornából / akármiből egyszerre, egyidejűleg csak egyetlenegy legyen a memóriában / fusson / legyen nyitva / akármi. Ez a feladat oly tipikus, hogy külön tervezési sablonja van: ez a singleton.

A Singleton

Annak biztosítása, hogy csak egyetlen bal külső visszapillantónk legyen, viszonylag egyszerű: a legelső felszerelése után elfogynak a furatok, nincs hova szerelni a másodikat. Azonban a Windowson nincsenek furatok, ha egyetlenegy EXCHSRVR.EXE futtatása a kívánatos, ezt magának az adott objektumnak kell elintéznie: mintha a visszapillantó tükör maga szabályozná, hogy belőle csak egy lehet. Bizonyos programozási nyelvek és keretrendszerek nem sok lehetőséggel szolgálnak a probléma megoldására, barkácsolásra kényszerülünk: jönnek az IF-ek és a globális változók. A C# nyelv viszont lehetővé teszi, hogy az objektum definíciója, osztálya rendelkezhessen az objektum létrehozásáról, azaz a visszapillantó képes megszámolni saját példányait. Az alábbi kódrészlet ezt a lehetőséget szemlélteti:

```
class Singleton
{
  private static Singleton singleton = null;
  //az osztály tartalmaz egy saját típusú változót
  public static Singleton Instance()
  //minden új objektum születése előtt hívódik meg
  {
    if (null == singleton)
    //ha nincs belőle példány, legyen!
    {
        singleton = new Singleton();
    }
    return singleton;
    //nesze, hívó! lehet, hogy nem is újat kapsz!
  }

private Singleton()
  {
  }
}
```

Nincs globális számláló, mi több, a hívó alkalmazásnak nem is kell tudnia, hogy az objektum singletonként tengeti életét. Mi csak meghívjuk, és ő gondoskodik saját kényelméről! A lényeg hat sorban elfért, a többi zárójel, soremelés stb. a könnyebb olvashatóság végett. Ezek után nem kell más, mint hogy a programozó mindig felismerje, hogy egy singleton problémával áll szemben, és levegye a polcról ezt a sablont. Nem bírom abbahagyni e minta dicséretét: borzasztó egyszerűen átalakítható tetszőleges számú példány engedélyezésére. *(A Network Load Balancing is egy singleton: egy ponton férünk hozzá, és ő eldönti, hogy hány példányt enged, és mi melyikkel kommunikálunk!).* Ha egy alkalmazás azt kérdi, hogy hány ezmegazt *(kockát, végrehajtási szálat, párhuzamos szólamot, akármit)* szeretnénk, akkor kibővített singletonnal állunk szemben.

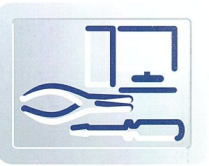
További csinos tervezési minták is léteznek, ezekről szól a következő cikk. Végül hadd hívjam fel a figyelmet egy másik módszertanra, a Microsoft Solutions Frameworkre is, melyről a kilencedik oldalon kezdődő cikk közöl részleteket.

Fóti Marcell
marcellf@netacademia.net
A szerző a NetAcademia vezető oktatója
MCSE, MCT, MCDBA, MZ/X

Kapcsolódó NetAcademia tanfolyamok

Design Patterns Workshop
Microsoft Solutions Framework

Design Patterns bevezetés



Szoftvertervezési kérdésekről beszélgetve előbb-utóbb beleütközünk a Design Patterns-ekbe. A követelmények feltérképezése után egy nagy szakadék tátong az emberi nyelven leírt követelmények és az azt megvalósító kód között. A szakadékot a tervezési folyamatnak kell*(ene)* áthidalnia. Az építészek vagy a gépészek évekig tanulnak rendszereket tervezni. Nem kellene ennek így lenni az informatikában is? Elő hát a Tervezési mintákkal, elő a Design Patternekkel!

Történelem

A Patternek fogalma Christopher Alexander építészhez kapcsolható. Építészhez? Bizony, bizony. Ha belegondolunk, az építészet pont egy olyan szakma, ahol tervezni kell, apró alkotóelemekből kell összerakni egy lehetőleg szervesen egyként viselkedő alkotást, például az épületet. Mitől otthonos egy lakás, és mitől idegen egy másik? Sándor elgondolkodott ezen, és rájött, hogy a profik ösztönükre hagyatkozva valamint a múltbéli tapasztalataik alapján jó terveket készítenek. Például minden terasz esetén megfogható 10-20 olyan alapvető tulajdonság, amely ha szerepel a terasz tervében, az hangulatos, barátságos lesz. Misztérium? Mindjárt kiderül, nem.

Felfigyelt arra, hogy bizonyos problémák újra és újra felbukkannak a tervezőmunka során, és ezekre mindig más, de jellegében mindig nagyon hasonló megoldást kellett adni.

A folyamatot a probléma, környezet, megoldás hármassal jellemezte, ahol tulajdonképpen a környezeti feltételek és maga a probléma ismeretében leírt egy általánosan használható megoldást. Negyedszázada jelent meg könyve „A Pattern Language: Towns, Buildings, Construction" címmel. Ebben 250 tervet írt le, amelynek segítségével még a tapasztalatlanabb tervezők is sokkal gyorsabban tudtak épületeket tervezni, ráadásul – és ez nagyon fontos – az elkészült tervek magukon hordozták a mintákban leírt tudást, így azok nagyobb valószínűséggel lettek szebbek, jobbak vagy használhatóbbak.

Chris könyve elgondolkodtatta a szoftvertervezőket is, hisz a programtervezés folyama hasonló problémákat vet fel, mint az építészet. Kis részeket kell kidolgozni, majd abból összerakni egy teljes egészet. Először Smalltalk környezetben használták fel a patternek mentén vezetett gondolkodásmódot, amelyben dokumentumok, a megjelenítés és a felhasználói beavatkozás elválasztására megalkották a Model-Controller-View mintát. Aki ismeri az MFC-t, annak Document-View architektúra néven ismerős lehet ez.

Aztán a 90-es évek elején elkezdett szerveződni egy csoport, amely később a Gang of Four *(GoF)*, a négyek bandája nevet kapta. Tagjai: Erich Gamma, Richard Helm, Ralph Johnson és John Vlissides. 1995-re elkészült közös könyvük, melynek címe: Design Patterns, Elements of Reusable Object-Oriented Software *(Addison-Wesley, 1995)*. Általában e könyv megjelenését tekintik a szoftver DP-ek kiindulási idejének. A továbbiakban egyszerűen a GoF néven fogok rájuk hivatkozni, könyvükre pedig a DP könyvként.

Mi az a Design Pattern?

A legáltalánosabb definíció:

```
Egy-egy ismétlődő szoftvertervezési probléma
megnevezett, újrafelhasználható megoldása egy
bizonyos környezet és feltételek mellett.
```

A szoftvertervezés egyik legnagyobb problémája, hogy nem algoritmizálható, nagyon intuitív folyamat. Egy-egy problémára általában végtelen sok megoldás adható, csak éppen az egyik egyszerű lesz, a másik meg bonyolult. Az egyik hatékony, a másik csapnivalóan lassú. Az egyik könnyen módosítható, a másik egy nagy, összefüggő gubanc.

Egy tapasztalt szoftvertervező már nem talál ki minden egyes problémára új megoldásokat, hanem vannak a fejében olyan megoldás-sablonok, amelyeket korábbi tervezési feladatokból szűrt le. A kezdő fejében – bármilyen okos is – még nincsenek azonnal alkalmazható megoldások, így neki is végig kell járni azt az utat, amit a tapasztalt kollégája már bejárt.

Kivéve, ha ... valaki leírja, mi jár a fejében. Ha valaki precízen dokumentálja, hogy adott környezeti feltételek mellet találkozott egy ilyen és ilyen tervezési problémával, és a sok megoldás közül ez bizonyult a legjobbnak. Éles környezetben többször kipróbálta különböző feladatokra, és sikeresen tudta alkalmazni rájuk. Tulajdonképpen ezt tette meg a GoF, ezekből születtek meg a könyvben leírt DP-k.

Gyakorlati projektekben kielemezték nagyobb architektúrák részmegoldásait, és megfogalmazták, hogy adott típusú problémára milyen megoldást lehet adni. A megoldások általában nem triviálisak. Pont az a jó a DP-kben, hogy azáltal, hogy az első ötletnél valamivel bonyolultabb megoldást adnak egy problémára, a kapott eredmény jól bővíthető, könnyen módosítható, azaz könnyebben képes alkalmazkodni a megváltozott üzleti igényekhez. Aki már dolgozott valamilyen tényleges megrendelésen, tudja, hogy ez milyen fontos tényező egy munka sikere szempontjából. Alapvetően, életbevágóan fontos.

Milyen előnyökkel járnak a DP-k?

A DP-k számtalan ponton segíthetnek a tervezésben. Nézzük át a legfontosabbakat!

- Lerövidíthetik a tervező betanulási idejét. A DP-ket olyan emberek írták, akik tényleg értenek a tervezéshez. Nem kizárt, hogy valaki tud frappánsabb vagy valamilyen szempontból jobb megoldást szülni egy problémára mint a GoF, de általában érdemes megérteni, hogyan gondolkodtak a szerzők, ha más miatt nem, hát azért, mert sokat lehet belőle tanulni.
- Lerövidíthetik a tervezés idejét. Ha valakinek már a fejében van néhány minta, és esetleg sikeresen alkalmazta is őket valamilyen feladat megoldásában, előbb-utóbb azon kapja magát, egy tervezés során beugrik neki, hogy most, adott viszonyok között ide egy mondjuk Visitor patternre volna szükség. Az

így születő megoldások lehet, hogy valamivel összetettebbek lesznek, mintha saját kútfőből szültük volna meg őket *(képzavar – a szerk.)*, de cserébe szinte biztos, hogy sokkal jobban bővíthető és módosítható lesz a kapott architektúra.

- Szótárat ad a programozók kezébe. Amikor azt mondom autó, nem kell körbemagyaráznom, hogy egy olyan dobozról van szó, aminek négy kereke van, emberek közlekednek vele, hanem mindenki tudja mire gondolok. Ez az absztrakció lényege. Az absztrakció a gondolkodás egy újabb szintje, ennek köszönhető, hogy *(viszonylag)* hatékonyan tudunk egymással kommunikálni.
  Ha azt mondom olló, mindenki tudja mire gondolok. Ha azt mondom Observer DP, már sokkal kevesebben. Miért, mi a különbség a kettő között? Gyakorlatilag csak az, hogy az egyiket már ismerjük, mert annyiszor láttuk és megtapasztaltuk, a másikat pedig még nem. Amint valaki megismer egy fogalmat vagy tárgyat, azonnal él vele a kommunikációban, mert így sokkal tömörebben és hatékonyabban tudja kifejezni magát.
  Ezek után ha két DP-ket ismerő tervező beszélget egymással, nem kell nekik kilométeres mondatokkal mindent körbemagyarázni, ha van egy egységes szótár, ami alapján mindketten tudják miről van szó. Emiatt nagyon fontos, hogy minden egyes DP-nek van egy könnyen megjegyezhető neve, így az beépülhet a tervezők szótárába, megkönnyítve az írásos és a szóbeli kommunikációt.
- A legtöbb DP leghosszabb része a Következmények *(Consequences)* rész. Ebben nagyon alaposan kielemzik a DP által a kiinduló problémára felvázolt megoldás előnyeit-hátrányait, alkalmazhatóságát. E részekből rengeteg tervezési meggondolást lehet tanulni, megismerni, hogyan gondolkodnak a profik, amit aztán mi is beépíthetünk a saját gondolkodásmódunkba. Azaz a DP-ket tanulmányozva olyan tervezési megfontolásokra tehetünk szert, amelyeket talán évek alatt sem tudnánk magunktól összeszedni.

Vannak más patternek is, mint a GoF DP-k?

A szakmában számtalan egyéb pattern is létezik a GoF-os DP-k mellett, bár tény, hogy a GoF könyv a legismertebb igehirdetője a témának. Ha pontosak akarunk lenni, akkor a GoF által leírt minták OOP tervezési minták. De emellett még van nagyon sokféle pattern, hisz a szoftvertervezés-írás nem csak OOP alapon történhet, valamint különböző szinteken is lehet megoldásokat keresni.
A legnagyobb léptékű patterneket általában Architecture Patterneknek hívják. Ezek több alrendszerből álló nagy építőkockákat mutatnak be.
Közepes bonyolultságúak a jelen cikk és a GoF könyv fő témakörét adó Design Patternek, amelyek egy-egy részfeladatra, rész-architektúrára adnak általános, nyelvfüggetlen megoldást, egymással kommunikáló komponensek képében.
A legegyszerűbb minták az idiomok vagy más néven Programming Patternek. Ezek egy objektum belsejének részleteivel vagy egy nyelv elemeinek használatával foglalkoznak. Például az egy C++-os idiom, hogy a konstruktorban lefoglalunk erőforrásokat, a metódusok használják őket, majd a destruktor felszabadítja őket. Vagy az is egy gyakori idiom, hogy egy metódus visszatérési értékét eltároljuk egy változóba, de rögtön össze is hasonlítjuk egy értékkel:

```
if ((ido = GetTime()) > 60) ...
```

Patterneket vethetünk be a tervezést megelőző fázisban, az analízisben is. Martin Fowler például írt egy könyvet Analysis Patterns címmel. Ebben egészségügyi és pénzügyi programok írásához kapunk 70 analízis- és tervezési mintát. Ez egy specializált könyv, viszont pont emiatt az adott területekre *(OOA)* konkrétabb megoldásokkal tud szolgálni.
Érdemes még tudni egy másik négyes által írt könyvcsaládról: ezek a Pattern-Oriented Software Architecture vagy POSA könyvek. A második rész címe: Patterns for Concurrent and Networked Objects. Ebben elosztott, hálózati alkalmazások fejlesztéséhez kapunk 17 DP-t.
Azaz a GoF-ban leírtakon kívül létezik még jó pár specializált pattern leírás, amely egy-egy területre koncentrál. Amint a programozás egyre inkább átmegy művészetből tudományba, várhatóan egyre inkább nő a patternek jelentősége is. Jelen pillanatban ugyanis ez tűnik az egyik leghatékonyabb tudás-átviteli formának.

Mi a DP-k lényege?

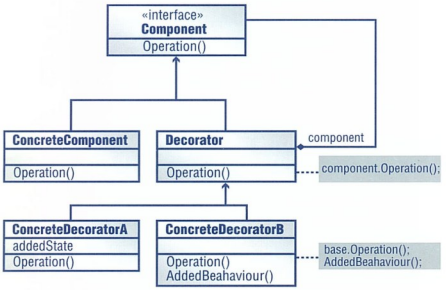
A DP-k három alapvető technikát favorizálnak:

- Részesítsd előnyben a delegálást a leszármaztatással szemben;
- Programozz interfészekkel szemben;
- Rejtsd el, ami változhat.

Miért nem ajánlják a leszármaztatást? Amikor valaki elkezd OOP-t tanulni, általában nagyon megtetszik neki az, hogy egyik osztályból egy másikat leszármaztatva annak minden implementációja *(kód és adattagok)* rendelkezésre áll a leszármazott osztályokban is.
Ez nagyon jó, azonban az öröklés statikus kapcsolat két osztály között, amely fordítási időben értékelődik ki. Ha egyszer a Kutya osztály az Állat osztályból származik, az összes viselkedése, tudása *(metódusok)* amit az állattól kap, fordításkor eldől, futásidőben már nem lehet rajta változtatni.
Emellett a leszármaztatás merev is. Gondoljuk arra, hogy leszármaztatok egy vezérlőt a TextBox osztályból. Ebben specializálom a TextBoxot, hogy mondjuk csak telefonszámokat fogadjon el.
Ezek után szükség van egy olyan TextBoxra, aminek a háttere piros. Leszármaztatom a TextBox osztályból, és mondjuk a konstruktorban pirosra állítom a háttérszínt. Ezek után szükség van egy piros hátterű, telefonszámokat elfogadó Textboxra. Még többszörös leszármaztatást támogató rendszerben sem tudom egyszerűen leszármaztatni a piros és a telefonszámos TextBoxból az új TexBoxot, hisz az így valójában kétféle TextBoxot is tartalmazna. Nincs mese, le kell származtatnom újra az alap TextBoxból, és újra kell implementálnom a két funkciót. Programozásban az egyik fő ellenségünk az ismétlődő kód. Itt triviálisan kopi-paszta öröklést használtunk *(kódot, implementációt másoltunk)*.
Mi az egész problémakör oka? Az, hogy az öröklés túl mereven bedrótozza a leszármazott osztály lehetőségeit. Nem tudjuk könnyedén kombinálni a szükséges szolgáltatásokat, így n féle lehetőség összes kombinációjának kihasználásához 2n féle osztályt kellene implementálni. Nem túl kecsegtető terv.
Mit javasol erre a problémára a GoF könyv? Emlékezzünk csak az első alapelvre:

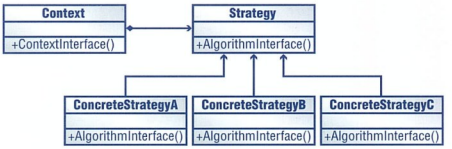
- részesítsd előnyben a delegálást a leszármaztatással szemben.

Delegálás = ne az adott osztályban valósítsunk meg egy működést, hanem egy másik osztályban, és delegáljuk, dobjuk át a metódushívásokat.
Keressünk egy olyan DP-t, ami passzol a problémánkra. Minden DP-nek van egy Intent – Mire való? fejezete. Ebben a következőt találjuk a Decorator patternnél:
Cél – „Dinamikusan *(futásidőben)* járulékos felelősségek hozzáadása egy objektumhoz. A dekorátorok rugalmasabb funkcióbővítési alternatívát biztosítanak a leszármaztatással szemben.„
Ez nagyon úgy néz ki, hogy passzol a problémánkra. A responsibility (felelősség) talán furcsa szó, de OOP terminológiában ez egy objektumtól elvárható szolgáltatásokra utal, azaz milyen működést, funkciókat várhatunk el tőle.
A pattern struktúrája a következő:

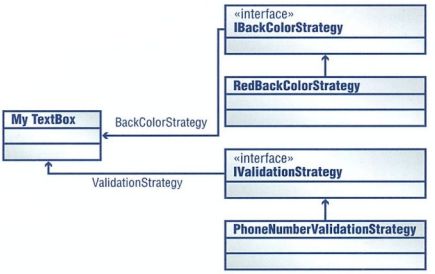


És itt kerülünk bajba. A pattern használatához kellene egy alapinterfész *(Component)*, amelyben minden egyes TextBox metódus össze van szedve. Ez nem feltétlenül egy interfész, hanem lehet absztrakt alaposztály is, vagy egy olyan konkrét osztály, amelyben minden metódus virtuális, azaz az összes metódusa polimorf módon újraimplementálható a leszármazott osztályokban. Esetünkben azonban van egy konkrét TextBox osztály, amelynek van egy pár tucat metódusa. A pattern használatához mindegyiket tovább kellene delegálni a Decoratoroknak, ami esetünkben nagyon nagy munka lenne, illetve a nem virtuális metódusok miatt egy részével nem tudnánk mit kezdeni. A pattern Implementation fejezetében is írják, hogy a Component alaposztály legyen egyszerű, kevés metódust tartalmazó. Esetünkben ez nem igaz.
Így vérzett el a dekorátor pattern. A példa mutatja, hogy a patternek bevetésekor nagyon oda kell figyelni, mert egy, a problémára látszólag kiválóan alkalmazható pattern lehet, hogy valami apróság miatt mégsem használható. Szerencsére a Decorator pattern leírása során felhívják a figyelmünket arra, hogy abban az esetben, ha a Component alaposztály *(TextBox)* túl összetett és a Decorator osztályok túl bonyolultak lennének, gondolkodjunk el a Strategy pattern használatán.
A Strategy célja:
„Definiálni egy algoritmuscsaládot, egységbezárva és cserélhetővé téve őket. A Stategy lehetővé teszi, hogy az algoritmus anélkül változzon, hogy az ügyfél *(hívó)* észrevenné ezt."
Esetünkben két algoritmusról van szó. Az egyik megmondja, milyen színű legyen a háttérszín, a másik pedig leellenőrzi, hogy a begépelt karakterfüzér telefonszám-e? Azaz kétszer is be kell vetnünk a Strategy patternt.

A pattern statikus UML osztálydiagramja a következő:



Esetünkben a Context, aki használja a Strategy interfészen definiált algoritmusokat a saját „okos" TextBoxunk. Látható, hogy a Context aggregálja a Strategy interfészt, azaz a TextBoxunknak lesz egy-egy referenciája a háttérszínállító és a validáló stratégiára is. UML-ben így néz ki az osztálydiagramunk:



A megfelelő algoritmust tároló konkrét stratégiaimplementációt a MyTextBox konstruktorban adjuk át a textboxnak. Nézzük meg az UML diagramon látható osztályok konkrét implementációját:

```
public class MyTextBox: TextBox {
  IBackColorStrategy bs;
  IValidationStrategy vs;

  public MyTextBox(IBackColorStrategy bs,
 IValidationStrategy vs) {
    this.bs = bs;
    this.vs = vs;

    if (bs != null) {
      //Felhasználjuk az IBackColorStrategy-t.
      BackColor = bs.GetBackColor();
    }
    Validating += new
  CancelEventHandler(MyTextBox_Validating);
  }

  private void MyTextBox_Validating(object source,
 CancelEventArgs e) {
    if (vs != null) {
      //Felhasználjuk az IValidationStrategy-t.
      if (!vs.IsValid(Text)) {
        e.Cancel = true;
        MessageBox.Show("A helyes formátum: " +
        vs.GetValidFormat());
      }
    }
  }
}

public interface IBackColorStrategy {
  Color GetBackColor();
}

public class RedBackColorStrategy:
IBackColorStrategy {
  public Color GetBackColor() {
```

```
    return Color.Red;
  }
}
public interface IValidationStrategy {
  bool IsValid(string text);
  string GetValidFormat();
}

public class PhoneNumberValidationStrategy:
 IValidationStrategy {
  public bool IsValid(string text) {
    Regex re = new
    Regex(@"\d{2}-\d{2}-\d{3}-\d{4}");
    return re.Match(text).Success;
  }
  public string GetValidFormat() {
    return "dd-dd-ddd-dddd - Országhívó-
    Körzetszám-Tel1-Tel2";
  }
}
```

Hogyan hozhatunk létre különböző típusú TextBoxokat?

```
//egyelőre bedrótozott a statégiaobjektumok
//létrehozása
IBackColorStrategy bs = new
RedBackColorStrategy();
IValidationStrategy vs = new
PhoneNumberValidationStrategy();

int i = -20;
MyTextBox t = new MyTextBox(bs, vs);
t.Location = new System.Drawing.Point(20, i+=40);
t.Text = "Piros és validál";
Controls.Add(t);

t = new MyTextBox(null, vs);
t.Location = new System.Drawing.Point(20, i+=40);
t.Text = "Csak validál";
Controls.Add(t);

t = new MyTextBox(bs, null);
t.Location = new System.Drawing.Point(20, i+=40);
t.Text = "Csak piros";
Controls.Add(t);
```

Látható, hogy attól függően, hogy milyen stratégiaobjektumokat kap egy-egy MyTextBox példány, mindegyik másként viselkedik.
Mi történik, ha kiderül, hogy kellene egy irányítószámokat is ellenőrző TextBox *(ami persze lehet piros is)*? Semmi gond, az előbbi ellenőrző stratégiánk telesen alkalmas erre a célra is, csak létre kell hozni egy új stratégiaosztályt, ami az IValidationStrategy-t implementálja:

```
public class ZipValidationStrategy:
IValidationStrategy
{
  public bool IsValid(string text)
  {
    Regex re = new Regex(@"\d{4}");
    return re.Match(text).Success;
  }
  public string GetValidFormat()
  {
    return "dddd - Az irányítószám négy
    számjegye";
  }
}
```

A megoldás *(és általában a DP-k használatának)* talán legszebb része, hogy az új funkcionalitás használatához semmit nem kell átírni, csak új kódot hozzáadni! Az új stratégiaosztály készen van, már csak meg kell mondani egy új textboxunknak, hogy használja:

```
IValidationStrategy zvs = new
ZipValidationStrategy();
t = new MyTextBox(null, zvs);
t.Location = new System.Drawing.Point(20, i+=40);
t.Text = "Csak ZIP";
```

Ha megfigyeljük, a megoldás nagyon jól bővíthető, és magán viseli a DP-k mindhárom alapelvét:

- Részesítsd előnyben a delegálást a leszármaztatással szemben;
- Programozz interfészekkel szemben;
- Rejtsd el, ami változhat.

Mi változhat? A validálás vagy a háttérszín algoritmusa. Elrejtettük őket a MyTextBox elől, így azokat legószerűen bármikor kicserélhetjük, anélkül, hogy erről bármi tudomása lenne a TextBoxkánknak.

Összefoglalás

A példa alapján sejthető, hogy DP-k ismeretében sokkal jobban bővíthető és karbantartható kódokat írhatunk, mint tisztán saját kútfőből merítve. Akiket érdekelnek az objektumorientált analízis és tervezés részletei DP-k bevetésével, azoknak mindenképpen ajánljuk a 3 napos Design Patterns tanfolyamunkat, amelyben egy többnyelvű, többfelé adatbázison működő webalkalmazást építünk fel DP technikák felhasználásával, a cikkszerző vezetésével.

A cikkben szereplő URL-ek:
Letölthető példakód
http://technet.netacademia.net/download/dp

Soczó Zsolt MCSE, MCSD, MCDBA, MCAD, MCT
Zsolt.Soczo@netacademia.NET

Kapcsolódó tanfolyamaink:
DP – Design Patterns
http://www.netacademia.net/workshop/DP

Microsoft Solutions Framework

Avagy ki látott már sikeres, határidőre befejezett, költségkereten belül maradó szoftverfejlesztési projectet?

Ha egy fejlesztőcsapat fejébe veszi, hogy következő projektjét már nem bukja el, érdemes megismerkednie az MSF keretrendszerrel. Ez ugyanis nem mást tartalmaz, mint a világ legnagyobb szoftvergyárában korábban előfordult bukásokból levont tanulságokat, és a „kihívások" elkerülésének lehetőségeit.

Az elmúlt évezred talán legnagyobb szoftverproject-bukása a Visual C++ kifejlesztéséhez fűződik. A munka során ugyanis minden lehetséges és lehetetlen határon sikerült túlmennie a fejlesztőcsapatnak. Sem a határidők, sem a költségkeret nem állták ki a gyakorlat próbáját. A felelősök utólagos megkeresése és „kijelölése" szintén kudarcot vallott. Vajon miért?

Jim McCarthy, a fent nevezett Visual C++ Business Unit vezetője a projekt után 23 ökölszabályban foglalta össze a sikeres fejlesztési projekt alapelveit *("23 rules of thumb for shipping great software on time")*. Ebben rámutat, hogy annak ellenére az ügyfél és a managerek határozzák meg a projektek határidejét, hogy a fejlesztés pontos időigényéről a munka dandárját végző fejlesztőknek sincs halvány elképzelésük sem! Nem meglepő módon az így kitalált határidők SOHASEM lesznek betartva.

Az MSF-keretrendszerben éppen ezért javasoljuk a „bottom up scheduling" eljárást, amikor a programozók *(alsó szint!)* becslik meg az egyes részfeladatok határidejét, amit azután a managerek csak összesítenek. Ez az eljárás azonban azt feltételezi, hogy a határidők felállítása előtt már tudjuk, hogy mit is akarunk csinálni. És ha ez sem teljesül…?

Az informatikai projektek sikeressége a Standish Group kutatásai alapján

Kézenfekvő megoldásnak tűnik: kérdezzük meg az ügyfelet, vajon mit akar? Sajnos azonban az ügyfelek döntő többsége legfeljebb azt tudja, mi nem megy neki *(sokszor még azt sem)*. Tehát adva van egy fejlesztési project, amiben nem ismert célokat kell fix határidőre teljesíteni. Ha ehhez hozzávesszük, hogy az ügyfél minél több drága funkciót követel minél kevesebb pénzért *(és idő alatt)* mi pedig minél több pénzt akarunk keresni, minél kevesebb munkával, látható, hogy a bukás garantált!

Most nézzük meg, hogy egy ilyen, eleve bukásra ítélt fejlesztési projekt során milyen további lehetőségek adódnak saját munkánk megnehezítésére!

Kihívásokkal küszködik…

Igen gyakori erőforráspazarlás, hogy a fejlesztők a szépség jegyében módosítanak egy már letesztelt kódot *(például belekezdenek egy jó nagy tárolt eljárás optimalizálásába)*, ami az ügyfélnek egyáltalán nem is fontos: maximum havonta egyszer futtatja, és teljesen mindegy, hogy ilyenkor 2 perc vagy 10 másodperc alatt fut le – igazából az sem zavarná, ha 1 órába telne.

Ez teljesen fölösleges munka, és elveszi az időt a fontos, naponta 1000-szer futtatott kódágak alapos elemzésétől, bugokat visz a már letesztelt kódba.

Apropó bug: mi az a bug? Mit kell vele tenni? Gyakori, hogy a fejlesztők minden bugot ki akarnak javítani. Ezt tudományosan issue *(ügy)* menedzsmentnek hívjuk. Vegyünk egy konkrét példát: ha mondjuk egy HR szoftverben ellenőrzés híján a munkatárs életkora 134 vagy 1 millió év is lehet, azt bugnak tekintjük-e? Az ügyfélszoftver használatával a rendszer jól viselkedik, de ha valami őrült közvetlenül matat az adatbázisban, és hülyeséget ír be, akkor azt elfogadja, további számításokat végez vele *(mikor megy nyugdíjba egy egymillió éves munkatárs?)*. Az ilyen bug kijavítását a project végére kell hagyni, mert nem fontos a javítása, a legtöbbször észre sem veszik. Kritikusabb a helyzet, ha elfogadunk mondjuk 100 %-nál nagyobb kamatot, vagy éppen február 31-i teljesítést. De vajon TÉNYLEG ki kell ezeket javítani? Kockáztatni a build sikerességét és az átadást? Nem elég csak ledokumentálni *(it's a feature, not a bug)*, és majd a következő verzióban kijavítani? Hol a határ?

Ebben a nehéz kérdésben segít az MSF bug/issue management része.

Hogyan kezeljem azt, ha az ügyfél mindent akar?

A legjobb terv is arra való, hogy azután legyen mitől eltérni. Gyakran előfordul, hogy valami közbejön, és a csapat nem ké-

pes mindent határidőre „megkódolni". Az emberi hibákon túl *(síbalesetet szenved az egyik főkóderünk)* a környezetből *(például lefagy a fejlesztői gépem két napra)* és az ügyfél hozzáállásából fakadó hibák is közrejátszanak szép terveink rombadőlésében. Tekintettel arra, hogy MINDEN szoftverprojekt alatt amúgy is ingoványos a talaj *(lásd: az ügyfél nem is tudja, mit akar)*, érdemes lehet a szerződésbe belevenni, hogy milyen irányelvek szerint térünk el, ha gáz van. GÁZ PEDIG VAN!

Egy sikeres gáztervezési esetünkben az ügyfelünkkel sikerült előre megállapodni, hogy mely funkciók eshetnek áldozatul egy esetleges gáztámadásnak. Egyetértett, hogy a portálja fórum nélkül is elindulhat, és az adminisztrációban is találtunk tíz olyan kényelmi funkciót, ami nélkül tulajdonképpen simán el lehet kezelgetni az alkalmazást.

Mit ad Isten, a cég PR-főnöke kitalálta, hogy egy héttel hamarabb kell elindulni a tervezett időpontnál! A céltudatos tervezés miatt ez esetben nem szenvedtünk gázmérgezést, mert a buildünk stabil volt minden nap, és ami utolsónak maradt *(a fórum és a 10 kényelmi funkció)* azt egyszerűen kidobtuk a projektből és nem is próbáltuk meg egy héttel kevesebb idő alatt befejezni, hisz jól tudtuk *(szerencsére ez a szerződésben is benne volt!)* hogy ez lehetetlen. A portál határidő előtt egy héttel elindult, és mindenki örült. *(Persze még így is további egy hétig reszeltük az éles rendszert. Elképzelhető, mi lett volna, ha még a fórummal is megpróbálkozunk!)*

Sokan félnek az őszinteségtől, pedig a bizalom megszerzése a legfontosabb. Ha nem csalódnak bennünk, legközelebb is minket választanak.

Az MSF trade-off matrix megtette a hatását.

Titkolózás vs. őszinteség

Az alábbi esettanulmány címe ez is lehetne: ki mit ért 60 kilobájtos képek kezelésén? Adva van egy project, melynek részeként kisebb képfájlokat kell megjeleníteni a képernyőn. A megrendelő csak annyit mond a képekről: ne izguljunk, maximum 60-70 kilobájtosak.

Korábbi projektekben megedződött lelki szemeink előtt kicsi, színes igazolványképek lebegnek, és ennek megfelelően rittyentünk egy túlbiztosított, akár 100 k-s képek megjelenítésére is alkalmas ActiveX-vezérlőt.

Minden rendben van, a build jó, stabil és gyors. Csakhogy a megrendelőnél a képkezelő ActiveX hatására az egész oprendszer simán, hibaüzenet nélkül újrabootolt! Mi az ördög? Reboot funkciót is belefejlesztettünk?

1. Mi NT4-et használtunk 128 MB RAM-mal *(jó régen volt)* a külföldi ügyfelünk gépein viszont Win95 volt 32 megával.
2. Nem tudtuk, hogy ők a képkezelőt hatalmas, 5000 x 3000 pixeles képekkel fogják túráztatni! A fájlméret stimmelt, 50-60k. Mivel csak ennyit tudtunk, azt hittük ezek maximum 800x600-as JPG-k. 24 bites színmélységgel egy az egyben kirakattuk a Windows GDI-vel a képernyőre a megfelelő ablakba.

A mi képeinkkel mindez 5-10 MB memóriát evett – hadd egyen! Csakhogy a 60k-s képek, amiket ők akartak kezelni, fekete-fehér 1 bites TIFF-ek *(beszkennelt tervrajzok)* voltak, és az 24 bitben 120 MB RAM-ot kért. A Win95 ettől egyszerűen és csendesen lehasalt. Ki a hibás? Miért VB-ben írtuk meg ezt az egyszerű vezérlőt? Miért nem optimalizált C++ kódot raktunk alá?

Azért mert nem tudtuk mi a cél, nem láttuk a kockázatokat, nem volt jó a management.

Ha **MSF shared vision modell** szerint dolgoztunk volna, mindenkiben tudatosult volna a cél: „bazi nagy fekete-fehér képeket kell kezelni". Ennek tudatában biztosan nem VB-vel állunk neki a feladatnak, sőt, lehet, hogy nem is kliens-, hanem szerveroldali képfeldolgozást javaslunk.

Az MSF-keretrendszer

Ennyi probléma után essék szó a megoldás lehetőségéről is. A Microsoft Solution Framework egy keretrendszer, amely szabályozza az IT projektek folyamatait, ám mégis szabad utat enged a kreativitásnak, az újszerű megoldásoknak. Az MSF modellek gyűjteménye, melyeket mint szerszámokat bármikor előhúzhatunk a szerszámosládából, ha a szükség úgy hozza. Ennek a megközelítésnek az a legfőbb előnye, hogy nem kényszerít ránk olyan modelleket vagy szabályokat, amelyek egy adott projekt lefolytatásához és sikerre viteléhez nem szükségesek.

Az MSF alapjai a csapatmodell, a folyamatmodell, a proaktív kockázatkezelés, a korszerű hibakövetés, a minőségirányítás támogatására bevezetett követhetőség, és egyéb hasznos ötletek. Ezen modellek alkalmazása több előnnyel is jár, ezek közül a legfontosabb, hogy a csapat tagjai mindig tudják, mi történik, illetve mindig pontosan tisztában vannak azzal, hogy az adott részért ki a felelős. Így a csapat hatékonysága megnő, a határidő pontosan tartható, nincs költségtúllépés.

Meggyőződésem, hogy a fenti mondatokat más szakértők anyagaiban is olvasták, és most az fordul meg a fejükben, hogy leírni könnyű, de teljesíteni valószínűleg lehetetlen. Az igazság viszont az, hogy noha ideális projekt nem is létezik, az MSF alapelveinek okos használatával több sikeres projektet végigvittem, gyakran ***a megrendelő legnagyobb meglepetésére***. Az MSF alkalmazása nem könnyű, sok gyakorlás kell hozzá. Meg kell küzdeni feletteseinkkel, ügyfeleinkkel, hogy rávegyük őket az alapelvek használatára. Az MSF tartogat néhány meglepetést, amiből párat alább felsorolok.

Meglepetések

Első meglepetés, hogy az MSF-csapatmodell elveti az egyszemélyes projektfelelős koncepcióját, mert készítői azt vallják *(és milyen igazuk van!)*, hogy egy átlagos IT projekt terheit nem képes egyetlen ember a vállán viselni, legyen bármilyen jó szakember. Ha mégis kiváló vezető, akkor is nehézségbe ütközik, hiszen ellentmondásos érdekeket kell magában egyeztetnie. Az MSF-csapat ezzel szemben hat egyenlő szerepkörre épül, ahol mindenkinek megvan a maga felelősségi területe. A szerepkörök feladatai sokszor ellentmondanak egymásnak. A csapat akkor működik jól, ha a hat szerep képviselői konszenzusra jutnak, amire az MSF konkrét utakat, módszereket jelöl ki. A szerepek kisebb projekteknél összevonhatók, így a legkisebb MSF-csapat háromtagú lehet.

A csapatmodell hat szerepköre és felelősségi területei a következők: a Product Manager felelős az ügyfél elégedettségéért, a Program Manager felelős a fejlesztés gördülékeny haladásáért, a Development felelős azért, hogy a rendszer a specifikációnak megfelelően, határidőre készüljön el, a Test felelős a termék minőségéért, a User Experience felelős a végfelhasználók igényeinek figyelembevételéért és oktatásáért, és végül, de semmiképpen sem utolsó sorban a Release Manager felelős a rendszer üzembehelyezéséért. Jól látszik, hogy a Product Manager és a User Experience között konfliktushelyzet alakulhat ki, hiszen a termékfelelőst az alacsony ár érdekli, míg a felhasználók felelősét a minél gazdagabb felhasználói élmény, ami pedig költséges. Ezt egy csapat sokkal hatékonyabban tudja feloldani, mint egyetlen felelős.

Az MSF folyamatmodellje egy módosított spirálmodellre épül, amely ötvözi a vízesésmodell és a spirálmodell előnyeit. Az MSF legújabb, harmadik változatában egy projekt öt részből áll. Az első az Envisioning, melynek során megszületik a projekt víziója, és eldöntetik, hogy annak mely részei fogják a projekt kereteit képezni. A következő fázis a Planning, melynek során a vízió megvalósításra kiválasztott részeit részletesen megtervezik és elkészül a projektterv. A harmadik a Development, ennek a végén már kész termék jelenik meg, ami már csak stabilizálásra szorul. A Stabilizing fázis végén tehát a termék elnyeri végső, telepítésre alkalmas alakját, és ezzel belépünk az ötödik, Deployment fázisba. Fontos, hogy a hat szerep minden negyedben egyformán aktív. A Test szerep például már a víziót vagy a terveket is ellenőrzi, teszteli, állandóan hibákat keres. A Release Manager szintén már a legelején aktív, minden felmerülő ötletet elemez az üzemeltetés szempontjából, így nem a végén derül ki, hogy egy nagyszerű funkció esetleg nem képes működni a rendszer valós környezetében.
A kockázatkezelés a projekt teljes időtartama alatt működik, ez miden csapattag felelőssége. A kockázati tényezőket dokumentáljuk, rangsoroljuk, majd felelősöket rendelünk hozzájuk. Így ha a kockázat problémává válik, azaz bekövetkezik egy nem várt esemény, az valójában várt esemény lesz, és dokumentálva lesznek a tennivalók. Ennek a főnökség és a megrendelő egyformán örül, de csak akkor működik, ha őszinték vagyunk egymáshoz, és elsősorban magunkhoz.
Az utolsó három fázis alatt működik a hibakövetés, feladata a szoftverben keletkező hibák felderítése, illetve feldolgozása. Fő felelőse a Test szerep, illetve a Development csapat.

Az MSF ereje

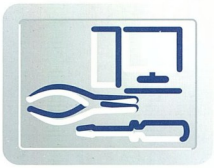
Az MSF ereje abban rejlik, hogy a megrendelőt is a fejlesztői csapat részeként kezeli, nem zárja ki a tervezésből, a tesztelésből és a kockázatok kezeléséből. Ha van olyan érv, ami igazán az MSF alkalmazása mellett szól, akkor ez az. Ilyen kérdés a határidők megszabása, amely hagyományosan már akkor lezajlik, amikor még azt sem tudjuk pontosan, mi a feladat.
Az MSF a határidőket úgy tartja be, hogy pontos modellt ad a határidők, a rendelkezésre álló emberi és anyagi erőforrások és az elvégzendő feladatok közötti egyensúly meghatározására. Ez párosul azzal az alapelvvel, hogy a határidőket azok becslik meg az egyes feladatokra, akik a feladatot végzik. Így a projektmenedzsment nem kerül abba a helyzetbe, hogy irreális határidőket szabjon meg a programozóknak.

MSF a Microsoftnál – a Daily Build

Végül megemlítek egy kevéssé ismert fejlesztőcéget, akik sikeresen használják az MSF-keretrendszert: ez a Microsoft maga. A teljes rendszerből az úgynevezett napi buildet vegyük górcső alá.
A Microsoft saját magán szerzett keserű tapasztalataiból származik a mindennapra egy build technika. A build nem más, mint az összes forráskód összefordítása egy *(fél)*késztermékké. A napi build arra jó, hogy azonnal kiderülhessen, vajon egyáltalán összefordítható-e még a sok fejlesztő által elszigetelten készített forráskód, vagy szétszaladtak a lovak.
Emellett – ha minden jól megy – minden nap, minden este készül egy stabil rendszer, amit ha kell, akár holnap szállíthatunk! A napi build abban is segít, hogy minden főnök láthassa, a szoftver „hogy is áll" egy adott pillanatban. Ha a daily build nem készül el napok óta, az jelzés, hogy valami nincs rendben. A daily build úgy készül, hogy minden fejlesztő minden reggel leszedi a többi fejlesztő előző stabil kódját, és annak birtokában fejleszti tovább a maga részét. Így elkerülhető, hogy ősi verziókkal feleslegesen küzdjön. Ha valamit befejezett, azt a saját gépén „buildeli" és teszteli is egyben. *(Nem vár a tesztelőkre! Ő maga is tesztel!)*
Ha ez a lépés sikeres, beküldi a kódot a buildelőknek. A napi build sok-sok programozó összes beküldött kódjából áll össze – ha összeáll. Ha nem sikerül a build, azt a kódot, ami miatt nem áll össze, visszadobják a bűnösnek, tegnapi kódja kerül a buildbe, és másnap már az új buildhez kell hozzáigazítania a kódját. Fordítás és linkelés után az új build lesz az aktuális, ezt lehet hagyományos módszerekkel tesztelni. A Microsoftnál minden éjszaka tesztelnek, aminek eredménye, hogy a másnap reggel 9-kor tartott meetingen már hibalisták *(bugreport)* állnak rendelkezésre! Fontos megjegyezni, hogy a bűnös programozó, aki miatt a build nem állt össze nem büntetést, hanem segítséget kap. Meglepő, ugye?
Az MSF működik. Eddigi legnagyobb projektem, amit az MSF segítségével határidőre végeztünk el, 1 millió euró költségvetésű volt.

Bíró Tamás (MCSD)
bt@sensenet.hu

Kapcsolódó NetAcademia tanfolyam:
Microsoft Solutions Framework



Portál Para[digma]

III. rész – the X files

Sorozatunk első két részében sokat emlegettük a különféle x-szel kezdődő fájlokat, elsősorban az XML-t és az XSLT-t. Arról azonban kevés szó esett, hogy pontosan milyen szerepet is kapnak az X-fájlok egy portál életében, illetve hogyan kell őket alkalmazni egy többrétegű korszerű portálmegoldás fejlesztése közben. Az igazság ezúttal ideát van.

Az XML és XSLT transzformációt a legtöbb portálmegoldásban az XML-fájlok HTML-re konvertálására használják, azaz az XML a kapcsolat az üzleti logika és a prezentáció között, és az XSLT maga a prezentációs réteg. Az XML azonban ennél többet tud.

Az XML-technológia helyes alkalmazásával valóban könnyen fejleszthetők igazi többrétegű, komponensalapú alkalmazások. Amennyiben a részegységek XML-fájlokon *(streameken, adatfolyamokon)* keresztül kommunikálnak egymással, amelyek előre definiált szerkezettel *(sémával)* rendelkeznek, és pengeélesen elhatárolódnak egymástól, lehetővé válik, hogy egymástól függetlenül fejlesszük, vagy akár kicseréljük őket. A régi, nem XML-alapú komponensek egy kommunikációs réteg megírásával könnyen illeszthetők XML-hez. A már eleve xml-t küldő és fogadó alkalmazásokat pedig egyszerűen lehet illeszteni: vagy módosítani kell a kommunikációs rétegen, vagy XSLT transzformációt kell alkalmazni.

Az XML technológia tehát nemcsak a prezentációban kap szerepet. Bárhol használhatjuk, ahol komponensek kapcsolódnak, azzal a különbséggel, hogy a prezentáció mindenképpen megkívánja az XML transzformálását valamilyen más fogyasztható formátumra, míg az egyéb komponensek közötti kommunikációhoz nem feltétlenül kell átalakítást végezni. A dolog igazi szépsége a szerepek megmaradása. Az adatforrás előállít, a megjelenítő megjelenít. Például egy ügyféllistát előállító komponensnek nem kell foglalkoznia azzal, hogy a kimenete hova kerül, hiszen az általa szolgáltatott XML transzformálható HTML-lé, vagy – további feldolgozás céljából – illeszthető egy másik komponens bemenetére is. Ezáltal érvényesül az xml egyik legfontosabb elve, miszerint az adat elválik a megjelenéstől, csak a tartalomról hordoz metainformációt, annak feldolgozásáról nem.

Az xml technológia a strukturáltságot és az uniformizmust hozza be a komponensek közötti kommunikációba. A modern portálmotoroknál ez a tény fokozottan előtérbe kerül, mivel nemcsak az egyes belső komponensekkel kell sakkozni, hanem a portált a környezetbe is be kell illeszteni, már meglévő komponenseket és külső alkalmazásokat is integrálni kell. Ekkor realizálódik az az előny, hogy a komponensek azonos nyelven beszélnek.

Az X-fájlok

A technológia mindhárom főszereplője W3C szabvány. Azonos nyelvtant használnak, hiszen az xslt és az xsd is xml fájlok. Így ha valaki *(vagy valami)* tud xml-t olvasni, akkor sémát és transformert is képes.

Az XML az eXtensible Markup Language, az XSLT az eXtensible Stylesheet Language for Transformations, az XSD pedig az XML Schema Definition szavakat takarja. Félreértésre adhat okot az XML értelmezésben a language, és az XSLT-ben a stylesheet szó. Helyesebb értelmezés szerint az XML nem nyelv, csak szintaxis, nyelvvé akkor válik, amikor egy sémával *(XSD)* előírjuk a szerkezeti és tartalmi elemeit. Tehát itt a kulcsszó az extensible. Valamint az XSLT nem stylesheet, hanem teljes értékű programnyelv, amit XML forrás átalakítására találtak ki, itt a kulcsszó inkább a language és a transformation.

> Az XML-technológia helyes alkalmazásával valóban könnyen fejleszthetők igazi többrétegű, komponensalapú alkalmazások.

Az XML és az XSD

Abból, hogy a komponensek közötti kommunikáció nyelve xml, két fontos előnyünk származik. Tudjuk, hogy egy üzleti szoftver fejlesztése ma már nem képzelhető el mások által megírt komponensek felhasználása nélkül. Mivel a komponensek a legkülönfélébb adatokat illetve objektumokat kérnek és szolgáltatnak, a fejlesztés idejének nem elhanyagolható része a komponensek közötti adatcsere összehangolása. Az egyik előny ennek az időnek a jelentős csökkenése, hiszen az illesztés az XML transzformációjával megoldható. A másik, hogy így az alkalmazások akár a weben is kommunikálhatnak egymással *(pl: webservice, SOAP)*, hiszen az XML karakteres adatfolyam, így átvitele semmilyen problémába nem ütközik.

Az XML tehát adathordozó. Mivel bonyolult struktúrák is leírhatók vele, helyes alkalmazásával nemcsak egyes adatok, tömbök, hanem teljes objektumok is átvihetők, szerializálhatók vele *(természetesen a metódusok nem)*. A .NET framework meg is valósít egy olyan névteret, amelyben XML-szerializációt segítő osztályok vannak. Valójában a webszolgáltatások által használt SOAP-protokoll is így visz át objektumokat.

Ha szoftverkomponenst készítünk, annak be- és kimeneti adatstruktúráit alaposan dokumentálnunk, valamint a bemeneti adatokat ellenőriznünk kell. Ha komponensünk XML bemeneteket vár és XML kimeneteket produkál, ismét előnyben vagyunk. Az XML formátumát sémával *(XSD)* írjuk le. Már az is előny, hogy ha példa XML-eket mellékelünk a be- és kimenetekre, azokat egy gyakorlott szem általában azonnal átlátja, ám a séma használatával teljes értékű és szabványos nyelvű dokumentációt, valamint validációs eszközt nyújtunk.

Az XSD-vel *(lásd a szabványt a **[schema]** címen)*, szinte min-

dent előírhatunk a várt formátummal kapcsolatban. Szabályozhatjuk az elemek sorrendjét, mennyiségét és adattípusát is. Az XML érvényesítése sem probléma, hiszen ez teljes egészében a technológia sajátja. Egy XML támogatást nyújtó környezetnek tudnia kell validációt végezni akár úgy is, hogy a séma nincs is a lokális gépen, csak az Interneten át érhető el.

Az XSLT

Az X-fájlok közül az XSLT megértése szokta a legtöbb problémát okozni, mivel elsőre nem igazán érthető, hogyan működik. A most következő fejezetek nem kívánják az XSLT nyelv részletes bemutatását, azt megtették mások, csupán gondolatébresztőnek szánom őket, valamint egy-két érdekes probléma megoldását szeretném megosztani a kedves Olvasóval.

Az XSLT feladata a transzformálás. A bemenete mindig XML, a kimenete pedig háromféle lehet. Ha a feladat csatolás egy másik komponenshez, a kimenet pedig XML, ekkor transzformernek szoktuk hívni. Ha a feladat prezentáció, lehet text *(pl. email body)*, vagy HTML, ekkor megjelenítőnek, renderernek is szoktuk nevezni. A felosztás nem teljesen egzakt, mivel például WML prezentáció is generálható, ami szintén XML szintaxisú. Az XSLT működése ugyanaz akármilyen kimenete is legyen. Az XSLT jól megfér a megszokott CSS-sel is, mivel más a feladatuk. A stíluslap ugyanúgy használható külső fájlként, vagy inline scriptként, mint eddig. A prezentáció teljesen transzparens, hiszen szerveroldalon generált HTML is csak HTML, a lényeg a bemeneten van. Ha a bemenet XML, azt a leghatékonyabban XSLT-vel tudjuk transzformálni, megjeleníteni, akár spártai egyszerűségű, akár csilli-villi izgő-mozgó oldalakról van szó.

Fontos előny a szerveroldali scriptekkel szemben, hogy míg azok csak a webszerveren futnak, addig az XSLT kliensoldalon is futtatható. Ez esetben a szervernek csak a nyers adatokat kell szolgáltatnia, a prezentációt és a felhasználói inputok ellenőrzésének egy részét kliensoldalon is el lehet végezni. Ehhez a meglévő adatokon, és az üzleti logikán semmit, vagy csak alig kell módosítanunk. Az is nagy erőssége az XSLT-nek, hogy nem értelmezett, hanem lefordított kódként fut. Ha a futtatóobjektumot létrehoztuk, és feltöltöttük a forráskóddal, az lefordul, és ezt a csőre töltött objektumot akár cache-elhetjük is. A transzformálás így annyira gyors lehet, hogy még egy nagyon nagy portál HTML fájljait sem kell statikusan legenerálni.

Most pedig lássuk az XSLT működését közelebbről!

XSLT at work

Minden kódvégrehajtó egység rendelkezik bizonyos láthatatlan alapfunkciókkal, amelyek mindig végrehajtódnak. A szekvenciális végrehajtók *(mikroproci, értelmezőprogram)* például az utasítás beolvasása után a következő utasításra állnak, vagy függvényhíváskor veremben tárolják a visszatérés helyét. Az XSLT nem így működik, hanem context-nodeokat jelöl meg a forrás XML-ben, node listákba válogatja az elemeket, és sablonokat párosít, beépített algoritmust nyújt a rekurzív fabejárásra is. A dolog egyáltalán nem öncélú. A rekurzív fabejárást a szokásos iteratív nyelveken sem nehéz megvalósítani, de ezt teljesen felesleges minden alkalmazásban megírni, mivel a feldolgozandó adatstruktúra mindig egy fa, és mivel minden fabejáró ugyanazt a logikát követi. Gyakorlatilag az XSLT ennek minden gondját-baját leveszi a vállunkról, és elrejti.

Az XSLT rekurzív nyelv. Nincs FOR... NEXT, DO... LOOP, nincs tömb, sőt még a változók is más értelmet nyernek. Sok minden másképp működik, mint a megszokott programnyelvekben, de ez egyáltalán nem idegen az emberi gondolkodásmódtól, főleg a programozókétól. A programozók már a kezdeti időktől fogva makrókat, blokkokat, függvényeket írnak, és később a híváskor egyetlen utasításban hivatkoznak rá. Ezekben a nagyobb egységekben természetesen további hivatkozások lehetnek – elvben akármilyen mélységig. Az így kapott algoritmusok szorosan összefüggenek a feldolgozandó adathalmaz struktúrájával.

Az XSLT filozófiája hasonló, csak nem a szekvencia, hanem a blokk kerül előtérbe, kiegészülve a szelekcióval, valamint azzal, hogy mindig van *(legalább egy)* meghatározott bemeneti fastruktúrájú adathalmaz. A szelektálás XPath nyelvű, amely egyszerűsége mellett, igen bonyolult lekérdezéseket tesz lehetővé a bemeneti XML-ből.

A szelektálás miatt az XSLT-vel való munka inkább az SQL-re hasonlít, mint a szekvenciális programvégrehajtásra: egy lépést egy szelektált halmazon kell értelmezni. A szelektált halmaz egyes elemein az XSLT végrehajtó végiglépked *(mintha egy forward-only kurzor lenne)*, egy illeszkedő sablont keres, és ha talál, azonnal elkezdi annak végrehajtását. A halmaz éppen aktuális elemét hívjuk context-node-nak *(az adott sablonban a context-node XPath reprezentációja mindig egy pont)*. A megkezdett új sablon új lekérdezéseket tartalmazhat, ami egy új halmazt hoz létre, és annak elemeire szintén sablonokat keres a végrehajtó. A halmaz következő elemére csak akkor lép, ha egy sablonban nincs szelekció, vagy az adott elemre nincs találat. A halmazok előállítása, az azon való lépegetés, és a sablonok keresése automatikus, nekünk nincs más dolgunk, mint előírni a sablonokat, a lekérdezéseket, és természetesen azokat a statikus elemeket, amelyek változtatás nélkül kerülnek a kimenetre.

A szelektálás miatt az XSLT-vel való munka inkább az SQL-re hasonlít, mint a szekvenciális programvégrehajtásra.

Tulajdonképpen az XSLT-kód néhány definíció-, és feldolgozási utasítást leszámítva nem is áll másból csak sablonokból, azokon belül pedig statikus outputból, és szelekciókból.

Az alábbiakban nézzünk meg egy-két gyakorlati példát, miközben nagy lépésekben átvesszük az XSLT alapjait is!

Egy példa: adatáramlás a legalsó szintről a legfölsőre

Legyen a feladat egy *(nem létező)* ügyféllista megjelenítése. Ehhez adott egy SQL adatbázistábla *(tblCompanies)* az alábbi mezőkkel: CompanyID, Name, Address, Email. Az adatkezelő réteg egyik metódusa, mondjuk a GetCompanyList(), lekérdezi a táblát az SQL Server XML-támogatását segítségül hívva *(SELECT * FROM tblCompanies FOR XML AUTO, ELEMENTS)*, ami a következő adatfolyamot produkálja:

```
<tblCompanies>
  <CompanyID>1</CompanyID>
  <Name>Targonca Kft.</Name>
  <Address>Bp. 1921.</Address>
  <Email>info@targonca.hu</Email>
</tblCompanies>

<tblCompanies>
  <CompanyID>2</CompanyID>
  <Name>Duna Bt.</Name>
  <Address>Bp. XXXI. Ezer tér 1.</Address>
```

```
  <Email>info@duna.hu</Email>
</tblCompanies>

<tblCompanies>
  <CompanyID>3</CompanyID>
  <Name>Dollar + Cent inc.</Name>
  <Address>Nagybér. Gazdag u. 328.</Address>
  <Email>info@DCI.hu</Email>
</tblCompanies>
```

Ez egy XML fragment, ami önmagában nem érvényes XML, mivel nincs sem XML deklarációja, sem gyökéreleme. Ez a fajta lekérdezés így működik és kész. Ezzel az adatfolyammal többféle dolgot tehetünk. Betölthetjük egy már létező DOM document valamelyik element-jébe mint inner-xml. Vagy ami számunkra most fontosabb: átalakíthatjuk egy másfajta XML-lé. Szerencsére az XSLT transformert nem zavarja, hogy csak XML fragmentet kell feldolgoznia. Tehát a kapott adatfolyamot a következő XSLT transzformerrel illesztjük az üzleti logika bemenetére, amely XML-XML transzformációt végez *(a leírására még visszatérek)*:

```
<?xml version="1.0"?>
<xsl:transform version="1.0" xmlns:xsl="
➥http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"
➥  omit-xml-declaration="yes"/>

  <xsl:template match="/">
    <xsl:element name="Companies">
      <xsl:apply-templates select="tblCompanies"/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="tblCompanies">
    <xsl:element name="Company">
      <xsl:copy-of select="Name | Address | Email"/>
    </xsl:element>
  </xsl:template>
</xsl:transform>
```

A transzformer kimenete lesz az üzleti logika bemenete, ami egy szerializált objektumkollekció. Ez az első kapcsolódási pont az alkalmazásunk komponensei között. Mivel a jelenlegi példánkban az üzleti logika nem nyúl a listához, csak például ellenőrzi a jogosultságot, és bejegyzi a lekérdezés tényét a logfájlba, ez lesz a kimenet is a második kapcsolódási pont, a prezentációs réteg felé:

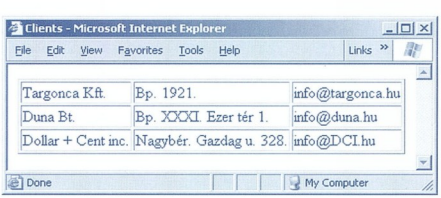
```
<Companies>
  <Company>
    <Name>Targonca Kft.</Name>
    <Address>Bp. 1921.</Address>
    <Email>info@targonca.hu</Email>
  </Company>

  <Company>
    <Name>Duna Bt.</Name>
    <Address>Bp. XXXI. Ezer tér 1.</Address>
    <Email>info@duna.hu</Email>
  </Company>

  <Company>
    <Name>Dollar + Cent inc.</Name>
    <Address>Nagybér. Gazdag u. 328.</Address>
    <Email>info@DCI.hu</Email>
  </Company>
</Companies>
```

A prezentációs rétegnek, jelen esetben egy böngészőnek az alábbi scriptet szánjuk:

```
<HTML>
  <HEAD>
    <TITLE>Clients</TITLE>
  </HEAD>
  <BODY>
    <TABLE BORDER="1">
      <TR>
        <TD>Targonca Kft.</TD>
        <TD>Bp. 1921.</TD>
        <TD>info@targonca.hu</TD>
      </TR>
      <TR>
        <TD>Duna Bt.</TD>
        <TD>Bp. XXXI. Ezer tér 1.</TD>
        <TD>info@duna.hu</TD>
      </TR>
      <TR>
        <TD>Dollar + Cent inc.</TD>
        <TD>Nagybér. Gazdag u. 328.</TD>
        <TD>info@DCI.hu</TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

Ezt megkaphatjuk például a következő egyszerű, de teljes értékű megjelenítőtől:

```
 1 <?xml version="1.0" ?>
 2 <xsl:transform version="1.0" xmlns:xsl="
➥http://www.w3.org/1999/XSL/Transform">
 3   <xsl:output method="html"/>
 4   <xsl:template match="/">
 5     <HTML>
 6       <HEAD>
 7         <TITLE>Clients</TITLE>
 8       </HEAD>
 9       <BODY>
10         <xsl:apply-templates select="Companies"/>
11       </BODY>
12     </HTML>
13   </xsl:template>
14   <xsl:template match="Companies">
15     <TABLE BORDER="1">
16       <xsl:apply-templates select="Company"/>
17     </TABLE>
18   </xsl:template>
19   <xsl:template match="Company">
20     <TR>
21       <xsl:apply-templates/>
22     </TR>
23   </xsl:template>
24   <xsl:template match="*">
25     <TD>
26       <xsl:value-of select="."/>
27     </TD>
28   </xsl:template>
29 </xsl:transform>
```



Ezt a megjelenést várjuk el

A fent leírt folyamat teljesen tipikus. Jól látható az alkalmazás több rétege és az azok közötti illesztés. Ha a fenti alkalmazásban például az adatkezelő réteget le kell cserélni *(mert például*

más a platform és nem áll rendelkezésre SQL Server vagy más alkalmazásból jön az adat), csak az adatkezelő- és az üzleti logikai rétegek közötti illesztést kell átalakítani. Az, hogy a transzformálást melyik réteg végzi, az eredmény szempontjából lényegtelen. A példában a prezentációs réteg tulajdonképpen az XSLT, valamint az a logika, amelyik meghatározza, hogy melyik megjelenítőt kell használni, és az a kód, amelyik fizikailag meghívja a transzformáló metódust.

És most nézzük meg részletesen a legutóbb leírt megjelenítőt! Az első sorból megtudjuk, hogy XML szintaxisú fájlt olvasunk, a 2.-ból, hogy a tartalma egy XSLT, és a 3.-ból, hogy hatására HTML output keletkezik. Az 1-2. és 29. sorok szükségesek minimálisan egy helyes transformerhez. Az 'xsl:template' elemek *(4-13, 14-18, 19-23, 24-28. sorok)* az igazi műveletvégzők, ezek alakítják a kérdéses XML elemet a megfelelő kimeneti részfává. Az 'xsl:apply-templates' elemek *(10, 16, 21. sorok)* végzik a rekurziót az adott elemen belül, ezek állítják elő azt a halmazt, amely elemeire template-t kell keresni. Végül a 26. sorban lévő 'xsl:value-of' elem az, amelyik a forrás XML-ből adatot emel át a kimenetre. A többi sor statikus tartalom, amely változatlanul kerül a kimenetre.

A fent leírt megjelenítő azért lehet ennyire egyszerű, mert a bemeneti és a kimeneti fa nagyon hasonló, nem ugyanaz, de 'ugyanarra megy': ahol a forrásfa bejárásakor lefelé lépünk, ott a kimeneti fa generálásakor is al-elemet hozunk létre. A másik egyszerűsítés, hogy minden információhordozó elemet azonos módon jelenítettünk meg, sőt azok logikailag is egyenértékűek. Ez a való világban természetesen sohasem fordul elő, de arra jó, hogy áttekintsük a transformer működését.

Ami nem egészen nyilvánvaló: az XSLT rendelkezik egy beépített sablonnal, amely alapállapotban a gyökérelem text() node-jait jeleníti meg *(vagyis összefüggő szövegfolyamként, szóközök, tabok és enterek nélkül megkapjuk az XML-ben lévő összes olyan szöveget, amelyek nincsenek a '<' és a '>' közé zárva).* A részletek a **[built-in-rule]** címen olvashatók. A lényeg, hogy ezt felül kell definiálnunk, azaz meg kell adnunk egy alapszabályt, amely akkor lép érvénybe, amikor elkezdődik a transzformálás, ha magyarul mondanám: a context-node az XML Document. Az alapszabály leggyakrabban így kezdődik:

```
<xsl:template match="/">
```

Fontos tudni, hogy ebben a sablonban nincs context-node *(más felfogás szerint a context-node a teljes dokumentum),* tehát ebbe a sablonba azok a dolgok kerülhetnek, amelyeknek az adattól függetlenül meg kell jelenniük a kimeneten. Az alapszabályban leírt 'apply-templates' utasítás *(10. sor)* az, amely használható context-node-ot választ ki a forrás XML-ből. Példánkban ez a <Companies> elem. Erre az elemre a 14. sorban kezdődő sablon illeszkedik, tehát annak végrehajtása következik teljes rekurzióval. Majd ha az teljes egészében befejeződött, folytatódik a 11. sorban ennek a sablonnak a végrehajtása.

Az 'apply-templates' alakjai

Ez az XSLT nyelv egyik legfontosabb utasítása, amely több alakban létezik. Az egyik, a 'select' attributummal ellátott verzió hatására összeáll egy halmaz *(node-list),* amely a context-node-hoz képest relatív XPath-nak megfelelő elemekből fog állni, majd ezekre egyenként sablonokat keres a végrehajtó. A másik alak a 'select' nélküli verzió, ez arra utasítja a végrehajtót, hogy a context-node részfáját teljes egészében járja be, és úgy alkalmazza a sablonokat. Ezzel a 'mindent egyben' szelekcióval vigyázni kell, mert amennyiben a találati node-list valamelyikére nem lehet sablont illeszteni, az a beépített sablonnal megy ki, vagyis a text() node-jai lesznek az outputra másolva. Ilyen esetben célszerű egy olyan sablont alkalmazni, amelyik lenyeli a sablon nélküli node-okat. Alább egy ilyen sablon látható, ez ki is írja a kérdéses elem nevét három csillaggal megjelölve, ami komolyabb XSLT építése közben hasznos lehet:

```
<xsl:template match="*">
  <xsl:value-of select="name()" />***
</xsl:template>
```

Ez a megoldás az XSLT-sablon beépített prioritásai miatt működhet, amely szerint a nevesített sablonok *(amelyek 'match' attributumában egzakt név van)* nagyobb prioritást élveznek a wildcarddal szemben. A prioritásokról érdemes megnézni a szabványt a **[conflict]** címen.

Ha több azonos nevű elem van az xml-ben, de másképp kell őket megjeleníteni, nem a prioritást kell változtatni, hanem a sablon 'match' attributumában az elem nevén kívül például szerepeltessünk annyi parentet, amennyi elég a megkülönböztetéshez, pl: Name helyett Company/Name.

Melyik legyen: apply-templates vagy for-each?

Feljebb láttuk, hogy az XSLT alapműködése a szelekcióra alkalmazott automatikus sablonkeresés és végrehajtás. Azonban nemcsak az 'apply-templates' utasítást használhatjuk szelektálásra, hanem a 'for-each' utasítást is. Ebben az esetben elegendő egyetlen sablon is. Felfedezhető a hasonlóság az ASP-vel, miszerint adott a HTML váz, amelybe a megfelelő helyre beiktatjuk a dinamikus elemeket. Ez az egyszerűsége és áttekinthetősége miatt csábító lehet. Viszont csak abban az esetben ésszerű a használata, ha egy XML elemet csak egy helyen kell megjeleníteni. Az első példa 4–28. sorait cseréljük le az alábbiakra:

```
<xsl:template match="/">
 <HTML>
  <HEAD>
   <TITLE>Clients</TITLE>
  </HEAD>
  <BODY>
   <TABLE BORDER="1">
    <xsl:for-each select="/Companies/Company">
     <TR>
      <TD><xsl:value-of select="Name" /></TD>
      <TD><xsl:value-of select="Address" /></TD>
      <TD><xsl:value-of select="Email" /></TD>
     </TR>
    </xsl:for-each>
   </TABLE>
  </BODY>
 </HTML>
</xsl:template>
```

Ezen kívül látható, hogy az egyes sorok context-node-jai *(a <Company> elemek)* abszolút eléréssel vannak szelektálva, ami rugalmatlan, mivel ha megváltozik az XML-ben a <Companies> helye, át kell írni a transformer minden abszolút XPath-át, míg az eredeti példában csak a <Companies> elemét érintené a változás.

Ha az eredeti példa sablonjait összehasonlítjuk a bemeneti XML-lel, nem nehéz észrevenni az objektumorientált gondolkodást sem. Mivel a bemenet egy objektum-kollekció, van benne egy sablon, amelyik az egész kollekcióra vonatkozik:

```
14  <xsl:template match="Companies"> ...
```

Van egy sablon, az egyes objektumokra:

```
19  <xsl:template match="Company"> ...
```

És van egy, amelyik az egyes property-kre:

```
24  <xsl:template match="*"> ...
```

Ez a felfogás egyrészt átláthatóbbá teszi az egész alkalmazást, mivel az egyes objektumok belső reprezentációja megegyezhet. Ugyanilyen property-kkel rendelkezhetnek, tehát csak egy modelre van szükség. Másrészt bonyolult megjelenítés esetén maga az XSLT is könnyebben fejleszthető.

Változók és paraméterek

Az XSLT-ben használhatunk a teljes transformerre nézve globális, és az egyes sablonokban érvényes lokális változókat is. Ezen kívül a sablonokat paraméterekkel is elláthatjuk, sőt globális paramétereket adhatunk át a transzformert létrehozó objektumból. Ilyen paraméter lehet például színkód, CSS fájlnév, vagy bármilyen adat, amely a megjelenítést vezérelheti. A paraméterátadás jól látható az alábbi C# függvényben:

```
private string transform(string input)
{
 XmlDocument xd = new XmlDocument();
 XslTransform tr = new XslTransform();
 xd.LoadXml(input); tr.Load("tr.xslt");
 StringWriter strwr = new StringWriter();
 XmlTextWriter xmlwr = new XmlTextWriter(strwr);
 //---- parameter letrehozas es atadas ----
 string MOST = DateTime.Now.ToString()
 XsltArgumentList args = new XsltArgumentList();
 args.AddParam("date", "", MOST);
 tr.Transform(xd, args, xmlwr);
 return strwr.ToString();
}
```

Ez a függvény egy stringben kapott XML-t transzformál úgy, hogy az XSLT-forrást a fájlrendszerből veszi, valamint az éppen aktuális időt átadja a transzformer objektumnak. Az XSLT-t természetesen fel kell készíteni a paraméter fogadására. Az átadott paramétereket egyedi névvel kell ellátni, mindegyiknek léteznie kell, és az XSLT-beli deklarációt csak a 'transform' elem közvetlen gyermekeként lehet szerepeltetni.
A paraméter deklarációja:

```
<xsl:transform version="1.0" ...
 <xsl:param name="date"/>
```

És felhasználása:

```
<xsl:value-of select="$date"/>
```

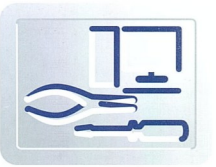
Végül álljon itt egy érdekes effektus. Ha az első példakód 19-23. sorait az alábbira cseréljük, a táblázatunk sorai váltakozó színnel jelennek meg, és a kurzor alatti sor is meg van jelölve *(highlight)*.

```
 1 <xsl:template match="Company">
 2  <xsl:variable name="color">
 3   <xsl:choose>
 4    <xsl:when test="position() mod 2 = 0">
 5 lightyellow</xsl:when>
 6    <xsl:otherwise>white</xsl:otherwise>
 7   </xsl:choose>
 8  </xsl:variable>
 9  <TR ONMOUSEOVER="javascript:
10  this.style.backgroundColor='lightgrey'">
11   <xsl:attribute name="ONMOUSEOUT">
12    <xsl:text>javascript:
13  this.style.backgroundColor='</xsl:text>
14    <xsl:value-of select="$color"/>
15    <xsl:text>';</xsl:text>
16   </xsl:attribute>
17   <xsl:attribute name="BGCOLOR">
18    <xsl:value-of select="$color"/>
19   </xsl:attribute>
20   <xsl:apply-templates />
21  </TR>
22 </xsl:template>
```

Az effektus egy egyszerű HTML trükk. A hatást a TR-elemek háttérszínének váltogatásával érjük el. A háttérszínt egy előre beállított változó *(color)* segítségével írjuk be a megfelelő helyekre *(14. és 18. sor)*. A változó deklarálása és értékadása a 2-8. sorokban történik.
Ennek a példának, és még sok más *(ennél jóval összetettebb)* megoldásnak az ismertetését a következő cikkünkben folytatjuk.

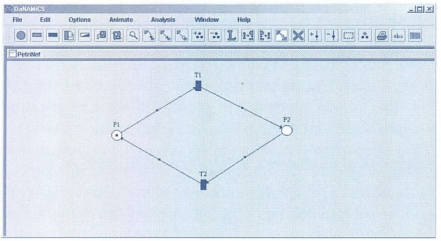
Gyebrovszki Zoltán
gz@sensenet.hu

Petri-hálók modellezése és analízise



A korábbiakban bemutattam a Petri-hálók működését. Most egy kicsit gyakorlatiasabb oldaláról igyekszem megközelíteni a dolgot, így egy olyan eszközt mutatok be, amely modellezési- és különböző analízis célokra egyaránt alkalmas.

Az eszköz, amit választottam, bárki számára hozzáférhető, és kellően sokoldalú ahhoz, hogy ne csak prezentációs célokra használjuk.
Ez az eszköz a DaNAMiCS *(Data Network Architecture: Modelling Concurrent Systems)*, amely letölthető a **[danamics]** címről *(egy demo változat, amely maximum 10 place kezelésére képes)*.



A DaNAMiCS felülete

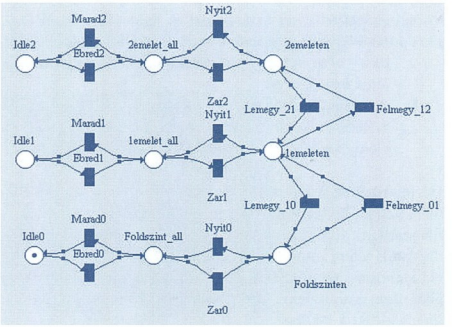
A felső soron szereplő legfontosabb gombok jelentése:

Gomb	Jelentés
●	Place elhelyezése
▭	Időzített illetve időzítés nélküli tranzíció elhelyezése
	Tranzíció elhelyezkedésének módosítása *(vízszintes/függőleges)*
	Tranzíció típusának megváltoztatása *(időzített/időzítés nélküli)*
	Alhálózat beszúrása. Az alhálózatnak már készen kell lennie egy DaNAMiCS *(.bim)* fájlban.
	Alhálózat megtekintése.
	Él felvétele a hálózatba. A két gomb között az a különbség, hogy az előbbi ívelt éleket rajzol, míg az utóbbi egyeneseket.
	Tiltó él beszúrása a hálózatba.
	Token hozzáadása illetve elvétele egy helyről *(place)*.
	Alkotóelem *(place, tranzíció, él stb.)* tulajdonságainak megtekintése.
	A nézet nagyítása/kicsinyítése.
	Alkotóelem mozgatása illetve törlése.
	Kijelölés.
	Tokenek színének változtatása *(lásd a színezett Petri-hálóknál)*.

Ennyi bevezető után lássunk most egy egyszerű példát, amelyen keresztül egyrészt egy tervezési feladat megoldását igyekszem bemutatni, másrészt természetesen a DaNaMiCS működését is.

Példa: Lift Petri-hálója

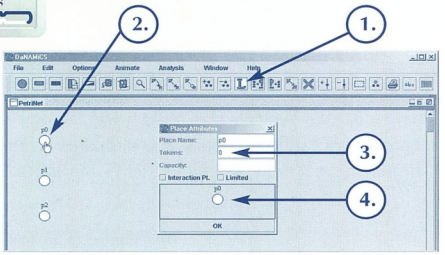
Egy korábbi cikkemben szerepelt egy lift állapotautomatája. Most ezt szeretném feleleveníteni azzal az apró módosítással, hogy az ajtó nyitását és zárását egyszerűen megállásnak nevezzük.
Első megközelítésben az alábbi módon rajzoljuk fel a lifthez tartozó Petri-hálót:



Liftautomata kezdetleges Petri-hálója

Hogyan hozzuk létre a feni Petri-hálót? A DaNAMiCS segítségével először a place-eket helyezzük el: ráklikkelünk a „Place elhelyezése" gombra, majd az ablakban tetszőleges helyre kattintva elhelyezünk oda egy helyet. Ha többször, több helyen kattintunk, természetesen több place kerül elhelyezésre. Hasonló módon rajzoljuk meg a tranzíciókat is. A place-ek és tranzíciók összekötése úgy történik, hogy az „Él felvétele" gombra kattintunk, ezután pedig behúzzuk az élet – lenyomva tartott egérgombbal berajzoljuk a megfelelő éleket *(ne feledjük, hogy él csak place-ből tranzícióba, vagy tranzícióból place-be futhat!)*.
Ha egy-egy place-t vagy tranzíciót elhelyezünk, természetesen a rendszer még nem tudja, hogy azokat nem p0, p1, p2, stb. illetve t0, t1, t2, stb. néven szeretnénk használni. Nekünk azért nem célszerűek ezek az elnevezések, mert bár egyediek, egyáltalán nem beszédesek, nem túl sokat árulnak el arról, hogy tulajdonképpen mit is akartunk modellezni az adott komponenssel. Ezért amint megrajzolunk egy-egy alko-

tóelemet vagy alkotóelem-csoportot, érdemes rögtön átnevezni. Ezt mutatja az alábbi ábra:



Place tulajdonságainak megváltoztatása

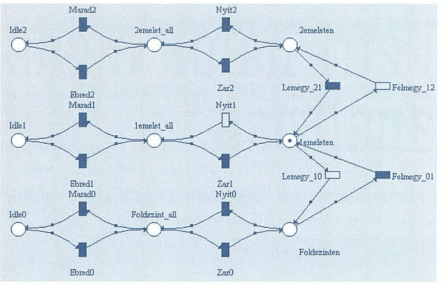
(1) Első lépésként klikkeljünk az „Alkotóelem tulajdonságai" gombra. *(Az éppen aktív funkciót megtestesítő gomb mindig sárga háttérrel jelenik meg).*

(2) Ezt követően ha bármilyen komponens fölé visszük egerünket, a kurzor „mutató kezecskére" változik. Ha ekkor rákattintunk az elemre, egy tulajdonságablak jelenik meg.

(3) Ezen ablak felső részében néhány beviteli mezőt találhatunk. Legfelül a komponens nevét állíthatjuk be. Place esetében ezt a tokenszám követi, azaz hogy hány token kívánunk elhelyezni az adott helyen a kiinduló állapotban.
A következő mezőben a kapacitást állíthatjuk be *(igen, ez a – viszonylag új – alkalmazás képes a kapacitáskorlátok kezelésére).*

(4) Az ablak legalsó részében egy kis ábra található az aktuálisan kijelölt alkotóelemről. Erre kattintgatva az elem és a felirat kölcsönös pozícióját állíthatjuk be.

Ha felrajzoltuk a hálózatot, nyilván szeretnénk látni működés közben. Ehhez először is tokeneket kell elhelyeznünk a hálózatban, hiszen ezek adják az egész működés lényegét. De mit is jelent esetünkben egy-egy token?
Ha alaposan megnézzük a hálózatot, a place-ek jelentik a lift állapotait, a tranzíciók pedig az állapotátmeneteket. Ennek megfelelően a tokenek egy-egy liftet jelölnek, amelyek egymástól függetlenül mozognak az épületben. Egyliftes rendszert akkor kapunk, ha egyetlen tokent veszünk fel a kiindulási állapotba. Tegyük ezt a tokent az Idle0-ba, vagyis feltételezzük, hogy kezdetben liftünk a földszinten áll.
Tokent kétféleképp helyezhetünk el. Egyrészt az adott place tulajdonságlapján is beállíthatjuk, hogy hány token jelöli azt a kezdőállapotban. Másrészt a „Token hozzáadása" funkció segítségével minden egérkattintás eggyel növeli a kijelölt place tokenszámát.
A rendszer működését az alábbiak szerint modellezhetjük: Az AnimatePlay menüpontot kiválasztva elindul a szimuláció. Mivel a Petri-hálók működése nemdeterminisztikus, lépésről lépésre láthatjuk a tüzelhető tranzíciókat, és választhatjuk ki, melyik tüzeljen a következő lépésben. Ezáltal mi biztosítjuk a nemdeterminisztikusságot, ráadásul a különböző tüzelési szekvenciákat így könnyedén, saját igényeinknek megfelelően tesztelhetjük.
Az alábbi ábra azt az állapotot mutatja, amikor a lift az első emeleten van, ajtaja pedig zárva:

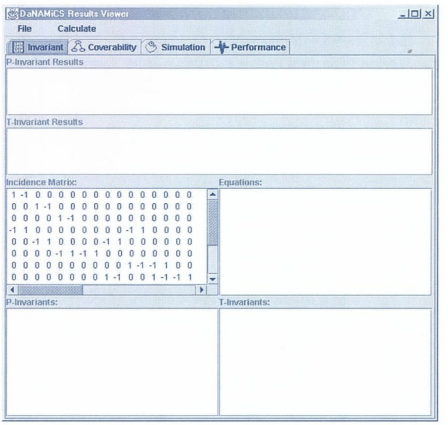


Szimuláció a DaNAMiCS segítségével

Az ábrán színes téglalapok jelölik a tüzelhető tranzíciókat. Ha ezek közül valamelyikre ráklikkelünk, következő lépésként az tüzel, és a Petri-háló átlép a következő állapotba: az új tokeneloszlással a tüzelhető tranzíciók halmaza is megváltozik.
Ezzel eljátszogatva láthatjuk, ahogyan a lift egyik emeletről a másikra „vándorol", illetve az emeleteken nyitja-zárja ajtaját, stb. Ránézésre a működés „korrektnek" tűnik, azaz a lift soha nem kerül holtpontra, illetve soha nem lesz egy liftből kettő, ne adj' isten, még több.
Természetesen ezen intuíciók bizonyíthatók illetve megcáfolhatók pontos matematikai módszerekkel. Lássunk most ezek közül néhányat!

Matematikai módszerek a Petri-háló analízisére

A DaNAMiCS a legtöbb analízismódszert támogatja, most ezek közül mutatok be néhányat. Az AnalysisInvariants menüpontot kiválasztva egy új ablakot kapunk, amelyben több funkciót kijelölhetünk. Válasszuk ki a „Show Incidence Matrix" sort *(a többi egyenlőre nem érdekes számunkra)*, majd nyomjuk le a Calculate gombot.
Ekkor egy olyan ablak jelenik meg, amely több részből áll, valahogy így:



A szomszédsági mátrix

A szomszédsági mátrix esetünkben azt mutatja, hogy mely tranzíciók mely helyekkel szomszédosak *(vannak élekkel összekötve)*, és hogy az adott tranzíció a helyhez hány tokent ad, vagy vesz el onnan.
Lássuk, pontosan hogyan is fest a liftautomata szomszédsági mátrixa. A könnyebbség kedvéért a mátrixot elláttam a megfelelő fejlécekkel, amelyek azt mutatják, hogy a sorok- illetve oszlopok mely tranzícióknak/placeknek felelnek meg:

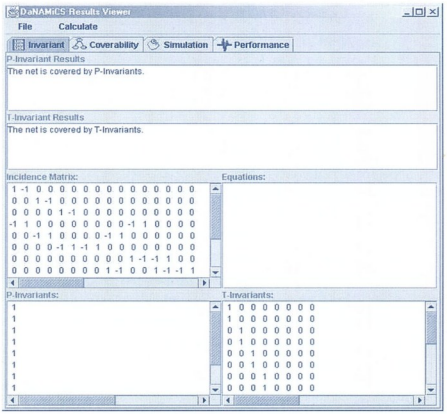
Hely \ Tranzíció	Marad2	Ebred2	Marad1	Ebred1	Marad0	Ebred0	Zar0	Nyit0	Zar1	Nyit1	Zar2	Nyit2	Lemegy_21	Felmegy_12	Lemegy_10	Felmegy_01
Idle2	+	-														
Idle1			+	-												
Idle0					+	-										
2emelet_all	-	+									-	+				
1emelet_all			-	+					-	+						
Foldszint_all					-	+	-	+								
2emeleten											+	-	-	+		
1emeleten									+	-			+	-	-	+
Fsz.-en							+	-							+	-

A fenti táblázatban üres cellák jelölik, ha az adott tranzíció nincs közvetlen kapcsolatban az adott place-szel. Mivel esetünkben minden tranzíció súlya 1, a tokenszámok helyett *(+1 és -1)* csupán az előjeleket tüntettem fel. Így például az első oszlop azt mutatja, hogy a Marad2 tranzíció tüzeléskor az Idle2 helyhez ad egy tokent, a 2emelet_all helyről pedig elvesz egyet.
A táblázat további cellái hasonló módon értelmezhetők.
Természetesen, ha egy tranzíció nem csak egy tokent vesz el egy helyről, illetve ad hozzá helyhez, nem elég az előjeleket feltüntetni, a pontos értékeket is szerepeltetni kell a mátrixban.

A szomszédossági mátrix kitöltése nem túl bonyolult feladat, viszont időigényes, hiszen a mátrix mérete a háló méretének növekedésével rohamosan nő. Ugyanakkor rendkívüli figyelmet is igényel, hiszen a nagy számú mező kitöltése rengeteg hibázási lehetőséget tartogat.

Invariánsok a Petri-hálóban

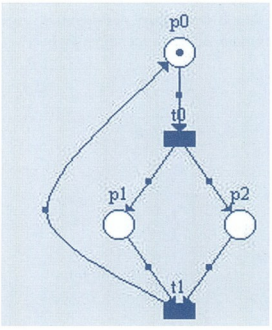
A Petri-hálók elemzésében nagy segítséget jelentenek az úgynevezett invariánsok, melyek a háló különböző jellemzőiből számíthatók. Kétféle invarianst különböztetünk meg: a T- *(tranzíció-)* és a P- *(place-)* invariánsokat.
(Az invariánsok olyan jellemzők a matematikában, amelyek nem változnak meg akkor sem, ha a kérdéses objektumot mindenféle kegyetlen, bár szabályos módosításnak vetem alá. Például a bűvös kockáról a megfelelő invariáns kiszámításával azonnal kideríthető, hogy szabályosan tekergetve keverték össze, vagy kifeszegették a sarokelemeit és elforgatva rakták vissza – a szerk.)
A T-invariáns olyan tüzelési szekvencia, amely végrehajtása nem változtatja meg a Petri-háló tokeneloszlását. A T-invariáns létezésével bizonyítható a rendszer ciklikus működése. Ha létezik T-invariáns, akkor a rendszer élő, és holtpontmentes.
A P-invariáns által kijelölt helyeken a tokenek súlyozott összege nem változik, azaz a tokenek egy részhalmaza *(vagy akár teljes halmaza)* a helyek egy *(rész)*halmazában kering. A P-invariánsok által meghatározott helyeken biztosan nem nő a tokenek száma *(a háló korlátos)*, és el sem veszik token *(a háló élő)*.
Ezen invariánsok kézzel is számolhatók a szomszédsági mátrixból, az úgynevezett Martinez-Silva algoritmussal. Ezt az algoritmust használja a DaNaMiCS is *(Calculate P-invariants illetve Calculate T-invariants)*:



Invariánsok számítása a DaNAMiCS segítségével

A Martinez-Silva algoritmus

Kiindulásként vegyünk fel egy mátrixot *(Q0')*, amely úgy néz ki, hogy leírjuk a szomszédossági mátrixot *(W)*, majd elé írunk egy megfelelő méretű *(a lift esetében egy 9*9-es)* egységmátrixot. Hogy egyszerűen és látványosan be is tudjam mutatni a működését, most egy kicsit félreteszem a liftes példát, helyette tekintsük az alábbi Petri-hálót:



Egyszerű példa a Martinez-Silva algoritmus bemutatására

Ennek a Petri-hálónak a szomszédsági mátrixa:

$$\begin{array}{c} \\ p0 \\ p1 \\ p1 \end{array} \begin{array}{c} \begin{array}{cc} t0 & t1 \end{array} \\ \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix} \end{array}$$

Ez alapján az algoritmus kiinduló mátrixa így írható fel:

$$Q_0 = \begin{bmatrix} 1 & 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 & -1 \end{bmatrix} \begin{array}{l} p0 \\ p1 \\ p2 \end{array}$$

Az algoritmus menete a következő: minden tranzícióra végrehajtunk egy iterációt, amelyben megvizsgáljuk az összes olyan place-párt, amelynek egyik tagja bemenő, másik pedig kimenő

helye az aktuális tranzíciónak. A t0 esetében például a p0-p1 és a p0-p2 párokat kell figyelembe venni, mégpedig úgy, hogy mátrixunkba új sorokat hozunk létre:

$$Q_1 = \begin{bmatrix} 1 & 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 & -1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} p0 \\ p1 \\ p2 \\ p0-p1 \\ p0-p2 \end{matrix}$$

Hogyan jönnek létre ezek az új sorok? Roppant egyszerűen: az adott place-eknek megfelelő eredeti sorokban szereplő értékeket mezőről mezőre összeadjuk. *(A fenti mátrixban például a felső két sor összege a bekeretezett 4. sor.)*
Ezeket az új sorokat felvesszük minden, a fenti feltételeknek megfelelő place-párra. Amikor ezzel készen vagyunk, a lépés során figyelembe vett place-ek sorait elhagyjuk a mátrixból:

$$Q_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} p0-p1 \\ p0-p2 \end{matrix}$$

Az algoritmust addig folytatjuk, amíg nem fogynak el az eredeti place-sorok, vagy amíg nem mentünk végig minden tranzíción.
Az algoritmus végén egyszerűen kiolvashatjuk a P-invariánsokat: p0-p1 és p0-p2.

E rövid kitérő után térjünk vissza a liftautomatához, és lássuk, mit is jelent pontosan a kapott eredmény? A P-invariáns egy oszlopvektor, amelynek minden sora *(eleme)* egy-egy helyhez tartozik, azaz:

$$\begin{matrix} Idle2 \\ Idle1 \\ Idle0 \\ 2emelet\_all \\ 1emelet\_all \\ Földszint\_all \\ 2emeleten \\ 1emeleten \\ Földszinten \end{matrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

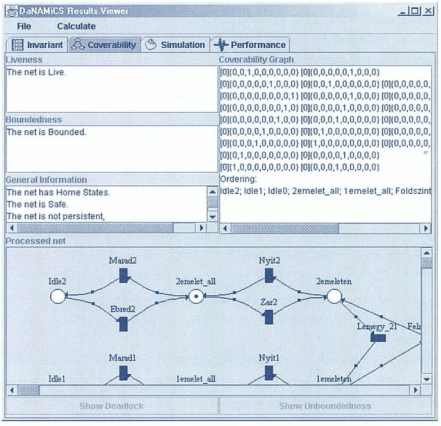
Ez azt jelenti, hogy minden egyetlen P-invariáns van, ebben minden place egyszeresen szerepel, azaz a keringő token sehol nem tűnik el, és sehol nem „osztódik".
A T-invariáns mezőben egy mátrixot találunk. Sorainak itt a tranzíciók felelnek meg, oszlopai pedig egy-egy T-invariánst jelölnek. A kapott mátrix a lift esetében így néz ki:

$$\begin{matrix} Marad2 \\ Ebred2 \\ Marad1 \\ Ebred1 \\ Marad0 \\ Ebred0 \\ Zar0 \\ Nyit0 \\ Zar1 \\ Nyit1 \\ Zar2 \\ Nyit2 \\ Lemegy\_21 \\ Felmegy\_12 \\ Lemegy\_10 \\ Felmegy\_01 \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Mit is jelent pontosan a fenti mátrix? Lássuk az első oszlopát: a Marad2 és az Ebred2 tranzíciót tartalmazza, azaz ha ezek a tranzíciók egymás után egy-egy alkalommal tüzelnek, az nem változtat a rendszer tokeneloszlásán. *(Természetesen itt a minimális T-invariánsokat számoljuk, hiszen ennél hosszabb szekvenciák is létezhetnek [pl. az Ebred0, Zar0, Nyit0, Marad0]*, de ezek mindig elemi T-invariánsokból állnak össze.

Elérhetőség

A DanaMiCS segítségével a Petri-háló egyéb tulajdonságai is könnyedén vizsgálhatók. Az *Analysis → Coverability Graph* menüpont segítségével *(vagy a már megnyitott Results Viewer megfelelő fülének kiválasztásával)* egy újabb listát kapunk a meghatározható jellemzőkről. A Calculate gombra kattintva egy ilyen ablak jelenik meg:



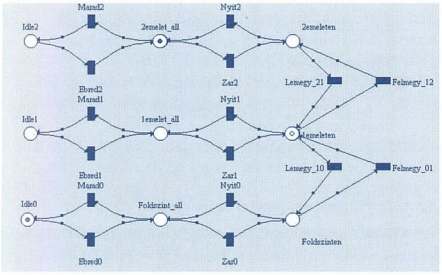
Elérhetőség vizsgálata a DaNAMiCS segítségével

A színezett Petri-hálók

A fent bemutatott analíziseken kívül egyéb vizsgálatokat is végezhetünk a DaNAMiCS segítségével, ezekre azonban most nem térnék ki. Ehelyett a színezett Petri-hálókról ejtenék néhány szót.

Eddig a tokeneket egyszerűen felvettük a hálóba, nem tettünk különbséget közöttük. *(Nem volt fontos, hogy melyik lift van az adott emeleten, az számított csupán, hogy jöjjön valamelyik lift.)* Ez azonban az esetek jelentős részében nem elég, szükségünk van arra, hogy különböző típusú tokeneket hozzunk létre, amelyek vagy egyes példányokat jelölnek, vagy egy-egy csoportot *(például egyfajta tokenek a lányokat, másmilyenek a fiúkat)*.

Több megoldás létezhet a tokenek megkülönböztetésére, például a különböző típusokat jelölhetnénk különböző alakú jelekkel *(például pöttyökkel, háromszögekkel, négyzetekkel, stb.)*, sokkal átláthatóbb és esztétikusabb megoldás azonban, ha a tokeneket színükkel különböztetjük meg egymástól. Az eddig egyformán fekete tokenek helyett színes tokeneket rajzolunk, és a különböző színű tokenek szolgálnak a megkülönböztetésre. Így például már azt is meg tudjuk mondani, hogy melyik lift tartózkodik egy-egy adott helyen, *nemcsak azt, hogy van-e ott lift (kétszínű nyomtatásban minden tehén fekete – a szerk.)*:



Az egyes liftek megkülönböztetése színezett tokenek segítségével

A fenti ábra például azt az állapotot *(tokeneloszlást)* mutatja, amikor a piros lift a földszinten van nyugalomban, a zöld az első emeleten mozog *(felfelé vagy lefelé halad, illetve épp most nyit majd ajtót, vagy most zárta azt)*, a kék pedig a második emeleten áll nyitott ajtókkal. Annak kezelése, hogy az egyes színek a valóságban mely lifteket jelölik, a mi felelősségünk *(ugyanúgy, ahogyan egy programkódban is nekünk kell tudnunk, hogy egy adott változó a valóságban minek felel meg)*.

Remélem, mindenki érdeklődését sikerült felkeltenem a Petri-hálók iránt. A piacon természetesen sok olyan eszköz található még, amely segíti a modellezést és az analizálást. A következő hónapokban már nem ezekkel foglalkozom *(bízom abban, hogy mindenkinek sikerült kellő kiindulást biztosítanom így is)*, hanem újabb vizekre evezem.

Molnár Ágnes
agi@harpia.homeip.net

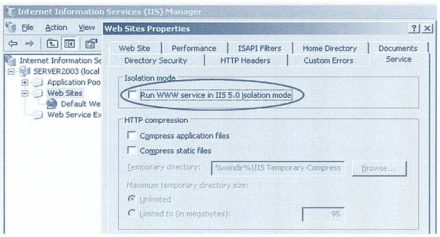
Internet Information Services 6.0

II. rész: A Windows 2003 Server webkiszolgálójának újdonságai

Az előző részben megismertük az IIS 6.0 webkiszolgálójának új belső felépítését. Most tovább ismerkedünk a webkiszolgálóval és terítékre kerül az FTP szolgáltatás is.

Az IIS 5 Isolation Mode

Az előző számban leírt, alapértelmezett Worker Process Isolation Mode mellett az IIS6 kompatibilitási okokból képes az úgynevezett „IIS 5 Isolation Mode"-ban is futni. A kiszolgáló illetve a webalkalmazások ilyenkor az IIS 5 felépítéséhez nagyon hasonlóan működnek.



Az IIS 5 Isolation Mode a webkiszolgálók gyökerén kapcsolható be, hatása a kiszolgáló minden virtuális webkiszolgálójára érvényes!

A beállítás módosítása után a webszolgáltatások újraindulnak, a felhasználói felületről pedig eltűnnek az Application Pool-ok, helyettük a webalkalmazásoknál az IIS5-ben már jók megszokott, és az előző részben részletesebben is leírt „Application Protection" mezőben állíthatjuk be a „Low *(IIS Process)*", „Medium *(Pooled)*", illetve „High *(Isolated)*" értékeket.

A http.sys és a WAS természetesen ebben az üzemmódban is megmarad, csak a w3wp.exe-k szerepét veszi át az inetinfo.exe és a dllhost.exe.

Összehasonlításul álljon itt egy táblázat a különböző funkciókról és az őket megvalósító komponensekről, az IIS különböző üzemmódjaiban:

Funkció	IIS 5.0 (Windows 2000)	IIS 6.0 (IIS 5.0 Isolation Mode)	IIS 6.0 (Worker Process Isolation Mode)
IIS metabase	Inetinfo.exe (bináris)	Inetinfo.exe (XML)	Inetinfo.exe (XML)
http.sys - konfiguráció	WAS	WAS	
Worker Process	-	-	W3wp.exe
WP menedzsment	-	-	WAS
In-Process ISAPI alk.	Inetinfo.exe	Inetinfo.exe	W3wp.exe
Out-of-Process ISAPI alk.	Dllhost.exe	Dllhost.exe	-
ISAPI szűrők	Inetinfo.exe	Inetinfo.exe	W3wp.exe
HTTP protokoll	Inetinfo.exe (winsock)	http.sys	http.sys
FTP, SMTP, NNTP	Inetinfo.exe	Inetinfo.exe	Inetinfo.exe

Az IIS6 alapértelmezett üzemmódját az dönti el, hogy a kiszolgálót hogyan telepítettük:

- Tiszta telepítésnél: Worker Process Isolation Mode
- Az IIS6 korábbi *(béta)* verziójának frissítésénél: megmarad a frissítés előtt beállított üzemmód
- IIS4, IIS5-ről történő frissítéskor: IIS 5 Isolation Mode

Az ASP.NET Process Model és az IIS6

Az ASP.NET programozók jól tudják, hogy az ASP.NET saját processzmodellt is tartalmaz. Amikor az ASP.NET a webkiszolgáló „IIS 5 Isolation Mode"-jában fut, akkor a saját processzmodelljét és beállításait használja *(a machine.config fájlból)*. Amikor azonban a webkiszolgáló az új Working Process Isolation Mode-ban fut, az ASP.NET letiltja a saját processzmodelljét és ő is aláveti magát a WAS akaratának.

Ez önmagában általában nem jelent túl nagy gondot, mert az ASP.NET processzmodellje sokban hasonlít az IIS6-ban megvalósítotthoz *(talán nem véletlenül)*. Azonban, a Working Process Isolation Mode két beállítás kivételével figyelmen kívül hagyja az ASP.NET machine.config fájl tartalmát, kivéve a maxIOThreads és a maxWorkerThreads értékeket. Minden más itt megadott beállítás érvénytelen marad, ezért szükség lehet arra, hogy azokat kézzel „másoljuk" át az IIS *(egészen pontosan a megfelelő Application Pool)* beállításai közé.

Az IIS 6.0 és a biztonság

Tanulva az elmúlt évek – sokszor nagyon is jogos – kritikáiból, végre az IIS programozói is csatlakoztak a „minden tilos, kivéve amit engedélyezünk" felfogáshoz. *(Emlékeztetőül: az IIS 5.0 még a „mindent szabad, kivéve amit tiltunk" elv jegyében működött, és ezt a hozzáállást használják ki a mai napig is az IIS-eken terjedő férgek és egyéb kártevők)*. Egy frissen telepített IIS 6.0 csak és kizárólag statikus tartalom *(.html oldalak, képek, egyebek)* kiszolgálására alkalmas, minden más extra szolgáltatást *(például az ASP-t, ASP.NET-et, WebDAV-ot,*

FrontPage Server Extensionst) a telepítés után külön kell engedélyeznünk! Amíg ezt meg nem tesszük, az IIS az ilyen jellegű erőforrásokra utaló kérésekre HTTP 404-es *(Not Found)* hibát küld vissza, nagyon helyesen.
Az engedélyezhető webkiszolgáló-bővítményeket *(például az ASP motor is ez)*, az IIS Managerben, a Web Service Extensions alatt találjuk:

Vasszigor: a bővítmények alapértelmezett beállításai

Láthatjuk, hogy a telepítés után rendelkezésre álló bővítmények mindegyike tiltott *(Prohibited)*. Lássuk, melyik mire is való:

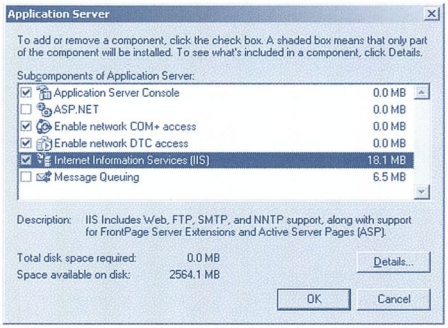
- All Unknown ISAPI Extensions: minden olyan ISAPI bővítmény *(azaz bizonyos típusú erőforrásokhoz rendelt „motor", általában .dll)*, ami a listában külön nem szerepel
- All Unknown CGI Extensions: minden olyan külső, webkiszolgálóból futtatható alkalmazás *(.exe)*, ami a listában külön nem szerepel.

A fenti két opció általános beállítás, a következő négy azonban már konkrét telepített és engedélyezhető bővítményt takar. Ezeknél a tiltás/engedélyezés mellett egy tulajdonságlapot is találunk, ahol megtekinthetjük a bővítményhez tartozó erőforrásokat *(.dll-eket)*, és akár egyenként is engedélyezhetjük vagy tilthatjuk őket!

- **Active Server Pages:** .asp alkalmazások futtatásához szükséges bővítmény *(igényli például az Exchange 2000 és a Certificate Services)* – asp.dll
- **WebDAV:** a kiszolgálón megosztott webes mappák írható-olvasható hozzáféréséhez használt protokoll – httpext.dll
- **Server Side Includes:** Kiszolgálóoldali SSI direktívák *(#exec, #config, stb.)* végrehajtását vezérlő bővítmény – ssinc.dll. Hacsak nincs rá kifejezetten szükségünk *(általában .shtm illetve .shtml fájlok kezeléséhez kell)*, hagyjuk tiltva! A SSI funkciókat ASP segítségével 100%-ban meg lehet valósítani.
- Internet Data Connector: .idc fájlok végrehajtását végző komponens, webes ODBC adatbázis-hozzáféréshez – httpodbc.dll. Funkcióját az ASP-ben használt adatbáziskezelő komponensek már szinte teljesen kiváltották. Kompatibilitási okokból maradt bennt, ha nincs rá szükségünk, ne engedélyezzük!

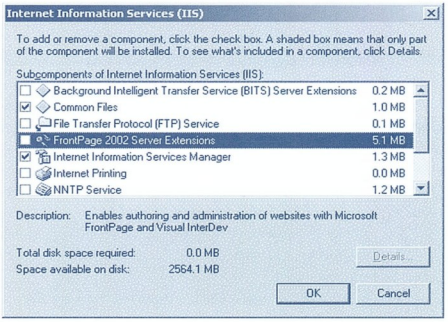
További komponensek telepítése

Feltűnhetett, hogy a fenti listából sok minden hiányzik. Nincs ott például a Microsoft egyik legfontosabb zászlóshajója, az ASP.NET, de a FrontPage Server Extensions bővítményeket sem látjuk sehol. Ezeket külön kell telepítenünk, mielőtt engedélyezhetnénk őket. Ehhez indítsuk el a Control Panel → Add or Remove Programs → Add/Remove Windows Components varázslót!
Az Internet Information Services-t és a kapcsolódó szolgáltatásokat az „Application Server" csoportban találjuk:



Az Internet Information Services az Application Server szolgáltatáscsoport része lett. Ugyanitt kapcsolható be az ASP.NET-támogatás is

Az ASP.NET telepítéséhez tegyünk mellé pipát; az IIS komponenseinek *(például a FrontPage Server Extensions)* kiválasztásához pedig jelöljük ki az IIS-t és kattintsunk a „Details" gombra:



Az Internet Information Services telepíthető összetevői

Miután az ASP.NET és a FrontPage 2002 Server Extensions feltelepült, az IIS konzolban a webkiszolgáló bővítményei között *(rögtön engedélyezve)* megtaláljuk majd:

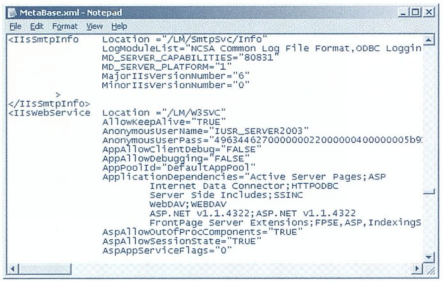
- ASP.NET v1.1.4322 – aspnet_isapi.dll
- FrontPage Server Extensions 2002 – admin.dll, fpadmdll.dll, author.dll, fpcount.exe, shtml.dll

Természetesen a beállítások között minden fájlnév a teljes elérési utat tartalmazza, azaz a FrontPage Server Extensions engedélyezése nem jelenti azt, hogy bármilyen más admin.dll-t is engedélyeznénk!

XML MetaBase

A MetaBase az Internet Information Services saját konfigurációs adatbázisa. Ez az adatbázis tárol minden olyan információt, ami az IIS működéséhez szükséges *(az adminisztratív esz-*

közökkel is többnyire ebben szerkesztgetünk, az IIS beállításainak mentése-visszaállítása pedig gyakorlatilag a MetaBase mentését is visszaállítását jelenti). IIS5-ig ez az adatbázis egy speciális felépítésű bináris fájl volt *(metabase.bin)*; szerkesztéséhez külön eszközöket *(MetaEdit)* és scripteket kaptunk. Ilyen scripteket egyébként magunk is írhatunk.



Az IIS6 MetaBase részlete

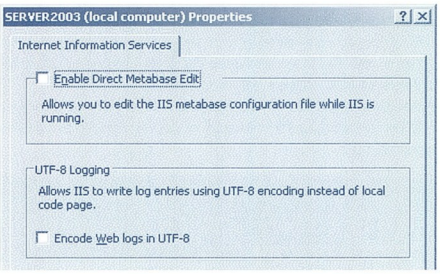
Az IIS6 MetaBase legfőbb újdonsága, hogy az adatbázis immár nem bináris, hanem XML formátumban található a lemezen. Ennek legnagyobb előnye, hogy ember által olvasható *(már amennyire egy hatvan kilobájtos XML dokumentum az)*, és nincs akadálya annak, hogy a beállításokat akár egy Notepad segítségével módosítsuk a MetaBase belsejében. A fájlt egyébként a

```
\WINDOWS\System32\inetsrv\MetaBase.xml
```

útvonalon találjuk meg. Az inetinfo.exe ezt minden indulásakor a memóriába tölti és onnan dolgozik az adatokkal. Éppen ezért, ha a beállításokon valamelyik grafikus eszközzel módosítunk, az először csak a memóriában történik meg, a lemezre pedig csak a kiszolgáló leállásakor kerül. Ha egy opciót szeretnénk rögtön a beállítás után a MetaBase.xml fájlban látni, az IIS Manager-ben kattintsunk jobb gombbal a webkiszolgáló nevére, majd a menüből válasszuk az All Tasks → Save Configuration to Disk parancsot!
Ha a kiszolgáló tulajdonságai között engedélyezzük az „Enable Direct Metabase Edit" opciót, az IIS futás közben észleli a MetaBase esetleges változtatásait és azonnal újra betölti azt. Az új MetaBase egyébként továbbra is használható minden olyan eszközzel, scripttel vagy technológiával *(például ADSI vagy WMI)*, amivel a korábbi változatok. Az alapértelmezett beállítás szerint minden változtatás előtt a MetaBase-ből készül egy biztonsági másolat a

```
\WINDOWS\system32\inetsrv\History\
```

mappába, így a webkiszolgáló bármelyik állapota könnyen visszaállítható.



A webkiszolgáló globális beállításai

Webnaplók UTF-8 formátumban

Ugyanitt, a webkiszolgáló globális tulajdonságai között kapcsolhatjuk be a webnaplók UTF-8 formátumát. Ez a formátum képes a webkiszolgálóhoz beérkező Unicode kódolású kérések naplózását is. Ennek a funkciónak nálunk leginkább csak hibakeresési szempontjai lesznek, de a távol-keleti világban ennek nagyobb jelentősége lesz.

Az FTP kiszolgáló újdonságai

Az újrakezdhető le- és feltöltések mellett *(ami már a Windows 2000-ben megjelent)*, az FTP szolgáltatásnak két újdonsága van *(halkan jegyzem meg, hogy volt, ahol ezek az „újdonságok" évtizedekkel ezelőtt természetesen voltak…)*:

- Server-to-Server FTP: adatmásolás két FTP kiszolgáló között úgy, hogy az adat az ügyfélhez nem jut el, csak a kiszolgálók között mozog
- FTP User Isolation: a felhasználók gyökérkönyvtárainak beállítása után a felhasználó azt semmilyen módon nem hagyhatja el *(a saját könyvtárát látja az FTP kiszolgáló gyökérkönyvtárának)*.

Server-to-Server FTP

A szolgáltatás engedélyezéséhez két registry-értéket kell beállítanunk:

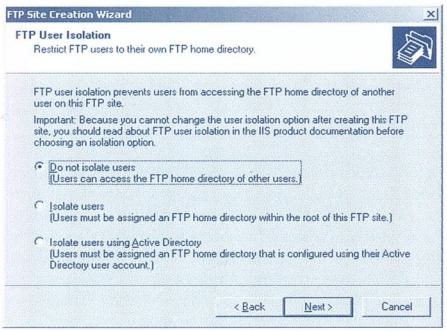
```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
Services\MSFTPSVC\Parameters\
        EnableDataConnTo3rdIP=1
        EnablePasvConnFrom3rdIP=1
```

Ezután az FTP kiszolgáló engedélyezni fogja a kiszolgálók közötti fájlátvitelt. Nagyon fontos, hogy az EnablePasvConnFrom3rdIP engedélyezésével biztonsági rést ütünk a pajzson, ezért internetes környezetben ennek beállítása nem javasolt.

FTP User Isolation

A felhasználónként beállítható gyökérkönyvtár régóta jogos igény a Windows FTP szolgáltatásában. Eddig *(valamint az IIS6-ban az alapértelmezett beállítások mellett)* legfeljebb annyit tudtunk elérni, hogy az FTP gyökérkönyvtárában *(\InetPub\FTPRoot)* létrehoztunk egy, a felhasználó nevével megegyező mappát *(vagy virtuális könyvtárat)*. Ekkor, ha a felhasználó bejelentkezett, alapértelmezésben a nevének megfelelő könyvtárba került *(anonymous felhasználóhoz értelemszerűen az anonymous nevű könyvtárat kellett létrehozni)*. Mivel azonban a felhasználónak a valódi gyökérkönyvtárhoz *(azaz a saját könyvtárának szülőkönyvtárához)* legalább browse joggal kell

rendelkeznie, a cd.. parancs segítségével a saját mappáját elhagyva kiléphetett oda.
Az FTP User Isolation Mode ezt segít elkerülni. Ha bekapcsoljuk, a felhasználó a saját könyvtárát az FTP kiszolgáló „gyökerének" látja, azt tehát nem tudja elhagyni. Az FTP User Isolation Mode virtuális FTP kiszolgálónként kapcsolható be *(vállalkozó kedvű olvasóink máris elkezdhetik keresgélni a MetaBase-ben az IisFtpServer elem UserIsolationMode attribútumát)*. Nem biztos, hogy elsőre sikerül megtalálni, ugyanis az üzemmód csak a virtuális FTP kiszolgáló létrehozásakor határozható meg, és annak későbbi módosítására sincs mód *(a Default FTP Site pedig nem támogatja az FTP User Isolation módot)*.
A dolog kipróbálásához tehát egy új virtuális FTP kiszolgálót kell létrehoznunk. Az IIS Manager-ben az FTP Sites-re kattintva válasszuk a New… → FTP Site… parancsot, adjunk meg egy nevet, majd válasszuk ki a használni kívánt IP-címet és portot. Ezután a következő beállítópanellel találkozunk:



Az FTP User Isolation beállítás csak a virtuális FTP kiszolgáló létrehozásakor határozható meg

Itt három lehetőségünk van:

- **Do not isolate users:** a felhasználókat nem izoláljuk *(az NTFS jogosultságok ettől függetlenül természetesen érvényesek!!!)*
- **Isolate users:** a helyi és tartományi felhasználók izolálása
- **Isolate users using Active Directory:** a helyi és tartományi felhasználók izolálása az Active Directory-ban tárolt információk segítségével. Ekkor meg kell még adnunk egy felhasználónevet, aminek segítségével az IIS az Active Directory-ban keresgélni tud majd, valamint egy alapértelmezett tartománynevet arra az esetre, ha a felhasználó tartománynév nélkül jelentkezne be.

Felhasználók izolálása Active Directory nélkül

Ha nincs Active Directory, a felhasználókat egyszerű könyvtárstruktúra segítségével irányítgatjuk.

```
C:\INETPUB\FTPROOT
+---Falatrax2003
|   +---teszt
+---LocalUser
|   +---public
|   +---teszt
```

Az FTPRoot alatti könyvtárstruktúra Active Directory nélküli FTP User Isolation esetén

A teendő az, hogy az FTPRoot alatt hozzuk létre a következő alkönyvtárakat:

- **LocalUser:** a helyi felhasználói könyvtárainak gyökere
- **LocalUser\public:** az anonymous FTP felhasználó könyvtára
- **LocalUser\<felhasználó>:** a helyi <felhasználó> nevű felhasználó könyvtára *(itt: teszt)*
- **<domainnév>:** a <domainnév> tartomány felhasználói könyvtárainak gyökere *(a tartomány neve ebben a példában FALATRAX2003 volt)*
- **<domainnév>\<felhasználó>:** a <domainnév> tartomány <felhasználó> felhasználójának könyvtára *(itt FALATRAX2003\teszt)*

Természetesen a listába akármennyi tartományt felvehetünk. Figyeljünk arra hogy ha – bár biztonsági szempontból nem szerencsés – az FTP kiszolgáló tartományvezérlőn fut, akkor mindenképpen a <domainnév>\<felhasználó> formátumot kell használnunk.

A felhasználók izolálása az Active Directory segítségével

Ebben az üzemmódban a felhasználó gyökérkönyvtárát a felhasználó Active Directory-objektumában tárolt adatok határozzák meg. Ehhez a Windows 2003 Active Directory-sémában a felhasználói fiók objektuma két új jellemzőt kapott: ezek az msIIS-FTPRoot és az msIIS-FTPDir. A mezők jelentése:

- **FTPRoot:** a felhasználó könyvtárának elérési útja *(lehet UNC útvonal is)*
- **FTPDir:** a felhasználó könyvtárának neve

A könyvtár teljes nevét az FTPRoot és az FTPDir mezők „összeolvasása" adja. Az alapértelmezett rendszergazdai felületen ezek a mezők nem láthatók, de az IIS-hez adott iisftp.vbs script segítségével lekérdezhetők illetve beállíthatók. Példák a mezők beállítására és lekérdezésére:

```
C:\>iisftp /setADProp teszt FTPRoot
\\server2003\users
The value of FTPRoot for user teszt has been set
to \\server2003\users

C:\>iisftp /setADProp teszt FTPDir teszt
The value of FTPDir for user teszt has been set
to teszt

C:\>iisftp /getADProp teszt FTPRoot
The value of FTPRoot for user teszt is:
\\server2003\users

C:\>iisftp /getADProp teszt FTPDir
The value of FTPDir for user teszt is:
teszt
```

…folytatjuk!

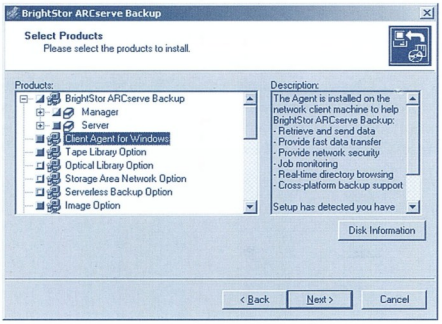
Fülöp Miklós
mick@netacademia.net

BrightStor ARCserve v9 for Windows

Foglalt állományok és alkalmazásszerverek mentése

Az adatmentés teljességének feltétele a mentendő adatok állományszintű változatlansága. Ez a feltétel feloldandó érdekütközéshez vezet a gazdaalkalmazás és mentőmegoldás között.

A Computer Associates Brightstor ARCserve nevű adatmentő és -helyreállító szoftverterméke a rendhagyó számozású 2000-es után a 9-es verzióval tart. Néhány kevésbé elterjedt operációs rendszertől és alkalmazástól eltekintve tetszőleges, az eredeti gyártó által még támogatott operációs rendszer illetve alkalmazáskiszolgáló adatainak mentésére, helyreállítására alkalmas.
A BrightStor ARCserve Backup v9 for Windows, WindowsNT/2000/XP/2003-környezetben telepíthető: a telepítés egylépéses, gyors, kevés felhasználói beavatkozást igényel.
A szerver- és kliensoldali komponensek a megfelelő jogosultságok birtokában a helyi és távoli gép*(ek)*re is telepíthetők. A telepítőprogram felismeri a telepíthető összetevőket, és ajánlatot tesz a telepítésükre.



Telepítés egy lépésben

A mentési rendszer

Egy mentési rendszer nélkülözhetetlen összetevői a következők:

- adatforrás
- feladatütemező és –kiosztó
- adattároló.

Bár a feladatütemező és- kiosztó szerepe nagyon fontos a mentési rendszerben, ez a cikk csak az adatforgalom két végével, az adatforrással és adattárolóval foglalkozik.
Hálózati környezetben az adatforrást kétféleképpen érhetjük el. A mentőszerver gazdagépe által ismert valamennyi állománymegosztási protokollt *(pl. SMB, NFS, NCP(IPX/SPX))* felhasználhatjuk távoli gépen levő, de a mentőszerveren helyi állományként látszó állomány mentésére.
A javasolt megoldásnál célalkalmazást, ún. mentőügynököt telepítünk a mentendő adatokat szolgáltató számítógépre.

A mentőügynök

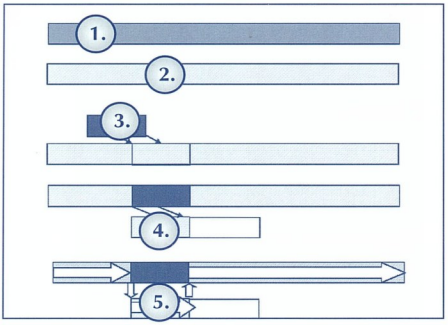
A mentőügynök, fajtájától függően alkalmas lehet a rendszer- és felhasználói állományok, vagy a támogatott alkalmazásszerverek állományainak on-line, azaz üzem közbeni mentésére. Az on-line mentés során a mentendő szerver átmeneti teljesítménycsökkenés árán, de folyamatosan kiszolgálja a felhasználói igényeket is. Az operációs-rendszert mentő ügynök *(**U**niversal **C**lient **A**gent)* számos feladatot lát el: fellapozza a gazdagép meghajtóit/köteteit és állományrendszerét, rendszerleíró állományait az esetleges fürtözést leíró állományokkal együtt, és ha van, az Active Directoryt, illetve Netware-mentés esetében a Novell címtárat.
Ha telepítettük az egylépéses rendszervisszaállítási opciót *(**D**isaster **R**ecovery **O**ption)* is, a hálózati beállításokat és a hálózatikártya-meghajtót, továbbá a partíciós adatokat is összegyűjti. A mentésre kijelölt állományokat valós időben tömörítve küldi el a mentőszervernek: ezzel a mentendő adatok tömöríthetőségével arányosan időt és sávszélességet takarít meg.

Foglalt állományok

A mentőrendszer tervezésénél kiválasztjuk azt az időszakot *(Backup Window)*, amelyben a mentendő rendszer terheletlen. Ebben az időtartományban a rendszererőforrások szinte korlátozás nélkül használhatók mentési célokra, a mentendő állományok a mentés szempontjából ideális, nyugalmi állapotban vannak.
A mentés teljességének feltétele a mentendő adatok állományszintű változatlansága, azaz annak szavatolása, hogy egy adott állomány a mentés sikeres befejezéséig nem fog megváltozni.
A folyamatos üzemű számítógépes környezetben lerövidül, sőt adott esetben teljesen el is tűnik a nyugalmi időszak, legfeljebb a terhelés napszak-, vagy más jellemzőtől függő átmeneti csökkenéséről beszélhetünk. Ilyen körülmények között nem szavatolható a mentendő állományok változatlansága, tehát a gazda- és a mentőalkalmazás érdekei ütköznek.
Az ütközést az ú.n. nyitottállomány-ügynök *(**O**pen **F**ile **A**gent, OFA)* oldja fel.
Az OFA rendszermag-szinten működik, a mentőszoftver felől érkező kéréseket szolgálja ki. Az OFA mindaddig passzív, amíg egy bizonyos állomány mentésére nem érkezik kérés, és csak

akkor éled fel, ha a mentendő állomány a mentés kezdetének pillanatában foglalt. Ekkor elindul egy időzítő, amely egy olyan öt másodperces intervallumot keres, amelyben az állomány állományrendszerbeli megjelenítése változatlan. Előfordulhat, hogy a memóriában időközben már megváltozott az állomány, de ezt a változást nem lehet menteni, emiatt figyelmen kívül hagyja. Ha záros határidőn belül sikerül egy nyugalmi periódust találnia, „pillanatképet" készít az állományról. Ez a „pillanatkép" más hasonló megoldásoktól eltérően nem az eredeti állomány másolatát jelenti, hanem azt a rögzítési tevékenységet, melynek során a mentés ideje alatt megváltoztatandó állományrészek eredeti tartalma egy átmeneti pufferbe, az ún. Preview File-ba kerül. Amikor a mentési folyamat eléri az időközben megváltozott részt, az OFA az eredeti adatokat küldi a pufferből.

Azokat az alkalmazáskiszolgálókat, amelyekhez a BrightStor ARCserve v9 nem rendelkezik külön mentőügynökkel, szintén a nyitottállomány-ügynökkel menthetjük. Ha ezek az alkalmazások több változó tartalmú állományból állnak, valamennyi érintett állományt egyidejűleg kell elmenteni. Ezt a feltételt az OFA úgy teljesíti, hogy az alkalmazáshoz tartozó állománycsoportra végez foglaltságvizsgálatot, és minden egyes állomány változásait rögzíti.



A nyitott állományok mentése

(1) Az állomány a mentés kezdetén nyitott
(2) Az OFA nyugalmi állapotra vár, majd „fényképez"
(3) Az állomány egy része megváltozik
(4) Az OFA a változás előtti állapotot eltárolja
(5) A mentés eredeti állapot szerint történik

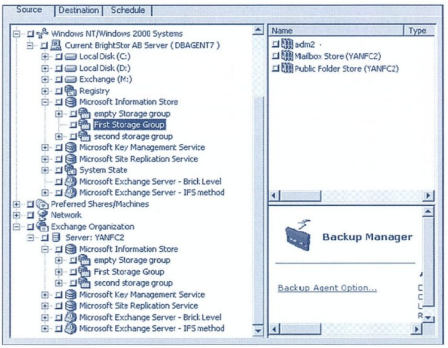
Microsoft Exchange-kiszolgáló mentése

A Microsoft Exchange-hez kifejlesztett mentőügynök alapfeladata az Exchange levelező-adatbázis *(Information Store)* valósidejű mentése, illetve Exchange 2000 esetében még a Key Management Service- és a Site Replication Service-adatbázis, Exchange 5.5 esetében pedig a címtár mentése is. Natív Windows 2000-es környezetben az Active Directoryt a rendszermentő ügynökkel mentjük.

Az Exchange-ügynök további szolgáltatása a felhasználói postafiókok és a publikus mappákban található levelek, névjegyek, feljegyzések, találkozók stb. mentése *(Brick Level Backup)*. A két mentési eljárás kiegészíti egymást: a teljes levelezőszervert adatbázismentésből, az egyes postafiókok tartalmát pedig postafiókmentésből lehet helyreállítani.

Amikor a BrightStor ARCserve Backup hozzáfog az adatbázis, vagy annak egy részének mentéséhez, kérést küld a mentőügynöknek. A mentőügynök a kért adatokat az Exchange Server Backup/Restore API-csatolón keresztül éri el.

Az Exchange-kiszolgáló mentése lehet teljes, vagy különbségi *(inkrementális vagy differenciális)*. Mindhárom módon ugyanazt az eredményt kapjuk, az eltérés a helyreállítás bonyolultságában van.



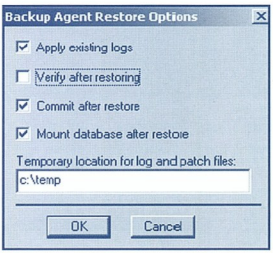
Exchange-kiszolgáló mentése

Ha a mentés sebessége az elsődleges, akkor valamelyik különbségi módszert, ha pedig a helyreállítás sebessége a fontosabb, a teljes mentést célszerű választani.

Az adatbázis helyreállítását követően konzisztencia-vizsgálatot kell végeznünk. Erre elsősorban az Exchange-kiszolgálóhoz adott `eseutil.exe` szolgál,

```
eseutil /mh dbase-file.ext | find /i "consistent"
```

de alapszintű vizsgálatra az Exchange-ügynök is képes.



Az Exchange-ügynök beállítási lehetőségei

Az Exchange-kiszolgáló biztonsági mentésének legcélszerűbb módja az, amikor a nyugalmi állapotú kiszolgálóról, leállított Exchange-szolgáltatások mellett teljes mentést készítünk, majd a továbbiakban csak on-line mentéseket végzünk. A nyugalmi állapotban készített teljes mentésből gyorsan helyreállítható egy működőképes rendszer, ezt követően már csak az aktuális adatbázisállapotot kell visszatölteni. Az adatbázis visszaállítása során a levelezőszerver felhasználókat nem szolgál ki.

A postafiókok mentése MAPI-csatolón keresztül történik. Ezt a csatolót az Exchange-kliens vagy a Microsoft Outlook tetszőleges változata tartalmazza.

Az egyes postafiókok tartalmának helyreállításához az Exchange-kiszolgáló normál üzeme szükséges, a felhasználó levelezőkliensében a visszatöltést követően azonnal megjelennek a mentés óta törölt levelek. Természetesen több szűrési felté-

telből is választhatunk a levelek visszatöltése kapcsán.

Egylépéses rendszervisszaállítás *(Disaster Recovery)*

A működőképes rendszer gyors helyreállításának feltétele a lehető legkevesebb kézi beavatkozás a rendszer visszatöltése során.
A hagyományos adatvisszaállítás rendszerépítéssel kezdődik. Ennek során:

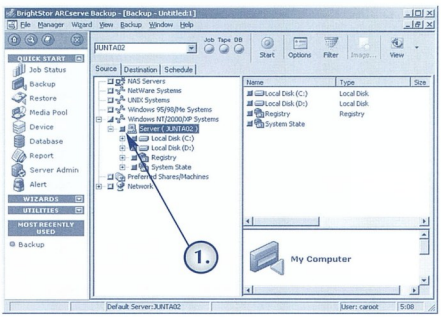
- operációs rendszert
- operációs rendszer-javítást
- eszközmeghajtót
- eszközmeghajtó-javítást
- alkalmazásszervert
- alkalmazásszerver-javítást

telepítünk.

Szükséges kellékek: a telepítendő összetevők telepítőkészletei. Mindeközben a rendszeren legalább az alábbiakat kell beállítanunk:

- partíciós információk
- hálózati beállítások
- az alkalmazásszerverek beállításai.

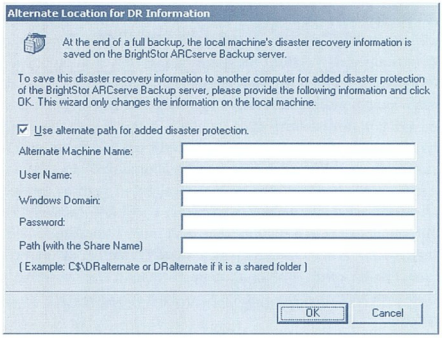
Szükséges kellékek: a beállítások aktuális változata, nyomtatott és/vagy elektronikus formában.
Ezt követően láthatunk hozzá a felhasználói adatok és az alkalmazásszerverek adatainak helyreállításához.
A mentőügynökkel készített teljes rendszermentésünk azáltal válik alkalmassá az egylépéses rendszervisszaállításra, hogy legyártunk egy indítókészletet.
Az indítókészlet tartalmaz egy floppy-lemezt a mentett gép egyedi beállításaival, egy módosított operációs-rendszer CD-t, és a teljes szalagos mentést.



Adatforrás kijelölése mentéshez

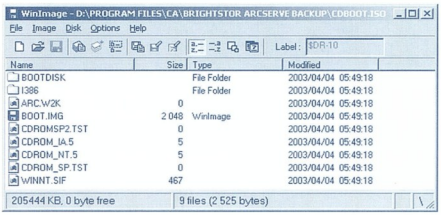
(1) A mentett gép egyedi beállításait a rendszermentőügynök gyűjti össze, ha a teljes gépet kijelöljük mentésre.

A floppy-lemezre a partícionálási adatok és a hálózati beállítások kerülnek, valamint ide másolódik a hálózati kártya meghajtója is.
Az egyes mentett számítógépek beállításait az ARCserve-kiszolgáló tárolja. A nagyobb biztonság érdekében ezekről az adatokról másolat készíthető egy kiválasztott kiszolgálón.



A konfigurációs adatok biztonsági másolata

A rendszerindító CD-ből mentett operációs rendszer-fajtánként egy darabra van szükség, ugyanazzal a CD-vel tudunk egy Windows 2000 munkaállomást és egy szervert visszaállítani.
Ezt a CD-t a BrightStore ARCserve 9 állítja elő ISO-állomány alakjában az eredeti operációs rendszer telepítőkészletének i386-os könyvtárából és az ARCserve állományokból.



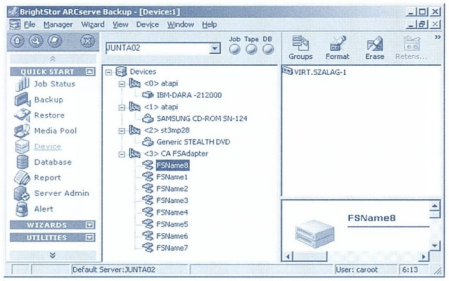
A rendszerindító CD tartalma

A gyakorlatban előfordulhat olyan eset, hogy a rendszert hálózaton keresztül mentjük, de a visszaállítás helyi szalagos egységgel történik. Ilyen esetben a rendszervisszatöltés során a számítógép úgy viselkedik, mintha ARCserve-kiszolgáló lenne. Természetesen van mód a hálózati adatvisszatöltésre is, ehhez a floppy-lemezen található hálózati meghajtót és -beállításokat használjuk.
Ha a szalagos egységünk támogatja a szalagról történő rendszerindítást *(**O**ne **B**utton **D**isaster **R**ecovery)*, kizárólag a teljes mentést tartalmazó szalagból is visszaállítható a teljes rendszer. Az OBDR kétségtelen előnye az egyszerűség, ám a szalagos rendszerindítás sajátosságaiból *(az állományok nem okvetlenül a rendszerindításhoz szükséges sorrendben helyezkednek el a szalagon)* adódóan az első rendszerindulás NAGYON lassú, a szalagos egység sebességének függvényében órákig is eltarthat.

Virtuális szalag: merevlemez-alapú mentés

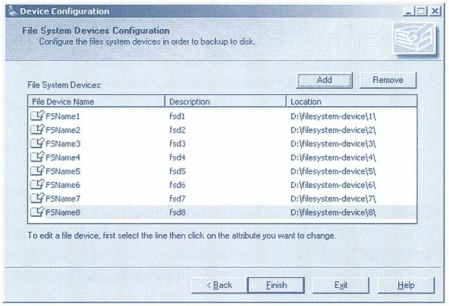
A szalagos mentési rendszerek szűk keresztmetszete rendszerint a szalagos egység teljesítménye. A mentési munkákat be kell sorolni. A szekvenciális írás hosszú nyugalmi időt igényelt, a mentési folyamat hosszasan leterhelte a helyi hálózatot.
A folyamatos üzemű számítógépes rendszereknél nincsen mód a mentendő kiszolgálók tartós leterhelésére mentési célokból; a lehető leggyorsabban el kell vinni a mentés adatait, hogy a kiszolgáló visszatérhessen az alapfeladatához.

A BrightStor ARCserve mentési teljesítménye elméletileg tetszőlegesen növelhető a mentőkiszolgálóra csatlakoztatott szalagos egységek számának növelésével, de a gyakorlatban a nagyszámú fizikai mentőeszköz egyetlen kiszolgálóra történő csatlakoztatása nehezen valósítható meg.
A merevlemezek árának rohamos zuhanása oda vezetett, hogy a merevlemez-alapú mentés ésszerű választási lehetőséggé vált.
Az ARCserve a lemeztárat virtuális szalagnak tekinti,



Az ARCserve eszközkezelője

annyi eszközre tud párhuzamosan írni, ahány eszközt létrehoztunk.

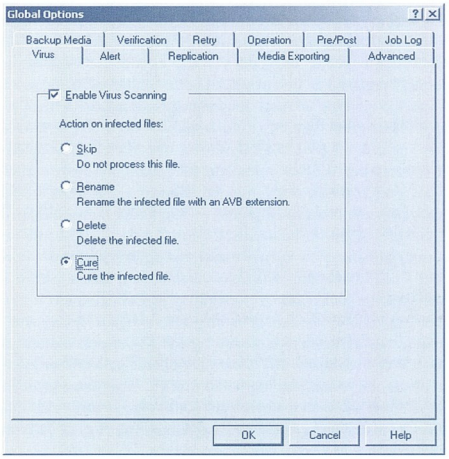


A virtuális szalag létrehozása egyszerű

Az így létrehozott virtuális szalagos meghajtók megfelelő méretezés és hálózati kapacitás mellett alkalmasak arra, hogy az alkalmazásszerverek adatait nagyon gyorsan befogadják, és később ugyanezeket a mentéseket a hagyományos szalagra másolják.
A merevlemez-alapú mentés legfőbb erénye az azonnali adathozzáférés. Ez az előny azokban az esetekben a legszembetűnőbb, amikor egy-egy viszonylag kisméretű, de fontos állományt kell visszaállítani. Szalagos megoldásnál a hozzáférés idejének túlnyomó részét az adott szalag fizikai megkeresése és előrecsévélése emészti fel.

Vírusvédelem

A számítógépes rendszerben számos ponton célszerű ellenőrizni az adatok vírusmentességét, az egyik ilyen pont a mentési rendszer. Ha ezen a helyen is ellenőrizzük az adatokat, nem fordulhat elő, hogy a vírusfertőzést idegen helyről származó szalag okozza, és elkerülhető az egyszer már kiirtott vírus véletlen visszafertőzése egy régebbi mentésből történő adatvisszaállítás során.
A BrightStor ARCserve v9 telepítőkészlete tartalmazza az eTrust InoculateIT 6 engine-t, amely az azonos nevű víruskereső lecsupaszított központi eleme. A mentés illetve visszaállítás során a víruskeresés opcionális, feladatonként állítható.



A vírusdefiníciós állomány önműködő frissítése az alábbi parancssorral történhet,

```
C:\Program files\Common
↳ Files\CA\ScanEngine\inodist.exe /cfg C:\Program
↳ files\Common Files\CA\ScanEngine\inodist.ini
```

akár ARCserve-feladatként, akár külső időzítéssel.

Jung Tamás
tamas.jung@ca.com
A szerző rendszermérnök, a Computer Associates tanácsadója
MCSE, CUE, CSS, CNS

A DHCP rejtett szépségei – I.

Szinte minden TCP/IP hálózat rendelkezik DHCP szolgáltatással. Ez az alkalmazás háttérbe húzódik, a felhasználók sohasem találkoznak vele, és ha jól működik, a rendszergazdák is csak hébe-hóba vetnek rá pillantást. Itt is érvényes az ókori bölcsesség: ami nem látható, az fontosabb, mint ami látható.

Az Internet elterjedése technikai értelemben a TCP/IP protokollcsalád széles körű alkalmazását jelenti. Szemben azonban más protokollokkal, amelyek nem igényelnek különösebb konfigurálást, a TCP/IP negyedik verziója csak akkor nyeri el robosztusságát, ha az üzemeltetők precíz beállításokat végeznek rajta. Ha igen sok állomás dolgozik egy hálózatban, a beállítások sokszorosan terhelik a rendszergazdát. Kézzel kellene azokat végezni, méghozzá a helyszínen, hiszen csak a konfiguráció után lehet hálózati kapcsolatot létrehozni, másrészt a szabvány megköveteli, hogy minden állomás egyedi IP-címmel rendelkezzék – ezt csak precíz és naprakész nyilvántartással lehet biztosítani. Az adatbázist folyton frissíteni kellene amikor egy új gép érkezik, vagy egy eszközt az egyik hálózatból a másikba kell költöztetni. A problémahalmazra a válasz a DHCP szolgáltatás kialakítása. A Dynamic Host Configuration Protocol, vagyis a dinamikus címkonfigurációs eljárás képes automatikusan az ügyfelek számára egyedi IP-címeket osztani. A DHCP egy szabvány *(RFC)*, bármely operációs rendszer használhatja, ha ismeri. Ma már talán nincs is olyan, amelyiket ne készítettek volna fel a címek automatikus kezelésére. Akár egy Windows for Workgroups 3.11, egy Linux, vagy egy OS/2 is lehet ügyfél. A kliensek viselkedése különböző helyzetekben és konfigurációs igényei eltérőek lehetnek, ahogy ezt később majd látni fogjuk.

DHCP alapok

A DHCP-szabvány megértéséhez néhány TCP/IP alapfogalmat precízen ismerni kell. Ezek a következők: IP-cím, alhálózati maszk, alapértelmezett átjáró, szórt üzenet *(broadcast)*, útválasztó *(router)*. Lakonikus tömörséggel nézzük, mi mit jelent.

IP-cím: négy számjegyből álló, az adott állomást a hálózaton egyértelműen azonosító egyedi cím. A cím két részből áll: a hálózati címből és az állomás címéből. Ránézésre nem állapítható meg, hogy hol ér véget az egyik, és hol kezdődik a másik. Pl.: 172.16.3.14

Alhálózati maszk: az a szám, amely meghatározza, hogy az IP-cím mely része hálózati, és mely része állomáscím. Ennek segítségével lehet megállapítani egy másik IP-címről, hogy az a mi hálózatunkba tartozik-e vagy sem. Pl: 255.255.0.0

Alapértelmezett átjáró: ha egy olyan címre kell csomagokat küldeni, amely nem a mi hálózatunkban van, az állomások az alapértelmezett átjáró felé továbbítják az adatokat. A valóságban az alapértelemzett átjárók többnyire útválasztók *(routerek)*.

Szórt üzenet *(broadcast)*: egy speciális csomag, amelyet minden állomás megkap, és fel is dolgoz. A szórt üzenetek nem jutnak át az útválasztón, mivel általában lokális jelentőségük van. Ethernet hálózaton a szórt üzenet MAC célcíme FF-FF-FF-FF-FF-FF.

Útválasztó *(router)*: egy olyan speciális állomás, amely kettő vagy ennél több IP-alhálózatot kapcsol össze, és rendelkezik azzal a képességgel, hogy az egyik hálózatból érkezett csomagot a megfelelő útvonalat kiválasztva egy másik hálózatba juttatja.

Az alapvető TCP/IP fogalmak mellett meg kell barátkoznunk néhány speciális, a DHCP szabványra vagy kiszolgálóra jellemző definícióval is.

> A folyamat megold egy paradoxont: miként lehet egy állomással TCP/IP szabvány szerint kommunikálni úgy, hogy az nem rendelkezik IP-címmel, ergo képtelen a kommunikációra.

Scope: Egy folytonos címtartomány *(Pl.: 10.10.10.1-10.10.10.254)*, amelyből a DHCP kiszolgáló címeket oszt az ügyfeleknek. *(Egy vödör IP-cím – a szerk.)* Egyszerűbb esetekben a scope egy alhálózatot reprezentál.

Superscope: Több scope rendszergazdák által meghatározott csoportja, amely képes több logikai alhálózat kezelésére egy fizikai alhálózaton.

Exclusion range: Azon címek listája, tartománya, amelyeket a DHCP kiszolgáló nem fog kiajánlani az ügyfeleknek – például mert már foglaltak.

Address pool: A címtartomány kiosztható része.

Lease *(bérlet)*: a DHCP szerver által meghatározott időszak, amely idő alatt egy ügyfél használhatja a szervertől kapott címet. A bérletet meg kell újítani, különben lejár és tovább nem használható. A kiadott bérlet aktív. Ha nem történik megújítás, a bérlet aktív állapotból felhasználható *(szabad)* állapotba kerül, vagyis egy másik állomás számára kiadhatóvá válik.

Reservation *(lefoglalás)*: Állandó cím hozzárendelése egy ügyfélhez. Ezzel biztosítani lehet, hogy egy adott eszköznek mindig ugyanaz marad az IP-címe.

Option types: Az ügyfeleknek átadandó konfigurációs paraméter. Általában egy scope-hoz adunk meg opciókat, de léteznek más lehetőségek is.

Option classes: Egy másik módszer, hogy az ügyfelek részére konfigurációs beállításokat adjunk ki. Az IP-címet igénylő rendszerek tisztában vannak a saját gyártójukkal és egyéb tulajdonságaikkal, ezek alapján speciális konfigurációs beállításokat is megértenek. Az adott osztályba tartozó ügyfelek érvényesíteni fogják a beállításokat, másokra viszont hatástalan lesz.

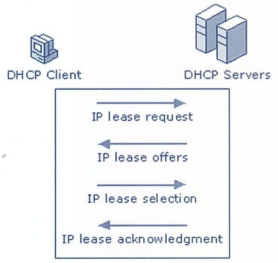
A címkiosztás célja és elvei

A DHCP szolgáltatás legfontosabb feladata, hogy egyedi címekkel lássa el azokat az állomásokat, amelyek ezt igénylik. A címeket egy előre kijelölt címtartományból, a scope-ból veszi. A folyamat szépsége abban rejlik, hogy megold egy parado-

xont: miként lehet egy állomással TCP/IP szabvány szerint kommunikálni úgy, hogy az nem rendelkezik IP-címmel, ergo képtelen a kommunikációra. A megoldás a szórt üzenetek alkalmazása. Ez az egyetlen csomagtípus, amelyet minden állomás feldolgoz, és csak a csomag tartalma alapján dönti el, az vajon neki szól-e vagy sem. Lássuk pontosan a folyamatot.

Az IP-címkiosztás menetrendje

Egy állomás indulásakor érzékeli, hogy az üzemeltetők statikus IP konfiguráció helyett dinamikus cím kérésére állították. A 68-as UPD-forrásportról a 67-es UDP-célportra egy ún. DHCP-Discover *(DHCP-felfedezés)* szórt üzenetet indít a hálón. A csomag célja, hogy az állomás felhívja magára a figyelmet, és arra kérje az ilyen csomagokat figyelő DHCP-kiszolgálókat, hogy ajánljanak fel számára címet.



Az ügyfél és a DHCP szerver kezdeti csomagváltásai

A szerver vagy szerverek az üzenetet feldolgozzák, és szintén broadcast módszerrel elküldik felajánlásukat *(feltéve, ha van kiosztható címük)*. Ezt a csomagot nevezzük DHCP-Offernek *(DHCP ajánlat)*.

A beérkezett ajánlatból vagy ajánlatokból az állomás kiválasztja a neki megfelelőt, és még mindig a szórt üzenettípust használva egy DHCP-Request *(DHCP kérés)* csomagot küld el a hálózaton. A csomagban elhelyezi annak a szervernek az azonosítóját, amelytől a címet kéri. A szórt üzenet azért hasznos, mert bár az állomás már tudja a kiválasztott szerver címét, a broadcast csomaggal könnyen értesítheti a többi szervert a választásáról, így azok erre a szórt üzenetre úgy reagálnak, hogy korábbi ajánlatukat visszavonják. A visszavonás nem jár hálózati forgalommal, csupán a felajánlott cím kerül újra felajánlható státuszba.

Ha a DHCP-szerver elfogadja a kérést, egy DHCP-Acknowledgement *(DHCP-jóváhagyás)* csomaggal nyugtázza az ügyfél felé, továbbra is szórt üzenettel. Ekkor kapja meg az ügyfél az egyéb DHCP-opciókat is. A kiszolgáló rögzíti kliense legfontosabb adatait az adatbázisban, többek között a host nevét, MAC-címét, és a bérlet lejáratának időpontját.

Igen ritka esetben az is előfordulhat, hogy nem fogadja el a szerver a kérést, ekkor egy negative acknowledgement *(DHCP-Nack)* csomagot kap az IP-címet kérő gép, amely azután újrakezdi az eljárást.

Látnunk kell, hogy nem igazán vagy nem csupán egy IP-címet kap a számítógépünk, inkább egy engedélyt arra vonatkozóan, hogy egy meghatározott ideig használhat egy adott IP-címet – néhány további paraméterrel együtt. Ebből következik, hogy nem a fenti az egyetlen kommunikáció a DHCP-szerver és az ügyfél között.

Kommunikáció a DHCP-szerverrel

Az IP-cím boldog tulajdonosa legközelebb egy újraindításkor fog kapcsolatba lépni a címkiosztó szolgáltatással. Azt várnánk, hogy a fentiekkel teljesen megegyező párbeszéd zajlik majd le, de tévedünk.

Az ügyfelek ugyanis eltárolják a bérletre vonatkozó összes információt, vagyis tisztában vannak azzal, hogy őket megilleti egy cím. Ezért induláskor DHCP-Discover helyett egy DHCP-Request csomagot küldenek, amelyben a korábban már elnyert IP-címet kérik célzott üzenet formájában, megspórolva két csomagot, némi időt és sávszélességet. A kiszolgáló ezt rendesen DHCP-ACK üzenettel nyugtázza, és az élet megy tovább.

Ha azonban a ki- és bekapcsolás között fontos változások történtek, például a hordozható gépünket egy másik telephelyen kapcsoltuk be, más eredménye lesz a csomagváltásnak. Az új alhálózatban a másik DHCP-szerver megkapja a DHCP-Request csomagot, mert a szórt üzenetet fel kell dolgoznia. Mivel a kért cím nem az ő hálózatából való, DHCP-Nack csomaggal válaszol a kérésre. Az ügyfél ezután „elfelejti" a korábbi címét, és egy szabályos IP-cím igénylésbe kezd. No, és mi van, ha a notebook-ot az otthoni hálózatba kapcsoltam be, ahol nincs is DHCP szerver?

A szituációban a különböző operációs rendszerek eltérő módon viselkednek. Ha a rendszer Windows 95, vagy Windows NT 4.0, esetleg ezeknél korábbi változat, az ügyfélszoftver azt feltételezi, hogy nem változott a helyzete, csak a DHCP-szerver nem érhető el! A válasz nélkül maradt DHCP-Request csomag arra készteti az ügyfelet, hogy ellenőrizze, érvényes-e még a bérlete. Amennyiben a válasz igen, a rendszer a korábban megkapott IP beállításokkal elindul. Ha a bérlet már nem érvényes, a hálózati protokollt nem tölti be a szoftver.

Windows 98-tól felfelé, valamint Windows 2000-nél és Windows XP-nél már okosabbak az eszközök. Ha nem jön válasz a DHCP-Request csomagra, az operációs rendszer kísérletet tesz arra, hogy eldöntse, vajon a „helyén" van-e még és csak a DHCP-kiszolgáló nem érhető el, vagy egy új hálózatban működik, ahol nincs DHCP-szolgáltatás. A trükk egyszerű, egy ICMP *(ping)* csomagot küld az alapértelmezett átjáró felé. Ha választ kap, a rendszer ott ébredt fel, ahol kikapcsolták, és a DHCP-szolgáltatással van probléma. Ellenkező esetben, tehát ha az átjáró sem válaszol, a feltételezés az, hogy nem a saját helyén indították a rendszert – ekkor használja a szoftver az APIPA képességét. Tegyünk egy kis kitérőt, és nézzük meg, miről van szó.

Csináld magad DHCP: az APIPA

A rövidítés az Automatic Private Internet Protocol Addressing kifejezés rövidítése, magyarul automatikus magán IP-cím kiosztási eljárás. A Microsoft otthoni és kisebb irodai hálózatokhoz vezette be a még csak draft formájában létező APIPA-t, olyan helyekre, ahol bizonyosan nincs kiszolgáló, mert nem érné meg, és nincs szaktudás sem a hálózat konfigurálására.

Az APIPA működése egyszerű: ha induláskor az operációs rendszer nem talál DHCP kiszolgálót, a draft által lefoglalt, B-típusú IP-címtartományból *(169.254.0.0, subnet mask: 255.255.0.0)* véletlenszerűen kiválaszt egy címet, meggyőződik arról, hogy azt más nem használja, majd elindul. A meggyőződés annyit tesz, hogy egy ICMP csomagot indít a kiválasztott cím felé. Ha érkezik rá válasz, már létezik a cím a hálózatban, tehát másikat kell keresni. Tízszer próbál így címhez jutni, és tekintve, hogy 65535 a lehetséges címek száma, kicsi az esélye, hogy nem találja meg az „igazit".

Az automatikusan meghatározott címhez azután nem ragaszkodik, és minden ötödik percben kibocsát egy DHCP-Discover csomagot, hátha meggondolták magukat az üzemeltetők és működőképes állapotba hoztak egy címkiosztó szolgáltatást.

A DHCP és az APIPA azért fér meg egymás mellett, mert az operációs rendszer el tudja dönteni, hogy milyen szituációban van. Ha korábban már kapott IP-címet, most pedig nem, ugyanakkor az alapértelmezett átjáró működik, vélhetően a DHCP-szerver áll. Sebaj, a korábban kapott, és érvényes bérletet lehet használni. Ha az alapértelmezett átjáró sem működik, úgy nincs is DHCP a közelben. Sebaj, ad magának címet az APIPA eljárással. Így vagy úgy, de az ügyfél IP-címhez jut. Persze, ha olyan régóta nincs már DHCP-szolgáltatás, hogy a bérlet lejárt, a rendszergazdák szándéka ellenére az APIPA segítségével éled fel a hálózat, de ez már nem az operációs rendszer felelőssége.

További DHCP-üzenetváltások

A bérlet egy meghatározott időtartamra vonatkozik, a bérlet lejárati dátummal rendelkezik. Az időtartamba beletartozik a kikapcsolt állapotban eltöltött idő is. Ha eltelik a bérleti idő fele, az ügyfél egy megújítási kérelmet küld annak a DHCP-kiszolgálónak, amelytől a címet kapta. Nincs szükség szórt csomagokra, az üzenetváltás közvetlen. Ha nincs válasz, 4, 8, majd 16 másodperccel később újra próbálkozik az ügyfél. A szerver egy DHCP-Ack csomaggal jelzi, hogy elfogadja a bérlet meghosszabbítását.

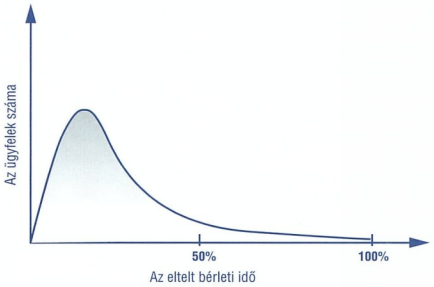
Amennyiben egy szerencsétlen véletlen miatt nem sikerül megújítani a bérletet félidőben, nem történik semmi, az ügyfél továbbra is kommunikációképes. Amikor a bérlet időtartamának 87,5%-a is eltelt, az igénylő újra próbálkozik. Ha ezúttal sem jár szerencsével, már felébred benne a gyanú, hogy valami gond van a kiszolgálóval, ezért a unicast, vagyis direkt csomag helyett ismét előveszi a régi trükköt, szórt üzenetet küld a hálózatra, hátha van ott DHCP-szolgáltatás, csak egy másik szerveren. Más kiszolgálók persze DHCP-Nack csomagot küldenek, de sebaj, ekkor újraindul a címigénylési eljárás, a végeredmény pedig egy új bérlet kiadása.

Ha minden jóakaratunk ellenére 87,5%-nál sem sikerül a bérlet megújítása *(még az újra bevetett 4, 8, és 16 másodperces újrapróbálkozás ellenére sem)* a következő kérelem a bérlet lejáratakor esedékes. A sikertelenségnek már kommunikációs következményei vannak. Ha a rendszer ismeri az APIPA módszert, azzal szerez címet, de ha nem ismeri *(Win95, NT4)*, cím *(és hálózat)* nélkül marad.

Néhány szó elöljáróban a DHCP megbízhatóságáról

Miután áttekintettük a lehetséges üzenetváltásokat, vizsgáljuk meg, hogy egyetlen kiszolgálóval, és némi ügyes beállítással milyen rendelkezésre állást tudunk biztosítani. A DHCP egy alapvető szolgáltatás, a legfontosabb kérdés mindenek előtt tehát az, vajon mennyire bízhatunk benne.

Láttuk, bármely ügyfél, amely képes eltárolni a saját bérletére vonatkozó információkat, a DHCP-kiszolgáló nélkül is képes használni a bérletét, sőt az sem probléma egy darabig, ha a megújítás nem sikerül. Egy egyszerű újraindítást a felhasználók nem vesznek észre. Az is bizonyos, hogy a kiszolgáló kiesését nem ugyanakkor észlelik majd a kliensek, hanem a bérletük lejáratakor. Az alábbi ábra mutatja, hogy egy átlagos ügyfél vélhetően a bérletidő negyedénél jár *(de nem zárható ki, hogy van olyan delikvens, amely már kitöltötte a saját idejét, pl. napokig nem volt bekapcsolva)*.



Az ügyfélszámítógépek eloszlása az eltelt bérletidő függvényében

Mennyi ideig állhat egy DHCP-szolgáltatás? A hiba nem azonnal jelentkezik, hanem apránként. Előbb egy gép esik ki, majd még egy, majd egyre több. Ha a bérletidő felénél tovább áll a szerver, az ügyfelek többségét érinteni fogja. Ebből viszont az következik, hogy a bérletidő növelése a hiba eszkalálódásának sebességét csökkenti. Ha tehát csupán egyetlen szerverünk van, és vélhetően csak lassan tudjuk megjavítani a DHCP-kiszolgálót, nyugodtan növeljük meg a bérleti idő hosszát *(persze még a hiba előtt)*, így vélhetően nagyon kevés gép esik ki, amíg a problémát elhárítjuk. Szegény ember statisztikai alapon biztosíthat magasabb rendelkezésre állást.

No persze gép és gép között jelentős különbség lehet. Ha ne adj' Isten épp a főnökünk gépe, vagy az ő főnökének a PC-je nem működik, nincs mérlegelési lehetőségünk, gyorsan kell cselekednünk és javítanunk. A jövőbeli fejmosások elkerülése érdekében pedig előbb-utóbb gondoskodni kell valahogyan a kiszolgáló többszörözéséről. Sokféle lehetőségünk van, kényelmesek és drágák, olcsóbbak és több beavatkozást igénylők. A kiszolgáló funkcióinak ismertetése után visszatérünk még a rendelkezésre állás kérdésére, már csak azért is, mert összetett DHCP beállításoknál nem is mindig könnyű megoldani a problémát.

Lepenye Tamás, MCSE 2000
lepenyet@mal.hu

Parancssori környezet és eszközök

Halál a GUI-ra

Sok UNIX-os szakember nézte le a Windows kiszolgálókat, mert csak grafikus környezetből lehetett üzemeltetni őket. Megpróbálom bebizonyítani, hogy az új generációs Windows kiszolgálókat immáron kizárólag parancssorból, manuálisan *(és automatizáltan is)* lehet mindenre kiterjedően adminisztrálni, konfigurálni és felügyelni. A cikkben – ismétlésképpen – némi DOS-ismeretet is megemlítek azon fiatalok kedvéért, akik akkor még nem éltek...

A Windows Server 2003 családban történt fejlesztésekkel a Microsoft nagy lépést tett előre a kiszolgálók kezelésével, felügyeletével és konfigurálásával kapcsolatban. Sok rendszergazdának és mérnöknek jelent könnyebbséget ezen eszközök manuális vagy batch, esetleg scriptalapú automatikus használata. Közepes és nagy hálózatokban, heterogén környezetben a Windows 2000 még hagy*(ott)* némi kívánnivalót maga után, ugyanis még a resource kit eszközeivel sem volt megoldható minden erről a felületről. Azért egyáltalán nem mindegy, hogy egy eszközt konyhakészen kapunk, vagy nekünk kell összevarázsolni, barkácsolni valamit!
Mivel még kiadás előtt álló, nem végleges szoftverről van szó, pontosítanám hogy a Release Candidate 2 verzió publikus *(CPP, build 3718)* változatából kiindulva dolgoztam.

Az újdonság erejével

A Windows XP-vel kezdődött minden. Külön csoport jött létre a fejlesztők között, hogy megfelelő előkészítő felmérések után offenzívát indíthassanak a parancssori eszközök komoly bővítéséért. A felismeréshez hozzátartozik, hogy a teljes birtoklási költség *(TBK – TCO)* jelentősen csökkenthető a rutin felügyeleti feladatok megfelelő automatizálással, amely nagy mértékben befolyásolhatja a nagyvállalati IT vezetők beszerzési döntéseit.
Mindegyik parancsnak ismernie kell a /? kapcsolót, amelyre a szintaksizát és a súgóját kell válaszként visszaadnia. Telneten és terminál-szolgáltatási felületeken keresztül is teljes funkcionalitásukat kell megbízhatóan nyújtani. És talán a legfontosabb, hogy a hagyományos parancssori konvencióknak megfelelően az Stdin, Stdout és Stderr portokkal kell működniük. A távoli futtatás egységesítése a /S kapcsolóval megoldott, valamint a telneten és a terminál-szolgáltatáson keresztüli működés is garantált.
Az XP-s fejlesztések, egységesítések, és eszközök igen nagy részét egy az egyben átvették a Windows Server 2003-ba is.

Support tools

Mivel szükségünk lehet az alaptermékben kívüli – Support Tools – eszközökre is, a telepítésükhöz indítsuk el a Server CD-ről az alábbi állományt a Windows Explorerben:

```
<x>:\Support\Tools\Suptools.msi
```

A telepítési varázsló használata egyértelmű, ezért nem részleteznem külön. A teljes telepítést stílszerűen tisztán parancssorból is megtehetjük, íme:

```
msiexec /i x:\support\tools\suptools.msi
/q ADDLOCAL=ALL
```

Ezen csomag tartalmát eredetileg Microsoft támogatási mérnökök – azon belül is specialisták – munkájának elősegítésére készítették, ezért ne lepődjünk meg ha nem svájci bicskaként funkcionálnak. Céleszközök. Azonban jónéhány, hétköznapi hibakeresésben is hasznos diagnosztikai eszköz található meg benne: a netdiag.exe és a dcdiag.exe. Ezen felül sok grafikus felületű segédprogram is található a csomagban, amelyekre nem fogok kitérni. Van rá példa, hogy egy segédprogram parancssori változata mellé külön grafikus felületű verziót is adnak. Például az installációs készleteket pucoló eszközök esetében is így van – az MsiZap.exe-re épül az MsiCuu.exe grafikus programocska.

> Ezidáig az egér nélkül kezelt Windows volt a szakmai mérce, mostantól tisztán parancssori rendszerüzemeltetés lesz az...

Segítség! Help!

A súgó. A színpadon oly fontos erőforrást hálistennek itt már nagyon komolyan készítették el, a korábbiakban mostohaként és számkivetettként kezelt parancssori eszközök dokumentációja immáron bőven eléri a kívánatos szintet. Az alaptermékben található parancssori eszközök a ntcmds.chm, a Support Toolsban találhatóak pedig a suptools.chm behívásával ismerhetők meg. A .chm típusú súgóállományokat vagy Windows Intézőből duplaklikkel, vagy a parancssorból a

```
hh.exe helpfilename.chm
```

futtatásával indíthatjuk el – a .exe kiterjesztés persze elhagyható. Ilyen korrekt, részletes súgóállományt már régen láttam, főleg a Support Tools esetében. Egy egyszerű „whoami" parancs a /? paraméteres futtatásával 57 soros választ adott vissza *(üres sorok nélkül)*. Egyes eszközökről bővebben olvashatunk a Support Knowledge Base-ben is.
Ha kollegiális segítséget keresünk ezen eszközökkel kapcsolatban, sajnos még nem érhetőek el a Windows Server 2003-as

dedikált hírcsoportok. Azonban a Tech.NET levlistákon kívül a news.microsoft.com NNTP news szerverén keresztül a **[cmdadmin]** című hírcsoportban találhatók a Windows 2000-re vonatkozó információk. Remélhetőleg a végleges termék megjelenésével egy időben elindul a Windows Server 2003-as megfelelője is.

Batch, script – automatizáció

Ez a téma kevésbé kötődik az új Windows Server verziókhoz, azonban az új eszközök leghatékonyabb felhasználási területe talán az automatizáció, és ezen területet immáron a Microsoft is kellő súllyal kezeli. Felhasználási körük szinte végtelen, leginkább a login scriptek, a tömeges címtári felhasználó vagy egyéb objektum-módosítások, illetve időzített feladatok korrekt kivitelezésénél jönnek jól. Számomra a komplex környezetek testreszabott, offline mentéseinél nyújt nagy segítséget a batchek használata.

Sok helyen összemosódik ez a két kvázi-programozási fogalom *(még hivatalos MS dokumentációban is – lásd login script, ami ebből a szempontból félrevezető)*. Én az alábbiakban definiálnám őket, mivel a komolyabb scriptelés ezen cikk határain kívül esik. A batch programozás az alap parancssori környezet beépített képességeire építve képes futtatni, valamint egymásba fűzni és átirányítani egyes programok futását, kimenetét. Paraméteres futtatással behelyettesítődnek a megadott értékek. A batch programozásnak DOS-os időkből nagy hagyományai vannak, alapjaiban nem változott meg semmi, de sokkal egyszerűbben tudjuk használni a Windows Server parancssori eszközöket ilyen módon.

Komplexebb, és inkább programozói vénát igénylő feladat scripteket készíteni. A Cscript és Wscript beépített interpreterek segítségével futtatható programok egy adott scriptnyelven *(VBScript vagy JScript)* írhatók, amelyekben változók és egyéb programozási erőforrások is rendelkezésre állnak.

Egy adott automatizációs feladat általában elvégezhető mindkét típusú megközelítéssel, a batch használata azonban gyorsabb tud lenni kevésbé összetett feladatoknál. A scripteléssel kapcsolatos fenntartásaikat pedig érdemes levetniük a tartózkodóknak, ugyanis millió sornyi minta-kód adatbázis érhető el a Microsoft Technet és MSDN weboldalán a **[scriptcenter]** és **[scr_samples]** címeken.

Az StdIn, StdOut, StdErr portok egységes használata teszi lehetővé, hogy a batchfolyamatokba programok kimenetét átfűzzük, esetleg átkössük egy másik parancs bemenetére *(pipe)*. Az Stderr hibakezelés kapcsán megemlítendő, hogy megszokás szerint a 0-át visszaadó program hibátlanul futott le, míg integer számot visszadva azonosítható a futtatási hiba. Ezen hibakódok megismeréséhez az alapdokumentáció kevés, néhány eszköz esetében a /? paraméter kilistázza, de a Microsoft a még el nem érhető Resource Kit-re hivatkozik ezügyben **[reskit2003]**.

Érdekes lehetőségeket biztosítanak több parancs összekötött végrehajtására a feltételes feldolgozási jelek. Két parancs közül, amelyek ilyen jellel lettek összefűzve, mindig a bal oldali futási eredménye alapján indul el a jobb oldali:

- & egyszerű, feltétel nélküli összefűzés;
- && a második parancs csak az első sikere esetén fut le *(if errorcode = 0)*;
- || az előző fordítottja, azaz csak sikertelenség esetén indul a második *(if errocode > 0)*.

Megjegyzendő, hogy hagyományos zárójelekkel tovább csoportosíthatóak a parancsok, valamint hogy ezen speciális jeleket idézőjelben kell használni, ha argumentumként akarjuk továbbadni őket.

Az alábbi parancsokat kifejezetten batch programozáshoz használhatjuk:

- **Call:** másik, külső batch meghívása, párhuzamos futtatással
- **Choice:** felhasználói bevitel bekérése, pl. IGEN/NEM
- **Echo:** a kimeneti kiíratás szabályozása
- **Endlocal:** környezeti beállítások lezárása, a setlocal után
- **For:** ciklikus futtatási függvény
- **Goto:** a jó öreg programozási ellenség itt megváltó is tud lenni – ugrás megadott sorra, címkére
- **If:** feltételes végrehajtás
- **Pause:** futtatás felfüggesztése, felhasználói beavatkozással továbblép
- **Rem:** megjegyzések beszúrása a „forráskódba"
- **Setlocal:** helyi környezeti beállítások
- **Shift:** futtatási paraméterek átadása

Ezekről bővebbet az eljövendő cikkekben próbálok átadni.

Az átirányítással lehet olyan feladatokat megoldani, ahol az egyes programok között együttműködés, paraméter-átadás vagy naplózás szükséges. Az itt bevezetendő „handle" kifejezést talán kezelési felületként lehetne fordítani. Ez lenne az összefoglaló neve a StdIn, StdOut, StdErr, valamint további ki és bemeneti adatfolyamoknak, amelyeket 0-tól 9-ig számoznak. Egy átirányítás ugyanis nemcsak fájlba történhet,

```
dir > dir.txt
```

hanem bármely kezelési felületről megtehetjük ezt bármely másikra. Az alábbi operátorokkal dolgozhatunk:

- > Parancs kimenetének átirányítása fájlba vagy eszközre *(pl. nyomtatóra)*, a parancssori ablak vagy kezelési felület helyett.
- < Parancs bementének beolvasása fájlból, parancssori bevitel vagy kezelési felület helyett.
- >> Kimenet átirányítása fájlba, nem felülírva, hanem kiegészítve annak tartalmát.
- >& Adott kimeneti kezelési felület átirányítása másik bemeneti kezelési felületre.
- <& Adott bemeneti kezelési felület átirányítása másik kimeneti kezelési felületre.

Utóbbi két esetben pontosan azonosítanunk kell a kezelési felületet. Ezek a következő azonosítók: StdIn 0, StdOut 1, StdErr 2, majd 9-ig az adott program által egyedileg definiáltak. Ha egy írásvédett fájl törlésének futtatási eredményét simán *(>)* átirányítjuk egy fájlba, csak a fájl nevét kapjuk meg. Azonban ha a hibakezelési kimenetet *(StdErr, vagy 2)* is bevesszük a képbe az alábbiak szerint, a megadott fájlban a hibaüzenet lesz!

```
del irasvedett.exe > kimenet.log 2>&1
```

Speciális alváltozata az átirányításnak a „csövezés" *(pipe)*, ahol az először lefutó program teljes kimenetét *(StdOut)* beadjuk a következő program bemenetére *(StdIn)*. Jól ismert példája a

```
parancs.exe | more
```

ahol is a „more.com" program formázása oldalakra tördelve jeleníti meg az adott parancs kimenetét.
A két típusú átirányítást kombinálva hasznos szűréseket végezhetünk:

```
dir /b | find "LOG" > loglist.txt
```

Ez esetben a dir listázás eredményei közül a find.exe segítségével kiszűrt „log" szót tartalmazó állományok/könyvtárak listáját a loglist.txt-ben kapjuk meg.
Szűréshez kapcsolódik még a clip nevű újdonság, amely a find és more parancsokon kívül a | csövezést igényli. Segítségével a Clipboardra *(vágólap)* irányítható egy parancssori futtatás kimenete.

```
dir | clip
```

A readme.txt tartalma ezzel másolható be a Vágólapra.

```
clip < readme.txt
```

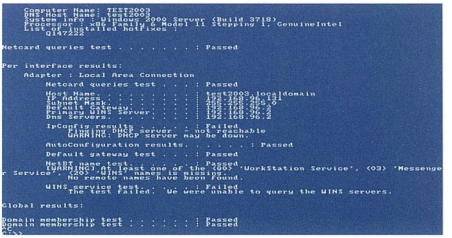
Parancssori futtatási környezet

A Cmd.exe, mint témánk központi személyisége biztosítja számunkra a parancssori munkakörnyezetet. A jó öreg Command.com leváltója komoly odafigyelést érdemel, testreszabása több szinten is megoldható és hasznos. A rendszerszintű beállítások mindenre kiterjednek, míg a helyi beállítások csak egy adott bejelentkezett felhasználó vagy egy konkrét cmd.exe példány futásakor érvényesek. További lehetőségeket biztosít batch környezetben a „nesting" technika, amely a cmd.exe többszöri, egymásból kiinduló futtatását jelenti. Az egyes cmd.exe futó példányra vonatkozó beállításokhoz használhatóak a setlocal/endlocal kapcsolók.
A környezeti változók közül talán a legismertebb %PATH%, de sok más, gépre és felhasználóra vonatkozó opció is létezik. A teljes lista az említett ntcmds.chm helpben található meg. Beállításuk a set parancssal történik, behelyettesítésük pedig parancssori és batch környezetben akkor történik meg, ha %változó% formában adjuk meg. Login scriptekben igen hasznosak a %USERNAME% és a %USERPROFILE% változók.

A parancssor testreszabásához kevésbé mission-critical, de kényelmes és hasznos lehet a könyvtár- és fájlnév-kiegészítés bekapcsolása. Ekkor kényelmesen, mondjuk a TAB billentyűvel tudunk válogatni az adott könyvtárban található bejegyzések között. A regisztrációs adatbázist kell módosítani az alábbiak szerint:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Command
Processor\CompletionChar
```

RegDWORD típusú értéket kell adni, a TAB esetében 9-et. Úgy tűnik, hogy gyárilag bekapcsolva kapjuk már ezt az opciót.
Magának az ikonról indítható Cmd.exe-nek a megjelenését a bal felső sarokban található c:\ ikonocskán keresztül elérhető Properties menüben szabályozhatjuk. A Layout fülön a Buffer Size Height megnövelésével a képernyőről már kifutott dolgokra is visszalapozhatunk. A Windows Size Height paraméterével pedig 25-ről mondjuk 50 sorosra növelhetjük az alap cmd.exe-t. Ha ezután az OK-t választjuk, a rendszer illedelmesen megkérdezi, hogy csak az e pillanatban futó parancssort változtassa meg, vagy magát az ikont, amivel mindig indítjuk a cmd.exe-t. A „Modify shortcut that started this window" ez utóbbit végzi el. A Colors fül babrálásával pedig akár az alábbi eredményt is elérhetjük:



The Matrix has you (fekete alapon zöld)

Hatásvadászok *(lásd Mátrix)* és nosztalgiázók *(lásd nagygépes terminál)* a parancssori környezetben a karakterek színét két mozdulattal zöldre állíthatják.
Amennyiben a Start menü Run segédletével indítottuk el a cmd.exe-t, az ilyen jellegű testreszabások után az OK az alábbi lehetőséget adja fel a „modify shortcut" opció helyett: „Save properties for future windows with same title". Ezzel örökíthetjük a beállításokat minden Run menüpontból indított cmd.exe-re.

Új eszközök a szerszámosládában

Nos, az említett ntcmds.chm help dokumentumból kiindulva megkülönböztethetjük a teljesen új eszközöket *(a Windows 2000 Server alaptermékhez képest)* a megváltoztatott, de már korábban is létező parancsoktól. Ezen felül egy homályos kategória, a származási törzskönyv is létezik, amit mindenkinek a történelem-ismeretére bízok: azaz honnan származik egy adott eszköz? Egyes elemek ugyanis korábbi Server változatok Support Tools vagy Resource Kit csomagjaiból erednek, sokan pedig a Windows XP-ben láttak napvilágot jelenlegi formájukban. Sok sikert a nyomozásban!
A régebbi, most megújult eszközök közül kiemelném a diskperf, az rmdir valamint az xcopy parancsokat. A diskperf ezennel hatályát vesztette, ugyanis mind a logikai mind a fizikai diszk teljesitmény számlálók automatikusan engedélyeződnek ha bekapcsoljuk őket. Kompatibilitási okokból maradt meg, az IOCTL_DISK_PERFORMANCE paramétert erőszakosan követelő applikációk számára. Az rmdir mostantól rendelkezik egy /s kapcsolóval, amivel nem üres könyvtárakat is hajlandó legyalulni. Az xcopy /g használatával a titkosított állományok másolása esetén a célkönyvtárba megérkező másolat már nem lesz titkosított *(ha megvan ehhez a megfelelő privát kulcsunk)*.
Az MSDOS óta megszüntetett parancsokról is található a helpben egy szép lista, nekrológgal együtt. Itt tudhatjuk meg hogy az fdisk helyett már a diskpart használandó partícionálásra.
Az új parancssori eszközök listája nem rövid. A Support Tools nélkül 63 bejegyzés található a helpben a „New commandline tools" címke alatt, azonban jónéhány új tétel kimaradt ebből a kigyűjtésből. A „Command-line reference A-Z" címke ABC sorrendben közli az összes parancsot, és részletes leírásukat. Funkcionalitás alapján csoportosítva sajnos csak a Support Tools eszközei vannak. Egyszer talán még megérjük, hogy a Microsoft a rendelkezésünkre bocsásson egy csoportosított, teljes indexet az alaptermék, a Support Tools és a Resource Kit parancssori eszközeiről.
Ha saját csoportosításomra építek, az alábbiakat lehet nagyjából megkülönböztetni *(a teljesség igénye nélkül)*:

Active Directory

- dsadd: objektum hozzáadása
- dsget: objektum attribútumának lekérdezése
- dsmod: objektum módosítása
- dsmove: objektum mozgatása
- dsquery: objektumok lekérdezése kritériumok alapján
- dsrm: objektum törlése
- adprep: migráció előkészítéséhez *(sémamódosítást végez)*

Internet Information Services

- iisapp: adott applikációs pool kiszolgálását végző szálak azonosítására
- iisback: metabase és séma adatbázisok mentéseit kezeli, helyben és távolról
- iiscnfg: az IIS konfigurációjának teljes vagy részleges importálására és exportálására
- iisext: kiterjesztések, alkalmazások vagy egyedi fájlok kezelésére
- iisftp: ftp helyek kialakítására, vezérlésére
- iisftpdr: ftp virtuális könyvtárak kezelésére
- iisvdir: www virtuális könyvtárak kezelésére
- iisweb: webhelyek kialakítására, vezérlésére

Printers

- prncnfg: printer konfiguráláshoz, paramétereinek megjelenítéséhez
- prndrvr: driverek telepítése, eltávolítása helyi vagy távoli printszerverekről
- prnjobs: nyomtatási munkák *(job-ok)* teljeskörű kezelésére
- prnmngr: nyomtató-kapcsolatok létrehozáshoz, vezérléséhez
- prnport: TCP/IP nyomtatóportok kezelésére
- prnqctl: tesztoldal nyomtatása, várakozási sor felügyelete

Disk

- diskpart: diszk, partíció és kötet teljeskörű kezelése *(hibatűrő és egyéb bonyolultabb konfigurációkat is kezel)*
- defrag: töredezettség-menetesítés
- fsutil: kötetkezelés, felcsatolás, lebontás
- freedisk: szabad tárkapacitás vizsgálata

Network

- getmac: hálózati adapterek MAC azonosítójának lekérdezése
- waitfor: több gépen futó végrehajtások szinkronizálása
- openfiles: nyitott fájlok kezelése, lezárása

Cluster/Nlb

- nlb: a Wlbs.exe utódja, hálózati terhelésmegosztás vezérlésére
- nlbmgr: NLB fürt konfigurálása, vezérlése

Performance, eventlog

- logman: teljesítmény-figyelés vezérlése
- perfmon: NT4 kompatibilis teljesítmény-figyelés
- relog: teljesítmény-naplók konverziója, adatkinyerés
- typeperf: teljesítmény-figyelés kiíratása emgadott helyre
- eventcreate: eseménynaplóba beíratás
- eventquery: eseménynapló lekérdezése
- eventtriggers: eseményi-kezdeményezések kezelése

Group policy

- gpresult: kiírja a GP beállításokat és Resultant Set of Policy kiértékelést végez
- gpupdate: a secedit /refreshpolicy utódja, az XP-ben már ismerkedhettünk vele
- dcgpofix: GP alapértelmezésre állítása, "full GP reset"

General

- cmdkey: felhasználói azonosítók letárolása, kezelése
- driverquery: driver-szoftverek listázása, tulajdonságok
- forfiles: parancsok automatikus futtatása nagy mennyiségű fájlon
- helpctr: Help and Support Center indítása
- pagefileconfig: a lapozóállomány teljeskörű vezérlése
- sc: szolgáltatások vezérlése
- shutdown: gépek leállítása, újraindítása helyben és távolról
- takeown: NTFS tulajdonjog átvétele
- taskkill: feladatok kilövése, a korábbi kill.exe utódja
- tasklist: szolgáltatások, alkalmazások, feladatok, szálak lekérdezése
- timeout: a parancsvégrehajtás felfüggesztése
- where: fájl beazonosítása, keresése paraméterek alapján
- whoami: aktuális felhasználó és jogainak azonosítása

UNIX héj

Ha valaki natív UNIX típusú tojáshéjra vágyik Windows Server 2003-as környezetben, annak a Services for UNIX jelenlegi, 3.0-ás változatát javaslom, ugyanis Korn Shellhez és egyéb finomságokhoz juthat. Bővebb info a **[winunix]** címen található, a termék elérhető jelképes összegért. Nem bírom megállni, hogy a GPL/GNU licenszelés alatti ingyenes eszközök forráskódjának letöltési címét ne közvetlenül ide másoljam be: ftp://ftp.microsoft.com/developr/Interix/sfu30/gnu/
És még azzal merik vádolni az MS-t, hogy nem támogatja az open source mozgalmat...

Radványszki Gábor
radvanszki.gabor@bcss.hu
konzulens, MCP
BCSS Kft.

A cikkben szereplő, kiemelt **[kulcsszo]** használata:
http://technet.netacademia.net/go?kulcsszo

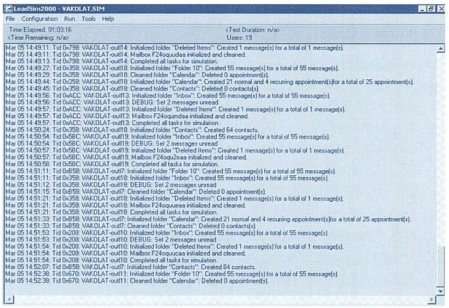
Communicare necesse est!

Használjuk erre a legmegfelelőbb eszközt!

A Microsoft Exchange a maga 110 millió felhasználójával magasan napjaink vezető üzenetkezelő és csoportmunka platformja. És hamarosan itt az Exchange 2003 *(Titanium)* az új verzió! Bátorfi Zsolt bevezető cikke után vizsgáljuk meg közelebbről, hogy mit rejtenek a csillogó korongok. Áttérjünk? Várjunk? Jobb lesz? Próbáljuk ki! Nézzük meg a gyakorlatban, hogy mi lakik a hangzatos név mögött!

A jelen

Tesztünk alanya egy kisvállalat – kedvenc téglagyárunk - egy telephellyel és pár tucatnyi felhasználóval. Kiindulási állapotunk egy Windows 2000 SP3 + Exchange 2000 SP3 és nagyon szeretnénk eljutni mindkét termékkel 2003-ba. A rendszer feltöltésére az Exchange Resource Kit loadsim nevű eszközét használtam, hogy a dolog életszerűbb legyen. Pontosabban annak a **[loadsin]** címről letölthető aktuális változatát.



Loadsim akcióban

Ez a kis program alapvetően egy Exchange terhelésszimulátor, azonban jelen esetben az volt a feladata, hogy egy szűz rendszerbe adott számú felhasználót, postaládát, nyilvános mappát, a felhasználók postafiókjába üzeneteket, kontaktokat, megbeszéléseket generáljon, majd néhány millió elvégzett művelettel pár évnyi múltat adjon a rendszernek.

Olvassunk!

Mielőtt túlságosan hamar megfeszítenénk klikkelő izmainkat a setup.exe felett, azt javaslom, hogy dőljünk kicsit hátra, olvassunk és gondoljuk át, hogy mit is szeretnénk csinálni.
Bár az Exchange 2003 még csak béta 2 fázisban van, máris tekintélyes mennyiségű dokumentáció áll rendelkezésünkre. A teszt során egy in-place *(eredetit felülíró)* upgrade-et játszunk el, nézzük, mi kell ehhez:

- Minimum Windows 2000 SP3
- Minimum Exchange 2000 SP3
- Az ADC szervereinken az Exchange 2003 ADC-nek kell *(ene)* futnia .

Eddig meg is volnánk, ADC-t nem használunk, a többi rendben. Ismét egy rövidítés. Mi is az az ADC és mire való? Ennek megértéséhez tolassunk kicsit vissza az időben, egészen az Exchange 5.5-ig. Mi volt vajon a legfontosabb különbség az Exchange 5.5 és az Exchange 2000 között?
A címtár. Az Exchange 5.5-nek bizony saját, külön címtára volt *(Exchange directory)*. Ebben tárolta a felhasználók levelezéssel kapcsolatos attribútumait és saját konfigurációs adatait. Maguk a felhasználók egy másik címtárban laktak és egy Exchange directory-ban beállított csatlakozás segítségével találtak be a postafiókjukba. Teljesen jogosan merült fel a kérdés: kell ennyi címtár? Hiszen ugyanannak a felhasználónak az attribútumairól van szó, miért tároljuk egy részét itt, egy részét ott? Dupla címtár, dupla adminisztráció, dupla fejfájás. Valószínűleg Redmondban is így gondolhatták, mert az Exchange 2000-nek már nincs külön címtára, az Active Directory-t használja *(Egy címtár mind felett…)*. Szép dolog, de vajon mi a helyzet az együttéléssel? Kidobhatjuk az Exchange 5.5-öt? Szerencsére nem, segít nekünk az ADC. Segítségével kapcsolatot teremthetünk a két címtár között, és a megfelelő konfigurációs és felhasználói adatokat szinkronizálva lehetővé válik az együttélés, vagy migráció. Az Exchange 2003 CD-n egy – főleg felhasználói visszacsatolások alapján – bővített és javított ADC található. Exchange 5.5-ről közvetlen frissítés Exchange 2003-ra nem lehetséges. Ebben az esetben az Exchange 5.5 mellé kell telepítenünk egy Exchange 2003 szervert és Active Directory konnektor és az Exchange Server Deployment Tools segítségével lehet elvégezni a migrációt.
A kompatibilitási mátrixot átnézve láthatjuk, hogy első lépésként az Exchange frissítése következik, mert az Exchange 2003 – ugyan nem teljes funkcionalitással, de – fut Windows 2000 operációs rendszeren, míg az Exchange 2000 – Windows 2003 párosítás nem megfelelő.

	Az Exchange telepíthető és futhat…		Támogatott Active Directory környezet	
Exchange verzió	Windows Server 2000-en	Windows Server 2003-on	Windows Server 2000	Windows Server 2003
Exchange 5.5 SP3	Igen	Nem	Nem szükséges	Nem szükséges
Exchange 2000 SP2	Igen	Nem	Igen	Igen
Exchange 2000 SP3	Igen	Nem	Igen	Igen
Exchange 2003	Igen (W2K SP3)	Igen	Igen (W2K SP3)	Igen

Windows – Exchange kompatibilitási mátrix

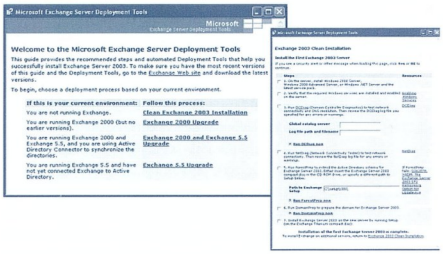
Eltávolítandó komponensek

Fontos tudnivaló, hogy – amennyiben telepítve vannak – néhány Exchange-komponenst el kell távolítanunk a frissítés előtt. Ilyen például:

- Az Instant Messaging *(a jövőben külön termékként fog megjelenni)*
- Chat
- Key Management Server *(Nincs már rá szükség. Hogy miért nem, az külön megér egy cikket)*
- MsMail konnektor, ccMail konnektor
- a Mobile Information Server Exchange Event Sink komponense.

Fúrógép, csavarhúzó, kalapács...

Örömmel állapíthatjuk meg, hogy a frissítésben alaposan kibővült eszközkészlet segít minket. A Microsoft Exchange Server Deployment Tools felkészített varázslókkal várja instrukcióinkat.

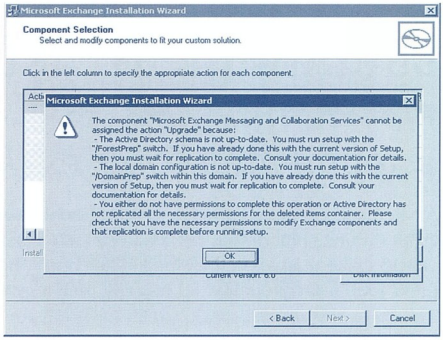


Exchange Server Deployment Tools

Ezeket az eszközöket futtathatjuk az Active Documentationból, ebben az esetben a dokumentáció lépésről lépésre vezet minket, és biztosabbak lehetünk benne, hogy a megfelelő paramétereket adjuk meg. Command line eszközből is van bőven, a teljesség igénye nélkül egy kis ízelítő: exdeploy, dsscopescan, dcdiag, orgcheck, polcheck...
Maga az Exchange 2003 telepítőprogram is változott. Például a telepítést végző felhasználónak az első szerver után nem szükséges az egész organizációban full exchange adminisztrátor jogosultság, elegendő az adott adminisztratív csoportra. Megjelent a /ChooseDC kapcsoló, ahol megadhatjuk, hogy a telepítés során melyik DC-t használjuk, és még számos változás teszi könnyebben használhatóvá a telepítőt.

Ágyazzunk!

Az Exchange 2003-nak külön meg kell ágyazni - az Exchange 2000-nél megismert módon. Az Exchange 2000 által módosított Active Directory sémát tovább kell bővíteni, ha ezt elmulasztjuk, a következő üzenet hívja fel a figyelmet erre:



Ne felejtsük el a sémát kibővíteni!

A séma bővítésében természetesen segítenek varázslóink is, de én jobban szeretek belátni Óz köntöse mögé, így mondjuk azt, hogy:

```
<CD meghajtó>:\SETUP\I386\setup /forestprep
```

Majd:

```
<CD meghajtó>:\SETUP\I386\setup /domainprep
```

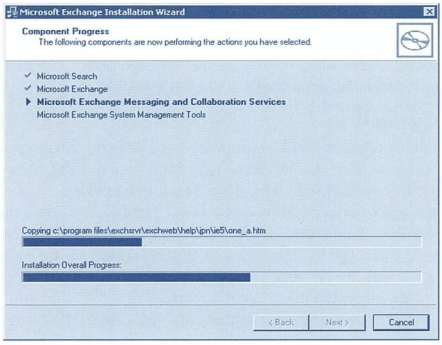
Exchange frissítés

Az előkészítés után következhet a frissítés. A következő service-re van szükség ehhez:

- .NET Framework
- ASP.NET
- World Wide Web service
- SMTP service
- NNTP service.

A .NET Framework és ASP.NET service-eket a telepítő Windows 2000-re automatikusan telepíti, azonban natív Windows 2003 forest esetében ezek le vannak tiltva, nekünk kell engedélyezni őket telepítés előtt.
Téglagyári gépünkön azonban csak elindítjuk a frissítést, és a végén örülünk.

```
<CD meghajtó>:\SETUP\I386\setup.exe
```

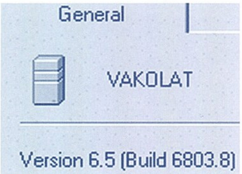


Ha eddig mindent jó csináltunk, elmehetünk kávézni

A frissítés után magasabb diszk aktivitás tapasztalható, a store.exe igen tevékeny. Ennek oka feltehetően az, hogy a verzióváltással változott a store.exe és az adatbázis szerkezete, és az adatbázis konverziója ilyenkor történik meg. Apropó store.exe: tehát maradt a jet/ese adatbázismotor.

Mit találunk a szerverünkön a frissítés után?

Egy Exchange 6.5-öt *(Buid 6803.8)*:



Exchange 2003= Exchange 6.5

Igen, az Exchange 2003 az Exchange 2000-hez hasonlóan, 6-os Exchange. A Nagy Váltás valóban várat magára az ugrás nem generációs. A dolog nem példa nélküli, gondoljunk csak a szép karriert befutott Exchange 5.5-re, ami az 5.0 gyermekbetegségeit kinőve évek óta kifogástalanul működik számos ügyfélnél.

Az Exchange 2000 SP3-tól az Exchange 2003 beta 2-ig elkészített 554 build-nek azonban meg van az eredménye, a termék számos újdonságot tartalmaz. Ezek a módosítások elsősorban a felhasználói visszajelzések alapján történtek és ennek eredményeképpen valóban a termék használhatóságát javítják. A témáról az előző számban remek összefoglaló található, de az egyes témakörök külön-külön is megérdemelnek egy önálló cikket.

Hová tűnt az M: meghajtó?

Az Exchange 2000-nél az information store tartalmát M: meghajtóként elérhettük a fájlrendszeren keresztül is. Ez azonban sokszor bajt is okozott. Az egységsugarú rendszergazda bizony késztetést érezhetett arra, hogy például a jogosultságokat – még leírni is rémes – az M: meghajtón könyvtárakként látszó mailbox-okon fájlszinten módosítsa. Ezt az Exchange „meg is hálálta" sok fejtörés okozva a helyrehozásra kiérkezett szakembereknek. További potenciális hibaforrást jelentett, hogy a fájlszintű vírusvédelmet biztosító programok és a mentőszoftverek az M: meghajtón különféle fájlszintű műveleteket végezve a levelezési adatbázis sérülését okozhatták. Jobb a békesség, az M: meghajtót elfelejthetjük.

Mit tegyenek azok, akik az adatbázis fájlrendszer driver-ét megfelelően kezelő alkalmazást használnak, és szükségük van a fájlszintű elérésre?

Az adatbázis tartalma fájlszinten továbbra is látható, azonban kicsit jobban el van rejtve. Az eléréshez szükséges névtér a következő:

```
\\.\BackofficeStorage\
```

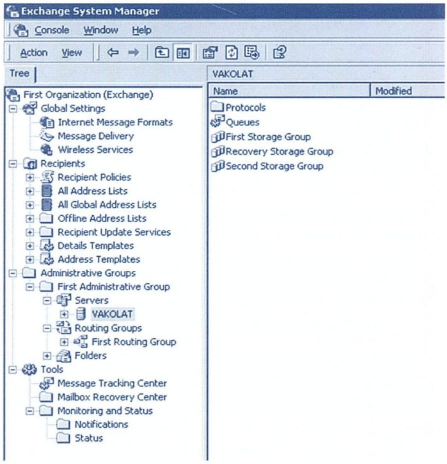
A postafiókok listáját például a

```
dir \\.\BackofficeStorage\teglagyar.net\mbx
```

paranccsal nézhetjük meg.

A „nagy piros gomb" és társai

Az Exchange frissítés tehát sikerült, az Exchange már 2003-as, az operációs rendszer még Windows 2000. Az Exchange System Managert elindítva ismerős kép fogad, nagyon hasonlít az Exchange 2000-ben megszokottra. Egy kis Exchange 2000 tapasztalattal könnyen el lehet rajta igazodni.



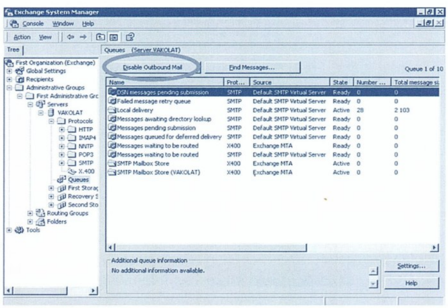
Exchange System Manager 2003

Ha egy kicsit alaposabban szétnézünk, számos változást látunk. Itt van például a wireless támogatás, ami belekerült a termékbe:

Integrált mobilitás

A termékbe integrált mobiltámogatás és a mobileszközök rohamos fejlődése az Exchange szolgáltatásainak jóval szélesebb körű elérését teszi majd lehetővé.

Újdonság a ***Queue Viewer*** is:



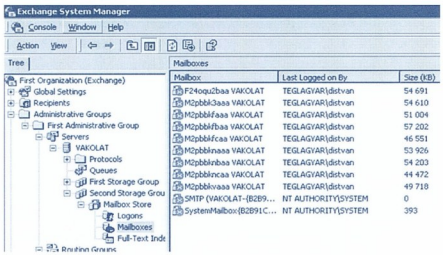
A „nagy piros gomb"

Végre átgondolt eszköz készült a queue-k kezelésére, egy helyen láthatjuk és kezelhetjük az SMTP és X.400 levelezési sorokat. Felhívnám a figyelmet a „nagy piros gombra", ami különösen hasznos lehet abban az esetben, ha egy kedves worm próbálja levelek ezreit kipumpálni szerverünkön keresztül. Persze egy jól menedzselt rendszerrel ilyen nem fordulhat elő. Már csak azért sem, mert az Exchange 2003 továbbfejlesztett és bővített antivírus API-t, a VSAPI 2.5-öt használ. Ez az Exchange 2000 SP1-ben megjelent VSAPI 2.0 utódja és lehetővé teszi többek között, hogy a mailboxot nem tartalmazó szerverek – például gateway, vagy bridgehead szerverek – is szűrjék az áthaladó forgalmat.

A ***nyilvános mappák*** kezelése is sokat fejlődött. Az Exchange 2000 esetében ez teljesen „automatikus" volt, és bizony néha megjelent az Exchange 2000 levelezési listán „mindent jól beállítottam, mégsem replikálja a nyilvános mappák tartalmát" témájú levél. Szerencsére az Exchange 2003-ban van lehetőség a replikáció manuális elindítására is, így ilyen esetben tudjuk noszogatni megfontoltan replikáló szerverünket.

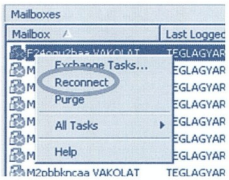
Az eszközök között megjelent a ***Mailbox Recovery Center,*** ami tulajdonképpen az Exchange 2000 CD-n található ***mbconn.exe*** nevű eszköz bővítése és integrációja a System Managerbe. Miért van erre szükség?

Képzeljük el, hogy megtörténik a katasztrófa. Az Exchange szerverünkre rászakad az ég. Enyhébb esetben leég, beázik, felrobban, vagy ellopják. A lényeg az, hogy teljesen és véglegesen működésképtelen lesz. Mit tesz ilyenkor a jól felkészült üzemeltető? Előveszi a tűzálló széfből a katasztrófatervet és megnézi, mi ilyenkor a teendő *(és persze azonnal hívja a Premier Supportot ☺).* Tegyük fel, hogy a címtár nem sérült – ha igen, az egy másik cikk lesz –, viszont a levelezőszerver teljesen megsemmisült. A katasztrófaterv tartalmazza, hogy ilyen esetben honnan kell előrántani egy megfelelő vasat, milyen beállításokkal kell feltelepíteni rá az Exchange-t *(a katasztrófa előtti állapot)* és pontosan hol található a legutolsó Exchange-mentés. Az utasításokat követve visszatöltik a mentést, és megpróbálják mountolni az adatbázist. Ha jó a terv és pontosan hajtották végre, ez sikerülni is fog, azonban a postafiókokat megnézve sok piros keresztet fognak látni:



Vajon sikerült a helyreállítás?

Nem kell rögtön a gyógyszeres szekrény felé nyúlni, nincs nagy baj, a helyreállítás egyik lépésében a címtárból el kellett távolítani a „halott" szerver bejegyzéseit. Ezért ezek a mailboxok „árván" maradtak. Az adatbázisban jelen vannak, de a címtárban rájuk vonatkozó bejegyzés nem megfelelő. Helyre kell hozni ezt a hivatkozást, vagyis újra kell csatlakoztatni a postafiókokat:



Csatlakozzunk?

Ezt megtehetjük egyesével a postafiókokra kattintva, de több száz felhasználónál ez nem túl szórakoztató időtöltés. Ilyen esetben nagyon fogjuk szeretni a Mailbox Recovery Center-t, mert az adott mailbox store-t hozzáadva megkeresi nekünk a postafiókokhoz tartozó felhasználókat és létrehozza a szükséges hivatkozásokat.

Szintén a helyreállításban nyújt segítséget a ***Recovery Storage Group.*** Ez egy speciális storage group, amibe visszatölthetjük az azonos adminisztratív csoportból származó adatbázisainkat. Az adatbázist ide visszatöltve az Exmerge segítségével visszamozgathatjuk a szükséges adatokat a megfelelő helyre. A Recovery Storage Group a felhasználók számára nem érhető el, így üzem közben is elvégezhető a visszaállítás. Miért is jó ez nekünk? Tegye fel a kezét, akihez érkezett már olyan kérés, hogy egy postafiók 1-2-3 héttel ezelőtti tartalmát kellene visszaállítani! Vagy a főnöknek nagyon szüksége van egy olyan levélre, amit 2 hónappal ezelőtt kitörölt... Mi ilyenkor a teendő, ha nincs postafiókszintű visszaállítást támogató mentőrendszerünk? Recovery szervert telepíteni, oda visszatölteni a megfelelő mentést, majd Exmerge segítségével kinyerni a postafiók tartalmát. Ez nem fél órás mutatvány. A Recovery Storage Group segítségével viszont igen.

Az újdonságok listája meglehetősen hosszú, terjedelmi okok miatt nem lehet célom ezeknek a tételes ismertetése. Egy változást azonban még megemlítek, mert már nagyon időszerű volt. Ez az „Out of Office" üzenetek kezelése. Gondolom minden levelezőlista tagnak ismerős, mikor a szabadságon lévő figyelmetlen listatagok minden listára érkező levelet egy ilyen üzenettel hálálnak meg. Mostantól kezdve biztosak lehetünk benne, hogy ők nem Exchange 2003-at használnak, mert Exchange 2003 esetében csak akkor megy el az „Out of Office" automatikus üzenet, ha a felhasználó szerepel a To: vagy a Cc: mezőben, tehát listás levelek esetében nem.

Folytassuk a migrációt a Windows frissítésével!

Windows 2003 upgrade

A Windows 2003 telepítéséhez az adprep segítségével elő kell készíteni az Active Directory erdőt és domaint. Az adprep használatának előfeltételeiről a Q331161-es cikk nyújt bővebb információt. A következő kapcsolókat használhatjuk:

```
D:\I386>adprep /?
The syntax of the command is:
adprep <cmd> [option]
Supported <cmd>:
/forestPrep       Update forest-wide information
                   Must be run on the schema role
                   master

/domainPrep       Update domain-wide information
                   Must be run on the infrastructure
                   role master
                   Must be run after /forestPrep is
                   finished
Supported [option]:
/noFileCopy       adprep will not copy any file from
                   source to local machine
/noSPWarning      adprep will suppress the
                   Windows 2000 service pack 2
```

```
                requirement warning during
                /forestprep
```

Az erdő előkészítése:

```
<CD meghajtó>:\I386\adprep /forestprep
```

Majd kapunk egy figyelmeztetést:

```
ADPREP WARNING:
Before running adprep, all Windows 2000 domain
controllers in the forest should be upgraded to
Windows 2000 Service Pack 1 (SP1) with QFE 265089,
or to Windows 2000 SP2 (or later).
QFE 265089 (included in Windows 2000 SP2 and
later) is required to prevent potential domain
controller corruption. For more information about
preparing your forest and domain see KB article
Q331161 at http://support.microsoft.com.
[User Action]
If ALL your existing Windows 2000 domain
controllers meet this requirement, type C and then
press ENTER to continue. Otherwise, type any other
key and press ENTER to quit.
```

Mivel domain kontrollerünk megfelel a feltételeknek,

```
C <ENTER>
```

Megtörténik a séma frissítése:

```
Opened Connection to VAKOLAT
SSPI Bind succeeded
Current Schema Version is 13
Upgrading schema to version 30
Connecting to "VAKOLAT"
Logging in as current user using SSPI
Importing directory from file
"C:\WINNT\System32\sch14.ldf"
Loading entries.............................
111 entries modified successfully.
.
Adprep successfully updated the forest-wide
information.
```

Majd:

```
<CD meghajtó>:\I386\adprep /domainprep
```

A sikeres adprep után indíthatjuk a frissítést:

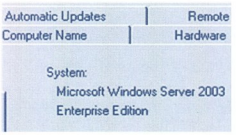
```
<CD meghajtó>:\I386\winnt32.exe
```

Windows 2003 upgrade

A frissítés a licenszkulcs megadása, újraindulás és egy kis várakozás után be is fejeződik, közben természetesen roppant hasznos dolgokat olvasgathatunk az új termékről:

Megérkeztünk?

A frissítést látszólag sikeresen megtörtént. Ellenőrizzük az operációs rendszerünk verzióját:




```
C:\ver
Microsoft Windows [Version 5.2.3763]
```

Éljenzés, görögtűz, mindkét termékkel megérkeztünk 2003-ba, a feladatot elvégeztük!
Ezzel az utolsó lépéssel hazai pályára helyeztük az Exchange 2003-at, Windows 2003 operációs rendszeren futtatva kapunk teljes funkcionalitást. A teljesség igénye nélkül nézzük át, hogy mivel lett okosabb az Exchange 2003 szerverünk:

- **Shadow Copy Backup:** A Windows Shadow Copy service segítségével a mentés terén új lehetőségeink nyílnak. Segítségével konzisztens pillanatfelvételt készíthetünk egy kötetről. *(Fontos téma, szintén megér egy cikket.)*
- **Outlook http access *(RPC over http)*:** Az IIS 6.0 és a Windows 2003-ban található Windows RPC Proxy service lehetővé teszi, hogy az Outlook 2003 és az Exchange 2003 közötti kommunikáció http, vagy https protokollokon keresztül történjen.
- **Fejlettebb clustertámogatás:** Mindkét termék fejlesztett és bővített clustertámogatással rendelkezik. Aki kinőtte a jelenlegi 2 vagy 4 node-os clusterét, annak van egy jó hírem: ezentúl építhet 8 node-os clustert! A függőségi viszonyok ésszerűsítése jelentősen jobb failover időt eredményez, vagyis probléma esetén az Exchange szolgáltatásai percekkel rövidebb idő alatt elérhetőek. Ez a javulás igen jelentős, ha figyelembe vesszük, hogy Windows 2000 – Exchange 2000 párosítás esetén egy átlagos failover – felhasználók számától függően – 3-8 percet vett igénybe. Végre a Kerberos az alapértelmezett autentikációs protokoll az Exchange virtuális szervereknél. Exchange 2000 esetében ez az NTLM volt, mivel a Windows Cluster service csak a Windows 2000 SP3 után volt képes Kerberost használni.

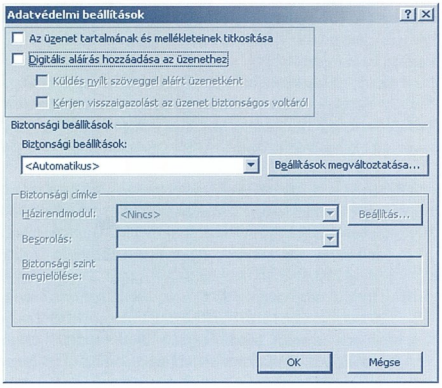
Eddig a szerveroldallal foglalkoztunk, vajon mi újság a végeken? Egy levelezőrendszernek alapvetően a felhasználók igényeinek kell megfelelnie. Az Exchange 2003 fejlesztésekor is ez volt a legfontosabb szempont, a legnagyobb fejlődés a kliensoldalon és a klienselérés területén történt. Teljesen megújult az Outlook Web Access, funkcionalitását tekintve felveszi a versenyt a szintén számos újdonságot tartalmazó Outlook 11-gyel. De ez már egy másik történet, és egy másik cikk. Ezzel folytatjuk.

Détári István
istvan.detari@getronics.hu
MCSE, MCSA, CCNP

Dupla KU

PKI ikonok az Outlookban

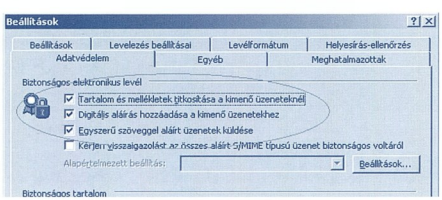
K: *Egyre gyakrabban küldök digitálisan aláírt vagy titkosított levelet Outlookkal. Ilyenkor minden esetben az üzenet biztonsági beállításainál kell pár pipát tennem, ami az alábbi képen is látszik.*



Üzenetek titkosítása és digitális aláírása

Hogyan lehet megoldani, hogy gyorsabban tudjak egy-egy levelet digitálisan aláírni?

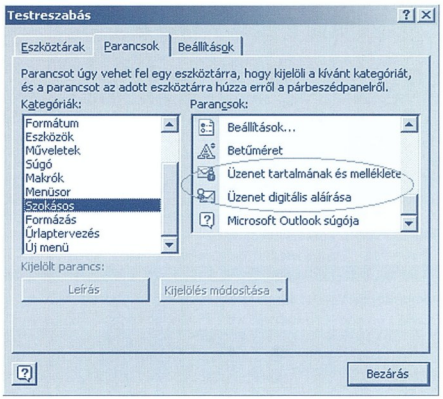
V: Ha az Outlookot a biztonsági beállítások közt *(Eszközök → Beállítások → Adatvédelem)* az alábbi képen is látható módon állítjuk be, akkor minden kimenő üzenet alapértelmezésként digitálisan aláírt és titkosított is lesz.



Minden kimenő levél aláírása és titkosítása

Ez a megoldás azonban a másik végletet jelenti, mert ha nem mindenhova küldünk titkosított és/vagy digitálisan aláírt leveleket, a pipákat a leveleknél egyenként ki kell szedni.
Ennél jobb megoldás, ha a titkosításhoz és aláíráshoz szükséges ikonokat megjelenítjük az eszköztáron. Nézzük lépésről lépésre:

1. Nyissunk egy új levelet
2. A levél Eszközök → Testreszabás menüjéből válasszuk a Parancsok tulajdonságlapot.
3. A baloldali kategóriák közül a Szokásosat *(Standard)* kell kiválasztani, és a jobb oldalon meg kell keresni a digitális aláíráshoz és a titkosításhoz használható parancsokat. Az alábbi ábra talán segít a keresésben:



A digitális aláírás és a titkosítás elrejtett ikonjai

Figyelem!
Hiába kutatunk a megfelelő gombok után, ha az alapértelmezett levélszerkesztőnk a Word. Ilyenkor nem bukkan fel a kategóriák közt a Szokásos *(Standard)*, és nem tudjuk használni ezeket az ikonokat.
A WordMail egyébként teljesen „üti" ezeket a gombokat, tehát még ha cselesen fel is tesszük őket a Word átmeneti lekapcsolásával, amint visszakapcsoljuk, a gombok eltűnnek. De folytassuk a lépéseket!

4. Ha rátaláltunk a listában a parancsokra, csak ki kell őket húzni az egérrel a levél eszköztárára. Tehát NEM az Outlook fő eszköztárára, hanem a levélére!

Így néz ki a végeredmény:

Felkerültek az eszköztárra az aláírásra és titkosításra szolgáló gyorsítógombok!

Ezentúl az eszköztárra kirakott ikonok segítségével gyorsabban tudjuk aláírni és titkosítani az üzeneteket, mert a két kis ikon ott lesz a fejlécen akkor is, ha új levelet írunk, de akkor is, amikor válaszolunk egy levélre, vagy épp továbbítjuk azt.

A .Net és a Tablet PC

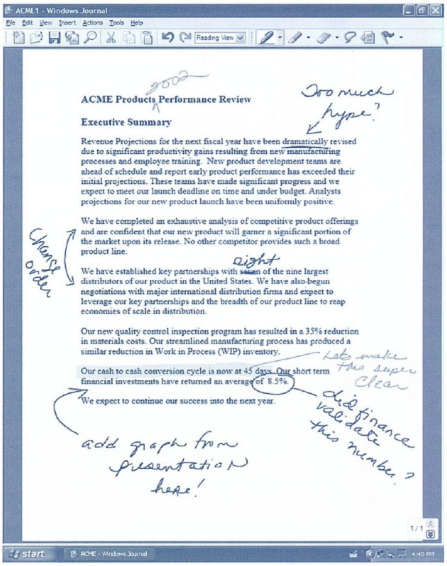
Miért fontos, hogy a Tablet PC a .Net keretrendszerre épül és a felhasználó számára milyen haszonnal jár a beépített .NET infrastruktúra? Hogyan illeszkedik a táblás modell a .NET stratégiába? A Microsoft PressPass kérdéseire Vic Gundotra, a Platform Strategy igazgatója és Alex Gounares, a Tablet PC vezetõ tervezõje válaszolt.

Gundotra szerint ahhoz, hogy megértsük, miért izgalmas a Tablet PC, érdemes elõbb egy pillantást vetnünk a.NET vízióra. Egyszerûen fogalmazva a .NET olyan platform, amely kapcsolatot teremt különbözõ rendszerek között. Ez talán túlzott egyszerûsítésnek hangzik – mondja Gundotra –, programozói szemmel nézve azonban a különbözõ üzleti megoldások integrációja, az ügyfelek és a vállalkozások közötti átjárhatóság megteremtése rendkívül költséges és bonyolult feladat. Az XML Web Services szabványos elemeire épülõ .NET keretrendszer azonban segít a költségek csökkentésében, az eltérõ rendszerek összekapcsolásában, együttmûködésében. A Tablet PC azért tekinthetõ a.NET-kész kliensek zászlóshajójának, mert a .NET keretrendszer az operációs rendszer része.

Miért fontos a felhasználó számára beépített .NET infrastruktúra? A .NET keretrendszer telepíthetõ ugyan egyedi PC-kre – válaszolta Gundotra –, de ezzel a feladatnak még nincs vége. Ám ha az alkalmazottak Tablet PC-t használnak, a beépített .NET keretrendszer révén minden egyes gép elõtt nyitva áll a .NET infrastruktúra minden lehetõsége. Látványosan könnyebb lesz a szabványos

Tablet PC-s alkalmazások bevezetése is, hiszen azokat csupán el kell helyezni a webszerveren, majd úgy beállítani a felhasználók Tablet PC-it, hogy erre mutassanak. A Tablet PC ezután automatikusan letölti és elindítja az alkalmazást.

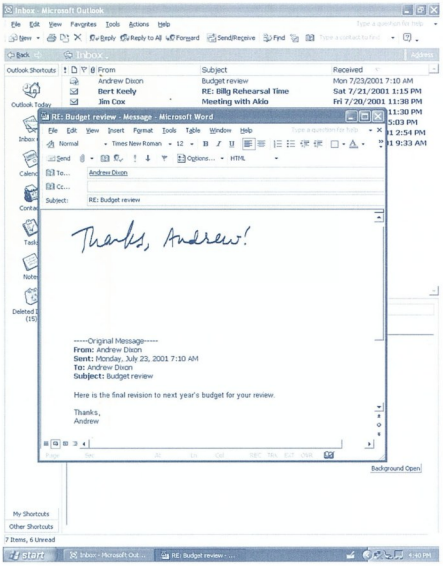
Gounares szerint a jövõben mind a horizontális, mind a vertikális alkalmazásokban az eddiginél nagyobb szerepet kaphat a kézírásos adatbevitel és a képállományok kommentálása. Vegyünk például egy biztosítótársaságot, amelynél kézírással kitölthetõ formanyomtatványokat vezetnek be. A biztosítási ügynök digitális fényképet csatolhat a kárfelvételi jegyzõkönyvhöz, a képen pedig digitális tintával bekarikázhatja az autón keletkezett sérüléseket. Az ilyen, testre szabott alkalmazások kifejlesztése és bevezetése a .NET keretrendszer jóvoltából egyszerûbb, mint valaha.



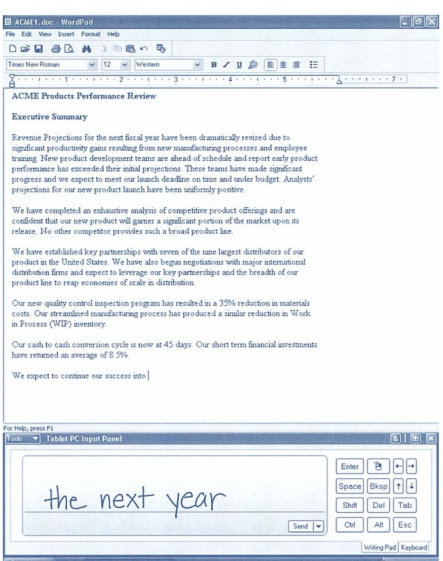
Minthogy a .NET keretrendszer a Visual Basictõl a C++-ig szinte valamennyi programnyelvet ismeri, minden fejlesztõ ugyanazokat a programozói könyvtárakat használhatja, ily módon a fejlesztés olcsóbbá és egyszerûbbé válik. Ahhoz – tette hozzá Gounares –, hogy egy, a .NET-re épülõ alkalmazáshoz kézírás-felismerési támogatást készítsen valaki, mindössze két programsort kell megírnia.

Hogyan illeszkedik az írótáblás modell a Microsoft .NET stratégiájába? – tette fel a kérdést a PressPass. Nos, Gundotra szerint a Tablet PC legfontosabb újdonsága, a kézírásos adatbevitel számos esetben sokkal természetesebb és hatékonyabb a billentyûzetnél. Ám a Tablet PC a felhasználó környezetére is hatással van. Ha egy tárgyaláson valaki hagyományos laptopot

vesz elő, és gépelni kezd, miközben az ügyféllel beszélget, óhatatlanul elvonja a figyelmét. Ezzel szemben, ha a szemébe nézve jegyzetel, az ügyfél úgy érzi, rá összpontosul a figyelem.



Korábban a méret volt a hordozható gépek legfőbb kritériumainak egyike. A Tablet PC esetében az adatbevitel – azaz a kézírás- és beszédvezérlés lehetősége – legalább ilyen fontos.



A .NET vízió lényege a rendszerek összekapcsolása, ennek pedig fontos része a mobil eszközökkel való integráció. E filozófia szerint – fejtegette Gundotra – egyre több érték a hálózat perifériáján képződik. Minél egyszerűbbé és hatékonyabbá válik a kapcsolat a Tablet PC-hez hasonló eszközökkel a különböző rendszerek és hálózatok között, annál kevésbé lesz értelme hálózati végpontokról beszélni. Gondoljunk csak bele annak a felhasználónak a helyzetébe, aki akárhonnan, akármikor, akármilyen eszközzel kapcsolódni szeretne a vállalati adatbázishoz! Amennyiben ez a feladat szabványos módon megoldható, megoldódik a vállalatok és belső rendszereik közötti kapcsolódás problémája is. Amennyiben az XML webszolgáltatásokat megfelelően használják, és köré olyan programozási modellt építenek, mint a .NET keretrendszer, mindazokat a problémákat elfelejthetjük, amelyekre elosztott számítástechnika néven szokás utalni. A .NET decentralizálja és ellaposítja a hálózatot, így többé nem lesz jelentősége annak, hogy a hálózat végpontján vagy a kellős közepén vagyunk-e éppen, mert az információkhoz és a szolgáltatásokhoz mindenki egyformán, szabványos módon férhet hozzá. Korábban – tette hozzá Gounares – a hálózat perifériáján meg kellett elégednünk a rádiótelefonokhoz hasonló egyszerű szerkezetekkel. Mostantól a hálózati végpont is lehet teljes értékű PC.

Ám hogy nem csak elméletekről van szó, arról orvosok, gyógyszergyártó cégek, kórház-informatikai szállítók és szoftverfejlesztők Tablet PC-vel szerzett tapasztalatai tanúskodnak. Mint arról a San Diegóban rendezett 2003 Healthcare Information and Management Systems Society *(HIMSS)* konferencián és kiállításon beszámoltak, a nagyjából kórlap méretű Tablet PC az orvosok számára a jegyzettábla és az egérrel, billentyűzettel, írósvesszővel egyaránt kezelhető, valós idejű, vezeték nélküli adatkapcsolatot nyújtó számítógép előnyeit egyesíti.

A San Franciscói Stentor kórház-informatikai cég elkészítette orvosi információ- és képkezelő rendszerének Windows XP Tablet PC kiadásra átdolgozott változatát. Az iSite Enterprise segítségével az orvosok egy szempillantás alatt lehívhatják a kórházi hálózatra kapcsolt számítógépekről a páciensről tárolt adatokat, felvételeket a tablet gépre.

S mint arról egy orvosgyakornok beszámolt, számára a Tablet PC olyan, mintha az íróasztalát tartaná a hóna alatt: megnézheti a laboreredményeket, a kórelőzményeket, vagyis mindazt, amire a kezelés során szüksége van, s mégsem akkora súlyt magával cipelnie, mint korábban, amikor az összes kórlapot magával hordta a nagyvizitre.

Kelenhegyi Péter
kelenhegyi@axelero.hu

Microsoft Project 2002
– hogy a tervekből valóság legyen!

A projektirányítás számos szervezet sikerében és üzletmenetében játszik fontos szerepet. Egy hatékony tervezési eszköz birtokában csökkenthetők a költségek és javítható a termelékenység, ami nyereségnövekedést eredményez. A Microsoft **Project 2002 Standard** minden eddiginél egyszerűbbé teszi az ütemezések és erőforrások kezelését, a projekt állapotának közlését és a projekt adatainak kimutatását.

Cégek, nagyvállalatok részére, a csoportmunkát teljes mértékben támogató **Project 2002 Professional** kínálja a Microsoft, mely segítségével és a központi nyilvántartást és kiszolgálást biztosító **Project 2002 Server** termékkel együttműködve teljeskörű nagyvállalati, projektirányítási, tervezési és döntéshozási feladatok végezhetők el, és annak adatai bármikor, bárhonnan hozzáférhetőek, publikálhatóak.

Ismerje meg közelebbről tanfolyamainkon!

Microsoft Project 2002 felhasználói kurzusok

Az érdeklődők **kétnapos, intenzív** tanfolyamok keretében ismerkedhetnek meg a Microsoft Project 2002-es verzióinak használatával.

Rugalmas időbeosztás

Megpróbáltuk figyelembe venni, hogy a cégek dolgozóikat nem tudják nélkülözni hosszabb időre, ezért a tanfolyamokat **Intenzív, egész napos formában** hirdettük meg, délelőtt 9.00 és 16.00 óra között. A tanfolyamok díja az oktatás és a tananyagok mellett az étkezést is tartalmazza a kurzus napjaira.

Project 2002 Professional és Server – testreszabott oktatások

Oktatóközpontunkon kívül, a megrendelő telephelyén, kihelyezett formában (vidéken is) is vállaljuk az adott cégprojektekhez kapcsolódva tanfolyamok vagy konzultációk megtartását és lebonyolítását, különös tekintettel az **összetett, Project 2002 Professional és Project 2002 Server használatával és bevezetésével** kapcsolatos témákban. Ezt követően igény esetén további **utólagos támogatást** és szaktanácsadást is nyújtunk.

Menjen biztosra!

Tervezze üzleti folyamatait Microsoft Project 2002-vel!

2 napos

PKI*-Workshop

rendszergazdáknak

A NetACADEMIA A LEGJOBBAKAT TANITJUK és a NETLOCK Az Első Hitelesítés Szolgáltató közös szervezésében

- *Elektronikus aláírás használata*
- *SSL, IPSec, EFS, S/MIME a gyakorlatban*
- *Ajándék kulcstároló eszköz és NetLock tanúsítvány*
- *Smart Card Logon*

Legyen az elsők között!

*Public Key Infrastructure, a nyílt kulcsú titkosítás köré, annak segítségével készített alkalmazások összessége

PKI-Workshop

Átfogó tudás az elektronikus aláírás és titkosítás vállalati szintű alkalmazásához.

A kétnapos tanfolyamon minden tudást, tapasztalatot és eszközt (USB token), továbbá a szükséges elektronikus tanúsítványokat átadjuk hallgatóinknak, akik a megszerzett gyakorlat birtokában fel tudják készíteni a technológia használatára azokat a munkatársakat, akiknek már ma szükségük van elektronikus aláírásra és adataik titkosítására.

Előadók:

Fóti Marcell, NetAcademia (kriptográfia, Smart Card Logon)
Boromisza Zsolt, NetLock (kulcstároló eszközök, kulcsok, tanúsítványok)
Fülöp Miklós, NetAcademia (Certificate Authority, SSL, IPSec, S/MIME stb.)
Dr. Mayer Erika, Páneurópa Jogász Unió (az elektronikus aláírás jogkövetkezményei)

Témakörök:

- Mi az a kriptográfia? Hogyan működik a **titkosítás**?
- Mi a **kulcspár**, a **tanúsítvány**, mi a szerepe az intelligens kulcstároló **eszközöknek**?
- Szimmetrikus algoritmusok (DES, 3DES, AES)
- Nyílt kulcsú algoritmusok (Diffie-Hellmen, RSA)
- Hash (MD4, MD5, SHA-1)
- Az **X.509** tanúsítványok szerepe és **használata**
- Hitelesítésszolgáltatók. A hierarchia felépítése
- Kulcstároló eszközök: **intelligens kártya** és **USB Token**
- Bejelentkezés tanúsítvánnyal (Single Sign On)
- SSL, S/MIME, HTTPS
- **Virtuális magánhálózatok**.

Jogi blokk:

Az elektronikus aláírási törvény célja, következményei, a végrehajtási rendeletek. Jogkövetkezmények, és a szükséges feltételek megléte.

Ajándék! Hallgatóink a tanfolyamon használt kriptográfiai eszközöket a tanfolyam után megtarthatják, és - a mellékelt NetLock tanúsítványokkal - hiteles elektronikus aláírást és nyílt kulcsú titkosítást használhatnak!

Aláíró és titkosító kulcsok tárolása USB-tokenen

+ NetLock B-osztályú tanúsítványpár!

A tanfolyam kezdete	Hossz [nap]	Bővebb információ	Ár [Ft]
2003. május 19-20. **2003. június 16-17.** **2003. augusztus 18-19.**	**2**	**http://www.netacademia.net/workshop/PKI** **Telefon: (1) 472-1214**	**140.000**

Jelentkezés weben:

- http://www.netacademia.net/workshop/pki

Jelentkezés faxon (472-1215) az alábbi adatok megadásával:

- cégnév
- számlázási cím
- fax, telefon
- hallgató(k) neve(i)
- e-mail
- időpont

A tanfolyamok helyszíne

A NetAcademia Kft. világörökségi környezetben várja hallgatóit Budapesten az **Andrássy út 62**-ben!

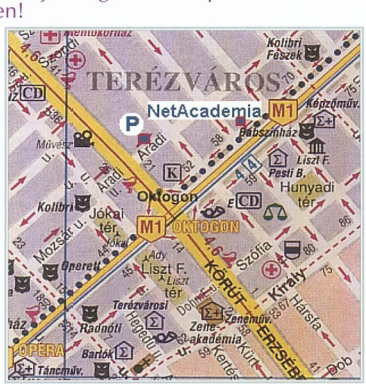
Megközelíthető az alábbi tömegközlekedési eszközökkel:

- a Milleniumi kisföldalattival (Vörösmarty utca)
- 4-es és 6-os villamossal (Oktogon)
- vonattal (a Nyugati pályaudvartól 10 perc séta)

Ingyenes parkolási lehetőség a Felvonulási téren.

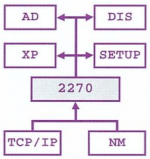
További elérhetőségeink:

Telefon: 1/472-1214
Fax: 1/472-1215
E-mail: info@netacademia.net
Weblap: http://www.netacademia.net



További tanfolyamaink rendszergazdáknak:

2270 - Windows 2003 Server és Active Directory alapismeretek és áttérés



A 2270-as sorszámot viselő tanfolyam az egyik legkomolyabb anyag a Microsoft kínálatából. Összetettségére, átgondoltságára jellemző, hogy négy (!) MCP-vizsga felkészítő anyagaként ajánlják.

W2003 - Windows 2003 Server Expert Workshop

Szakítson időt a Windows 2003 újdonságainak felfedezésére! Nálunk két nap alatt megtudhatja mindazt, amit egyedül másfél év alatt lehet összeszedni!

Témakörök:

- Az Active Directory újdonságai
- A Group Policy újdonságai
- Fejlesztői újdonságok rendszergazdáknak
- Rendszerújdonságok
- Biztonsági újdonságok
- IIS 6 újdonságok

SETUP - Felügyelet nélküli telepítés

A tanfolyamot rendszergazdáknak, üzemeltetőknek ajánljuk. Automatikus telepítési változatok. A RIS és a PXE indítókörnyezet. Válaszfájlok patkolása, OEM eszközmeghajtók telepítése, „idegen" merevlemez-vezérlők. Alkalmazások telepítése.

Ha további rendszergazdai vagy fejlesztői tanfolyamainkra is kíváncsi, kérje teljes körű, **ingyenes tanfolyami katalógusunkat**!

H-1062 Budapest, Andrássy út 62. • Tel.: 1/472-1214 • Fax: 1/472-1215
E-mail: info@netacademia.net • http://www.netacademia.net