Integrált Információs rendszerek

- előadás vázlat -

Készítette: Juhász Sándor
2003-2008

Tartalom

I Információs rendszerek jellemzői ... 5
I.1 Mi az információs rendszer? ... 5
I.2 Információs rendszerek típusai ... 6
I.3 Információs rendszerek teljesítménymérése ... 6
I.4 Információs rendszerek csoportosítása ... 7
I.4.1 Központosított és elosztott rendszerek ... 7
I.4.2 Főbb követelmények (elosztott megvalósításnál) ... 8
I.5 Jellemző felépítés: a rétegszerkezet ... 8
I.5.1 Kétrétegű architektúrák ... 9
I.5.2 Háromrétegű architektúrák ... 9
I.5.3 Rétegek szétválasztása ... 10
I.5.4 Rétegek elhelyezése ... 10
I.5.5 A rétegszerkezet kiterjesztése ... 11
II Információs rendszerek kliens oldalról ... 12
II.1 Vastag kliens alkalmazások ... 12
II.2 Vékony kliens alkalmazások ... 13
II.3 Okos kliens alkalmazások ... 15
II.3.1 SOA – Service Oriented Architecture ... 15
II.3.2 Okos kliens alkalmazások jellemzői ... 15
II.3.3 Mobil kliens megoldások ... 16
II.4 Az architektúrák tulajdonságainak összevetése ... 19
III Párhuzamos számítógép architektúrák ... 20
III.1 Teljesítmény növelésének módszerei ... 20
III.2 Neumann architektúra és cache ... 20
III.3 Párhuzamos számítógép architektúrák ... 21
III.3.1 Mai párhuzamos architektúrák ... 22
IV Klaszterek ... 25
IV.1 Mi a klaszter? ... 25
IV.2 Klaszter architektúrák ... 26
IV.2.1 Együttműködés lehetőségei ... 28
IV.2.2 Hálózatba kapcsolt számítógépek, mint platform ... 29
IV.3 Párhuzamos algoritmusok modellezése ... 30
IV.3.1 Fogalmak ... 30
IV.3.2 Párhuzamos algoritmus modell ... 30
V Szerver partícionálás és virtualizáció ... 31
V.1 A virtualizáció szerepe az informatikában ... 31
V.2 Virtualizáció legfontosabb területei ... 31
V.2.1 Erőforrás virtualizáció ... 31
V.2.2 Platform virtualizáció ... 31
V.2.3 Virtuális munkakörnyezetek ... 32
V.3 Számítógépek virtualizálásának módjai ... 32
V.3.1 Alkalmazás szinten történő virualizálás ... 32
V.3.2 Virtualizálás operációs rendszer szinten ... 32
V.3.3 Emuláció ... 32

V.3.4 Paravirtualizáció ... 33
V.3.5 Virtualizálás hardver támogatással ... 33
V.4 Virtualizálás vállalati rendszerekben ... 36
V.4.1 A virtualizálás motiváció információs rendszerekben ... 36
V.4.2 A szerverkonszolidáció hatásai ... 37
V.4.3 A virtualizálás kihasználásának egyéb területei ... 37
V.4.4 Tipikus szolgáltatások ... 37
VI Peer-to-Peer rendszerek ... 39
VI.1 Hogyan működik? ... 39
VI.1.1 Esettanulmány: Windows Peer-to-Peer Networking ... 39
VI.2 Jellemezők ... 41
VI.3 P2P fajtái ... 42
VI.3.1 Megosztatni kívánt szolgáltatás szerint ... 42
VI.3.2 Hálózati környezet szerint ... 42
VI.3.3 Csomópontok platformja szerint ... 42
VI.3.4 Architektúra szerint ... 43
VI.4 Más rendszerekkel való összehasonlítás ... 43
VI.4.1 Peer-to-peer és kliens-szerver rendszerek ... 44
VI.4.2 Peer-to-peer és a klaszterek ... 44
VI.5 P2P rétegek ... 45
VI.6 Topológiák ... 46
VI.6.1 Random Mesh ... 46
VI.6.2 Semi-Random Mesh ... 46
VI.6.3 Connected Mesh ... 46
VI.6.4 Connected Stars ... 46
VI.7 Adaptív TTL ... 47
VI.8 Esettanulmány: Gnutella ... 47
VII Vállalati alkalmazásintegráció ... 48
VII.1 Az alkalmazásintegráció története ... 48
VII.1.1 Jelentős versenyelőnyök ... 48
VII.1.2 Az alkalmazásintegráció előkészítése ... 48
VII.1.3 Az integrációs folyamat buktatói ... 49
VII.2 Az integráció folyamata ... 49
VII.3 Alkalmazáspárok integrálása ... 50
VII.4 Az integráció réteg szintjei ... 52
VII.4.1 Adat szintű integráció ... 52
VII.4.2 API szintű integráció ... 53
VII.4.3 Metódus szintű integráció ... 54
VII.4.4 User Interface szintű integráció ... 54
VIII Vállalati alkalmazás integráció ... 56
VIII.1 Spagetti ... 56
VIII.2 Integrációs alternatívák ... 57
VIII.2.1 Az integrációt előről kezdjük ... 57
VIII.2.2 Enterprise Resource Planning (ERP) ... 58
VIII.2.3 Middleware alapú alkamazás integráció ... 58
VIII.2.4 Integrációra tervezés ... 58
VIII.3 Middleware ... 59
VIII.4 Az integrációs központ ... 59
VIII.5 Business Process Management ... 60
VIII.6 Az alkalmazásintegráció funkcionális szintjei ... 61

VIII.6.1 Adatkapcsolat szintű integráció ... 61
VIII.6.2 Alkalmazás szintű integráció ... 61
VIII.6.3 Folyamatszintű integráció ... 62
VIII.7 Middleware rendszerek ... 63
VIII.7.1 Middleware típusok ... 63
VIII.7.1 RPC - Távoli eljáráshívás ... 65
VIII.7.2 Üzenet-orientált middleware (MOM, Message Oriented Middleware) ... 65
VIII.7.3 Elosztott objektum kezelők (Object Request Brokers) ... 66
VIII.7.4 Tranzakciós rendszerek ... 67
VIII.7.5 Komponens alapú alkalmazás-szerverek ... 68
VIII.7.6 Egyedi middleware megoldások ... 70
VIII.8 A middleware kiválasztása ... 70
VIII.8.1 Szempontok ... 70
VIII.8.2 Kiválasztás folyamata ... 71
VIII.8.3 Middleware típusok összehasonlítása ... 71
IX XML ... 72
IX.1 Felépítés ... 72
IX.2 Karakterek, elnevezések ... 73
IX.3 Névterek (Namespaces) ... 73
IX.3.1 Név konfliktus ... 74
IX.3.2 Megoldás név előtéttel ... 74
IX.3.3 Megoldás névtér használatával ... 74
IX.4 Tulajdonságok ... 75
IX.5 DTD (Document Type Definition) ... 76
IX.5.1 Elemek ... 76
IX.5.2 Tulajdonságok ... 77
IX.5.3 Egyedek ... 77
IX.6 Megjelenítés ... 78
IX.6.1 HTML ... 78
IX.6.2 CSS (Cascading Style Sheets) ... 79
IX.6.3 XSL eXtensible Stylesheet Language ... 80
IX.7 Programozás ... 81
IX.7.1 SAX ... 81
IX.7.2 XML DOM ... 81
IX.8 Egyéb alkalmazások ... 81
IX.8.1 XHTML ... 82
IX.8.2 XML Schema ... 82
IX.8.3 EbXML ... 82
X SOAP ... 82
X.1 Igény a Web szolgáltatások létrehozására ... 82
X.2 Együttműködési problémák komponens rendszereknél ... 83
X.2.1 Különböző adatreprezentáció ... 83
X.2.2 Nehézkes kliens-szerver kommunikáció ... 84
X.3 A megoldás ... 84
X.3.1 Az XML mint adatreprezentációs protokoll ... 84
X.3.2 A HTTP mint adattovábbítási protokoll ... 84
X.3.3 A megoldások megvalósítása ... 85
X.4 SOAP ... 85
X.4.1 Envelope ... 85
X.4.2 Body ... 85

X.4.3 Header ... 86
X.4.4 Adattípusok ... 86
X.4.5 Hibakezelés ... 86
X.4.6 HTTP fejléc ... 86
X.4.7 RPC és SOAP ... 86
X.4.8 Egy példa SOAP hívásra ... 87
XI Web szolgáltatások ... 88
Irodalom jegyzék ... 90

I Információs rendszerek jellemzői

I.1 Mi az információs rendszer?

Információs rendszerek = vállalati alkalmazások

Mivel foglalkozik: bérszámfejtés, ügyfélnyilvántartás, szállítmányozás, raktározás, költségelemzés, biztosítás, számlázás, költség elszámolás, hitelképesség vizsgálat

Mi <u>nem</u> tartozik az információs rendszerek körébe: vegyi folyamatok irányítása, üzemanyag befecskendezés, operációs rendszer, fordítók, szövegszerkesztő, táblázatkezelő, telefonközpont, lift vezérlés, szórakoztató elektronika, játékok

Jellemzők:

- nincsenek bonyolult szálszinkronizálási, időzítései és hardver-szoftver integrációs problémák, helyette:
- sok adat (rendszer nagy része ezek kezelésével foglalkozik): több 10 millió rekord, GB nagyságrendű adattárigény (adatok fájlokban, relációs és egyéb adatbázisokban rendezetten tárolva). Adattárolás szervezése önmagában is nagy feladat
- perzisztens adatok: a program újraindításai között is megmarad, sőt alkalmazás generáció váltásokat is túlél (import, export, biztonsági mentések és visszaállításuk, az adat kincs)
- konkurens hozzáférés (tranzakció kezelés): az akár több száz (Internet több ezer) felhasználó egyidejű adatmanipulációi nem szabad, hogy inkonzisztens állapotba vigyék a rendszert
- sok felhasználói képernyő: sok adat áttekintése és manipulációja bonyolult. Az adatokat általában többé-kevésbé laikus felhasználók kezelik
- üzleti logika: bonyolult és a legkevésbé sem logikus (complex business illogic). Ez általában a speciális (üzleti vagy marketing) szempontból hasznos kivételek és szabályoktól való eltérések következménye. Elképzelhető, hogy a hosszú időn át fennmaradó, vagy a máshonnan importált adatok egyes mezőinek eredeti jelentése nem ismert.
- integráció más vállalati alkalmazásokkal, ez adja az információs rendszerek egyik fő értékét
    - fizikai összekapcsolás szükséges (mai hálózati technológiák mellett nem probléma)
    - egyes rendszerek különféle értelmezik az azonos nevű adatoknak (conceptual dissonance)
    - az egyes rendszerek nem egy nyelvet beszélnek
    - ha valahol változik valami, az a teljes rendszerre kihathat
- nem feltétlenül nagy (sok kis projekt, integrációs spagetti), de ilykor is fontos a az összekapcsolásra tervezés, fejlesztési idő kis projekteknél általában rövid

I.2 Információs rendszerek típusai

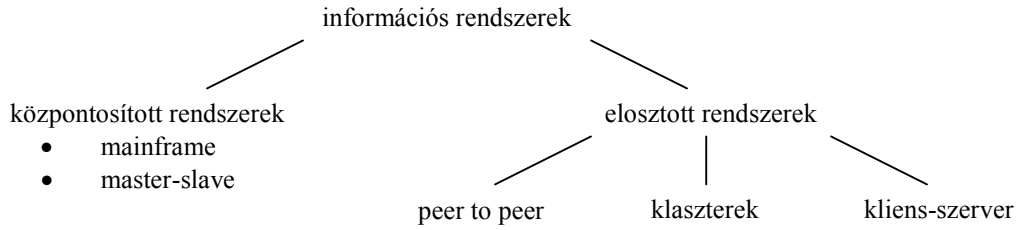
- *B2B (Business-to-business)*: együttműködés különböző vállalatok alkalmazásai között
    - szabványos kommunikációs megoldások szükségesek
    - biztonsági követelmények (adatforgalom titkosítása, partnerek jogosultságai)
- *B2C (Business-to-customer):*
    - egyszerűen kezelhető UI,
    - konkurenciakezelés (sok felhasználós rendszerek),
    - teljesítmény (a szervernek ki kell bírnia a nagyobb terheléseket is),
    - skálázhatóság (a teljesítmény az igényekhez igazítható legyen).
- Vállalaton belül
    - *Ügyfélszolgálat*
        - képzett és kevesebb felhasználó, mint az előző esetekben, de
        - a tranzakciók ideje hosszabb (másfajta konkurencia-kezelés szükséges)
        - összetettebb funkciók (kivételes események, pl. előtörlesztés, elveszett számlák kezelésére, alkalmazkodás az üzleti logikához).
    - *o Vállalaton belüli, saját nyilvántartás* (pénzmozgás követés, raktár nyilvántartás)
        - Ebben az esetben kritikus kérdés a költség (többnyire kisebb vállalkozások vágnak bele ilyen fejlesztésbe, mert a kész rendszerek túl költségesek a vállalat méretéhez képest)
        - a gyors fejlesztés, teljes kontroll az alkalmazás felett

I.3 Információs rendszerek teljesítménymérése

- *TPS (Transaction Per Second)*: Ez egy viszonylag szubjektív szám, ugyanis nem mindegy, hogy mit tekintünk tranzakciónak. Éppen ezért általában tipikus (?) tranzakció kombinációkat szoktak definiálni, hogy a tesztek egységes és összehasonlíthatók legyenek
- *egyidőben kiszolgálható felhasználók száma*
- *válaszidő (response time)*: Az az idő, ami a lekérdezés/módosítás elindítása és a válasz megérkezése között eltelik.
- *reakcióidő (responsiveness)*: Az az idő, amely a parancs kiadása és az első a visszajelzés között telik el (amíg rendszer valamilyen módon a felhasználó tudtára nem adja, hogy dolgozik). Progress bar, homokóra, várható hátralévő futási idő Sokszor ez legalább olyan fontos, mint a válaszidő, hiszen, frusztráló lehet a felhasználónak, ha nem látja, hogy történik valami
- *késleltetés (latency)*: egy tényleges feldolgozást nem végző parancs válaszideje
- *átbocsátóképesség (throughput)*: Egy bizonyos időszak alatt kiszolgált tranzakciók átlagos száma.
- *terhelés*: aktuális kihasználtság a maximális terhelhetőséghez (átbocsátó képesség, felhasználószám) viszonyítva
- *terhelés érzékenység* (pl. válaszidő változása a felhasználók vagy a tranzakció számának növekedésével)
- *skálázhatóság*: A rendszer skálázható, ha olyan kialakítású, hogy nagyobb teljesítményigény esetén könnyedén bővíthető, valamint ellenkező esetben, a kihasználatlanul erőforrások átcsoportosíthatók (a teljesítmény/feldolgozó egységek száma függvény alakulásából)
- optimalizálás szükséges,
    - de csak az első komoly konfiguráció változásig hatásos
    - a hardver bővítés olcsóbb lehet, mint a hosszasan elhúzódó finom beállítások

I.4 Információs rendszerek csoportosítása

- *Párhuzamosság*: egyidejű összehangolt működés
- *Elosztottság*: földrajzi értelemben elkülönültek, szétválasztható komponensek (lazán csatolt rendszerek)



- mester-szolga
  - mester kezdeményez
  - a szolgákban nincs feldolgozó képesség/intelligencia
- kliens-szerver
  - a kliens kezdeményez
  - a szerverek és a kliensek is intelligensek, mindkét típus végez feldolgozást

I.4.1 Központosított és elosztott rendszerek

- *történet és infrastuktúra fejlődés:*
  - mainframe, miniszámítógépek (szerverek), desktop PC-k
  - fejlődés mozgató rugói:
    - felhasználók helyi önállósági igénye (menedzsment),
    - multimédia igények felhasználói oldalon,
    - polcról levehető SW iránti igény,
    - rugalmas alkalmazkodás a piaci folyamatokhoz
    - kapcsolódás a partnerek (ügyfelek) információs rendszeréhez
    - alkalmazkodás a vállalat szervezeti ügyviteli tevékenységéhez
      - a teljesítmény magában már nem segít
      - vállalat működése (centralizáltság, formalizáltság, komplexitás)

- *központosított rendszerek:* egyetlen komoly hátránya a rugalmatlanság: nehezen bővíthető (esetleg az egészet ki kell cserélni), és a rendszer felhasználóinak önállósága korlátozott.
  - Minden adat egy helyen (kapcsolatok, integráltság)
  - Karbantartás, felügyelet
  - Fejlesztések frissítések egy helyen
  - Adminisztráció
  - Biztonság
    - fizikai (tűz, földrengés, fejsze)
    - adat megőrzés (HW és SW hibák, központi biztonsági mentések, tartalék rendszer)
    - adat lopás, változtatás (jogosultság rendszer, tűzfalak, idegen rendszerekkel való összekapcsolás veszélyei)
  - teljesítmény
  - megbízhatóság és elérhetőség (redundáns HW, 50 év SW fejlesztői tapasztalat)
  - képzett fejlesztőgárda, széleskörű felhasználói támogatás
  - de sajnos rugalmatlan

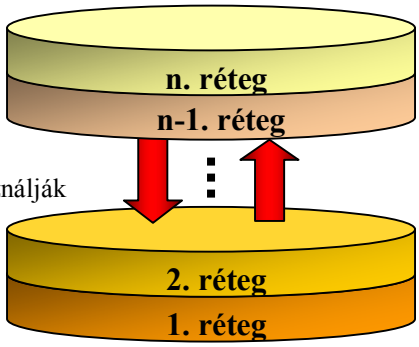
- *Elosztott rendszerek:* a központosított kezelés célja nem maga a központosítás, hanem az integráció, úgy kell tehát megtervezni az új adatkezelő rendszert, hogy az a felhasználók (adatrögzítők, alkalmazásfejlesztők, adat végfelhasználók) felé egy egységes, osztatlan arcot mutasson, bármilyen legyen is a belső megvalósítás. Óriási előny: nagyobb, olcsóbb *elérhetőség*
    - helyi önállóság (rugalmasság, alkalmazkodó képesség)
    - skálázhatóság
    - lokalizált biztonsági hiányosságok
    - megbízhatóság, elérhetőség
    - egységes rendszerkép kialakítása szükséges (integráció!!!)

I.4.2 Főbb követelmények (elosztott megvalósításnál)

Útépítés példa

- *Nagyobb teljesítmény, átbocsátó képesség*
- *Jobb erőforrás kihasználás*
- *Konfigurálhatóság, léptékezhetőség*
- *Hibatűrés*
- *Biztonság:* Autentikáció, hozzáférés szabályozás, adatvédelem
- *Egységes rendszerkép*

I.5 Jellemző felépítés: a rétegszerkezet



Rétegszerkezet felépítési szabályai:

- a rétegek csak közvetlen szomszédjaikkal kommunikálnak
- a rétegek (csak) a közvetlenül alattuk található réteg szolgáltatásait használják
- minden réteg eltakarja az alatta fekvő rétegeket a fölötte találhatók elől
- a rétegek közötti interfészek pontosan definiáltak

A rétegszerkezet kialakítása a következő előnyökkel jár:

- A rétegek a feladatot jól definiált, kisebb modulokra bontják. Minden réteg külön egységet alkot, mely a többi rétegtől függetlenül tervezhető és implementálható.
- Az egyes rétegek, az interfészek tiszteletben tartása mellett, kicserélhetők egy másik ugyanolyan funkciókat biztosító rétegre. Például a OSI 7 rétegű hálózati modelljében a legalsó, hálózati réteg az átviteli közeg (koaxiális kábel, UTP, üvegszál) függvényében önmagában kicserélhető úgy, hogy a további 6 réteg minden változtatás nélkül továbbra is működőképes marad.
- Ha rétegek közötti függőségek minimálisak (csak az eggyel alatta található rétegtől függ), a szolgáltatások alatti rétegek könnyen cserélhetők, illetve új rétegek könnyen behelyezhetők a már létezők közé.
- A rétegek remek, jól körülhatárolt lehetőséget biztosítanak a szabványosításra. A TCP vagy az IP azért szabványok, mert pontosan leírják, hogy kell működni a rétegüknek.

Bár a rétegszerkezet kialakítása kulcsfontosságú, és nagyon hasznos megközelítése a feladatok felbontásának, természetesen hátrányokkal is rendelkezik:

- A rétegek jól szétválasztanak számos problémát, de nem mindenre alkalmazhatók. Bizonyos esetekben a változásokat az összes rétegen végig kell futtatni. Információs rendszerek esetén klasszikus példa egy új adatmező felvétele, melyet az adatbázistól a felhasználói felületig mindenhol be kell illeszteni.
- A (felesleges) rétegeken történő áthaladás teljesítmény veszteséggel jár, mivel ezek a határátlépések tipikusan egyfajta ábrázolási transzformációval járnak. A funkciók azonos formában történő kezelése fejlesztési hatékonyság szempontjából azonban bőséges kárpótlást jelent.

layer: logikai szeparáció
tier: fizikai elkülönülés hangsúlyozása (a rétegek különböző számítógépeken futnak)

Rétegszerkezet kialakításának kérdései:

- Milyen rétegeket használjunk
- Mi legyen a rétegek feladata, funkciója
- Hová kerülnek a rétegek

I.5.1 Kétrétegű architektúrák

A kliens-szerver architektúra első megjelenésekor magától adódott a funkcionális elkülönítés: a közösen használt, fontos adatokat a sok kliens kiszolgálására képes adatbázis szerver tárolta, míg a kliensek az adatok megjelenítéséért és manipulációjáért voltak felelősek. Ez az elrendezés magában hordozta a központosított rendszerek összes előnyét:

- Minden adat egy helyen került tárolásra, így az adminisztráció, a karbantartás és a frissítés egy képzett rendszergazda kezében koncentrálódott.
- A biztonsági és megbízhatósági funkciók megoldása (szoftver, hardver redundancia, biztonsági mentések, hozzáférés szabályozás, fizikai védelem lopás, tűz, túlmelegedés ellen) szintén egy helyre összpontosult.
- A fejlesztések telepítése egyszerű, hiszen egyetlen kiszolgáló van a rendszerben.

2 rétegű alkalmazások

- tipikusan kliens-szerver (DBMS egyszerű funkcionalitású klienssel)
- probléma hová helyezzük a komplexebb logikát?
    - kliens: nehéz karbantartás, könnyen duplikálódhatnak az UI-be ágyazott kódok
    - szerver: tipikusan tárolt eljárások, de ezek sajnos nem gyártó függetlenek (hordozhatóság elveszik)

I.5.2 Háromrétegű architektúrák

A kétrétegű architektúrákkal kapcsolatos dilemmák megoldására az alkalmazáshoz kapcsolódó összetett funkciókat külön modulba helyezték, ezzel az üzleti logika számára külön réteg került kialakításra. A háromrétegű alkalmazások kialakításához a végső lökést a web felületen történő megjelenés iránti igény adta. A böngésző által biztosított korlátozott lehetőségek már semmiképpen sem voltak elegendők még az egyszerűbb üzleti logikák megvalósításához sem, és a harmadik, középső réteg kialakítása duplán megtérült, ha az alkalmazást hagyományos és webes felhasználó felülettel is el kellett látni, hiszen ilyenkor a középső réteg funkcióit mindkét megoldás használhatta. A háromrétegű architektúrákban a következő rétegeket különítették el:

- A ***prezentációs réteg*** (*presentation layer*) feladata a felhasználóval való kapcsolattartás. A prezentációs réteg alapvető funkciója, hogy az alkalmazás felhasználója számára szolgáltatásokat nyújtson. Lehet egy egyszerű parancssoros interfész, egy böngészőben megjelenő felület (továbbiakban vékony kliens, thin client) vagy egy összetett, multimédiás felületet biztosító, ablakokra épülő megoldás is (továbbiakban vastag kliens, rich client, fat client).
- Az ***adatkapcsolati réteg*** (*data access layer*) feladata a külső adatforrásokkal való kapcsolattartás. Az adatkapcsolati réteg fő funkciója, hogy a mások által nyújtott külső szolgáltatásokat az alkalmazáshoz illessze. Az adatkapcsolati réteg leggyakrabban relációs adatbázisokhoz kapcsolódik, de adatforrásnak számítanak az egyszerű állományok, a különféle formátumú üzenetek, vagy az XML-re épülő web szolgáltatások is.
- Az ***üzleti logika*** (domain logic, business logic) a rendszer alapvető funkcióit valósítja meg. Itt történik meg a prezentációs rétegből érkező parancsok értelmezése, és a bemenetek alapján a megfelelő számítások elvégzése és a kimeneti adatok előállítása.

3 rétegű alkalmazások

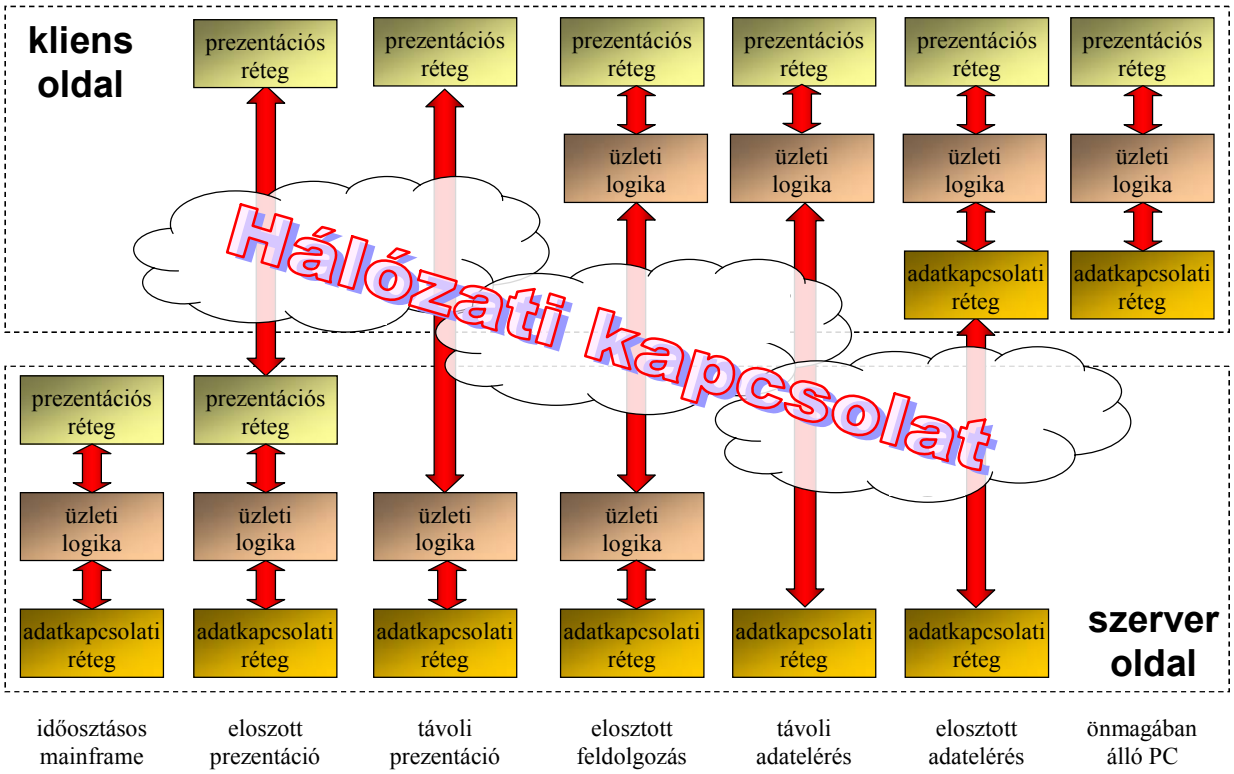
- bonyolultabb alkalmazás logikák külön modulba vitele
- fontos hajtóerő: web-es vékony kliens esetén a logika nem helyezhető el a kliensben és 3 réteg esetén sokkal egyszerűbb egy újabb web-es prezentációs réteg hozzáadása

- prezentáció: UI, kapcsolattartás a felhasználóval,
  - vékony és vastag kliens (vastag, ha grafikus, ablakozós kliens)
  - lehet gépi felhasználó is (web szolgáltatások)
  - hasonló az adat kapcsolathoz, de **szolgáltatásokat nyújt**
- adat kapcsolat: kapcsolat külső adatforrásokkal (adattár, alkalmazások)
  - DBMS, tranzakciós rendszerek
  - kommunikáció külső alkalmazásokkal (pl. üzenet alapú)
  - hasonló az prezentációhoz, de **szolgáltatásokat vesz igénybe**
- üzleti logika
  - a rendszer funkcióit valósítja meg

I.5.3 Rétegek szétválasztása

- prezentációt és üzleti logikát nem érdemes szétválasztani, ha
  - az üzleti logika majdnem semmi
  - jó fejlesztői 2 rétegű eszközeink vannak (Power Builder, Delphi, VB)
- prezentációt és üzleti logikát szét kell választani, amikor
  - UI-ben kellene adatot transzformálni
- az adat réteget és az üzleti logikát szét kell választani, ha
  - az adatok és a prezentáció struktúrája lényegesen eltér

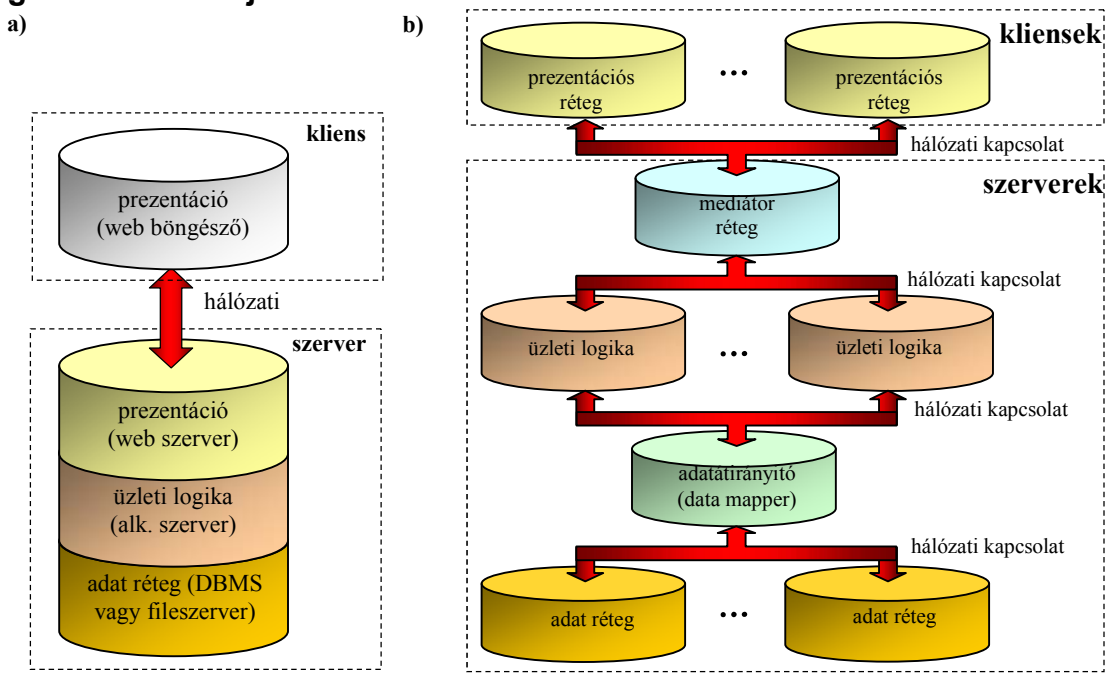
I.5.4 Rétegek elhelyezése



- két szélső rajz (a mester-szolga elven működő mainframe és termináljai, valamint az egyedül álló PC) nem kliens szerver architektúra, csupán a teljesség kedvéért kerültek az ábrára
- Minden a szerverre + web kliensek (központosított rendszerek összes előnye)
- Minden a kliensre, fő okok
  - kapcsolat nélküli működés (mobil)
  - reakcióidő (responsivness) javítása
  - távoli elérések minimalizálása (teljesítmény)

- Köztes megoldások
  - Adatréteg általában a szerver oldalon
  - Prezentáció általában kliens oldalon, de gyakran a szerver oldalra kerül (kliens csak web brower), 4 rétegű alkalmazások (kliens/prezentáció/üzleti logika/adat)
    - alkalmazások kompatibilitása
    - B2C: széles ügyfélkör, heterogén rendszerek
  - Üzleti logika mindkét oldalon lehet, de akár szét is vágható
    - Üzleti logika a kliensben: távoli adatelérés (fájl v. adatbázis rendszereket használó alkalmazások)
    - (elosztott feldolgozás, komponens alapú rendszerek, az üzleti logika a szerverek között is szétosztásra kerül)

I.5.5 A rétegszerkezet kiterjesztése



4 rétegű alkalmazások

Nagy jelentőséggel bír az elosztott prezentáció, mivel ez képezi a web alapú vékony architektúra alapját, a) ábra. Ekkor a prezentációs réteg funkciói szétosztásra kerülnek a kliens és a szerver oldal között. Ebben a négyrétegű megoldásban a kliens oldali prezentáció új nevet kap, a szakirodalomban gyakran kliens rétegként hivatkoznak rá.

5 rétegű alkalmazások

A különböző helyeken elosztva rendelkezésre álló szolgáltatások, vagy egyszerűen csak a nagyobb rendelkezésre állás vagy teljesítmény igénye miatt előfordul, hogy az üzleti logika vagy az adatkapcsolati réteg megvalósítása sok szerverrel történik. Ilyenkor újabb rétegek beiktatásával biztosítják az azonos szerepű szerverek címzési átlátszóságát, mely az átlátszó skálázhatóság (például szerver farmok), teljesítménynövelés (például cache) vagy hibatűrés (a hibás elem terhelését átveszik a többiek) megvalósításához szükséges. A szakirodalom az üzleti logikához kapcsoló szervezési réteget mediátornak (mediator), míg az adatkapcsolati réteghez tartozót adatátirányító (data mapper) rétegnek nevezi, b) ábra.

- a három fő réteg közé az elosztottság átlátszóságának biztosítása érdekében közbülső rétegek helyezhetők, *mediátor* és *data mapper* rétegek, funkciók
  - helyfüggetlen címzés
  - teljesítmény
  - terhelés elosztás)

prezentáció
mediátor
üzleti logika
data mapper
adat kapcsolat

II Információs rendszerek kliens oldalról

II.1 Vastag kliens alkalmazások

Feldolgozás túlnyomó többsége (kivéve a központi adatfeldolgozást) és a megjelenítés a kliens oldalon történik.

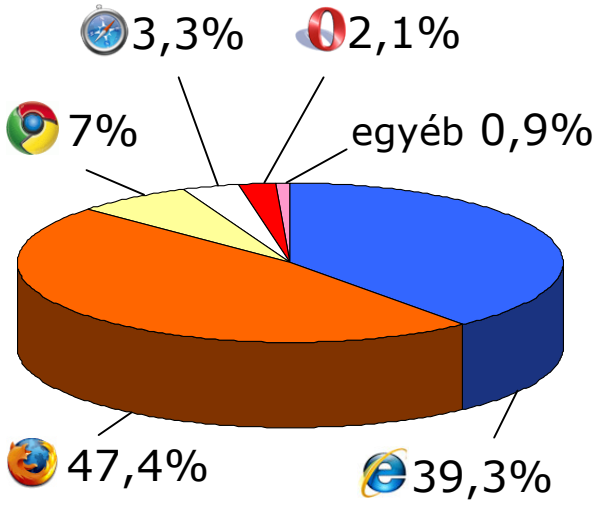
Tulajdonságok

- sok alkotóelem, szoros csatolással
    - *a szoros és laza csatolást a szerint különböztetjük meg, hogy a kapcsolódó rendszerek mennyi információt osztanak meg saját belső működésükről* (szoros: használnak közös erőforrásokat, laza: üzenetváltással kommunikálnak)
    - ennek hátrányai: nehéz telepítés, nem kívánt egymásrahatások (ld. DLL hell), verziókezelési és frissítési problémák)
- gazdag, ablakozós GUI (pl. horgonyzás, dock-olás, színek, összetett adatábrázolást segítő kontrollok, felhasználót segítő funkciók: inaktív/aktív parancsok, tooltip-ek, előnézet funkció))
- szokásos („szabványos”) alkalmazásvezérlés (menüsáv, toolbar – a szokásos funkciók a szokásos helyen; megszokott szerkezetű dialógusablakok (fileopen, színválasztás, stb.))
- multimédiatámogatás (dokumentumok és megjelenítés) – hangok, animációk, grafikák, illeszkedve a rendszer színvilágához és stílusához
- gyors reakció és válaszidő (nagy teljesítményigényű alkalmazások esetén is)
- helyi erőforrások kihasználhatósága (helyi erőforrások: processzor, memória, helyi lemezmeghajtó, alkalmazás specifikus helyi eszközök: pl. soros vonali eszközök)
- offline működés
- fejlesztőeszköz támogatás

II.2 Vékony kliens alkalmazások

Két fő formája

- Terminál alapú
  - hálózati számítógépekre (Sun JavaStations, IBM Network Station)
  - terminál emulátor típus (Citrix MetaFrame, Wyse Winterm, Windows Terminal Server)
- Webes vékony kliens (browser)



Az ***Internet Explorer***-t Microsoft 1995-ben vezette be, 1998-ban részesedése túllépte az addig piacvezető Netscape-ét.

A ***Firefox*** a Mozilla böngészője (a Netscape utódja). 2004-ben került piacra, ma az egyik legelterjedtebb böngésző.

A ***Netscape*** az első kereskedelmi böngésző. 1994-ben jelent meg, 2008 februárjában hivatalosan is befejezték a fejlesztését.

Az ***Opera*** egy gyors, kicsi nagyon szabvány követő böngésző. A kézi számítógépek és mobil telefonok egyik kedvelt böngészője.

A ***Chrome*** egy ingyenes, nyílt forráskódú böngésző a Google-től. 2008-ban jelent meg.

A ***Safari*** Macintosh rendszerek alapértelmezett böngészője. Az Apple a gyártója, és jól ismert az igényes kinézetéről.

Előnyök (mindkét típus)

- egyszerű telepítés – vagy nem is kell telepíteni (ezért egyszerű javítás, karbantartás, verziókezelés)
- *konfigurálás és karbantartás*
- kapacitástervezés (sok felhasználó együttes viselkedésének statisztikái és trendjei megfigyelhetők a szerver oldalon)
- olcsó, lassan avuló hardver
- hardver rugalmasság (különféle képességű eszközökre telepíthető kliens)
- kliens platform független, hordozható felhasználói interfész (roaming profile)
- rendelkezésre állás, mérsékelt kliens kockázatok a kliens oldalon (nincs adatvesztés, hiba esetén cserélhető a hardver)
- ellenőrzött környezet (biztonság, hatékonyság)

Hátrányok (mindkét típus)

- nagyobb válaszidők,
- rosszabb felhasználói élmény,
- korlátozott felhasználói szabadság (az ellenőrzött környezet miatt)
- kapcsolat függőség
- korlátozott számítási teljesítmény

Webes vékony kliens (browser) előnyök

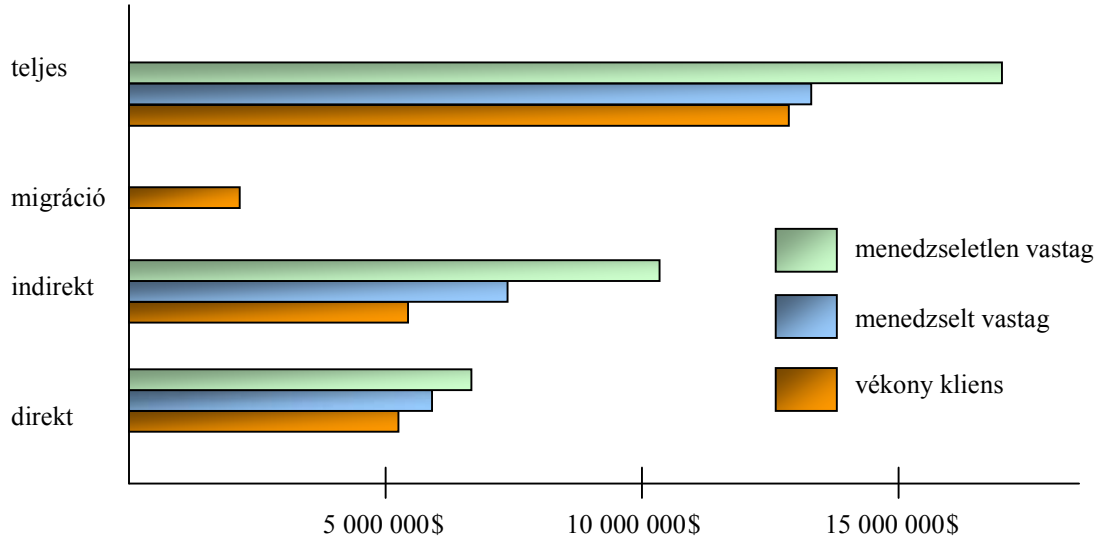
- széles közönség elérése szabványos módon a kliens típusától függetlenül
- skálázhatóság, sok felhasználós (állapotmentes protokollok, web szerver klaszterezhető)
- Egyszerű kezelés
- Hálózathoz kapcsolódó funkciók (pl. keresés, google map, szótár)
- Könnyen beágyazható multimédia elemek

Webes vékony kliens (browser) hátrányok

- kevésbé kiforrott fejlesztő eszközök
- a böngészők nem egységesek, inkonzistens alkalmazás környezet
- nincsen „szokásos" felhasználói felület – cél a figyelemfelhívás, de mindennapos használatra célszerűbb a funkcionalitás előtérbe helyezése
- egyszerű vagy nem hordozható GUI
- konzisztens állapotmentés lehetősége hiányzik (a session-ök közötti adatok tárolására nincs lehetőség)

Egy tanulmány foglalkozik azzal a témával, hogy egy cég konkrét rendszerében milyen nyereség érhető el, ha a meglévő klienseket vékony kliensekre cserélik. Lowber az elemzést 2001-ben, a Gartner Ti2 szoftverével végezte, 2500 felhasználót és 35 szervert tételezve fel, melyeket a felhasználók egy központi és két távoli telephelyről érnek el

Az áttérés direkt költségei között szerepelnek, hogy új terminálokat kell beszerezni (bár a továbbiakban nem kell költeni a HW fejlesztésére, cseréjére), illetve az alkalmazásokat át kell alakítani (de valószínűleg nem kell teljesen új programot írni). Indirekt költség, hogy az alkalmazottakat át kell képezni az új rendszer kezeléséhez, illetve terméktámogatást (support-ot) kell biztosítani, cserébe hatékonysági javulás várható. A tanulmány szerint vastag kliensről vékonyra áttérve 33%-os nyereség érhető el, míg menedzselt vastag kliensről vékonyra áttérve mindössze 1%.



II.3 Okos kliens alkalmazások

II.3.1 SOA – Service Oriented Architecture

Jellemzők

- egyszerű, általános interfész: a SOA egy szolgáltatás rendszer, a szolgáltatások elérésére szabványos interfészen történik
- laza csatolás: a SOA laza csatolású elosztott szolgáltatás rendszer, így az elérhető szolgáltatások listáját futási időben le lehet kérni
- üzenet alapú kommunikáción alapul
    - deszkriptív (nem instruktív) – leíró jellegű üzenetek
    - egyszerű formátum, korlátozott szókészlet (ezáltal egyszerű interfészek)
    - cél a bővíthetőség (az egyszerűség és a bővíthetőség összeegyeztethető, ld. XML)
- regisztrációs adatbázis (a szolgáltatások „telefonkönyve”)

II.3.2 Okos kliens alkalmazások jellemzői

Komponensek, szolgáltatások laza csatolásával áll össze (modularitás előnyeit használjuk ki, összekapcsolás üzenetalapú, szabványos interfészekkel). A szolgáltatás orientált architektúra lényegében olyan elosztott szolgáltatások rendszerére épít, melyek egy korlátozott formátumú, közös nyelvet használnak, azonban ez utóbbi tetszőlegesen bővíthető tartalmak leírására képes. A koncepció feltételezi a távoli használat lehetőségét, a több platformúságot és a laza csatolást, valamint törekszik arra, hogy a szolgáltatás végrehajtásának belső megvalósítását minél inkább eltakarja a szolgáltatás igénybevevők elő. A szerver oldal ilyen megvalósítása a kliensek kialakítására is jelentős hatással van.

Okos kliens követelmények

- helyi erőforrások használata (az okos kliens alkalmazás mindig rendelkezik egy a kliens oldali gépen futó kódrészlettel is, ld. vastag kliens előnyök)
- hálózati erőforrások használata
- nem folytonos kapcsolatú felhasználó támogatása
- intelligens SW frissítés (vastag klienseknél ezt a felhasználónak kell elvégeznie): online felkutatható, automatikusan feltelepülő komponensek, „one click deployment”
- kliens oldali eszközök rugalmas támogatása (nem az erőforrásokról van szó, hanem pl. mobiltelefon, PDA, stb.)

Okos kliens alkalmazások: a vékony és a vastag kliens előnyeinek ötvözése

Okos kliens típusok

- saját egyedi felülettel rendelkező okos kliensek (pl. egy teljes alkalmazás – Outlook)
- irodai alkalmazásokhoz kapcsolódó okos kliens (lehet, hogy nem is kell hozzá GUI, pl. egy Business Flow Control-hoz kapcsolódó feladat elvégzését segítő beépülő modul)
- mobil okos kliens (mobil eszközökhöz kapcsolódóan adatok gyűjtésére)

Jellemző	Vékony kliens	Okos kliens
Gazdag felületet biztosít	Igen, de bonyolult a fejlesztés, tesztelés és hibakeresés. Az összetett felhasználói felület rontja a hordozhatóságot, és az alkalmazást egyetlen böngészőtípushoz köti.	Igen, egyszerűbb a fejlesztés, tesztelés és hibakeresés.
Képes a helyi hardver erőforrások használatára	Igen, de csak speciális komponenseket keresztül (például COM).	igen
Együttműködés más helyi alkalmazásokkal	nincs	van
Többszálúság	nincs	van
Kapcsolat nélküli működés	nincs	van
Jól működhet nagy hálózati késleltetés és kis sávszélesség esetén	nem	igen
Könnyű telepíteni	igen	Változó. Az alkalmazás konkrét követelményein múlik.
Alacsony karbantartási és verzió váltási költségek	igen	Változó. Az alkalmazás konkrét követelményein múlik.
Különféle képességű hardver eszközökre is telepíthető	Igen, de a bonyolultabb alkalmazások egyetlen böngészőtípushoz kötődhetnek.	Igen, de a keretrendszerüknek működni kell az adott hardveren.

A vékony és az okos kliensek jellemzőinek összehasonlítása

II.3.3 Mobil kliens megoldások

Az utóbbi időben a mobil készülékek rohamos térhódítása figyelhető megaz informatika szinte minden területén. A mai fejlettebb mobiltelefonok számítási teljesítménye megközelíti egy 5-6 évvel ezelőtti asztali számítógépét. Az egyre növekedő háttértár kapacitás és a szabványos kommunikációs protokollok széles választékának támogatása révén ezek a készülékek már inkább tekinthetők kisméretű számítógépeknek, mint telefonoknak.

A kisebb (kijelző)méretű, korlátozottabb számítási és memória kapacitással rendelkező mobil készülékeken futó kliens alkalmazások szigorú értelemben nem alkotnak új kategóriát a vékony kliensek és a vastagkliensek mellett, leggyakrabban ezen architektúrák keverékeként működnek (hibrid kliensek). Mindamellett akár a hagyományos vékony kliens alkalmazások is közvetlenül futtathatók lehetnek mobiltelefonokon, hiszen a modernebb készülékek már nagyobb kijelzővel, teljes HTML támogatással és viszonylag gyors böngészőprogramokkal vannak ellátva, melyek lehetővé teszik az egyszerűbb web alkalmazások futtatását is.

Bár a mobil kliensek számos korlátozással rendelkeznek a nagyobb számítógépeken működő alkalmazásokkal szemben, a mobilitás mégis olyan komoly előnyt jelent, mely a legtöbb esetben a komolyabb kompromisszumokat is megér. Egyre több, hagyományosan vastag vagy vékony kliens architektúrára épülő, rendszerhez jelennek meg mobil kliensek, lehetővé téve, hogy a felhasználó mobiltelefonja segítségével útközben is elérje azokat a szolgáltatásokat, amiket eddig csak a számítógépe mellől használhatott (pl. blog készítő weboldalakhoz, közösségi site-okhoz, képmegosztóhoz kiadott mobil kliens). Ezek leggyakrabban egy külön installálandó (tulajdonképpen vastag kliens) alkalmazás révén teszik elérhetővé a rendszer szolgáltatásait, egy mobiltelefonokra tervezett felhasználói felülettel rendelkező különálló alkalmazás révén. A másik lehetőség, a vékony kliens alkalmazások mobiltelefonokon való elérhetővé tételéhez, a külön mobil böngészőkre tervezett weboldalak bevezetése, ilyen például a Google levelező rendszeréhez elérést biztosító Mobile Gmail.

II.3.3.1 A mobil kliensek jellemzői

- **Inkozisztens platformok:** Jelenleg négy nagyobb mobil szoftverplatform létezik és ezek teljes mértékben inkompatibilisek egymással. Egy Symbian OS-re írt szoftvert nem fut a Linux-os vagy a Windows Mobile-ra épülő készüléken. Az egyetlen kivétel a Java Mobile Edition, melyet elméletileg a készülékek jelentős hányada támogat, azonban a támogatottság foka készülékenként itt is jelentősen eltér (pl. nem minden kommunikációs protokoll áll rendelkezésre, vagy akár a fájlrendszer teljes hiánya is elképzelhető bizonyos egyszerűbb telefonoknál). Vannak törekvések egységes mobilplatform létrehozására, de ezek jelenleg inkább kísérleti projekteknek tekinthetők, mint életképes, üzleti alkalmazások készítésére alkalmas megoldásoknak. Manapság ha valaki széles körben használható alkalmazást szeretne készíteni, akkor a mobil klienst legalább két-három platformra külön-külön el kell készítenie.

- **Kisméretű, inkonzisztens kijelzők:** A hordozhatósághoz elengedhetetlen a minél kisebb készülékméret, azonban ez maga után vonzza a kijelző méretének lecsökkenését is. Egy kisméretű kijelzőre egészen más felhasználói felületet kell tervezni, hogy az kellő információt hordozzon és mindemellett mégis áttekinthető maradjon. Meg kell birkóznia a fejlesztőknek a készülékenként eltérő méretű kijelzők problémájával is. Bár a legtöbb szoftverplatform igyekszik megkönnyíteni a programozó dolgát, és automatikusan elvégzi a vezérlőelemek méretezését és pozícionálását, azonban ez nem minden esetben vezet tökéletes eredményhez. Gondoskodni kell továbbá a grafikus elemek (pl. képek, ikonok) több méretben való elkészítéséről is. Ezen utóbbi problémát több platformon vektorgrafikus elemek bevezetésével oldották meg (pl. SVG formátumú képek), melyeket a rendszer a kijelző méretétől függően információvesztés nélkül átméretezhet.

- **Eltérő beviteli eszközök:** A mobilkészülékek nemcsak eltérő méretű kijelzőkkel, de eltérő beviteli (input) perifériákkal is rendelkeznek. Itt a mobil kliensek készítésekor elsősorban két fő típust kell megkülönböztetnünk: a hagyományos, billentyű alapú felhasználói felületet támogató mobilkészülékeket, illetve az újabban egyre gyakoribb érintőképernyős megoldást alkalmazókat. Nyilvánvaló, hogy azokon az eszközökön, ahol csak a gombokkal lehet navigálni egészen más szempontokra kell oda figyelni a vezérlőelemek definiálásánál, mint az érintőképernyős készülékeknél.

- **Gyengébb hardver:** Bár ez a fejlett okostelefonoknál ez egyre kevésbé jelentős szempont, a tömegkészülékek teljesítménye még mindig jelentősen elmarad az asztali számítógépekétől A (alacsony fogyasztás, hosszabb akkumulátor élettartam miatt alkalmazott) gyengébb processzor mellett a kisméretű memória okozhat leginkább problémát adatorientált alkalmazásoknál. Fontos szempont lehet a szoftverek kialakításakor, hogy mekkora adatmennyiséget kell feldolgoznia a kliensnek az egyes szerver felé indított lekérdezések után.

- **Korlátozott üzemidő:** Mivel a mobil készülékek akkumulátorról üzemelnek, ezért egy alkalmazás tervezésekor fontos szempont az energiafogyasztás minimalizálása. Egy mobil kliensnél nem célszerű állandó aktív hálózati kapcsolatot fenntartani, túl gyakran szinkronizálni vagy folyamatosan számításigényes feladatokat végezni, mert ezek könnyen az akkumulátor gyors lemerüléséhez vezethetnek. A különféle vezeték nélküli hálózati interfészek aktív használata különösen nagy terhelést róhat az akkumulátorra.

- **Kommunikációs protokollok széleskörű támogatása:** A legtöbb mai mobilkészülék legalább 3-4 különféle vezeték nélküli hálózati interfészt támogat, kiemelnénk itt a WLAN-t, a Bluetooth-t és természetesen a GSM alapú GPRS-t ill. a különféle 3G-s internet elérési megoldásokat. Ezen technológiák révén a mobiltelefonok szinten minden helyzetben képesek lehetnek a szerverrel való kommunikációra. Fejlettebb országokban egyre általánosabb az átalánydíjas mobilinternet terjedése, ami lehetővé teszi, hogy plusz költség nélkül a készülék bárhol rákapcsolódhasson az Internetre, ahol csak van hálózati lefedettség. A HTML, és általában az XML támogatása is egyre gyakoribb, melyek révén a mobil kliens is ugyanazon a az interfészen keresztül éri el a szervert, mint a többi kliens alkalmazás.

II.3.3.2 Mobil kliensek teljesítménye

Az 1. táblázatban különféle mobilkészülékek, illetve egy mai modern asztali számítógép számítási teljesítményét mérjük össze. A π–t adott tizedes jegyig kiszámító algoritmus futásideje jól szemlélteti a nagyságrendbeli különbséget a PC, valamint a mobiltelefonok processzorának teljesítménye között. Továbbá ami még megfigyelhető, hogy egy a JAVA-ban írt alkalmazás csak fele olyan jól teljesített ugyanazon a készüléken mint a natív Symbian operációs rendszerre készített változat.

Készülék típusa	**Nokia N93**	**Nokia N91**	**Nokia 6630**	**SE K750i**	**PC**
Processzor	ARM11 332 MHz	ARM9 220 MHz	ARM9 220 MHz	ARM9 110 MHz	Intel Core 2 Duo 1830 MHz
PI-JAVA	3.702 sec	5.647 sec	5.703 sec	7.18 sec	0.032 sec
PI-Symbian	1.695 sec	2.690 sec	-	-	-

1. táblázat – Mobilkészülékek teljesítményének összehasonlítása

II.4 Az architektúrák tulajdonságainak összevetése

A 2. táblázatban összefoglaltuk a vékony, vastag, illetve a mobil kliensek általános tulajdonságait. Itt az önálló mobil kliens alkalmazásokat (tehát nem a mobil böngészőt használó vékony klienseket) vizsgáljuk. Megfigyelhető a párhuzam a mobil kliensek és a vastag kliensek között, a fő különbségek a mobil kliensek kisebb hardver kapacitása, a szoftverplatformok inkonzisztenciája, valamint a felhasználói felület eltérésének következményei.

	Vastag kliens	**Vékony kliens**	**Mobil kliens (vékony és vastag is lehet)**
Válaszidő	Gyors	Lassabb	Lassabb
Offline működés	Igen	Nem	Igen
Felhasználói felület	Megszokott, konzisztens	Inkonzisztens	Inkonzisztens
Frissítések	Nehézkes	Könnyű	Nehézkes
Üzemidő	Korlátlan	Korlátlan	Korlátozott
Fejlesztés	Megszokott környezet, nagy bonyolult alkalmazások, sok komponenssel	Kevésbé kifinomult fejlesztő eszközök, inkompatibilis kliens oldali eszközök	Kevésbé kifinomult fejlesztő eszközök, szimulátorok, inkompatibilis kliens oldali eszközök
Számításigényes funkciók kliensben	Igen	Nem	Nem

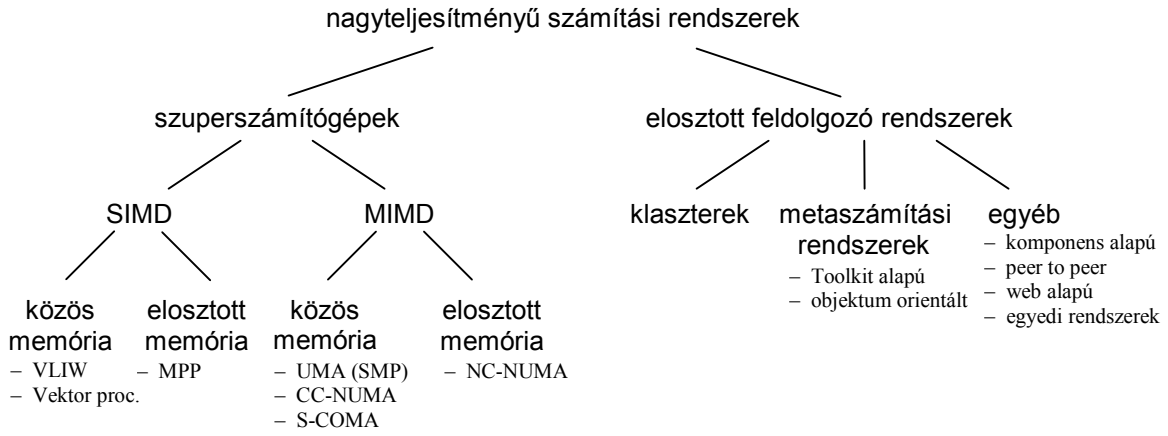
2. táblázat – A különféle kliens megoldások összehasonlítása

III Párhuzamos számítógép architektúrák

III.1 Teljesítmény növelésének módszerei

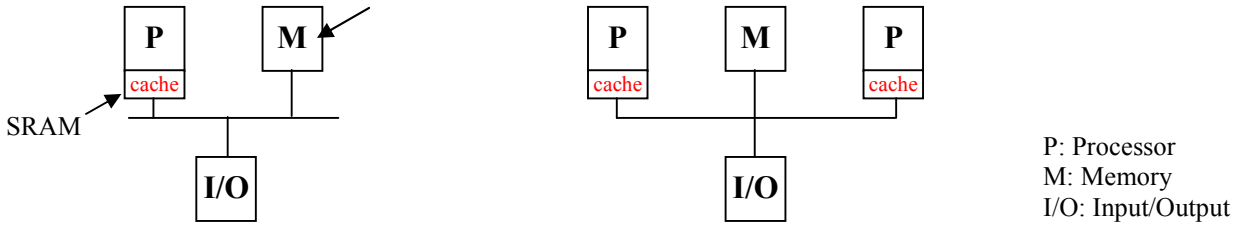
Ha adott egy feladat és annak megoldását szeretnénk valamilyen módon meggyorsítani, akkor három fő alternatíva között választhatunk.

- Többet dolgozni (processzor teljesítménye 18 havonta duplázódik, meddig folytatódhat?)
- Okosabban dolgozni (algoritmusok)
- Segítséget szerezni (párhuzamosság a jövő útja)



III.2 Neumann architektúra és cache

Neumann architecture:



SRAM	DRAM
- same speed as processor - expensive - needs a lot of space on the chip	- higher density - needs periodic updates - 10 x cheaper than SRAM - 10 x slower than SRAM

1 instruction <=>

1.5 memory access
x 2 for virtual memory access
x 4 superscalar processor
x 10 because memory is slower

= 120 (difference between processor speed and memory speed)

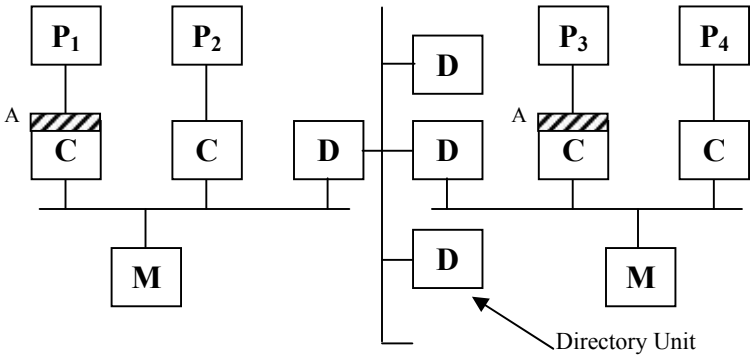
III.3 Párhuzamos számítógép architektúrák

- *Flynn csoportosítása*
  SIDI (hagyományos), MIMD(superskalár), MISD(pipeline), SIMD (vektor)

- *MIMD típusú számítógépek*

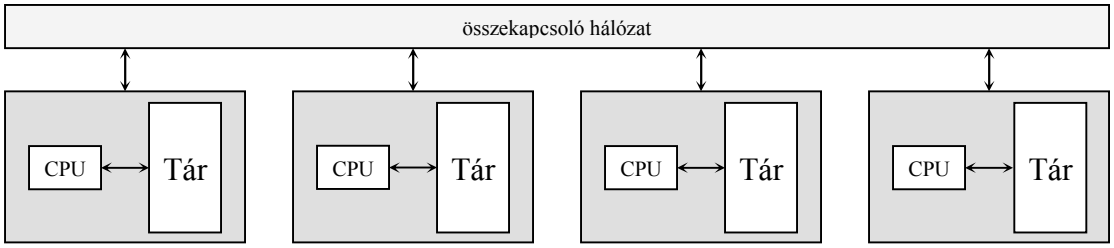
UMA = Uniform Memory Access (SMP)
NUMA = Non-Uniform Memory Access

"snoopy bus"

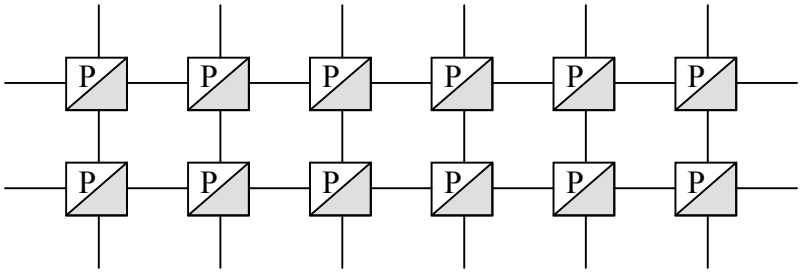


- *NUMA:*
  - CC-NUMA cache coherent(~ shared memory)
  - NC-NUMA non-coherent (~ LAN)

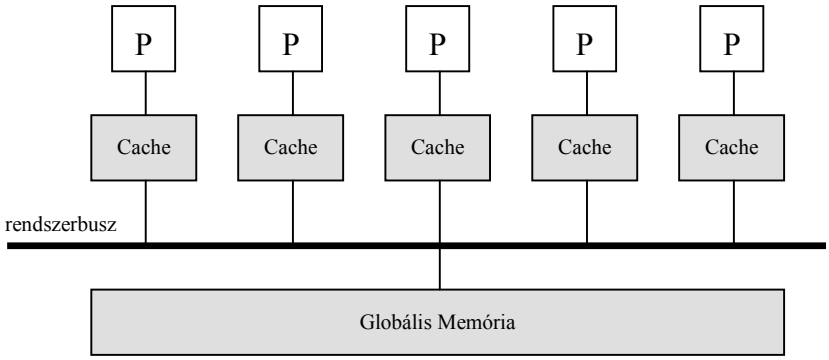
- *Multiszámítógép, Általánosított Neumann modell*



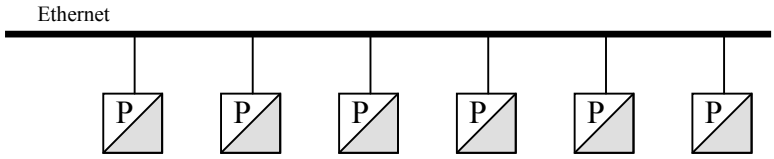
- *Elosztott memóriás MIMD számítógép, hálóba kapcsolva*



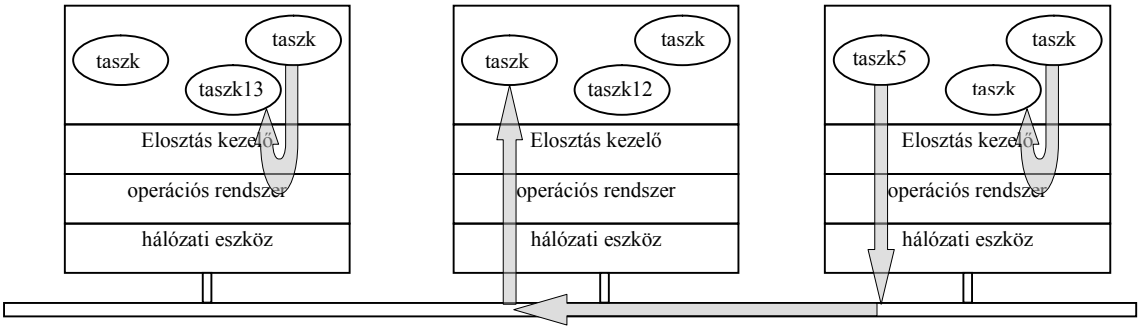
- *Osztott memóriás multiprocesszor modell*



- Hálózatba kapcsolt számítógépek



- Rétegszerkezetű virtuális gép (shared-nothing)



III.3.1 Mai párhuzamos architektúrák

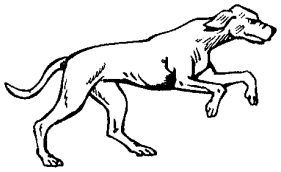
fejlődés: 1997 a szuperszámítógép éra vége, Cray Research felvásárlása 2 fő ok

- számítógépek teljesítménye gyorsan nő (164% évente, 9 havonta duplázódik, proc. órajel, architektúra, Pfister 10.old), a szükséges processzorok száma ezzel együtt csökken (nem kell *masszívan* párhuzamos megoldás), sőt gyakran 1 processzor is bőven elég
- Folyamatos elérhetőség igénye, *katasztrofális a meghibásodás* (űrkutatás, orvosi alkalmazások, tőzsde), *költséges leállás* (központi szerver, számlázó rendszer, internet), *olcsó gépek*, vegyünk kettőt (a gép nem olyan olcsó és elég megbízható, ezért a tartalék se porosodjon -> csináljunk klasztert)

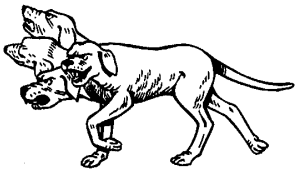
mai párhuzamos architektúrák:

- hagyományos masszív párhuzamos szuperszámítógépek, SMP, NUMA, klaszterek
- előnyök, hátrányok vannak, de az nyer akinek nagyobb a hangja, mert *különböznek a programozási modellek* egymás programját hatékonyan futtatni képtelenek, így a gyártók olyan típusú hardvert gyártanak, amihez van jó szoftver, illetve olyan szoftvert írnak, amihez sok hardver van (pozitív visszacsatolás, ördögi kör)

PC



SMP



Klaszter

	PC	Szuper	SMP	Klaszter
Teljesítmény	egységnyi	nagy	közepes (memória, I/O korlátoz)	nagy (a hálózat és a kommunikáció korlátoz)
Skálázhatóság	nincs	rossz (rugalmatlan)	korlátozott (HW kiépítés)	kiváló (SW kell csak hozzá)
Elérhetőség	egységnyi	jó	egységnyinél rosszabb	kiváló (SW kell csak hozzá)
Adminisztráció	egységnyi	egységes, egyszerű	egységes, egyszerű	speciális program kell hozzá
Licenszek száma	1 darab	speciális	1 darab	minden gépre külön kell
Biztonság	hagyomány os	jó	hagyományos	hagyományos + titkosítás (speciális megoldások)
SW ellátottság	sok	speciális	sok (PC)	speciális (Web serv, DB)
Számítási teljesítmény ára	1	>>1	>1	~1

Teljesítmény: az adott architektúrával elérhető legnagyobb számítási kapacitás. Ha egy átlagos PC-hez hasonlítjuk a többi rendszer teljesítményét, akkor a szuperszámítógép teljesítménye kimagaslóan nagy lesz, hisz az adott feladatra optimalizálták. Az SMP a PC-hez képest nagyobb teljesítményt mutat, de az elosztott erőforrásokon való osztozkodás a rendszer teljesítményét korlátozza. A klaszterben az egyes gépek jól tudnak egymás mellett dolgozni, csak a közös kommunikációs rendszer használata okoz korlátokat.

Skálázhatóság: Egy PC-t csak alacsony szinten lehet skálázni. Mert nem építhető plusz új elem bele, csak a meglévőket lehet gyorsabbakra cserélni, de így is meg kell tartani az interfészeket és a kompatibilitást a meglévő elemekkel. A szuperszámítógép korlátozottan bővíthető. Csak az előre megtervezett és megvalósított bővítési lehetőségeket lehet kihasználni. Például a még be nem töltött helyekre újabb processzorokat szerelni. Az SMP csak korlátozottan bővíthető, mert előre rögzített a hardver struktúrája, maximális processzorszám, rendszer memória méret. A klaszter tetszőlegesen bővíthető újabb rendszerbe kapcsolt számítógépekkel, szinte folytonos skálán, csak megfelelő szoftver kell hozzá, mely irányítja, kézben tartja a rendszert és elosztja a feladatokat.

Elérhetőség: PC esetén, miután minden alkatrészből csak egy van. Itt nincs külön hibajavítás és védelmi rendszer. De épp ezért csak kevés minden romolhat el. Szuper számítógép esetén jó a hibatűrő képesség, hisz ha megfelelően lett tervezve, akkor ellen kell tudjon állni. SMP-nél a PC-hez hasonló a hibatűrés, de azért annyival rosszabb, hogy itt több hibalehetőséget rejt magában a több processzor, bonyolultabb HW. A sok processzor közül elég, ha egy elromlik. Miután minden erőforrás közös, az a többi működését is előbb-utóbb lehetetlenné teszi. A klaszter esetén a hibatűrés kiváló, hisz teljesen elosztott a rendszer, az egyes elemek nem szűkítik egymás sávszélességét, hiba esetén átvehetik egymás feladatát. Természetesen ehhez szoftveres, hardveres támogatás is kell. Mint például: rendszer-adminisztrációs támogatás és folytonos monitorozás.

Adminisztráció: PC, SMP esetén nincs semmilyen különleges feladat. A szuperszámítógép és a klaszter speciális támogatást igényel, mert a feladatok elosztását nyomon kell követni, egyenletes teherelosztást kell biztosítani, és az egyes feladatok elvégzését is nyilván kell tartani.

Licenszek száma: A PC és SMP esetén egy darab kell, hisz csak egy operációs rendszer fut és a futtatott programokból is elég egyetlen példány. A szuper számítógépre természetesen speciális licence kell. A klaszter esetén viszont minden egyes gépre külön-külön kell egy licence, hisz minden gép független operációs rendszert futtat.

Biztonság: A PC és SMP esetén miután önálló gépek a külső rendszerből való támadás, lehallgatás nehézkes. Elegendő a megszokott biztonsági előírásokat betartani. A védelmet jelszó és tűzfal biztosíthatja. A szuperszámítógép biztonsága egyedi, jól tervezett. A klaszterben viszont komoly problémát okoz az egyes gépek közötti távolság, és a szabványos kommunikációs vonalak használata. Mindemellett a klaszterben sok kommunikáció folyik a hálózaton, ezért azt mindenképp titkosítani kell az átküldött adatokat.

Szoftver ellátottság: A PC-hez rengeteg szoftver kapható, vagy akár fejleszthető is. A szuperszámítógép speciális szoftvereket igényel, hisz egyedisége miatt rá kell szabni az optimális működéshez. Az SMP képes futtatni a PC-s szoftvereket, igaz ekkor nem használják ki a képességeit teljesen. De speciálisan rá fejlesztettekkel ez megtehető. Klaszterre már kaphatóak Web szerver, adatbázis kezelő szoftverek. A lehetőségek tárháza még igen szűk, de dinamikusan bővül a kínálat.

Számítási teljesítmény ára: A szuperszámítógép egyedisége miatt nagyon drága az egységnyi számítási kapacitása, igaz cserébe robosztus teljesítményt ad. Persze csak az adott feladatkörben. Az SMP esetén néhányszor többe kerül egy sima PC-nél a működéséhez szükséges speciális kialakítások miatt. De itt is nagyobb a kapott teljesítmény. A klaszter ára viszont erősen függ a becsatlakozó gépek számától. De, ha ezek a gépek már adottak, és csak össze kell őket kötni, akkor egész olcsó is lehet. Persze klaszter esetén nem kell minden géphez monitor és billentyűzet. Ha nem akarunk felhasználói felületet az adott géphez, akkor megspórolható.

IV Klaszterek

IV.1 Mi a klaszter?

A klaszter architektúra célok és felépítés szempontjából valahol az elosztott memóriát használó DM-MIMD szuperszámítógép architektúrák és a számítási teljesítményt hangsúlyozó elosztott rendszerek között helyezkedik el. Mindkét rendszerhez hasonlít abból a szempontból, hogy

- nagy számítási teljesítmény elérésére törekszik,
- a problémák megoldását párhuzamosítás alkalmazásával képes felgyorsítani,
- a csomópontok lazán csatoltak, a végrehajtó egységek üzenetalapú kommunikációt használnak,
- a hibatűrés és rendelkezésre állás fontos tervezési szempont,
- cél, hogy csomópontok elvétele (kiesése) esetén a rendszer teljesítménye fokozatosan romoljon (*gracefull degradation*), és újabb erőforrások hozzáadásakor fokozatosan nőjön (skálázhatóság).

Ugyanakkor a klaszterek lényegesen különböznek mindkét említett rendszertől. Eltérések a DM-MIMD architektúrához képest:

- Míg DM-MIMD szuperszámítógépek többszörözött processzor-memória párokból épülnek fel, addig a klaszterek építőelemét teljes kiépítésű számítógépek (processzor, cache, memória, I/O egységek) alkotják.
- A DM-MIMD rendszer memória csatornákat használ az üzenet alapú kommunikáció megvalósításához, a klaszter csomópontok tipikusan szabványos LAN hálózaton keresztül kapcsolódnak össze.
- Az egységes felügyeleti szoftverrel rendelkező szuperszámítógépekkel ellentétben a klaszterek minden csomópontja külön operációs rendszert futtat.

Az elosztott feldolgozást végző metaszámítási rendszerek és a klaszterek a következő alapvető különbségeket mutatják:

- Míg a metaszámítási rendszerek földrajzilag elosztottak, a klaszterek csomópontjai egy konkrét földrajzi helyen, egymás közelében találhatók.
- A klaszter csomópontjai személyi számítógépek vagy munkaállomások. A metaszámítási környezet csomópontjai tetszőleges számítógépek lehetnek, ideértve a klasszikus szuperszámítógépeket és a klasztereket is.
- A klaszterek általában homogén felépítésűek (nem mindig), a metaszámítási rendszer viszont tipikusan heterogén (nem mindig).
- A klaszter komponensei (csomópontjai) a külvilág számára nem láthatóak. A metaszámítási rendszerek elemei megkülönböztethetők és külön-külön címezhetők.

A fentiek alapján tehát a klaszterek olyan nagyteljesítményű párhuzamos számítási rendszerek, melyek a következő tulajdonságokkal rendelkeznek:

1. A klaszter teljes kiépítésű számítógépek (processzor, memória, I/O egységek) összekapcsolásával jön létre.
2. A klaszter csomópontjai lazán csatoltak, és a csomópontok összekapcsolása tipikusan helyi hálózat segítségével történik.
3. A klaszter egységes egyesített számítási erőforrásként használható.

Az első két pont a felépítésre utal, és leginkább a szuperszámítógépektől való eltéréseket hangsúlyozza, míg a harmadik pont a metaszámítási rendszerektől történő elkülöníthetőséget szolgálja.

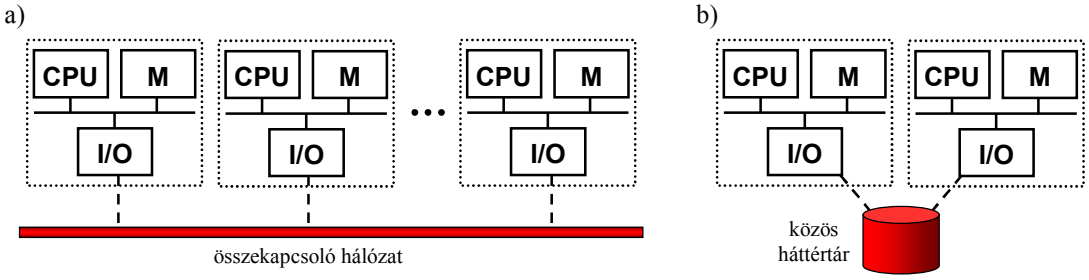
IV.2 Klaszter architektúrák

A különféle klaszterek hardver felépítés szempontjából jelentősen eltérhetnek egymástól, de a klaszter csomópontok összekötése tipikusan valamilyen I/O szintű kapcsolaton keresztül valósul meg. Az I/O szintű csatolás jóval lassabb a szuperszámítógépekben alkalmazott, processzorok és memóriák között kialakított nagysebességű buszoknál, de kiküszöböli annak fő hátrányait:

- a gyors buszok nagy számban történő összekapcsolása bonyolult és költséges,
- az építőelemek illesztéséhez architektúránként eltérő, különleges hardver szükséges,
- az operációs rendszerhez speciális memória kezelő szoftvert kell kidolgozni,
- a memóriakezelési többletköltségek (hozzáférés védelmek, szegmentálás, virtuális memória kezelés, cache) csökkenti a teljesítményt (korlátozott skálázhatóság).

Az I/O alrendszeren keresztül való kapcsolódás előnye, hogy a ki- és beviteli rendszert pontosan a különféle rendszerek egymáshoz kapcsolására tervezték, így a csomópontok összekapcsolására történő felhasználása könnyen kivitelezhető, olcsó megoldás. Nem elhanyagolható előny az sem, hogy az operációs rendszerek már eleve tartalmazzák az ilyen eszközök kezeléséhez szükséges illesztő programokat, így akár heterogén rendszerek esetében is külön erőfeszítések nélkül, azonnal működőképes. Az I/O csatolt megoldások hátránya a memóriabuszénál kisebb sávszélesség és a nagyobb késleltetések.

Az I/O csatolt rendszerek együttműködése, az ábrán látható módon, megszervezhető mind osztott erőforrás, mind üzenet alapú kommunikáció segítségével. A közös erőforrás (tipikusan diszk alrendszer) használata szűk keresztmetszetet jelent, ezért ez a megoldás csak a kicsi (maximum négy csomópont) vagy direkt a hibatűrés céljából létrehozott rendszerekben (például melegtartalék képzés) alkalmazható. Ilyen elven szerveződnek az adatbáziskezelő rendszerek klaszteres változatai is, ahol a tranzakciós naplókat a perzisztencia kezelés miatt már úgyis eleve a háttértárra kell írni.
Az üzenet alapú megoldással jóval nagyobb mértékű skálázhatóság érhető el. Ebben az esetben az I/O alrendszerek összekapcsolása valamilyen kommunikációs hálózaton keresztül valósul meg. Az összekapcsolható csomópontok számát csak a kommunikációs sávszélesség igény korlátozza, a különféle Internetes route-olási megoldásokkal fizikailag számítógépek milliói is összekapcsolhatók.

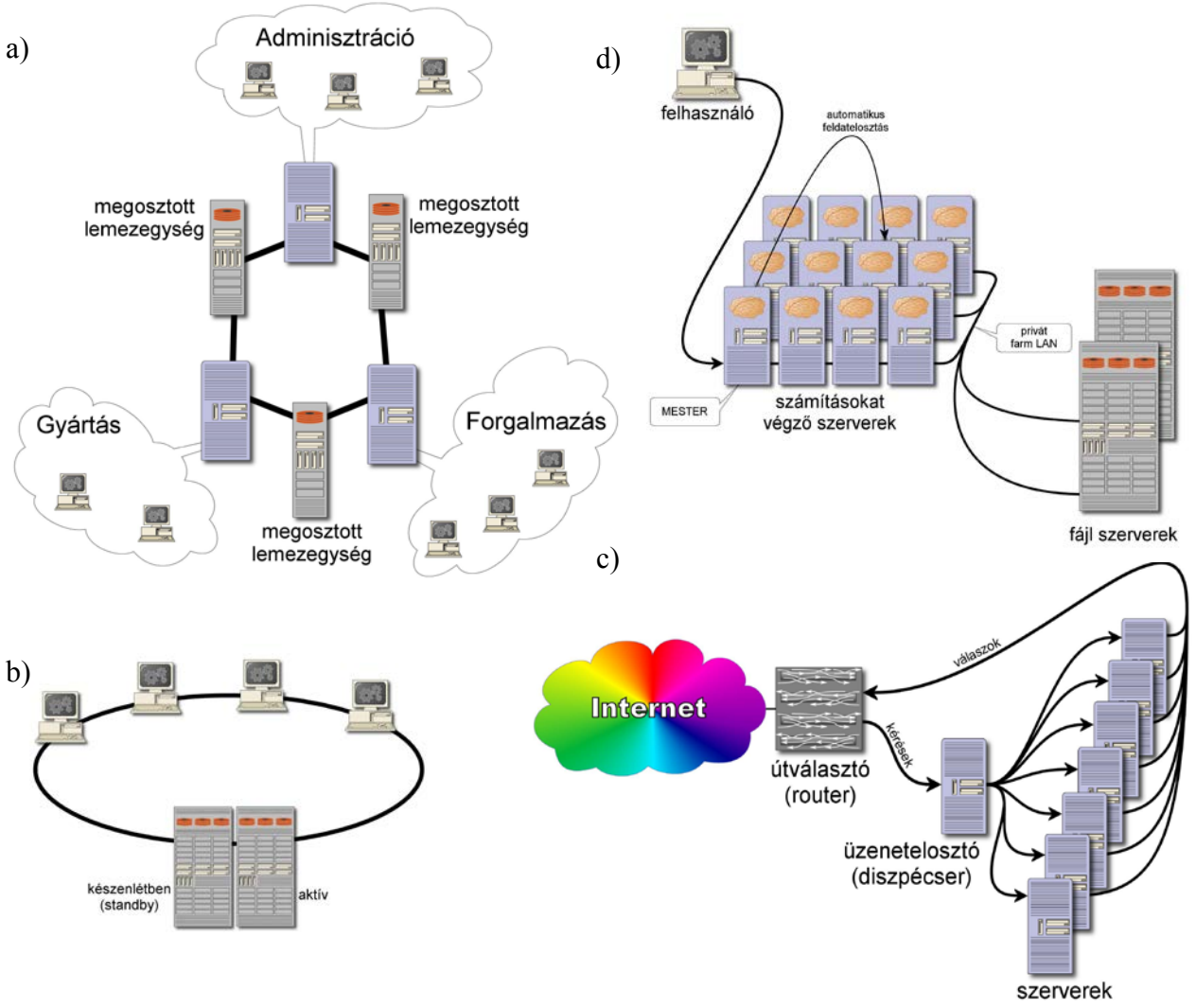


a) Üzenet alapú és b) megosztott erőforrást használó klaszterek architektúrák

Gyártók mindenféle architektúrát igyekeznek támogatni, ezért klasztert is “mindenki” gyárt: HP, Digital, Tandem, Compaq, IBM, Sun
Felhasználási példák:

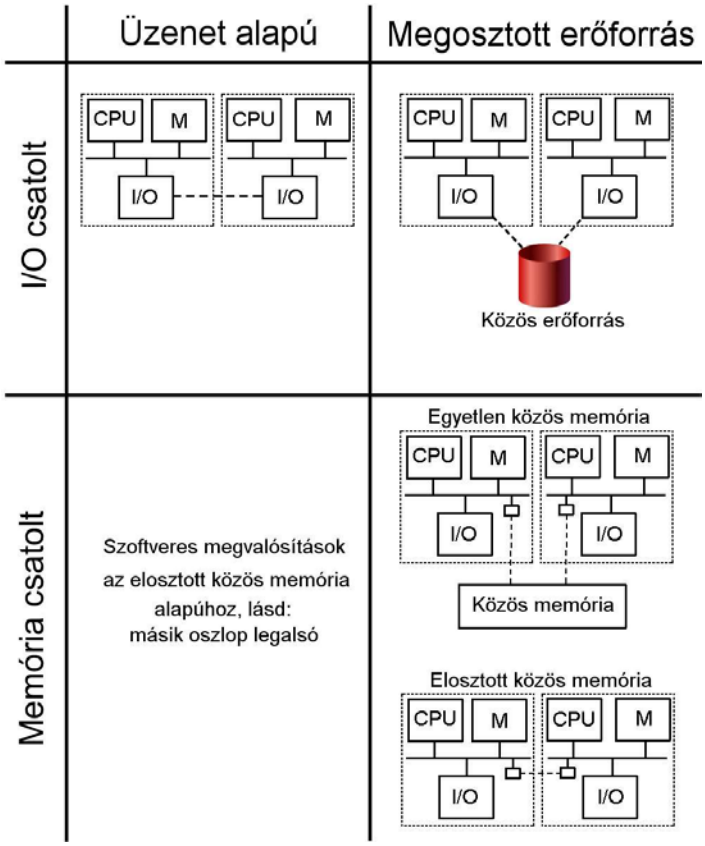
a. Sörfőzde: 3 szerver (adminisztráció, gyártás, szétosztás), egymás helyét átveszik, prioritások: gyártás, szétosztás, admin
b. Irodai fájlszerver (pl. ügyvédi iroda), 2 szerver, egyik tartalék, csak szinkronizálódik
c. Web szerver farm: sok azonos szerepű szerver egy kapcsoló mögött
d. Számítási farm (példa az ábrán: *Watson Research Central Compute Cluster*, már eredeti formájában nem létezik), A felhasználó egy terminálon bejelentkezett, ezt a vezér munkaállomás fogadta és kiválasztotta a legkisebb terhelésű szervert a felhasználó kéréseinek feldolgozásához. Minden program képes volt futni minden szerveren, azaz minden program és adat hozzáférhető volt minden szerveren. Ezt úgy érték el, hogy minden felhasználói adat a közös fájlszervereken volt tárolva, az adott szerveren csak a rendszer szoftverek, adat másolatok, és kisebb ideiglenes állományok voltak. A fájlszerverekhez közvetlenül csak a munkaállomások férhettek hozzá saját privát hálózatukon. A farm a felhasználóknak egyetlen gépnek mutatkozott. Bejelentkezni például külön az egyik gépére nem lehetett, csak az egész rendszerbe.

Valódi klaszterek: csomópontok egy csatoló elemen (alrendszeren) keresztül összekapcsolva (saját, szabványos háló, memória csatornák), különféle kommunikációs topológiák: központosított, hierarchikus busz , fa, hiperkocka, pillangó, gyűrű, háló / tórusz, teljes crossbar, virtuális crossbar (Internet)



IV.2.1 Együttműködés lehetőségei

- *I/O csatolt*: szokásos interfész és driverek, heterogén számítógépekre is, korlátozott sávszélesség
- *memória csatolt*: különleges HW (a memóriák illesztése speciális és architektúránként eltérő), spec. szoftver (operációs rendszer memória kezelőjébe), memória kezelési problémák (hozzáférés védelmek, szegmentálás, virtuális memória kezelés, cache), nagyobb teljesítmény, ha az előző overheadek nem túl nagyok

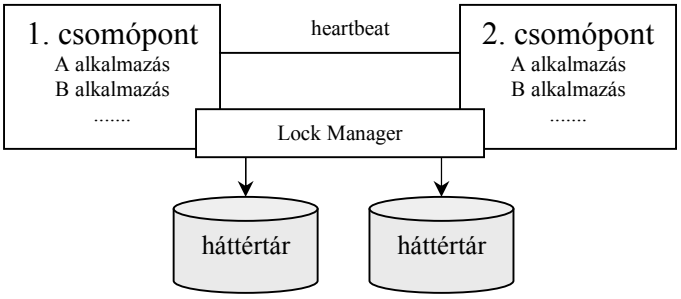


- *Közösen használt erőforrás* (memória, diszk): hagyományos programozási módszerek, cache probléma
- *Üzenet alapú:* szokatlanabb programozás, hordozható (plattform független), skálázható, nincs cache probléma

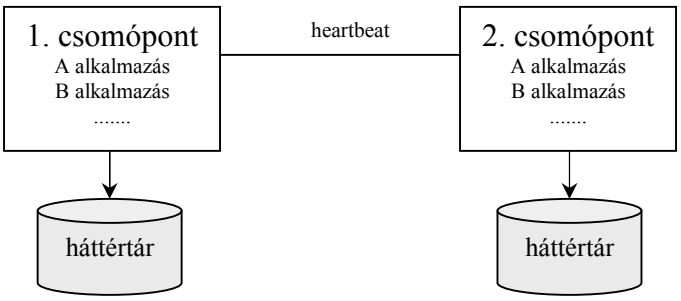
Tetszőleges hardver működés emulálható szoftverből (gyakori):

- osztott diszk v. memória szimuláció (egyszerűbb programozás vagy adaptálás, hatékonyság?)
- üzenet alapú kommunikáció osztott tárakkal megvalósítva (cache és szinkronizálási problémák elkerülésére)

- Közös háttértár (Oracle, VAX, Netware)



- Shared-Nothing (Microsoft)



IV.2.2 Hálózatba kapcsolt számítógépek, mint platform

IV.2.2.1 Tulajdonságok

- Nagy teljesítmény, redundancia olcsón
- Az új fejlesztésű gépek azonnal beépíthetők
- Meglévő szoftverek továbbra is használhatók
- Elérhetőség (kliensek folyamatos dinamikus kiszolgálása, “nincs” leállási idő)

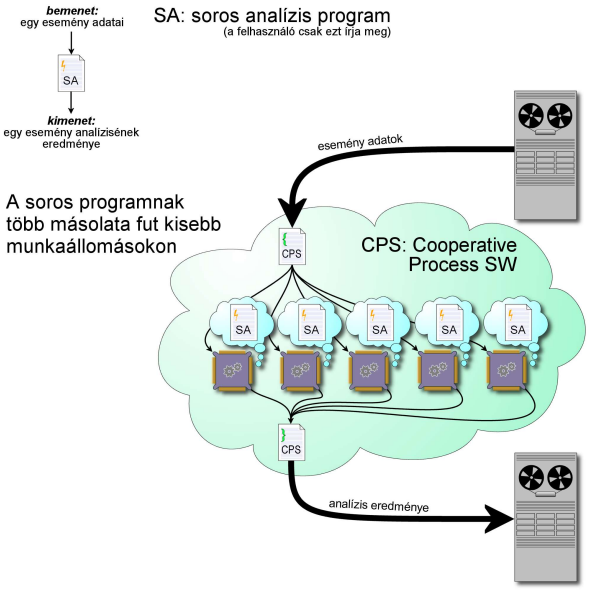
- Lassú kommunikáció
- biztonság, adminisztráció liceszek
- SW problémák: egységes rendszerkép kellene, hiányzó szoftverek (egyelőre csak adatbázis és tranzakció kezelők és web szerverek)

IV.2.2.2 Klaszterek programozása

A párhuzamos feldolgozást végző gépekre igen nehéz megfelelő szoftvert írni. Ezért a FermiLab emberei sem ilyen szoftvert írtak, hanem a rendszer lett úgy megépítve, hogy támogassa a párhuzamosságot. Ezt mutatja be az joob oldali ábra. A bonyolult, párhuzamos, egyszer megírt szoftverek és egyszerű, mások által programozható, soros alkalmazások szétválasztásának ezen módja máshol is jelentkezik (pl.: adatbázis kezelő rendszerek, tranzakció ellenőrzés). Külön nevet is kapott ez a technika: SPPS – *serial program, parallel (sub-) system*, azaz soros program párhuzamos (al)rendszer. Így el lehet rejteni az átlagos felhasználó elől a párhuzamos rendszerben rejlő nehézségeket, és ez már hasonlít a megszokottabb multiprogramozáshoz, mely egyetlen processzor esetén átfedést biztosít a számítások és I/O műveletek között. Ekkor az egyes programok nem futnak gyorsabban, csak kevesebbet kell várniuk az erőforrásokra. Párhuzamos rendszer esetén több erőforrás is, míg egyetlen processzor esetén csak átlapolás van.

- Batch alapú rendszerek, SPPS (Sequential Programming on Parallel System)
- Adatbázis kezelőkön keresztül (DB2, Oracle, Microsoft)
- Lehetőségek: új nyelv, egy meglévő kiegészítése, primitív könyvtárak
- PVM, PMI
    - Azonosítás
    - Kommunikáció (üzenet kezelő primitívek, kommunikátorok)
    - Biztonság
    - Erőforrás kezelés (task, memória)
    - Példa: Wilkinson: Parallel Pogramming PVM 53., MPI 57. oldal

- Komponens alapú programozás, információs adatbázis kell (COM, CORBA, Java Jini)
- Legion, Globus, Grid



IV.2.2.3 Feladat megoldási stratégiák

- Ideálisan párhuzamos számítások (master-slave, számítógépfarm, SPPS)
- Oszd meg és uralkodj, feladatdarabolási stratégiák
- Pipeline (pl. képfeldolgozás, Wilkinson: Parallel Pogramming144-151. oldal)
- Szinkronizált számítások (fizikai szimulációk)

IV.3 Párhuzamos algoritmusok modellezése

IV.3.1 Fogalmak

- *Probléma méret*: a feldolgozandó adategységek száma
- *Végrehajtási idő*: a feladat rendszerbe jutatásától kilépéséig eltelt idő.
- *Komplexitás*: (végrehajtási, memória) $O(f(N))$, létezik c és N_0, ha $N>N_0$, akkor $\textit{költség} \leq c*f(N)$. A költség többféleképpen definiálható, lehet például a végrehajtáshoz szükséges lépésszám vagy a megoldás létrehozásának ideje is.
- *Költség*: teljes lépésszám vagy futási idő: pl. $\textit{költség} = \textit{futási idő} \times \sum_{i=1}^{p} \textit{kihasználtság}_i$
- *Alsó korlát*: a számítási problémához meghatározható egy bizonyos komplexitású algoritmus, mely a megoldásához mindenképpen szükséges
- *Optimális algoritmus:* ha komplexitása a legnagyobb alsó korláttal megegyezik
- *Gyorsulás* (speedup): (feldolgozó egységek számának függvényében is, Amdahl törvénye)

$$\textit{gyorsulás} = \frac{\textit{A leggyorsabb szekvenciális algoritmus worst-case futási ideje}}{\textit{A vizsgált párhuzamos algoritmus worst-case futási ideje}}$$

- *Hatékonyság*:

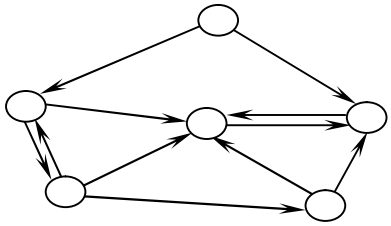
$$\textit{hatékonyság} = \frac{\textit{A leggyorsabb szekvenciális algoritmus worst-case költsége}}{\textit{A párhuzamos algoritmus költsége}}$$

- *Skálázhatóság*: a probléma méretének (komplexitás!) n* növelése esetén a futási idő változatlanul tartásához szükséges feldolgozó egységek száma (pl. mátrix szorzás)

IV.3.2 Párhuzamos algoritmus modell

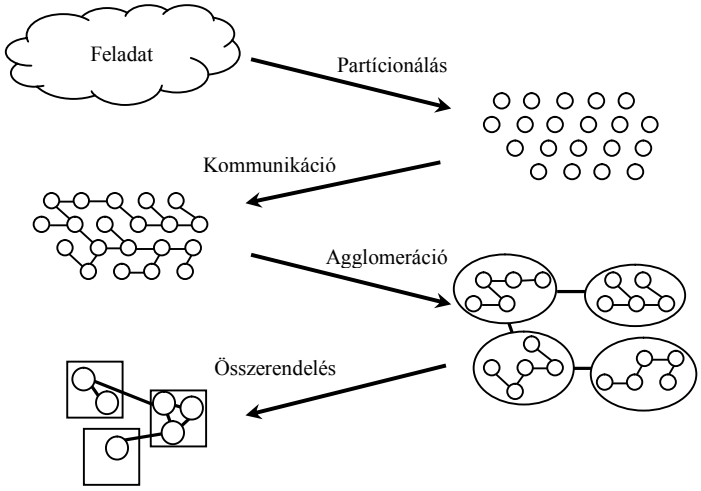
IV.3.2.1 **Taszkok és csatornák**:

A gráf csomópontjait a párhuzamosan futó feladatok alkotják, és élei az információ áramlás irányát mutatják.



IV.3.2.2 A párhuzamos algoritmustervezés folyamata

- *Partícionálás*: felosztása egymástól független feladatok sokaságára (adat és funkcionális partícionálás). A nem replikált műveleteket tartalmazó részek a skálázhatóságot, a finom felbontás pedig a következő lépések flexibilitását biztosítja.
- *Kommunikáció*: a közvetlen szomszédaikkal, jó skálázhatóság, ha az adatcsere szabályos struktúrájú (háló, fa, hiperkocka), és hasonló mennyiségű
- *Agglomeráció*: granularitásának növelése összevonással, a lokalitás elve, mérlegelni a replikáció lehetőségét, a skálázhatóságát megőrzése
- *Összerendelés*: az agglomeráció során létrehozott taszkok végrehajtó egységekhez rendelése tervezési vagy futási időben



V Szerver partícionálás és virtualizáció

V.1 A virtualizáció szerepe az informatikában

Hardverek sokféleség: az újabb típusok és megoldások bevezetésekor gyakran nehézséget okozott annak azonnali kihasználása és használatba vétele

- operációs rendszerek jelentették, melyek a hardver sokféleség egységes becsomagolására

Szoftver alapú sokféleség: elsősorban azokat az alkalmazásfejlesztőket sújtja, akik terméküket többféle platformon illetve operációs rendszer környezetben is alkalmazni kívánják.

- A fejlesztések és platformok további, immár operációs rendszerek fölötti összesítésére létrejöttek az újabb absztrakciós szintet biztosító virtuális környezetek (Java, .NET).

A különféle absztrakciós rétegek és virtuális környezetek sok helyen

- hardver megoldások,
- hardver erőforrások (SCSI, virtuális memória, ramdiszk),
- adatbázis elérések (ODBC)
- alkalmazások,
- a különféle alkalmazás rétegek (lásd például az ötrétegű alkalmazások felépítését)

A virtualizáció lényege a szolgáltatás igények elválasztása a szolgáltatás tényleges fizikai kivitelezésétől, vagyis *a virtualizáció egy az erőforrások fizikai jellemzőjének eltakarására szolgáló technika*. A virtualizáció különféle típusai:

- Egységesnek láttatják a több elemből álló rendszereket (elosztottság átlátszósága, egységes rendszerkép),
- Több egységre bontják a fizikailag egyetlen egységet képező rendszereket (partícionálás),
- Eltakarják egy rendszer belső komplexitását (egységes interfész szerep),
- Absztrakciót végeznek egy komplex rendszer könnyebb kezelhetősége érdekében.

V.2 Virtualizáció legfontosabb területei

V.2.1 Erőforrás virtualizáció

- virtuális memória
- háttértárak virtualizálása
  - több fizikai diszkből egy nagyobb logikai diszket képezzünk,
  - az adatátviteli teljesítmény illetve az elérhetőség növelése: RAID (mirroring, striping)
  - SAN (storage area network) távoli diszkek blokk szintű elérése
  - virtuális szalagkönyvtárak (VTL, Virtual Tape Library) a diszk léptékű teljesítményt "szalagszerű" jellemzőkkel és árral kombinálja.
- link aggregáció

V.2.2 Platform virtualizáció

A platform virtualizáció a futtató hardver minden erőforrására kiterjed, és egy teljes virtuális hardver bocsát a rajta működő alkalmazások rendelkezésére. Ez a megoldás egyszerre több (akár többféle) operációs rendszer számára kelti a dedikált számítógép illúzióját.

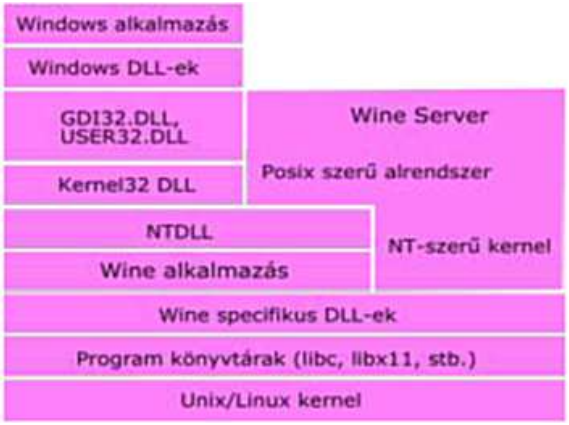
V.2.3 Virtuális munkakörnyezetek

- terminál szerverek
- távoli desktop elérés (Remote Desktop), távoli segélyszolgálat (remote assistance/support)
- helyi izolált környezetek (homokozó, sandbox)

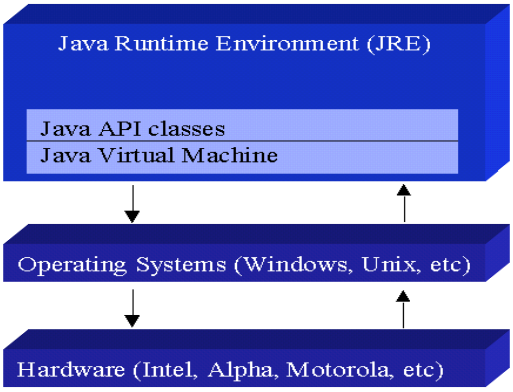
V.3 Számítógépek virtualizálásának módjai

V.3.1 Alkalmazás szinten történő virualizálás

Tiszta szoftver alapú: A gazda operációs rendszer által biztosított szolgáltatások felett egyfajta csomagolóréteget (wrapper) képez. Cygwin, Wine, JVM, .NET



Wine architektúra



JVM felépítése

V.3.2 Virtualizálás operációs rendszer szinten

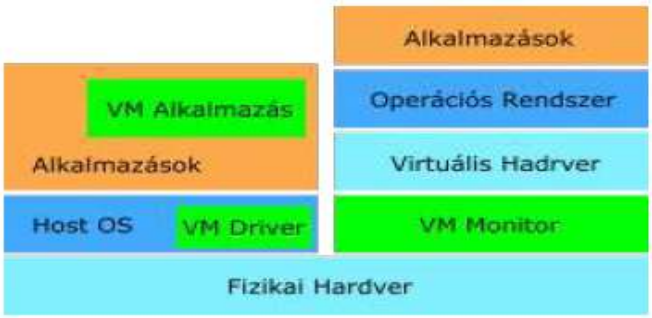
A virtuális környezetet kialakító szoftver szorosan együttműködik az operációs rendszerrel, esetleg a kettő el sem válik egymástól (Solaris Containers, BSD Jail, OpenVZ, Linux-VServer, Microsoft Virtual Server)

- A gazda operációs rendszert, hogy az önmagán belül képes legyen létrehozni egy vagy több virtuális környezetet (partíciók)
- a virtuális környezetetek ugyanannak gazda operációs rendszernek egy-egy újabb virtuális példányai
- teljesítménycsökkenés mindössze 1 százalék alá szorítható le
- több tíz, sőt akár több száz virtuális környezetet kialakítható ugyanazon a fizikai eszközön. (Virtuális Privát Szerver VPS szolgáltatók)

V.3.3 Emuláció

Az emuláció fogalmának jelentése, hogy egy adott fizikai eszközön részben vagy teljes egészében egy másik fajta eszköz fizikai felépítését és működését szimuláljuk. (x86 VMware termékek, általános processzor illetve rendszeremulációt végző nyílt forrású projekt a Qemu)

- az eredetihez teljesen hasonló virtuális környezetet nyújtanak és úgy futtathatók az arra az architektúrára készített operációs rendszerek és alkalmazások, mintha azt az eredeti fizikai eszközön tennénk
- viszonylag lassú, különösen teljesen idegen architektúrák emulációja esetén
- nagyon hasznosnak bizonyul az idegen rendszerre történő (ún. cross-platform) fejlesztés esetén (például beágyazott rendszerek vagy a mobiltelefon alkalmazások fejlesztése)



4. ábra: VMWare architektúra

V.3.4 Paravirtualizáció

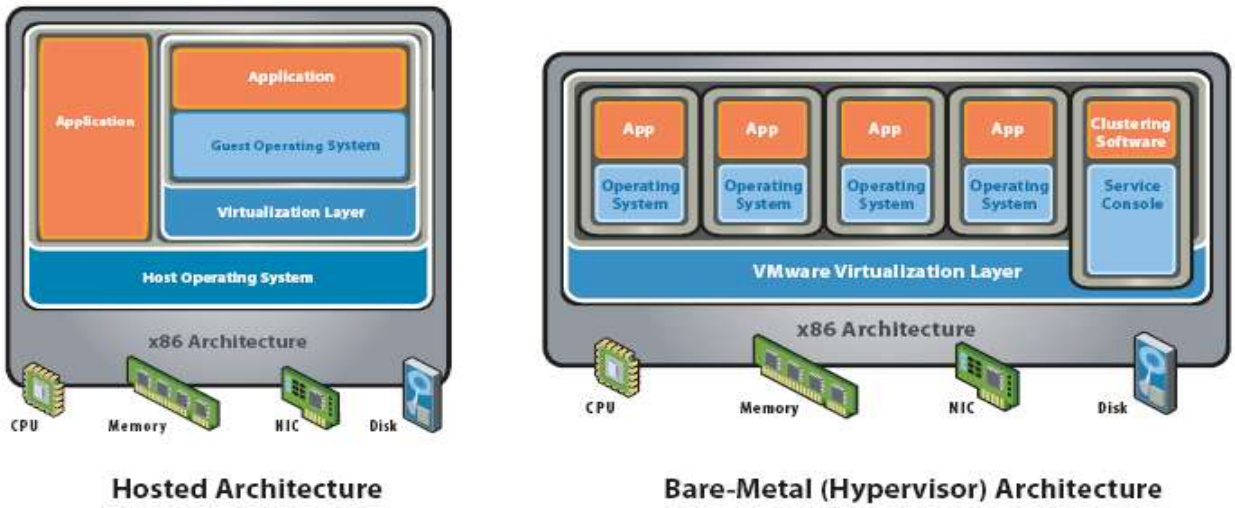
A virtualizálás olyan megvalósítását, ahol a vendég operációs rendszer tudatában van annak, hogy virtuális környezetben fut, és erre előre fel van készítve (a jobb adaptáció érdekében az eredetihez képest módosítottak rajta) paravirtualizációnak nevezzük. (User-Mode Linux, XEN)

- az emulációnál lényegesen jobb hatásfok érhető el, gyakran a fizikai gépek sebességéhez képesti veszteség 5% alatt marad. rendszer.
- A vendég operációs rendszereket adaptálni kell

V.3.5 Virtualizálás hardver támogatással

Eddig a virtuális környezetet kialakító alkalmazás (gazda operációs rendszer) feladat volt, hogy megoldja, hogy a több párhuzamos virtuális környezet hogyan érje el a valós fizikai eszközöket. A vendég operációs rendszerek nem 0-s címtől kezdődő folyamatos memória területet kaptak, a processzor beépített virtuális memória kezelése és védelme nem volt közvetlenül használható, illetve az I/O eszközök tényleges meghajtói is a gazda rendszerben kellett, hogy működjenek. A DMA képes (Direct Memory Access, közvetlen memória hozzáféréssel rendelkező) eszközök használatát is korlátozni kellett a vendég operációs rendszer számára. A fenti esetek mindegyik a virtuális környezet beavatkozását igényelte a folyamatba.

Hardver támogatás lényege hogy a processzort már a saját szintjén biztosítja a dedikált erőforrás hozzáférést több különálló operációs rendszer számára. Nincs szükség egy hagyományos értelemben vett gazda operációs rendszerre, hanem a virtuális gépek kezeléséről egy lényegesen vékonyabb réteg az ún. hypervisor gondoskodik.

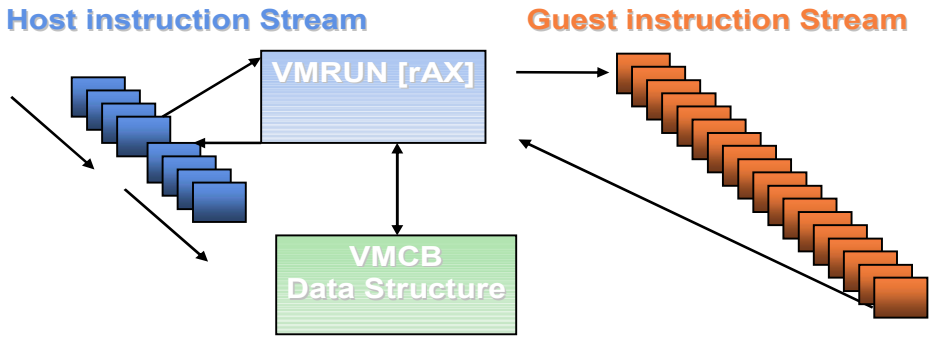


6. ábra: Hosztolt és hypervisor alapú virtualizálás

V.3.5.1 Az AMD "Pacifica" architektúrája

Az AMD által biztosított hardvertámogatás egy menet közben történő beavatkozás (intercept) alapú virtualizációt valósít meg. A processzorban bevezettek egy új privilégium szintet, az úgynevezett vendég védelmi üzemmódot (guest mode). A processzor alapvetően a gazda rendszer utasításait hajtja végre, és egy speciális utasítás a VMRUN hatására áttér guest módba, és ott makro szubritunszerűen futtatja a vendég operációs rendszert. A vendégrendszer kódjának futtatása addig tart, amíg valamilyen beavatkozást (interception) igénylő esemény (megszakítás, I/O hozzáférés) nem történik, vagy míg a környezet vezérlőt visszahívó VMMCALL utasítás kiadásra nem kerül. A gazda és a vendég operációs rendszerek közötti makró szintű átkapcsolást világváltásnak (world switch) nevezik.

A működés vezérlésében kulcsszerepet játszik a Virual Memory Control Block (VMCB) nevű adatszerkezet. Minden virtuális géphez tartozik egy ilyen, és nevével ellentétben nem csak a virtuális memória vezérlésében játszik szerepet, hanem világváltáskor ide mentődik a hozzá tartozó vendég operációs rendszer processzor és környezeti állapota, illetve itt kell konfigurálni, hogy mely események idézzék elő a gazda rendszer (hypervisor) beavatkozását a vendég környezetből történő visszalépéssel. A különféle beavatkozások két csoportra tagolódnak, a kivételekhez és a megszakításokhoz kapcsolódó kérelmekre. A szabályozás a hozzáférési és védelmi beállítások meglehetősen finom hangolását teszi így lehetővé külön-külön minden egyes vendég operációs rendszer számára. Itt találhatók az I/O műveletekre vonatkozó, és a megszakításkezelés módját szabályozó leíró szerkezetek is. A beavatkozást igénylő esemény megtörténtekor, a gazda rendszerhez való visszakapcsolás előtt az eseménnyel kapcsolatos információk szintén a VMCB-be kerülnek.



7. ábra: A gazda és a vendég rendszerek futtatási vázlata

- A virtualizálás egyik kulcskérdése a memória kezelés megvalósítása.
  - minden egyes vendég operációs rendszer számára külön laptáblát biztosít, 0-tól kezdődő folyamatos „fizikai" címtartománnyal a háttérben.
  - újabb absztrakciós szint bevezetése: a vendég által létrehozott „Guest Physical" címet még egy transzformációs táblázat segítségével kell „Host Physical" címmé alakítani, ami már ténylegesen a fizikai memóriában található (egymásba ágyazott, Nested Page Table technika).
  - Az új absztrakciós szint az árnyék laptábla (Shadow Page Table), melyet a gazda rendszer hoz létre, és tart karban, és melyet vendég rendszer csak olvasni tud a transzformációk guest módban való végrehajtása érdekében.
- megszakítás kezelés: A VMCB-ben beállított opció segítségével eldönthető, hogy az egyes megszakítás események kiszolgálása a gazda vagy a vendég rendszerben történjen-e. Ha a gazda rendszer szolgálja ki a megszakítást, akkor a kezében marad a megszakítás továbbításának menedzselése, illetve annak akár több vendég rendszer felé történő szétosztása is.

- DMA vezérlés
- a külső rendszermenedzsment eszközök támogatása
- „paged real” mód, virtuális laptáblákat nem használó, úgynevezett valós módját képes a virtuális környezetbe helyezni

A hardver támogatás jelentős terhet vesz le a virtuális környezetet kialakító szoftver válláról:

- a vendég rendszerekben valóban közvetlen memória és I/O hozzáférést biztosít
- komoly teljesítmény javulással
- a kompatibilitás és az vendég rendszerek közötti szeparáció megnő
- A vendég környezet (megfelelő engedélyek birtokában) ugyanúgy minden fizikai processzorhoz hozzáférhet, vagyis modern többmagos, illetve több processzoros rendszerekben akár valódi párhuzamos feldolgozást végző virtuális környezetek is megvalósíthatók.

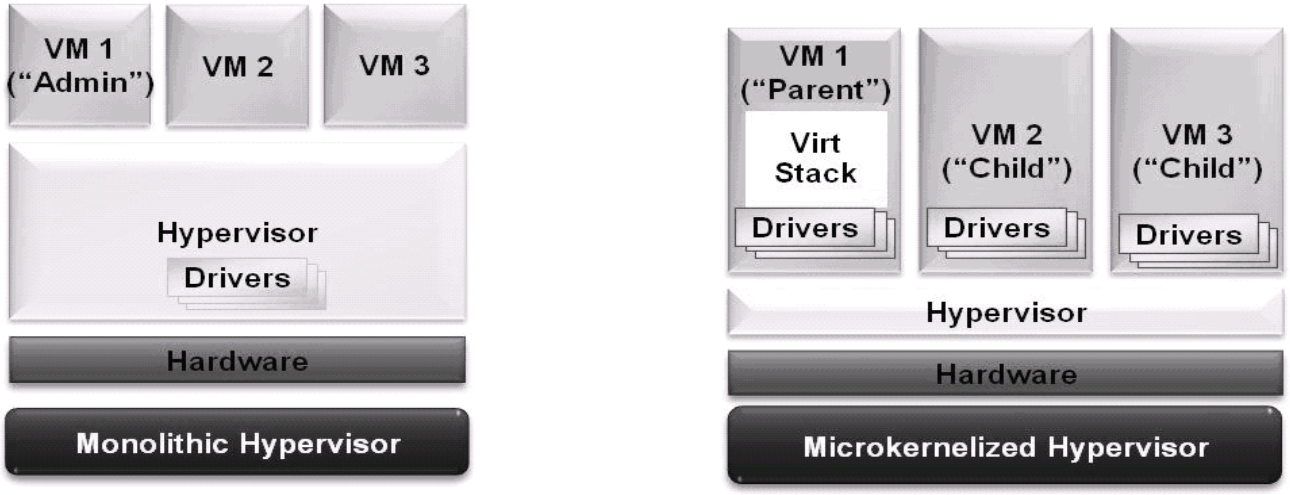
V.3.5.2 Monolitikus és mikrokernel alapú hypervisorok

A fő különbség a kettő között, hogy a monolitikus megoldás egy viszonylag nagyméretű szoftver. (VMware ESX Server)

- magába foglalja a fizikai eszközök meghajtó programjait, és ezek segítségével emulálja a virtuális eszközöket a vendégrendszerek számára
- a gazda és a vendég rendszer közötti gyakori kontextus váltások költségesek lehetnek
- a vendég operációs rendszereket nem kell adaptálni

A micro-kernel hypervisorok csupán egy vékony réteget képez a hardver fölött, mely csak és kizárólag partícionálási funkciókat lát el:

- nem tartalmaz harmadik fél által szállított eszközmeghajtókat, jelentősen csökkentve így a hibás, illetve rosszindulatú kódok bejutásának lehetőségét
- lényegesen kisebb támadási felületet és nagyobb biztonságot nyújt a rajta futó alkalmazások számára
- a vékonyabb réteg kevesebb kontextus váltás és gyorsabb működést is eredményez.
- a megoldás feltételezi, hogy a driverek a virtualizációs környezettel együttműködnek (például a XEN szabvány alapján). Ilyen elven működik a Microsoft Hiper-V architektúrája (Windows Server 2008).



9. ábra: Monolitikus és mikrokernel alapú hypervisor

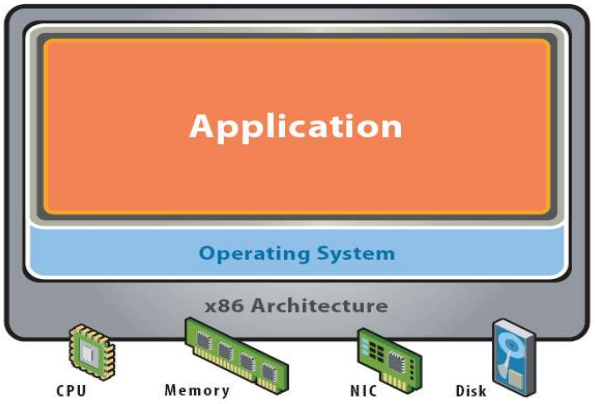
V.4 Virtualizálás vállalati rendszerekben

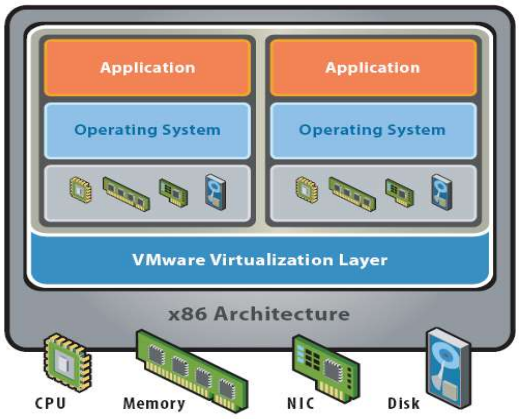
V.4.1 A virtualizálás motiváció információs rendszerekben

A vállalati szerverek általában sokféle hardvert tartalmaznak, egyetlen számítógépen egyféle operációs rendszer működik, és általában minden egyes funkció (web szerver, adatbázis szerverek, üzenetsor kiszolgáló, levelező szerver, stb.) külön számítógépen kerül elhelyezésre, akkor is, ha a szerver teljesítménye jóval meghaladja a szükséges mértéket. Ennek fő oka, hogy a közös gépen (operációs rendszeren) osztozó szolgáltatások esetén

- A szolgáltatások zavarhatják (lehetetlenné tehetik) egymás működését közös címek, fájlok, szolgáltatások használata miatt.
- A biztonsági intézkedések között a legkisebb közös nevező elvét kell alkalmazni, azaz a közös beállítások mindegyik rendszerénél megengedőbbek kell, hogy legyenek.
- Karbantartás (verzió frissítés) idején a szerver valamennyi alkalmazásának működése veszélybe kerülhet.
- A rendszer szállítója a hardver teljesítményt nem az átlagos, hanem a (várható) csúcsteljesítményre méretezi, mely csak úgy garantálható, ha ott más alkalmazás nem működik. A teljesítmény monitorozás és a skálázhatóság is csak ilyen formában kezelhető.
- Egyes régi rendszerek még a vállalat szempontjából fontos funkciót töltenek be, de az elavult hardver és szoftver környezetbe az új alkalmazások már nem illeszkednek.

A vállalat ezen okok miatt a kelleténél jóval több hardver vásárol, és tart fenn (áramfelvétel, hűtés, karbantartási mennyiség és szakismeretek), ami nem csupán költséges, hanem a nagyon rugalmatlan is. Ebben a rendszerfelállásban a kihasználatlan kapacitások nem csoportosíthatók át, és nem használhatók fel egyéb célokra. A fenti problémák megoldásában nagy segítséget jelent az adatközpont szervereinek konszolidálása, melynek kulcsfontosságú eszköze a virtualizálás. Ennek folyamatát és hatásait a következő pontban részletezzük.





Virtualizálás előtti állapot	Virtualizálás utáni állapot
• Sok különböző szerver • Minden számítógépen egyetlen operációs rendszer • A szoftver és a hardver szorosan csatolt • Az ugyanazon a számítógépen futó szolgáltatások zavarhatják egymást • Kihasználatlan erőforrások (10-15%) • Drága és rugalmatlan infrastruktúra	• Kevesebb szerver • Hardverfüggetlen alkalmazások és operációs rendszerek • Az operációs rendszerek és alkalmazások, együtt, egy csomagban kezelhetők (állapot mentés, mozgatás) • Rugalmas könnyen átkonfigurálható, skálázható infrastruktúra • Jobb erőforrás kihasználás (60-80%)

V.4.2 A szerverkonszolidáció hatásai

A vállalati infrastruktúra komplexitása és finanszírozás igénye jelentősen csökkenthető az infrastruktúra konszolidálásával. A konszolidáció fogalma a vállalati információs rendszerek egyfajta egységesítését és összesűrítését takarja. A konszolidáció a több vagy nagyobb méretű gépek vásárlása helyett a következő törekvések valamelyikének – vagy szerencsés esetben mindegyikének – megvalósulását jelenti:

- centralizáció: a meglévő szerverek elhelyezése kevesebb helyszínen,
- fizikai környezet egyesítése: több kiszolgáló gép helyettesítése egy géppel,
- alkalmazás csoportosítás: alkalmazások áthelyezése azonos platformra és/vagy szerverre,
- adatintegráció: különböző forrású és típusú adatok elhelyezése egy tárolón és azonos formátumban.

A sok kisebb egy-egy operációs rendszert és alkalmazást futtató szervert néhány nagyobbal helyettesítjük, és az addig különálló hardveren működő rendszereket az új szerveren kialakított virtuális környezetekbe (partíciókba) helyezzük.

- üzemeltetési költségek csökkenése,
- jobb hardver erőforrás kihasználással,
- Teljesítményjavulásával,
- leegyszerűsíti a rendszer mendzsmentet (központi admin, biztonsági mentések),
- biztonság,
- rugalmasság: Ha gazda rendszer kihasználatlannak, vagy éppenséggel túl szűknek bizonyulna, szabadon indíthatunk új partíciókat, vagy szüntethetünk meg közük, szétosztva a lefoglalt erőforrásokat a többi rendszer között. A vendég rendszerek bővítésre és skálázása is sokkal finomabb granularitással történhet, és a beavatkozás sikeressége illetve elégtelensége is azonnal szembetűnő.

V.4.3 A virtualizálás kihasználásának egyéb területei

- szerver konszolidáció
- nagyszámú különböző fejlesztő és teszt környezet alakítható ki
- új szolgáltatások bevezetése: Bármilyen legyen is az új szolgáltatás környezeti igénye, új hardver beszerzése nélkül azonnal felépíthetjük a neki megfelelő környezetet a virtuális szerverre. A virtuális környezet alkalmazható a vállalati migrációs tervek kipróbálására, ellenőrzése.
- nagyobb rendelkezésre állást, illetve üzletmenet folyamatosságának biztosítása: Ez a jellemző segíti a biztonsági mentések (backup) készítését, melyet lefagyás vagy más szoftverhiba esetén gyorsan, sőt akár automatikusan visszaállíthatunk, vagy melyet a gazdagép katasztrofális meghibásodása esetén egy tartalékgépen élesztünk föl.
- régebbi befektetések értékének védelme: A vállalat régi harmadik féle által készített vagy akár saját fejlesztésű alkalmazásainak karbantartása és üzemeltetése a kompatibilis hardver elérhetetlensége, illetve a szakértelem fokozatosan növekvő hiánya miatt egyre nagyobb költséggel jár. Egy virtuális gépben az eredeti környezetükben, a legmodernebb fizikai eszközökön futva élhetnek tovább.

V.4.4 Tipikus szolgáltatások

A szerver virtualizáció felhasználási területeinek bemutatása után végezetül tekintsük át, hogy a technológia szállítói milyen egyéb szolgáltatásokat biztosítanak a virtualizálást megvalósító alapfunkciókon kívül az általuk szállított rendszerekhez a lehetőségek minél teljesebb kihasználása

érdekében. Az itt említett példák mind a Microsoft Windows Server 2008, mind a VMware ESX Server termékhez kapcsolódó kínálatban megtalálhatók. Az egyes eszközök négy különféle cél megvalósítása köré csoportosulnak:

- Segítik a meglévő rendszerek virtuális környezetekbe történő migrálását, (meglevő fizikai rendszerek virtuális környezetbe mozgatása gyorsan, egyszerűen zajlik, mivel nincs szükség az alkalmazások újratelepítésére, konfigurálására).
- Segítik a rendszerüzemeltetést azáltal, hogy az egyes termékekhez kapcsolódó menedzsment eszközöket valamilyen általános ipari szabvány jelentő menedzsment környezettel helyettesítenek (biztonsági mentések, pillanatfelvételek működés közben, szolgáltatások csomópontok közötti mozgatása)
- Kiterjedt rendszer monitorozást valósítanak meg annak érdekében, hogy még idejében felhívják a rendszergazdák figyelmét a különféle folyamatok nemkívánatos változásaira, még mielőtt azok valós problémává alakulnának át.
- Automatizálnak egyes feladatokat, illetve lehetővé teszik az emberi beavatkozás nélküli, programozott reagálást különféle váratlan eseményekre (A rendszermonitorozó eszköz eseményt (riasztást) generálhat, amennyiben egy adott virtuális gép számára kezd szűkösnek bizonyulni a hozzárendelt hálózati sávszélesség. A riasztáshoz automatikus eseménykezelő rendelhető, mely a virtuális gép leállításával, a számára kiosztott hálózati sávszélesség megnövelésével, majd a szolgáltatás új konfiguráció szerinti újraindításával reagál a helyzetre.)
- Elérhetőség növelése (Több fizikai számítógép által közösen használható háttértár és fájlrendszer, klaszterezés támogatás)

V.4.4.1 Központi menedzsment

Minden virtualizálást támogató terméknek alapvető részét képezi egy egységes rendszerkezelő eszköz, mely a hardver környezet és rajta futó virtuális környezetet összerendelését és üzemeltetését szolgálja. Olyan monitorozó és rendszerfelügyeleti eszközöket értünk ez alatt, melyek központi kezelő felületet biztosítva szerverek százait képesek áttekinteni egyetlen központi helyről, lehetővé téve az állandó megfigyelés és az azonnali beavatkozást. A legfontosabb funkcióik a következők:

- A gazda gép konfigurálása (globális beállítások, külső háttértárak, háttértár partícionálás, virtuális háttértárak kezelése).
- Virtuális gépek létrehozása és futtatása. A virtuális gépek hardver kialakítása teljesen egyedi lehet, és általában testre szabható sablonok használatával történik. Megadható a virtuális gép által használható erőforrások (fizikai memória, fizikai processzorok száma, háttértárak) és hardver eszközök.
- Virtuális gép könyvtárak kezelése. A könyvtár kezelő virtuális gép image állományokat tárol és kezel, illetve lehetővé teszi az itt található alvó állapotú gépek fizikai környezetben történő aktiválását.
- A virtuális gépek fizikai számítógépekkel történő összerendelésének vezérlése, illetve a közöttük való mozgatása.
- Monitorozó és jelentés készítő modul. Képes megjeleníteni a fizikai illetve virtuális számítógépek aktuális terhelését, használati trendjeit, illetve figyelmemmel kísérhető a rendszerek működőképessége is. A rendszerek megjelenítésekor különféle csoportosításokat alkalmazhatunk, illetve gépenként testre szabhatjuk a különféle riasztások kritériumait is.
- Önkiszolgáló felhasználói felületet biztosíthatnak egyes nem rendszer adminisztrátor szerepű felhasználók (fejlesztők, tesztelők) számára egyedi virtuális gépek készítéséhez, illetve törléséhez, jóval nagyobb szabadságot adva a végfelhasználók kezébe. Természetesen az ehhez szükséges jogosultságok, illetve a létrehozható virtuális gépekre vonatkozó korlátozások beállítása továbbra is a rendszergazdák feladata.

VI Peer-to-Peer rendszerek

Elosztott rendszer alternatíva a kliens-szerver és a klaszter megoldások mellett. Egyenrangú csomópontok építik fel, cél valamilyen erőforrás vagy szolgáltatás megosztása.
Peer = kliens + szerver funkciók egyben

def: 1. Direkt módon cserélődő szolgáltatások a hálón
2. Olyan rendszer, mely elemei mutatják az alábbi tulajdonságokat
   - szerverszerű művelet
   - DNS független címzés (a kapcsolódó csomópontok nem feltétlenül rendelkeznek fix IP címmel, esetleg nincs is DNS bejegyzésük, mert nem osztanak meg erőforrásokat)
   - együttműködés a hálózaton

A peer alkalmazást futtató csomópontok gráfba szerveződnek, a logisztikai (overlay) és a fizikai hálózat elkülönül.

VI.1 Hogyan működik?

1. társak keresése, (TCP protokoll: broadcast, multicast, szerver lekérdezés)
2. keresés (szomszédoknak csomagok, Time to Live, kapott csomagok továbbítása, routolás, index szerver közvetlen lekérdezése)
3. találatok visszajelzése (közvetlen kapcsolat, vagy backward routing, a kérés csomagot már nem küldi tovább)
4. letöltés közvetlen kapcsolattal (vagy backward routing)

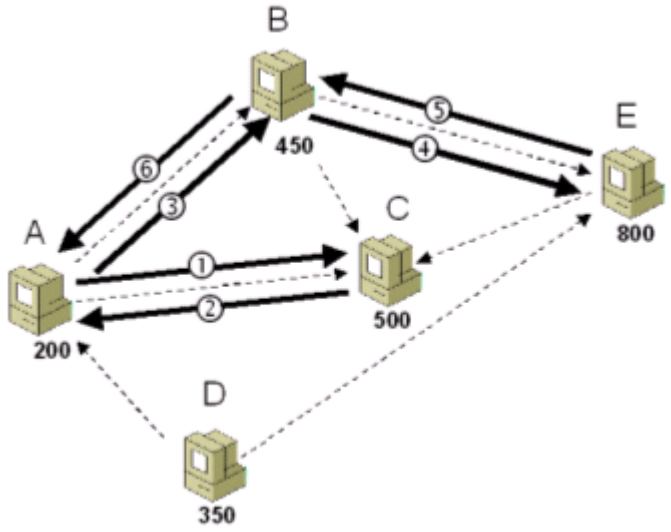
- topológia: jól összekapcsolt gráf (well connected graph) tulajdonságai
  - teljes kapcsolatrendszer (bármely két csomópont között van út, szigetek elkerülése)
  - minél kisebb átmérő (a legnagyobb úthossz a gráf tetszőleg két pontja között)
  - robosztus (néhány csomópont kiesése nem szünteti meg a kapcsolatrendszer teljességét)
- route-olás: gráfban a körök elkerülése (pl. GUID az üzenetekben), leggyorsabb útvonal megtalálása
- önszervezés: a szövevényes gráf kapcsolatok miatt egy csomóponthoz ugyanaz a kérés (azonos GUID-ból detektálható) többször is eljuthat, a csomagban talált TTL információ alapján az egyes kapcsolatok jósága, fontossága és korrelációja megállapítható, és az adatok folyamatos elemzésével számítható egy hasznossági index (utility index), melynek alapján egyes kapcsolatokat érdemes eldobni, és újat létesíteni helyettük
- találati valószínűség (figyelem: kikapcsolt csomópontok, nem teljes keresés): minél több helyen van meg egy dokumentum és minél teljesebb a keresés annál nagyobb

VI.1.1 Esettanulmány: Windows Peer-to-Peer Networking

- A jól összekapcsolt gráf kialakításához és fenntartása
  - a csomópontok globálisan egyedi azonosítóval rendelkeznek
  - a kialakuló gráfot aláírással látják el (ez a benne szereplő legalacsonyabb csomópont azonosítóval egyezik meg)
  - a gráf méretével arányos mennyiségű ún. kapcsolati (contact) csomópontok kijelölése
- szükséges műveletek

- Kapcsolódás a gráfhoz
  - egy már bent lévő csomópont felderítése
  - kapcsolódási kísérlet, ha a kapcsolatok maximális száma még nincs kihasználva, akkor a belépő megkapja a kontakt csomópontok listáját, ha már nincs több szabad kapcsolódási lehetőség, akkor a belépést elutasításra kerül (reject)
  - a reject message tartalmazza az elutasító csomópont szomszédjainak listáját, melyből az elutasított véletlenszerűen választ, és újra megkísérli a kapcsolódást
  - ha a bekapcsolódó új csomópont azonosítója kisebb, mint a gráf aláírása, akkor az aláírás megváltozik, broadcast (flooding) segítségével kell terjeszteni a gráf összes tagjának
- Szétkapcsolódás (leálláskor vagy felesleges link megszüntetésekor)
  - szétkapcsolódás ne okozzon szakadást a gráfban
  - akiről lekapcsolódik egy csomópont, annak megküldi a vele kapcsolatban álló többi csomópont listáját
  - szétkapcsolás után a másik csomópont a kilépő csomópont szomszédjainak listájából választ magának új partnert
- Hibadetektálás és javítás
  - a gráf aláírás bizonyos idő elteltével (pl. 5 perc) lejár. Az újra generálás módja, hogy minden csomópont az azonosítójával arányos ideig várakozik, majd ha addig nem érkezik frissítés a saját azonosítóját kezdi gráf aláírásként terjeszteni (legnagyobb valószínűséggel a legkisebb lesz az első, ha többen is elkezdték, akkor a legkisebbet tartják meg)
  - A gráfaláírás minden különálló részgráf területén azonos. A gráf szakadásai úgy detektálhatók, hogy a csomópontok saját gráfaláírásukat összehasonlítják azzal, amit a kontakt csomópontokkal közvetlen kapcsolatba lépve azoktól közvetlenül lekérdeznek. Ha a két aláírás különbözik, akkor a gráf szigetekre bomlott.
  - A gráf egységének helyreállítása a kontakt csomópontokkal (vagy szomszédaikkal) történő kapcsolatok újraépítésével történik

VI.1.1.1 Csomópontcím szerinti keresés



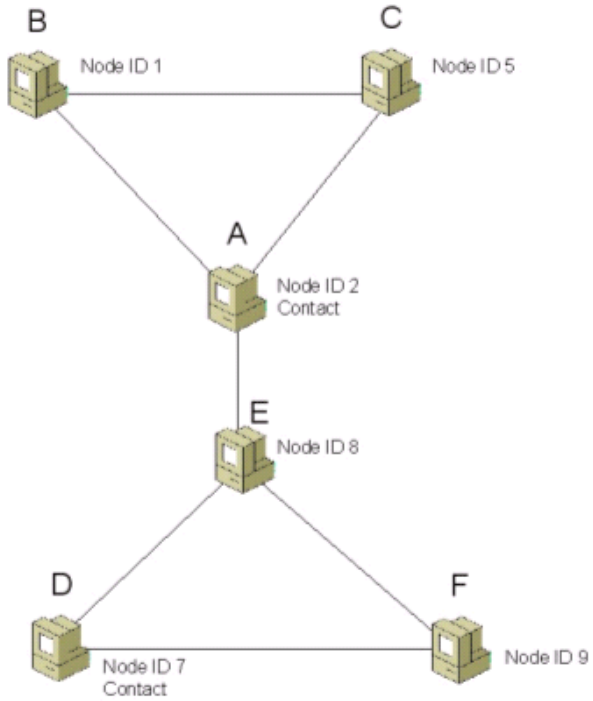
When Peer A wants to resolve the PNRP ID of 800, the following process occurs:

1. Because 500 is numerically closer to 800, Peer A sends a PNRP Request message to the node that registered the PNRP ID of 500 (Peer C).
2. Peer C does not have an entry for the PNRP ID of 800 and does not have any entries that are closer to 800. Peer C sends a response back to Peer A.
3. Because 450 is the next numerically closer PNRP ID to 800, Peer A sends a PNRP Request message to the node that registered the PNRP ID of 450 (Peer B).
4. Peer B forwards the request to the IP address that registered the PNRP ID of 800 (Peer E). Although Peer B has an entry for Peer E in its cache, the existence of the cache entry does not assure that Peer E is currently available on the network.
5. Peer E sends a response back to Peer B.
6. Peer B sends a response back to Peer A.

VI.1.1.2 Gráfjavítás kontakt csomópontok segítségével

The graph connection between Node A and Node E is severed. The following process repairs the partition:

- In the bottom partition of the graph—consisting of Node D, Node E, and Node F—the Signature record expires after a maximum time of 5 minutes because the refreshed Signature record sent by Node B is not propagated to nodes in the bottom partition.
- When the current Signature records expires, all the nodes in bottom partition of the graph calculate a random backoff delay that is proportional to their node IDs before flooding a new Signature record with their own node ID. Because Node D has lowest node ID, it floods a new Signature record with graph signature of 7 first. Node E and Node F receive the new Signature record and update their own Signature records.
- Node D updates its own Contact record with a graph signature of 7 and floods that new record to Node E and Node F. As the information is propagated to the graph nodes, each notes that the Contact record for Node A with a graph signature of 1 does not match the Contact record for Node D with graph signature of 7. To resolve the inconsistency, each node calculates a random backoff delay. After the delay, the node will attempt to connect with Node A to repair the partition. For this example, the delay for Node D is less than the delay for Node E or Node F. Node D connects with Node A.
- Node A and Node D exchange records. Node D notes that Node A sent it a new Signature record with graph signature of 1, which is lower than the current graph signature. Node D floods the new Signature record with graph signature of 1 to Node E and Node F.
- Node D updates its own Contact record for Node D with graph signature of 1 and floods the updated Contact record for Node D with graph signature of 1 to Node E and Node F. Then, Node D floods the new Contact record for Node A with graph signature of 1.



The graph has converged. All the nodes in the graph have the same set of graph records: a Signature record with graph signature of 1, a Contact record for Node A with graph signature of 1, and a Contact record for Node D with graph signature of 1.
If Node D, Node E, or Node F were not able to establish a graph connection with Node A or any other node in the upper partition (consisting of Node A, Node B, and Node C) due to network reachability problems, then two graphs would exist. There is another process to perform long-term partition repair.

VI.2 Jellemezők

Legfőbb előnyei:

- ár, karbantartás (meglévő infrastruktúrára épül rá, önszervezés)
- önszervezés (a nagyobb kapacitású csomópontok önmaguktól gerinchálózattá állnak össze, router funkció minden csomópontban, izolált szigetek)
- alacsony fejlesztési költség (a csomópontok egyenként egyszerűek)
- rendelkezésre állás (az idő hány százalékában működőképes, eszköz: redundancia, hibatűrés, átkonfigurálás)
- hibatűrés (detektálás, szeparálás, funkciók áttétele működő csomópontokra)
- megbízhatóság (a működési idő hány százalékában helyes a működés, P2P értelmezés?!)
- anonimitás
    - igény van rá, jó-e? jogi kérdések
    - megvalósítás: backward routing (minden csomópont a csomagazonosítók alapján tárolja, hogy melyik lekérdezés honnan érkezett, és a választ arra küldi vissza, a csomagazonosítók csak korlátos mennyiségben és ideig tárolódnak)
- hatékony multicast lehetőséget biztosít (pl. videó konferencia, on-line hang és képközvetítések) az IP multicast infrastruktúra alkalmazása nélkül

Hátrányok:

- biztonság (központi admin hiányzik, jogosultságok tárolása nem megoldható)
- sebesség (válasz idő), erőforrás igény (nagy sávszélességet fogyaszt)
- verzió kezelés
- konzisztens állapot mentés nem megoldható

Mérőszámok:

- találati arány (csak erőforrás megosztásnál)
- generált forgalom (algoritmusok erőforrás igénye és a generált forgalom fordítottan arányos)

VI.3 P2P fajtái

VI.3.1 Megosztatni kívánt szolgáltatás szerint

- szerver nélküli, valós idejű üzenettovábbítás kollaboráció (chat, video konferencia, egyéb groupware, játékok)
    - követelmény: veszteségmentes, minimális késleltetés, osztott objektumok szinkronizálása
- számítás kapacitás (SETI@Home)
    - központi elem szükséges (hibrid)
    - master-worker vagy oszd meg és urakodj jellegű megoldások
- tartalom megosztás (elsősorban fájlok)
    - követelmény: átlátszó osztott tár, felhasználók kapcsolódása, keresés, letöltés
    - szöveges: hír és egyéb tematikus listák (news groups)
    - hang és videó (terhelés elosztás, jogi kérdések)
    - szoftver frissítések szétosztása (nagy mennyiségű adat, sokan szeretnének rövid idő alatt hozzájutni)

VI.3.2 Hálózati környezet szerint

	Átviteli sebesség	Megbízhatóság	Minőség	Biztonság
Kábel kapcsolat	64Kbit/Sec- 1 Gbit/Sec	Jó/Kiváló	Jó/Kiváló	Jól megvalósítható
Modemes kapcsolat	3,6-64 Kbit/Sec	Közepes	Közepes	Változó
Ad-hoc (mobil hálózatok)	3,6-64 Kbit/Sec	Alacsony	Alacsony	Változó

VI.3.3 Csomópontok platformja szerint

Hardver eszközök		Operációs rendszerek	Fejlesztő eszközök
Asztali számítógépek (PC)		Linux/Unix	Java/C (.NET)
		Windows	Java, C, .NET
Mobil Eszközök	Kézi számítógépek	Windows CE	.NET
		Symbian, PalmOS	Java, C
	Mobil telefonok	Symbian	Java, C

VI.3.4 Architektúra szerint

- tiszta (teljesen elosztott, homogén, tökéletesen egyenrangú csomópontok, nagyobb forgalom, Gnutella)
- hibrid (szuper peer, hierarchikus általában kétszintű overlay háló, az alacsonyabb szintnek csak egy kapcsolata van egy szuper peer felé)
    - with dicovery server, csomópont lista a szerveren, (Kazza, Morpheus)
    - with dicovery and lookup server, csomópont lista és tartalom index a szerveren (Napster)
    - szélsőség: kliens-szerver, a tartalom is a szerveren van (fájl szerverek)

A tiszta és a hibrid modell tulajdonságainak összehasonlítása:

	Decentralizált peer-to-peer	**Hibrid peer-to-peer**
Rendelkezésre állás	A redundás tárolás miatt magas lehet.	A központosított szolgáltatások miatt alacsonyabb.
Megbízhatóság	A redundancia mértékével nő, de nem lehetünk biztosak benne, hogy mindig meg is találjuk. (TTL)	Lookup szerver sokat segíthet.
Skálázhatóság	Szinte problémamenetesen megoldható.	Általában kevés változtatással megoldható.
Hibatűrés	Nehéz a hibák felderítése és izolálása.	A központi csomópont segítségével, könnyebb lehet.
Biztonság	Nehezen megvalósítható.	A központi csomópont segítségével, elég jó megvalósítás érhető el.
Anonimitás	Jól megvalósítható.	Kevésbé mint a decentralizáltnál.
Csomópont felfedezés	Nehézkes a megvalósítása.	Viszonylag könnyű megvalósítani.
Átlátszóság	Jól megvalósítható.	Jól megvalósítható.
Elosztottság	Nagyon magas.	Magas lehet.

VI.4 Más rendszerekkel való összehasonlítás

Rendszer/ Tulajdonságok	**Rendszer típusa**		
	Központosított	**Kliens/Szerver**	**Peer-to-Peer**
Elosztottság	Alacsony	Magas	Nagyon magas
Rendszer költség	Nagyon magas	Közepes	Alacsony
Anonimitás	Alacsony (nincs)	Közepes	Nagyon magas
Skálázhatóság	Alacsony	Magas	Nagyon magas
Teljesítmény	Egymagában magas, egy az egy felhasználóra jutó alacsony lehet	Közepes	Egyenként alacsony, de összesítve magas
Robosztusság	Egymagában magas	Közepes	Egymagában alacsony
	Egyesített alacsony		Egyesített magas
Önszervezés	Nincs	Közepes	Kitűnő
Átlátszóság	Alacsony	Közepes	Közepes
Biztonság	Nagyon magas	Magas	Alacsony

VI.4.1 Peer-to-peer és kliens-szerver rendszerek

(érdemes-e vállalati fájl szervert peer-to-peer alapokra helyezni?)

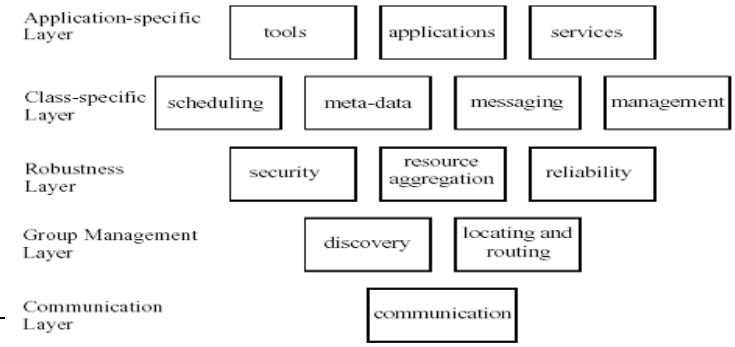
Peer-to-peer	Kliens/Szerver
az összes részvevő csomópont képes az erőforrásainak megosztására	csak a szerverek végeznek erőforrás megosztást
a sok csomópont együttesen nagyobb elérhetőséget biztosít	a szerver kritikus hibapont és szűk keretmetszet lehet
a csomópontok közvetlenül osztják meg erőforrásaikat	a publikálni kívánt szolgáltatást a szerveren kell elhelyezni
összességében nagy (felesleges?) forgalmat generál	csak a feltétlenül szükséges forgalom, de az mind áthalad a szerver szűk keresztmetszetén
a biztonságos működés nehezen megoldható	a biztonsági rendszer központilag megoldott

VI.4.2 Peer-to-peer és a klaszterek

(nagy számítási teljesítményű rendszerek kialakításához)

Peer-to-peer	Klaszter
a teljesítmény elsősorban a holtidők hasznosításából adódik	dedikált számítógépek teljesítményét egyesíti
földrajzilag elosztott	a fürt tagjai egymáshoz közel vannak
flooding technika széleskörű használata	közvetlen kommunikáció a csomópontok között, nincsenek felesleges üzenetek
a csomópont egymástól változó logikai távolságra (hops) vannak	a csomópontok logikai távolsága azonos, nincsenek gráfok
TCP fölötti rétegben kommunikál	a hálózati és szállítási protokollok egyediek lehetnek
lassú, Internet típusú kommunikáció	gyors, fully switched LAN típusú kommunikáció
csomópontok fluktuációja nagy	csomópontok megbízhatók, a kiesések és a hálózati hibák ritkák
teljesen elosztott működés	központi elemeket is használ (beléptetés, biztonság, közös fájl rendszer)
laza együttműködés, a csomópontok egymástól nem függnek	szoros együttműködés, a csomópontok egymás munkájára és eredményeire építenek
biztonsági megoldások kellene a csomópontok között	a csomópontok közötti kommunikációs közeg biztonságos, tűzfalak és titkosítás csak a klaszter külső felületén szükséges
csak master-slave vagy oszd meg és urajkodj séma	üzenet kezelésre építő összes programozási sablon használható

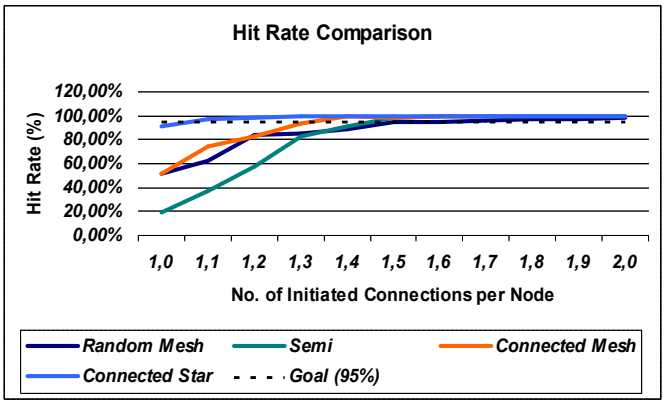
VI.5 P2P rétegek



Réteg	Funkció
Alkalmazás leíró réteg	• **Alkalmazások (applications):** az alatta elhelyezkedő komponensek összefogja, futtatja, és megteremti a felhasználói interfészt. • **Szolgáltatások (services):** azon komponensek sorolhatjuk ide, melyek futtatása a háttérben történik, pl. más csomópontok kéréseit dolgozzák fel (szerver folyamatok). • **Eszközök (tools):** a felhasználó rendelkezésére bocsátott eszközök összessége, melyek működését az alatta elhelyezkedő rétegek valósítják meg, vagy azok működését befolyásolják. Például környezeti beállításokat végző eszközök.
Osztály specifikus	típust ez a réteg határozza meg: • **Ütemező (scheduling):** pl. elosztott számításoknál a feladatok végrehajtásának az ütemezése. • **Adatokat leíró adatok (meta-data):** Ilyen adatok lehetnek például a tárolt dokumentumok verzió száma, szerzőjének neve stb., egymás közötti kommunikációja során használt üzenetek formátumának a leírása (A Gnutella esetében tipikusan csak ez a komponens lett meghatározva) • **Üzenet kezelés (messaging): (**kollaborációs alkalmazások esetén fontos) ez írja le a csomópontok közötti kommunikáció tulajdonságait (elemi események leírása, feldolgozása). Szoros együttműködésben dolgozik az ütemezővel. • **Menedzsment (management):** az alatta elhelyezkedő rétegek menedzselése a feladata.
Robosztussági	biztonság, erőforrás egyesítés (típus szerint: tárkapacitás, proc. idő), megbízhatóság (hibatűrés): • **Hiba detektálás (faulire detection)**, • **Kár felbecslés (damage assessment)**: meghatározni a rendszernek azon részét melyet érinti a hiba. • **Hiba kezelés (fault recovery)**: a rendszer eredeti állapotát vissza állítani, amennyiben ez nem lehetséges, biztonságos állapotba vinni. • **Hiba helyreállítás (fault repair)**: ha lehetséges a rendszert úgy módosítani, hogy a hiba többet ne következhessen be.
Csoport-menedzsment	csomópontok egymásra találása, routolás
Kommunikációs	felette elhelyezkedő rétegek számára elfedje a fizikai hálózatot (egyedi címzés)

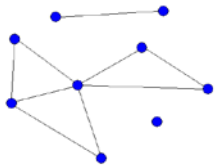
VI.6 Topológiák

- Random Mesh: teljesen véletlenszerű gráf, worst case
- Semi-Random Mesh: minden node-nak van legalább egy kapcsolata
- Connected Mesh: összefüggő gráf
- Connected Stars: összefüggő gráf, belső és levél csomópontok



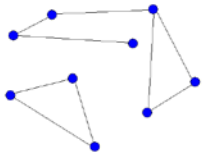
VI.6.1 Random Mesh

+Meglehetősen egyszerű
–Nincs túl sok köze a valósághoz
•Worst-case referenciának megteszi



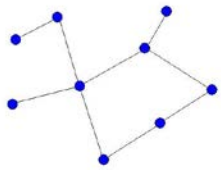
VI.6.2 Semi-Random Mesh

+Nem kifejezetten bonyolult
+A node-ok aktívan csatlakoznak
+Talán karban is tartható



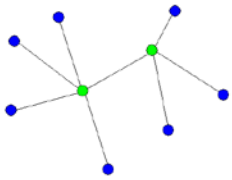
VI.6.3 Connected Mesh

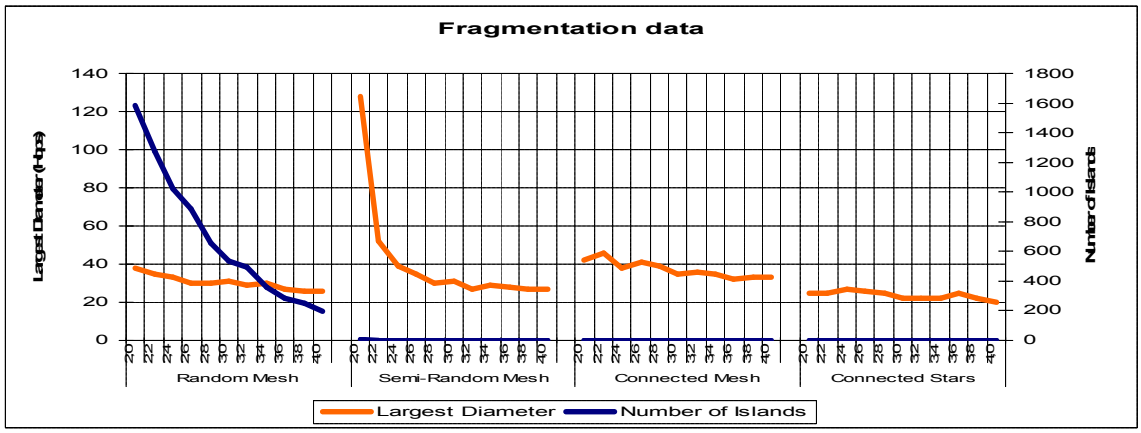
–Létrehozása nehéz
+Nincsenek benne szigetek
–Karbantartása nehéz

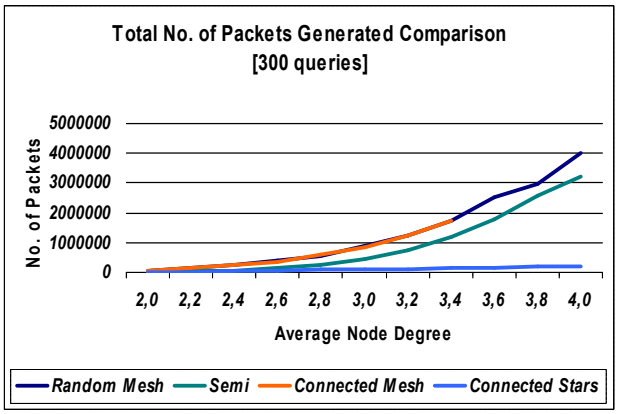


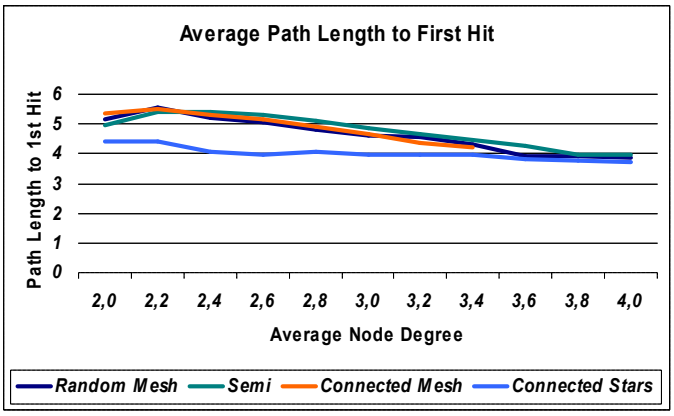
VI.6.4 Connected Stars

•Perzisztens maghálózat esetén
+Egyszerű létrehozni
+Nagyon valóságos
+Nincsenek benne szigetek
+Karbantartási igénye alacsony









VI.7 Adaptív TTL

- célja a forgalom csökkentése
- lényege, hogy több körben küldjük ki a kérést a TTL értéket fokozatosan a maximumig növelve

Type	Degree	HitRate	Packets from TTL1...7							Total #packets	Compared to TTL=7
Random	3,2	94,7%	0	0	0	45713	68880	113285	123557	351435	28%
	3,4	97,7%	0	4556	0	56068	78636	90175	71017	300453	17%
	3,6	96,7%	0	5045	0	62322	94024	86435	81563	329389	13%
Semi	2,8	96,0%	862	0	0	0	49506	52036	67349	169753	62%
	3,0	97,3%	0	0	9752	0	64373	58081	71217	203422	44%
	3,2	93,3%	0	0	0	34364	61282	55791	47675	199111	28%
Connected	2,6	93,0%	803	0	0	0	52389	73262	70169	196623	59%
	2,8	96,7%	0	0	10449	0	64989	79946	66155	221538	39%
	3,0	98,3%	0	0	12221	0	80161	94733	69511	256626	32%
ConStars	2,0	92,7%	0	0	19883	0	64852	0	89439	174175	63%
	2,2	96,7%	0	0	20346	0	86264	0	100156	206766	45%
	2,4	98,0%	0	0	21810	0	111492	57900	44213	235415	34%

A Semi és ConStars topológiák közül érdemes választani, a megcélzott hardverkörnyezet ismeretében:

Típus	Fokszám (95%)	Csomagok (300 Query)	ATTL	Megjegyzés
Random	3,4	1775000	20%	X
Semi	3,0	464000	60%	Egész jó
ConMesh	2,8	566000	40%	
ConStars	2,2	455000	50%	Ez is

VI.8 Esettanulmány: Gnutella

- Jól bevált, nyílt fájlmegosztó protokoll
- Teljesen elosztott szerveződésű, de alacsony számításigény jellemzi, elárasztó jellegű
- Az egész egy üzenetfejlécen alapul:
  - GUID (gráfkörök detektálásához, útvonal választás optimalizálása)
  - Típus (Ping/Pong, Query/QueryHit, PushReq)
  - TTL (általában 7-ről indul)
  - Hops
- TCP kapcsolatok gráfja

érdekességek:

- utolsó csomag azonosító megjegyzése (újrafeldolgozás és gráfkörök elkerülése)
- IP címet csak a visszairányú csomagok tartalmaznak (anomimitás, routolási táblában utolsó csomag mellett, hogy honnan jött, backward routing)
- push request kérés firewall miatt
- 8 kapcsolat/node esetén (TTL=7) ~87=2097152 üzenetet generál egy keresés

VII Vállalati alkalmazásintegráció

- rendszerszervezési módszertan az információs infrastruktúra hatékonyabbá tételéhez
- különös jelentősége van olyan vállalatoknál, ahol a nagyszámú ügyfélnek nyújtott szolgáltatások minősége a döntő választási szempont (bankok, bróker és telekommunikációs cégek)
- független alkalmazás szigetek összekapcsolása

VII.1 Az alkalmazásintegráció története

- kezdetben egy-egy kisebb, jól körülhatárolt terület munkafolyamatainak automatizálása
- az egyes részterületek automatizálása után a közöttük folyó, emberi közreműködést igénylő kommunikáció kiváltásával lehet további hatékonyság növekedést elérni
- Az integrált vállalati rendszerek célja
    - egységes felületen kell biztosítani az egyedi háttéralkalmazásokban tárolt és kezelt adatokhoz való hozzáférést
    - alkalmazások heterogenitása
    - az „ősi” rendszerekben rejtőző vállalati tudás hozzáférhetővé tétele
    - kapcsolódás más vállalatok alkalmazásaihoz
- szigetszerű alkalmazások együttműködésének biztosítása tipikusan egyesével, alkalmazáspáronként.
- Az alkalmazásintegrációs ráfordítás nagysága elsősorban az integrálandó alkalmazások mennyiségétől, és azok változatosságának mértékétől függ
    - néhány kritikus alkalmazás pár összekapcsolása nem jelent nehezen megoldható, jelentős ráfordítással járó feladatot
    - a kialakított kapcsolatok száma az összekapcsolni kívánt alkalmazások számával akár négyzetesen is nőhet
    - egy bizonyos méret elérésekor minden vállalat életében jelentkezik alkalmazások közötti teljes, szisztematikus integráció igénye
    - gyors (1 éven belüli) megtérülés csak nagy vállalatoknál

VII.1.1 Jelentős versenyelőnyök

- csökkenti az adminisztratív személyzet létszámát
- jelentősen felgyorsítja a munkafolyamatokat (csökkenti az emberi beavatkozásból és döntéshozásból származó szűk keresztmetszeteket)
- pontosabbá válik az információáramlás
- az egyes rendszerekben frissebb, naprakészebb kép alakul ki a velük kapcsolatban álló egyéb rendszerek állapotáról
- növelik a vállalat rugalmasságát és alkalmazkodóképességét
- az összekapcsolt rendszerekkel új, eddig elképzelhetetlen szolgáltatások válnak megvalósíthatóvá
- a vállalatok közötti kommunikáció, közvetlen adatcsere az információs rendszerek között, szorosabb, gördülékenyebb együttműködés.

VII.1.2 Az alkalmazásintegráció előkészítése

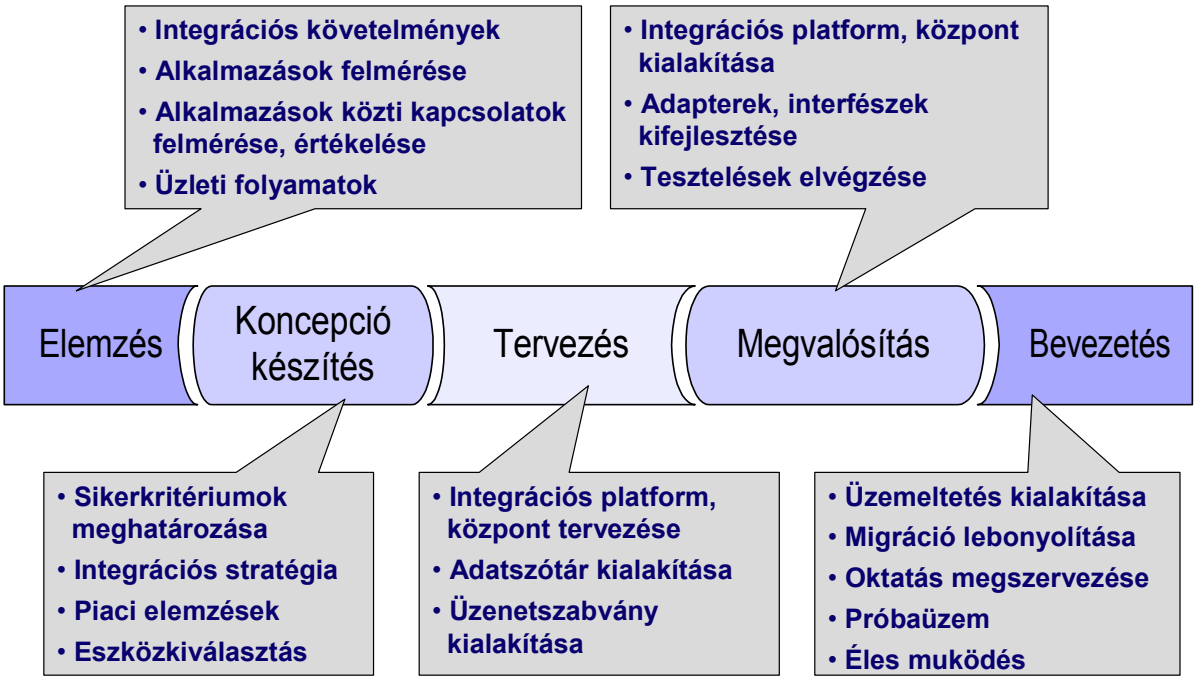
- Az alkalmazásintegráció egy számos előkészítő lépést igénylő és különféle döntési helyzeteket felvető hosszú folyamat.
- A legtöbb vállalati integrációs projekt kisebb, két alkalmazás pont-pont kapcsolatának kialakítására irányuló erőfeszítésnek indul, de a szerteágazó kapcsolatok és előre nem látott módon érintetté váló rendszerek nagy száma miatt hamarosan kezelhetetlen méretűvé eszkalálódik.

- Az integráció körültekintő előkészítést igényel
  - Érintett alkalmazások és kapcsolatok felmérése
  - Kockázatok felmérése
  - Határidők, költségek tervezése
  - Megfelelő integrációs eszközök és módszertan kiválasztása

VII.1.3 Az integrációs folyamat buktatói

- Változó alkalmazások és követelmények => könnyen hangolható és módosítható rendszerszerkezetre és jóval alkalmazkodóbb dinamikusabb projektvezetésre van szükség
- Az integrálandó alkalmazások sokfélék, változatos platformokon működnek
- Félig vak tervezés: az összekapcsolni kívánt rendszerek eredeti tervezői, fejlesztői általában elérhetetlenek, a dokumentációk hiányosak
- versengő, gyakran ellentmondó követelményekkel rendelkező szabványok: a módszertanok gyakran szorosan kötődnek egy-egy nagyobb cég termékeihez (IBM, Oracle, Microsoft) termékeihez
- az elhanyagolt apró, lényegtelennek tűnő részletek váratlanul és aránytalanul nagy szerephez juthatnak
- vállalat különféle szervezeti egységeinek eltérő prioritásai és dominancia harca => legyen egyértelmű ki hozza meg a végső döntéseket, illetve ki a felelős
- szervezeti egységek, vezetők, dolgozók protekcionizmusa, ellenállása, szabotázs akciói

VII.2 Az integráció folyamata



VII.3 Alkalmazáspárok integrálása

- Az integrálandó alkalmazások típusai
  - ***hagyományos (legacy) rendszerek***
    - jó minőségű, jól megtervezett
    - sok felhasználó együttes kiszolgálására képes
    - központi gép nagy terheléssel és magas kihasználtsággal működik
    - lassabb (több másodperces) reakcióidők
    - a nagy megbízhatóság
    - magas fejlesztési és fenntartási költség
  - ***saját fejlesztésű alkalmazások***
    - általában szélsőségesen a feladatra szabott, kisebb méretű alkalmazások
    - verziók kezelése nehézkes
    - drága karbantartás
    - alkalmazások minősége és dokumentáltsága változó
  - ***elosztott rendszerek***
    - kliens-szerver és a többrétegű architektúrák használata
    - A kritikus, nagyobb integráltságot igénylő részfeladatok adatbázis és állomány a szervereken
    - igényesebb, felhasználó barát grafikus felület
    - hibatűrés, skálázhatóság és terhelés-kiegyenlítés megvalósítása szerver oldalon
    - sokféle, viszonylag szorosan csatolt komponensből összeállított komplex alkalmazások
    - drága fejlesztés és karban tartás
  - ***készen kapható (off the shelf)*** kereskedelmi alkalmazások
    - olcsó
    - tipikus igényeket elégít ki
    - testre szabható
    - programozói interfészek (API)
    - jó minőségű, jól dokumentált alkalmazások
- rendszerkövetelmények tisztázása és formalizálása
  - milyen vállalati folyamatokat érint a kapcsolat kialakítására
  - hogyan (milyen formában) keletkeznek az adatok az egyik és hogyan dolgozhatók fel a másik rendszerben
  - az adatátvitel lehetséges módozatai időzítési, mennyiségi, biztonsági és elérhetőségi szempontból
- interfészek tervezése
  - fizikai összekapcsolás,
  - szóba jöhető protokollok,
  - időzítési lehetőségek, batch vagy valós idejű adatfeldolgozás,
  - biztonsági korlátozások,
  - forráskód szintű módosítás, API vagy valamilyen külső beavatkozás
  - konfigurálható paraméterek
- kapcsolatok kialakítása
  - élesben használt rendszer működésébe kell beavatkozni
  - utólagos módosítás, más célra készített ki- és bemenő interfészek használata komoly kockázatot jelent az addig probléma nélkül végzett feladatok szempontjából is
  - alkalmazások kódja vagy beállításai módosulnak, akkor újrafordítás, újratelepítés és újrakonfigurálás kellhet

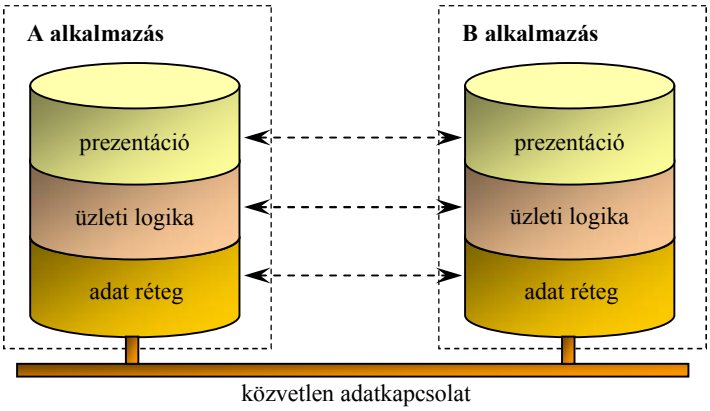
    - a módosítások nemcsak magára az alkalmazásra, hanem az azzal eddig is együttműködő vállalati környezetre is hatással lehetnek
- elkészült kapcsolatok, alkalmazások tesztelése
    - leválasztott teszt környezet kialakítása
    - teszt adatok biztosítása
    - nem kritikus időszakok (éjszaka, hétvége) kijelölése az éles tesztekhez
    - speciális terhelést generáló alkalmazások készítése elkészült kapcsolatok,
- áttérés
    - eredeti rendszer illetve az adatok mentése, és visszaállíthatóságának ellenőrzése
    - migrálni kell az eredeti rendszerben éppen aktuálisan megtalálható adatokat
    - éles környezetben és terheléssel megfigyelni a módosított rendszer működését
- karbantartás, utóélet
    - a kapcsolat is egy rendszerelem, működéséért felelni kell valakinek
    - terheltség, teljesítmény monitorozás
    - karbantartás, verziók, bugfix, változtatási igények kezelése
    - a kapcsolaton keresztül különféle, eddig ismeretlen hibák (például egy partnertől a belső logikának ellentmondó, vagy az adatintegritást sértő üzenet érkezik) és támadási felületek (például puffer túlírás, vagy illetéktelen hozzáférés lehetősége) jelennek meg a rendszerekben

Szempont	**Egyik választási lehetőség**	**Alternatív lehetőség(ek)**
Multiplicitás	Pont-pont: direkt kapcsolat pontosan két alkalmazás között. A dedikált csatorna előnye, a kisebb adminisztratív forgalom, és két résztvevő teljes szabadsága a kommunikáció lebonyolításában.	Egy-több: egyetlen alkalmazás több másikhoz kapcsolódik. Több-több: a kapcsolatot egyszerre több alkalmazás használja (például rádiós kapcsolat, vagy adatátviteli busz).
Időzítés	Szinkron: a kommunikáló felek egy időben vannak jelen. Gyors, egyszerű hibakezelés, pontos képet ad a kommunikáló partner állapotáról.	Aszinkron: A kommunikáló felek nem feltétlenül kell, hogy egy időben legyenek jelen. Nagyobb szabadság, jobb hibatűrés, nagyobb késleltetések jellemzik. A hibák kezelése nehezebb.
Indirekció	Közvetlen: a kommunikáló felek közvetlenül egymást címzik, egymáshoz beszélnek. Gyors, de megnehezíti a rendszer átkonfigurálását.	Indirekt: a felek valamilyen közvető, vagy átalakító közbeiktatásával kommunikálnak. Az indirekció jelentősen leegyszerűsíti az rendszer átkonfigurálását, az alkalmazások közötti újabb réteg a közvetlen csatolás megszüntetésével növeli rugalmasságot.
Állapot kezelés	Kapcsolatalapú: a felek közötti kapcsolat huzamosabb ideig fenáll, a kapcsolat teljes ideje alatt tisztában vannak a másik fél állapotával. Nagyobb erőforrás igény (kapcsolatok nyilván és életben tartása mind a két oldalon), kevesebb adatátvitel: minden kapcsolatfelvételnél elegendő egyszer autentikálni, nincs szükség minden műveletnél külön beléptetésre, protokol egyeztetésre, biztonsági és kódolási rendszerek tisztázására (például session kezelés).	Állapotmentes: minden művelet egymástól független. Nem kell állapotokat tárolni, viszont minden egyes alkalommal az összes adminisztratív információt (a felek kiléte, az üzenet kódolása, biztonsági beállítások) csatolni kell a kéréshez.
Nyugtázás	Kérdés-válasz (request-reply, handshaking): a protokoll szerint a felek felváltva kommunikálnak, explicit módon	Nyugtázás nélküli (fire and forget): A küldő üzeneteire nem vár választ, és nem lehet biztos abban, hogy az üzenet eléri a

	megerősítve (nyugtázva) a partner előző üzenetét. A válasz megérkezéséig nincs újabb forgalmazás, a válasz elmaradása hibát jelent.	célját. Aszinkron, vagy egyszerre több partner felé irányuló kommunikáció (broadcast, multicast) esetén használják. Nincs szükség nyugtára megbízható közegben. Sok változó elérhetőségű (például mobil) partner esetén a nyugták feldolgozása, vagy újraküldése járna túlzott erőforrás igénnyel (bizonyos eseményre beregisztrált partnerek tömeges értesítése).
Biztonság	Biztonságos: a kommunikáló azonosítják egymást, felek közötti adatátvitel kódoltan zajlik. Előny: az adatátvitel nem letagadható, visszakereshető a partnerek kiléte, illetve az adatátvitel tartalma.	Nem biztonságos: a kommunikáló felek közötti adatátvitel nyílt. Előny: egyszerű, gyors kommunikáció.

VII.4 Az integráció réteg szintjei

Az integráció folyamata szorosan összefügg rétegszerkezetek kialakításával.

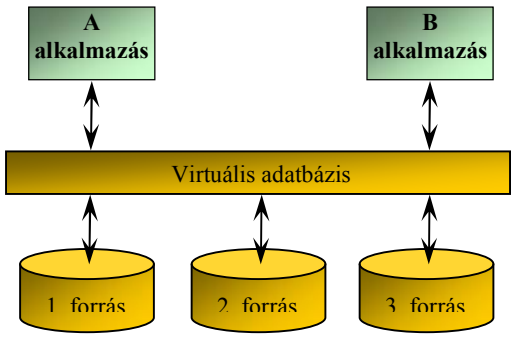


VII.4.1 Adat szintű integráció

Ez a legalsó szint, ilyenkor az adatok mentén integrálunk. Az integráció megvalósíthatóságának feltétele, hogy az adat kinyerhető legyen közvetlenül az állományokból vagy egy adatbázis interfészen keresztül.

- magukat a natív adatállományokat manipuláljuk (esetleg import/export egy szabványosabb, pl. szöveg, CVS vagy XML formátumba)
- kihasználjuk, hogy az adatkapcsolati réteg alsó részén gyakran egy szabványos adatbáziskezelő rendszert (DBMS) találunk

Inhomogén, eltérő adatmodellel, gyártóval vagy sémákkal rendelkező adatbázisok az egységes elérés érdekében érdemes egy közös homogén felületet kialakítani. Ekkor adat szinten egy virtuális adatbázist látunk, ami csak egy fedő rétégként létezik. A fejlesztők ehhez a virtuális adatbázishoz férnek hozzá.



Előnyök:

- ✓ nem igényli az alkalmazások változtatását (nincs verziózás, újratelepítés)
- ✓ egyszerű és olcsó (nem nyúlunk az alkalmazásokhoz)
- ✓ működik saját és kliens-szerver alkalmazásokra is, ha a DB a kezünkben van.

Hátrányok:

- ✗ csak akkor működik ha az adatbázis a kezünkben van, ami ritka
- ✗ más alkalmazások adatainak közvetlen manipulálása nagy kockázattal jár (elveszik az adatok és az azokat eredetileg kezelő alkalmazás egysége, így könnyen sérülhet az adatok integritására és biztonsága is)
- ✗ felesleges függőségi viszonyt hozhat létre az alkalmazott adatmodelltől és a kezeléséhez szükséges eszközöktől (extra elem a kommunikáló felek között)
- ✗ csak állapot és nem viselkedés szintű újrahasznosítás (a beolvasott állapot alapján újra meg kell valósítani az eredeti alkalmazás viselkedését). Nincs kód újrahasznosulás, ami további hibalehetőségeket teremt.

VII.4.2 API szintű integráció

Az alkalmazás programozói interfésze (API) egy olyan felület, amelyen keresztül szolgáltatásait más programok számára elérhetővé teszi. A minden egyes szolgáltatását ezen gépi interfészen keresztül érjük el, amit az alkalmazás becsomagolásával alakítunk ki. Általában ugyanazok a funkciók válnak gépi úton elérhetővé, mint az emberi felhasználók számára készített felhasználói felületről is kiválthatók.

Csomagolás módja:

- C/java nyelvű osztály vagy függvény könyvtárak
- web szolgáltatások
- valamilyen komponens alapú megoldás (COM, CORBA, J2EE, .NET)

Előnyök:

- ✓ egyfajta ipari szabvány az alkalmazások egymás általi vezérlésének megoldására
- ✓ minden alkalmazás a saját adatait kezeli, így elkerülhetők a nem megfelelő (inkonzisztens) külső adatmanipulációból fakadó problémák (nemcsak az adatok, hanem a viselkedés becsomagolását is megoldja, jobb kód újrahasznosítás)
- ✓ a polcról levehető alkalmazások szinte mindegyike rendelkezik valamilyen programozói felülettel

Hátrányok:

- ✗ változik az alkalmazás kódja, így újrafordítás szükséges
- ✗ az integráció általában hosszabb ideig tart és komoly költségekkel jár
- ✗ különféle felületetek nem feltétlenül biztosítanak teljes hozzáférést az alkalmazás minden funkciójához
- ✗ kellemetlen lehet, hogy a régebbi alkalmazások programozói felülete nem objektum orientált (például ilyen az ODBC is), hanem csak egyszerű függvénykönyvtárakat biztosít.

Akkor érdemes az alkalmazások összekapcsolását API szintű függvényhívások segítségével megoldani, ha az API már adott, és a közvetlen adatelérés integritási problémák miatt nem célszerű, vagy esetleg nem is lehetséges (akár az ismeretlen, akár nem hozzáférhető nem adatbázis sémák miatt).

VII.4.3 Metódus szintű integráció

Az üzleti logika újrahasznosítása itt úgy történik meg, hogy az egyes alkalmazások saját metódusaikat más alkalmazások számára is elérhető teszik. A gyakorlati megvalósításhoz különféle komponens technológiákat, alkalmazás szervereket és tranzakciós rendszereket (transaction processing monitors) alkalmaznak. A metódus szintű integráció lényegében az elosztott rendszerekkel történő alkalmazásfejlesztés vállalati alkalmazása, a legfőbb különleges szempont itt az üzleti logika szintjén értelmezhető funkciók megválasztása.

Előnyök:

- ✓ szabványos és jól dokumentált
- ✓ egyszerű, világos, objektum orientált szemléletű hozzáférést biztosít a közösen használt, gyakori vállalati funkciókhoz
- ✓ komponensnek függetlenek egymástól, egyenként telepíthetőek
- ✓ magas szintű kód újrafelhasználás: a konzisztens működés és az egyszerű karbantarthatóság (figyelni kell, hogy minden komponensből egy példány/verzió legyen)

Hátrányok:

- ✗ A komponens könyvtárak tárolásához, illetve futtatásához alkalmazásszerverek szükségesek.
- ✗ a különféle technikai jellegű (például tranzakciós vagy biztonsági) szolgáltatások támogatása komplikált anélkül, hogy minden metódusba külön beépítenénk azokat (a metódusokhoz való finom granularitású hozzáférési lehetőség)

VII.4.4 User Interface szintű integráció

A hagyományos mainframe (legacy) rendszerek konzolos elérésű alkalmazásainál terjedt el. Ebben az esetben csak az alkalmazás felhasználói interfészét használjuk fel, úgy hogy a felhasználót megszemélyesítjük programkóddal, így hívjuk az egyes alkalmazások funkcióit. Az eredményekhez a képernyőket dolgozzuk fel. Ezt a megoldást akkor használjuk, ha nincs más mód az integrációra, pl

- régi (legacy) alkalmazások esetében, melyek a régi technológiák (egzotikus fájlrendszerek, adatbázis kezelők) alkalmazása miatt abszolút semmilyen integrációnál használható felülettel nem rendelkeznek,
- azoknál az alkalmazásoknál, ahol az architektúrális eltérések, illetve a belső szerkezet, a gépi hozzáférési felületek vagy az állomány formátum ismeretének (dokumentáltságának) teljes hiánya miatt semmi egyéb lehetőség nem adódik
- ha az összekapcsolni kívánt alkalmazások semmilyen más módon nem férhetők hozzá egymás számára: különféle web alapú alkalmazások (portál technológiák), melyek a csak a prezentációs felületüket juttatják el a kliens oldali böngészőnek

Az ötlet nyilvánvalóan működőképes, hiszen az integráció megvalósítása előtt is a felhasználói felületen keresztül, természetesen emberi közreműködéssel, zajlott az alkalmazások közötti adatcsere.

- terminál emulátorok
- képolvasás, ami történhet sorosan, vagy DOM modell szerint
- Az adatkinyerés lehet statikus, amikor a sor, oszlop és pozíciót olvassuk ki, vagy lehet dinamikus, amikor is címkéket, szövegeket keresünk a képernyőn.
- A megfelelő volumenű információ kinyeréséhez párhuzamosítás szükséges. A mainframe szempontjából: sok konkurrens felhasználó.

Előnyök:

- ✓ mindig működik
- ✓ egyszerű megvalósítás
- ✓ a felhasználói interfészen keresztüli kapcsolattartás éppen olyan jól és egyszerűen működhet, mint a többi rétegben (sebesség növelés párhuzamosítással)

Hátrányok:

- × nagyon lassú
- × egyszerűsége ellenére munkaigényes és drága (sok képernyő és sok kézi munka)
- × az UI nem erre lett tervezve
- × nehéz a hibakezelés
- × a képernyő leképzések előállítása költséges és hosszú idő

VII.4.4.1 A megvalósítás alapvető lépései

- az alkalmazás működésének felmérése (logiaki sémák előállítása dokumentációból, forráskódból vagy akár intuitív módon)
- az adatok kinyerése képernyőképekből.(létre kell hozni egy képernyőkép katalógust)
    - képernyőképeket adatokká konvertáljuk (Screens-as-Data), és egy szöveges stream-mé alakítjuk
    - kinyert adatokat objektumokba rendezzük (Screens-as-Objects), az összegyűjtött adatokat nem közvetlenül adjuk tovább, hanem egy magasabb szintű nyelv (Java, Corba, COM) objektumaiba csomagoljuk
        - elegánsabb
        - összetett műveletek is definiálhatók a már kinyert adatokon
        - dokumentum szintű objektum modelt (DOM) is építhető
- a képernyőkép leképzések (mapping) előállítása. (Itt azt írjuk le, hogy az adott helyről kiolvasott adatot hogyan kell értelmezni a feldolgozás során. kimeneti formátumot, hibakezelés). A keresett információ helyének azonosítására két módszer:
    - Statikus: az adat helyét képernyő pozíció (sor-oszlop pár) adja. Ez egyszerű, mivel nem kell hozzá kereső logikát programozni, viszont gondot okoz, ha például a rekord típusától, vagy az adatok hosszától függően a képernyőkép változik.
    - Dinamikus: egy adott címke (szöveg) után keressük az adott adatot. Ez a módszer munkaigényesebb, viszont kevésbé érzékeny, ráadásul a kereső logika képes jelezni a hibát, ha nincs meg a keresett címke.

VIII Vállalati alkalmazás integráció

A vállalati alkalmazások összességének integrációja számos előnnyel jár

- a rendszerek közötti valós idejű információ megosztás,
- a vállalati folyamatok automatizálása, és ezáltal a szervezeti hatékonyság növelése,
- a rendszerek közötti információs konzisztencia fenntartása,
- a fejlesztések és a karbantartás megkönnyítése.

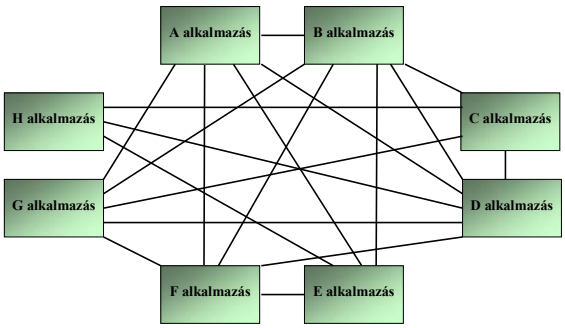
Nehézségek:

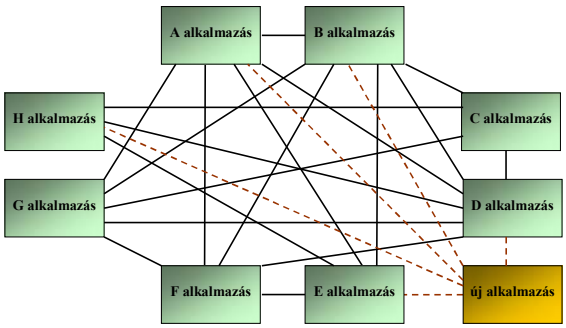
- az alkalmazás integráció nagy körültekintéssel végzendő, lassú és költséges folyamat
- hatóköre nehezen tervezhető, és könnyen eszkalálódik
- a megvalósítási költségek különösen a kis és középvállalatok költségvetéséhez képest riasztóan nagyok, a biztos és gyors megtérülés csak nagyvállalati keretek között

VIII.1 Spagetti

- Sok különféle vállalati terület, számos alrendszerrel
- A sokféle fent említett alkalmazás korántsem független egymástól, jellemző rájuk a kapcsolatok bonyolult hálózata
- Heterogén alkalmazások,
  - különféle informatikai kialakítással, eltérő platformok és operációs rendszerek
  - különbözhet az adatkezelési mechanizmusuk, rendszernyitvatartási idejük, folyamatkezelési módszereik és egyéb működési sajátságaik.
  - együttes üzemeltetésük már önmagában is nagy kihívást jelent
- A kapcsolatok kialakítása szintén egyesével, fontossági sorrendben történt.
  - Az egyes kapcsolatok egyediek
  - Elkészítésük két alkalmazás élébe is be kell, hogy avatkozzon (tervezés, változtatások, újrafordítás, tesztek)
  - A kapcsolatok száma nagy, általában az alkalmazások számával négyzetesen arányos
  - Teljes kapcsolatrendszer esetén $\frac{n(n-1)}{2}$ darab kapcsolat, szerencsétlen esetben $n(n\text{-}1)$ különböző interfész

Ez sok alrendszer esetén iszonyúan sok kapcsolat és nagyon bonyolult struktúra, gondoljunk csak a karbantartás nehézségére. Ezért ilyenkor integrációs módszerekhez kell folyamodni.





VIII.2 Integrációs alternatívák

A heterogén és strukturálatlan spagetti kapcsolatrendszernek a problémái:

- interfészek sokfélesége
  - az alkalmazások heterogének, különféle technológiákkal és egyedi szoftver megoldásokkal készültek, a kapcsolatok kialakítása is óhatatlanul egyedi lesz
  - áthidalás: az alkalmazások közötti kommunikáció szabványosításával (előírjuk az alkalmazások közötti kommunikáció során használt az adatok formátumát és az átviteli protokollt)
  - megvalósítás: middleware rendszerek:elegendő minden alkalmazást csupán a közös middleware rendszer nyelvére megtanítani ahhoz, hogy a rendszer bármelyik másik alkalmazásával képes legyen kommunikálni.
- a kapcsolatok nagy száma
  - az alkalmazások számát jóval meghaladja a kapcsolatok száma.
  - A kapcsolatok száma hatékonyan csökkenthető integrációs központ alkalmazásával: a kommunikáló alkalmazások közvetlen összeköttetését megszüntetjük, minden alkalmazáshoz csupán egy, az integrációs központ felé mutató kapcsolata lesz

Spagetti integráció helyett:

- MW réteg
  - szabványos kommunikáció
  - csak protokoll, nincs feltétlen központi elem (pl. RPC)
- Integrációs központ
  - megbízható, nagy rendelkezésre állású
  - indirekció bevezetése
  - biztonsági szolgáltatások
  - terheléselosztás

VIII.2.1 Az integrációt előről kezdjük

Az új igények figyelembevételével kidobunk mindent, és újrafejlesztünk egy integrált (esetleg monolitikus) megoldást

Előnyök:

- ✓ a vállalat igényei szabott integráció

Hátrányok:

- ✗ nagyon drága (alkalmazásonként kódmódosítást, újrafordítást, újratelepítést igényel)
- ✗ jelent kell lennie az alkalmazások értő módosításához szükséges szaktudásnak
- ✗ készen vásárolt alkalmazások képességei korlátoznak
- ✗ átalakítási folyamat tipikusan sokáig elhúzódik, a fokozat nélküli, gyors váltás megoldása általában nehéz
- ✗ csak egészen kis vállalatok esetében használható megoldás

VIII.2.2 Enterprise Resource Planning (ERP)

vállalatirányítási szoftver bevezetése: ilyenkor a vállalat megvesz egy dobozos terméket pl.: SAP, amely konkrét megoldásokat tartalmaz bizonyos vállalati egységekhez, és biztosítja az ezek közötti kapcsolatot is. A „best practice”-ek problémája: nem illeszkednek pontosan a konkrét vállalathoz, ezért mindig elengedhetetlen a konfiguráció, rendszerre szabás, amely sok időt vesz igénybe.

Előnyök:

- ✓ a szakterület „best practice”-einek átvétele a vállalat életébe
- ✓ szabványos megoldások
- ✓ támogatja a későbbi változtatásokat az integrálás és az egyes alkalmazások életében

Hátrányok:

- × drága
- × lassú, fokozatos áttérés az új rendszerre, ez nagy vállalatok életében akár 1-2 év is lehet
- × best practice nem feltétlenül illeszkedik a folyamatokhoz, akár a vállalati folyamatokon is alakítani kell
- × csak nagy vállalatok esetében éri meg

VIII.2.3 Middleware alapú alkalmazás integráció

nem maguknak a vállalatirányítás területeit lefedő alkalmazásoknak a helyettesítését tűzik ki célul, hanem megelégszenek a már meglévő alkalmazások kommunikációs rendszerének átalakításával

- nyílt szabványok, melyek tényleg csupán a kommunikáció szabványosítására koncentrálnak (RPC, CORBA, COM),
- termékek, melyek az integrációs központ előnyeit a kommunikációs szabványosítással kombinálva ennél lényegesen összetettebb, magasabb rendű integrációs feladatokat is megvalósít. Jó példa erre a relációs adatbázis alapú, vagy az üzenetkezelő és tranzakciós rendszerekre épülő middleware termékek alkalmazása.

Előnyök:

- ✓ jól dokumentált, ipari szabványok
- ✓ saját alkalmazásainkban könnyen implementálható
- ✓ nagy valószínűséggel a polcról levehető alkalmazások is támogatják
- ✓ az integrációs központtal: magasabb rendű szolgáltatások (biztonsági rendszer, aszinkron kommunikáció, tranzakció kezelés, naplózás, terhelés megosztás)

Hátrányok:

- × időigényes, drága
- × sokféle típus és gyártó van a piacon, ahol az egyes megoldások és termékek közötti átjárhatóság egyáltalán nem garantált

VIII.2.4 Integrációra tervezés

XML szabványok és web szolgáltatás szabványok felhasználásával tervezünk pl.: XML, XSLT, XSD, XML Schema, WSDL, UDDI, WS-SECURITY.

Előnyök:

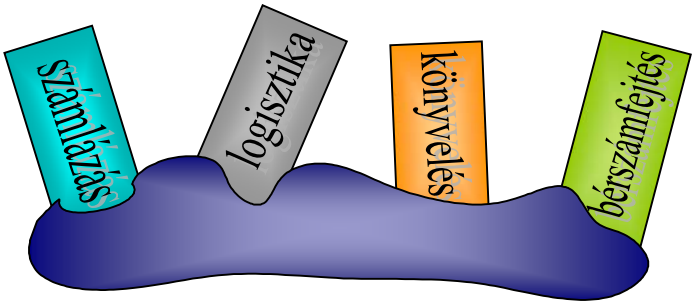
- ✓ jól dokumentált, ipari szabvány, széleskörű támogatottság
- ✓ olcsó megoldás
- ✓ eszközök leírósémák, transzformációk, interfészek létrehozásához

Hátrányok:

- × csak új alkalmazások és rendszerek esetén alkalmazható

VIII.3 Middleware

Jelenleg a middleware fogalma olyan rendszert takar, mely hálózati és adatbázis protokollok felhasználásával különböző platformok között egységes és biztonságos adattovábbítást biztosít.



Közös kommunikációs protokoll:

- közös adatábrázolás,
- időzítés,
- állapotkezelés,
- megbízható adatátvitel
- biztonság

Előnyök:

- ✓ egy interfészt kell készíteni, melyek az adott komponens a közös kommunikációs protokollhoz illesztik.
- ✓ alkalmazásonként nem egyedi, hiszen mindenhol azonos kommunikációs réteg fogadja a többi rendszerek interakcióit.
- ✓ vállalaton kívül eső rendszerekkel történő kapcsolattartás egységesül az üzenetek külalakja, szerkezete, ezzel az első lépés megtörténik a szervezet szintű üzenetszabvány kialakítására
- ✓ A szabályozott vállalati környezet megteremtéséhez az adatszótár, illetve a központi törzsadat kezelés kialakításának igénye is.

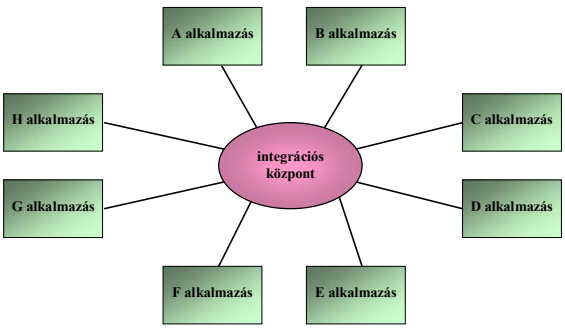
Megvalósítás:

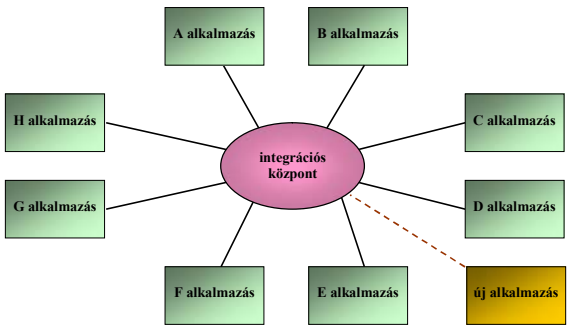
- Alkalmazásokba ágyazva
- Különálló komponensre is szükség lehet: integrációs központok

VIII.4 Az integrációs központ

Az integrációs központ kettős funkcionalitással rendelkezik.

- a közös kommunikációs rendszer alkalmazás független komponenseinek kialakítására (aszinkron kommunikáció, terhelés megosztás, indirekt kommunikáció, biztonsági adatbázis)
- a hub-and-spoke (kerékagy és küllők) architekturális tervezési minta megvalósítása





A rendezett kommunikációs struktúra legnagyobb előnyei:

- ✓ a kapcsolatok számának drasztikus csökkenése

- ✓ lényegesen egyszerűbb lesz az összekapcsolódó alkalmazás kommunikációja (egyetlen interfész szükséges)
- ✓ az alkalmazások közvetlenül nem látják egymást, az indirekt kommunikációs forma fontos előnye a rugalmasság
- ✓ a fejlesztők és üzemeltetők sokkal átláthatóbbnak találják a rendszert
- ✓ a kevesebb kapcsolaton keresztülfolyó üzenetek egyenletesebb terhelést biztosítanak (és kevesebb kapcsolat kisebb adminisztrációval jár).
- ✓ új rendszer információs áramba való bekapcsolása lényegesen leegyszerűsödik

Hátrányok:

- × központosítási séma: a feldolgozó- és áteresztőképesség korlátja, a kritikus hibapont létrejötte (megoldás több integrációs központ manuális kialakítása, illetve klaszterezés)
- × nagy hardver igény
- × a köztes állomás közbeiktatása miatt a kommunikáció késleltetése megnő, és az üzenetek átvitele általában is több erőforrást igényel a pont-pont megközelítéshez képest
- × magának az integrációs központnak a bevezetésével az új hardver, illetve új alkalmazás beszerzésével és működtetésével jár (eggyel több üzemeltetendő alkalmazást, és az ehhez szükséges szakértelem)

VIII.5 Business Process Management

Szükséges egy platform, ahol egyszerűen és olcsón megvalósíthatók a vállalati alkalmazások felett álló üzleti logikát megvalósító kisméretű rendszerek. A paradigma legfontosabb építőkövei az üzleti folyamat-támogató rendszerek (Business Process Management System – BPMS), valamint a vállalati szolgáltatás orientált architektúra (Service Oriented Architecture – SOA). A BPM egy keretrendszer, amely

- lehetővé teszi az üzleti folyamatok tervezését, fejlesztését, működtetését, monitorozását és automatizálását a folyamatban résztvevő emberek és különböző rendszerek tevékenységeinek összehangolásával
- üzleti folyamatok tervezésének, futtatásának és monitorozásának leválasztására a fizikai információs rendszerektől.
- A BPM eszközben futtatott folyamatok számára az egyes vállalati alkalmazások meghatározott, *ritkán változó szolgáltatásokat* (SOA elv) nyújtanak (pl. ügyfél számlaszámának megkeresése), de maga a *gyakran változó üzleti logika* ettől elválasztva működik.

A BPM rendszerek általában közvetlenül az integrációs központ funkcióira építenek:

- Az integrációs központ eredeti célja a spagetti jellegű kapcsolatok felszámolása
- a központi koordináció kihasználható: az egyes rendszerek központi vezénylésére, monitorozására, illetve működésük felügyeletére. Ezt a koncentrált információs csomópont az üzleti folyamatok pontosabb felmérésére, értékelésére és vezénylésére.

BPM feladatai

- a vállalaton belüli, illetve a más vállalati partnereket is érintő üzleti folyamatok koordinálása, az üzleti folyamatok felmérésére és monitorozására
- naprakész információkat biztosít az üzleti folyamatok aktuális állapotáról
- egyes folyamatok sebességéről és mennyiségéről statisztikák készíthetők
- kritikus folyamatok azonosítása és átszervezése, a változtatások eredményességének (hatásainak) a kiértékelésében is.
- dolgozók aktuális leterheltségének felmérése, a teljesítmény arányos értékelés, gyors és optimális átcsoportosítás
- ügyfelek kérdéseinek megválaszolását, panaszainak kezelését

VIII.6 Az alkalmazásintegráció funkcionális szintjei

Az alkalmazások közötti kapcsolatok kialakítása különféle mélységben történhet. A kapcsolatokat szorosságuk mértéke alapján három szintre szokták osztani:

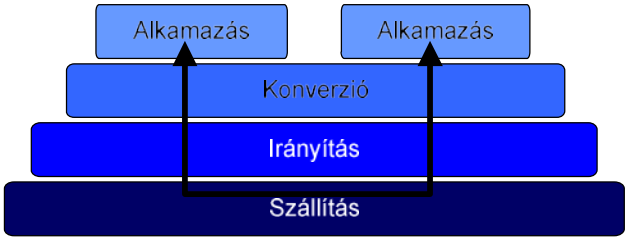
- az adatszintű kapcsolat kialakítása jelenti, amikor a vállalati alkalmazások csupán egymás adatait hasznosítják
- az alkalmazások már egymás funkcionális újrahasznosítására is képesek, vagyis az integráció már a szolgáltatások szintén történik meg.
- együttműködés a vállalati folyamatok mentén

VIII.6.1 Adatkapcsolat szintű integráció

Az adatszintű integráció (messaging):

- az alkalmazások ismerik a célrendszert
- egyszerű pont-pont protokollok segítségével adatokat szállítanak
- a vezérlő, konverziós funkciók megvalósítása az alkalmazások feladata
- A közös kommunikációs protokollok (middleware) azonban számos szolgáltatást nyújthat a prioritás-kezelés, garantált közvetítés, szegmentálás, kódlap konverzió területén.

VIII.6.2 Alkalmazás szintű integráció



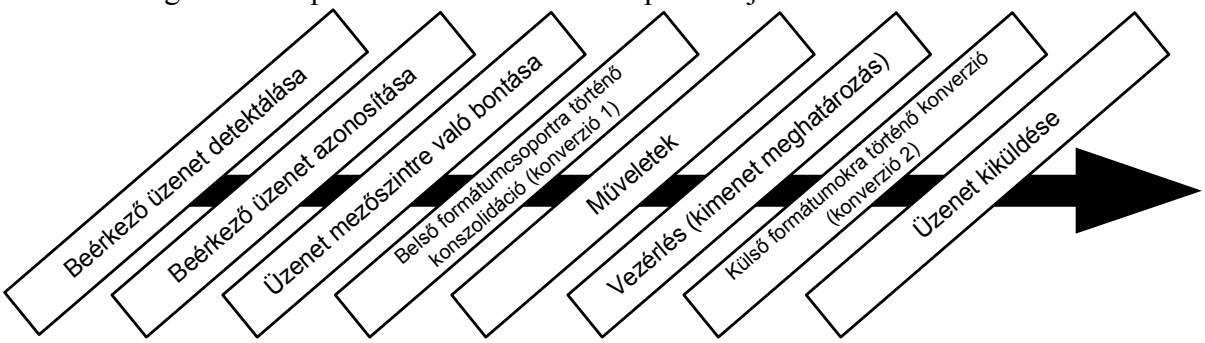
Alkalmazás szintű integrációnak (brokering): az alkalmazások már nem közvetlenül egymással kommunikálnak, hanem az adatcsere az integrációs központon (szolgáltatásgazda) keresztül valósul meg.

A szolgáltatásgazda végzi:

- az üzenetek alkalmazások közötti irányításából (routing)
  - irányítási tábla alapján
    - az üzenet fejlécében szerepel az üzenetfajta megnevezése, egy táblázatban tárolt vezérlő információk gyűjteményéből a cél rendszer beazonosítható
    - egy üzenetfajta csak egy irányban közlekedhet
    - . A több célállomásnak történő szétosztás egyszerűen megoldható
    - Ez az irányítási alternatíva a legelterjedtebb, azonban a valóságban gyakran alkalmazkodni kell a már létező rendszerek egyedi azonosítási struktúráihoz, eljárásaihoz
  - adattartalom alapján
    - a célállomás egy az üzenet belsejében található mező vagy mezőrészlet értékétől függ. az üzenetet az üzenetszerkezet ismeretében mezőkre kell bontani és a megfelelő mező értékkészletének megfelelően kell csak a továbbítást elvégezni.
    - erőforrás igényes, kisebb áteresztőképesség
    - csak elkerülhetetlen esetekben ajánlott az alkalmazása.
  - címzés alapján
    - gyors
    - Direkt címzés esetén hátránya, hogy a küldőnek ismernie kell azokat a modulokat és azonosítóit amelyekkel kapcsolatban áll

- nagyobb rugalmasságot, ha az alkalmazások nem indirekt módon (egy szolgáltatás névvel, vagy egy postaládát, üzenetsort megnevezve) az indirekció feloldása futási időben dinamikusan is megtörténhet.

- konverzió megvalósítása, a szintaktikai- (formai) és szemantikai (tartalmi) üzenet ellenőrzés
  - a különféle szabványok (SAP, natív, XML, delimitált szöveges) és ábrázolásmódok (fix pontos számok bájtsorrendje, nemzeti karakter kódlapok) közötti átváltásról.
  - üzenetek szerkezetének átalakítására is (mezők sorrendjének változása, esetleges ismétlődések eltüntetése).
  - adatformátum csere, itt elsősorban a valuták és mértékegységek konverziójára, illetve az eltérő nemzeti dátumábrázolási formátumokra kell gondolni.
  - Az integrációs központ üzenettovábbítási alapmodellje:



- Más üzemeltetést szolgáló feladatok:
  - modulkapcsolati vizsgálat
  - az üzenetszállító réteg működésének vizsgálata
  - üzenettorlódás monitorozás
- a középső rétegben tipikus szolgáltatásai
  - távoli eljáráshívás, szálkezelés, terhelés kiegyenlítés, átlátszó hibakezelés, perzisztencia és életciklus menedzsment, tranzakciókezelés, aszinkron üzenetkezelés, biztonsági funkciók.

Az integrációs központok több szabványos megoldást támogat, és az egyes alkalmazások adaptálásánál kiválasztható a legmegfelelőbb módszer. Az egyes protokollok közötti átlátszó konverziót az integrációs központ biztosítja.

VIII.6.3 Folyamatszintű integráció

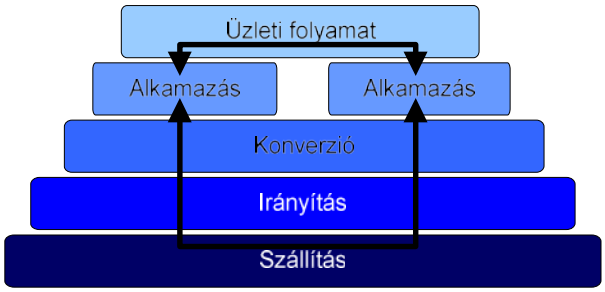
A legfelső szint az üzleti folyamatok integrálása (workflow), amikor az alkalmazások burkából kilépve a valós üzleti folyamatok mentén fűzzük fel a szolgáltatásokat Itt az alkalmazások fizikai tranzakcióinak sorozatát szerkezetbe rendezve egyetlen logikai tranzakcióként kezeljük. A folyamat szintű integrációnak kétféle típusa ismert:

Straight-Through Processing (STP)

- Emberi beavatkozás nélküli folyamatvezérlés;
- Munkát helyettesítő, teljesen automatizált folyamatok;
- Rövid lefutású tranzakciók;

Workflow

- Manuális beavatkozást igénylő folyamatok;
- Munkavégzést támogató kiegészítő funkciók támogatása;
- Hosszú lefutású tranzakciók;



VIII.7 Middleware rendszerek

- Magyarra fordítás: „köztes rendszer, alkalmazás”
- A middleware egy átvivő, transzport réteg különböző rendszerek közötti információ továbbításra, amely elfedi az OS-ek és a kommunikációs protokollok, hardverek specialitásait. A middleware szoftver biztosítja a kommunikációs alrendszert, amelyen keresztül az alkalmazások adatokat cserélhetnek.
- A middleware egy termék, mely **nem oldja meg automatikusan az információ-megosztást**, de **elrejti** a forrás és a célrendszer **komplexitását**, így a fejlesztőnek csak az adapterek elkészítésénél kell az alacsony szintű részletekkel foglalkozni (API, protokollok, stb.). Olyan **egységes felületet** biztosítanak az integrálandó rendszerek eléréséhez, hogy a **platformok különbözőségeit** áthidalják
- OS szintű támogatás
  - Windows: COM+ = COM + MSMQ + MTS
  - UNIX: DCE (Distributed Computing Environment)

VIII.7.1 Middleware típusok

A kereskedelmi forgalomban levő middleware-ek komplexek, sok funkcióval rendelkeznek. Egy szoftver **több megközelítést és modellt támogathat**, így a termékek kategorizálása nem egyszerű, és leginkább funkció alapján lehetséges. A middleware piaca lényegében hat különböző szegmensre bontható:

1. Távoli eljáráshívások (Remote Procedure Calls)
2. Üzenet-orientált middleware (Message Oriented Middleware)
3. Elosztott objektum kezelők (Object Request Brokers)
4. Tranzakciós rendszerek (Transaction Processing Monitors)
5. Komponens alapú alkalmazás-szerverek (Transactional Component Middleware)
6. Egyedi middleware megoldások (Homegrown middleware solutions)

VIII.7.1.1 Adatbázis alapú integráció

Egy **helyi** vagy **távoli** adatbázisok használ kommunikációs rétegként. Típusai:

- **CLI – Call Level Interface**: egy közös API, ami több DBMS-t támogat. Leggyakrabban relációs adatbáziskezelők esetén az **SQL** funkcionalitás:
  - **ODBC** (egy interfész, driverek)
  - **JDBC** (interfész szabvány: metódusok halmaza, azonos interfészen keresztül különböző adatbázisok érhetők el)
  - **OLE DB** (DB erőforrások objektumként elérhetők)
- **Natív** DB MW: Olyan DB elérési mechanizmusok, melyek **csak egy konkrét DBMS-sel működnek. Előnyük**, hogy a CLI-hez képest **jobb teljesítmény**t nyújtanak, kiküszöbölik a magasabb absztrakciós szint miatti overhead-et, és a DBMS minden funkciója elérhető

VIII.7.1.2 RPC

Nyelvfüggetlen, szabványos távoli eljáráshívás. Ma már nem használjuk, de a mai rendszerek őse.

- szabványos, egyszerű, jól érthető
- szinkron => lassú
- lassú vagy gyors? A lassúság kiküszöbölhető többszálúsággal
- direkt címzésű
- rugalmatlan
    - kis overhead
    - nehezen, drágán módosítható

Hiányosságok:

- hibakezelés
- csak pont-pont kapcsolatot kezel, többszörös kapcsolatkezelés hiányzik
- hordozhatóság csak IDL szinten
- pufferkezelés és ürítés
- szinkronizáció a processzek között

VIII.7.1.3 CORBA, DCOM (ORB)

Nyelvfüggetlen, objektum-orientált távoli eljáráshívás.

- RPC alapú
- kiegészítések
    - hibakezeelés
    - objektum-orientált csomagplás
    - hordozhatóság kód szinten
- örökli
    - direkt címzés
    - kis teljesítmény
    - szinkron

VIII.7.1.4 Házi rendszer

- alapos tervezés kell
- lehet, hogy első pillantásra olcsóbbnak tűnik
- skálázható?
- új emberek meg fogják tudni érteni?

VIII.7.1.5 MOM

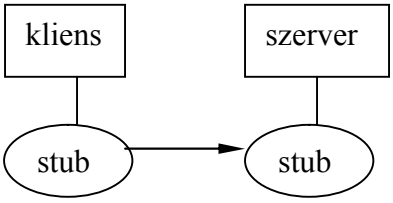
- aszinkron működés
- indirekt címzés
    - laza csatolás
    - nagy throughput
    - egyes üzenetek lassabbak lehetnek
- nagyobb overhead
- könnyebb karbantartás, átkonfigurálás
- extra szolgáltatások
    - terheléselosztás
    - biztonság
    - hibatűrés

- különböző rendelkezésre állású rendszerek közt is működik
- több-több kommunikáció lehetősége

VIII.7.1 RPC - Távoli eljáráshívás

Az egyik legrégebbi típus. Használata **egyszerű**, jól érthető: a kliens egy függvényt hív, melynek végrehajtása a szerveren (távoli gép) történik. A műveletet **szinkron** módban futtatja (a kliens blokkolódik). **Hátránya** a lassú működés, mivel hogy nincs aszinkron végrehajtása. Request/reply kommunikációs modellt használnak. Az RPC-k jól működnek kisebb, egyszerű alkalmazások esetén, ahol alapvetően pont-pont kapcsolatú kommunikáció van, nehezen skálázhatók és túl sok kritikus részlet függ a programozótól:

- hálózati vagy rendszerhibák kezelése;
- többszörös kapcsolatok kezelése;
- hordozhatóság;
- pufferelési és ürítési ellenőrzés;
- processzek közti szinkronizáció.



A szinkron működés miatt a távoli eljáráshívások nem használhatók vállalatméretű alkalmazások építőköveinek, ahol nagy teljesítmény mellett magas megbízhatóságra van szükség. Az RPC szabványait már nem fejlesztik, de sok helyen használják: az operációs rendszerekbe észrevétlenül integrálódott: **DCE**, **Unix** (rpcd, lib, fejlesztőeszközök), **Windows NT**-től felfelé.Az **elosztott objektumkezelő rendszerek** (Java RMI, COM, CORBA) is RPC alapokon nyugszanak.

VIII.7.2 Üzenet-orientált middleware (MOM, Message Oriented Middleware)

Az RPC hibáinak orvoslása találták ki. Egy tipikusan **üzenetsor** menedzselő szoftver, ami **pont-pont** információtovábbítást tesz lehetővé. A **perzisztens** üzenet aszinkron módon jut el a címzetthez, ezért a küldő **nem blokkol**ódik. Az üzenetsor **laza csatolás**t eredményez, az alkalmazások egymástól függetlenül működnek.

Előnye: hogy az üzenetek kicsik, továbbításuk olcsó (séma/adat, egy rekord kell), jó teljesítményű (üzenetek prioritásai, terhelésmegosztás, thread pooling), megbízható (üzenet nem vész el, ha megszakad a hálózati kapcsolat). **Megvalósítások**: MSMQ, MQSeries.

A MOM termékek nem csak kizárólag üzenettovábbításra használhatók; általában tartalmaznak üzenetfordító szolgáltatásokat, biztonsági modult, illesztő adaptereket többféle programhoz, hibajavítást, hálózati erőforrás-allokációt, routolási költségbecslést, üzenet priorizálási és igénylési, illetve hatékony debuggolási eszközöket.

Ellentétben az ORB-bal és az RPC termékekkel, a MOM többnyire nem feltételezi, hogy a rendszer megbízható hálózati réteggel rendelkezik. Ha az átviteli réteg megbízhatatlan, megpróbálja a problémákat az adott szinten helyrehozni, amelyen kommunikál. Ez például WAN vagy Internet alapú kommunikáció esetén szokott előfordulni.

Két különböző típusú MOM alakult ki:

- sorban állásos (message queuing),
- üzenet átadásos (message passing).

VIII.7.2.1 Message Queueing

A program-program kommunikáció egy soron (queue-n) keresztül történik, ami általában egy fájl. Ez lehetővé teszi, hogy a programok egymás közötti közvetlen kapcsolat nélkül küldjenek és fogadjanak információt. A kommunikáció a következő módon történik:

1. az alkalmazás **üzenetet küld** a queue menedzsernek

2. a QM **továbbít**ja az üzenetet a célalkalmazásnak
3. a célalkalmazás akkor **dolgozza fel** az üzenetet, amikor ideje van rá, így annak **nem is kell futnia**, amikor a forrás elküldi az üzenetet

Nagy, vállalati méretű alkalmazásokban a sorokat route-olásra is lehet használni, mivel beállíthatók üzenettovábbításra is, azaz a bejövő üzenetet tovább adhatják más soroknak is.

A módszer használata akkor indokolt, ha az alkalmazások nem kapcsolhatók össze közvetlenül (például a mobil számítástechnikában). Elengedhetetlen viszont megbízható és erős konfigurációt a rendszer alá tenni, mert igényli a pontos beállításokat és ellenkező esetben a teljesítménye nagyon gyenge lehet: ha valamilyen okból elveszik a hozzáférés egy sorhoz, akkor ez az egész rendszerre hatással lehet.

VIII.7.2.2 Message Passing (Publish-Subscribe)

Ez a módszer nagy, elosztott rendszerek építésénél különösen népszerű. Az alapvető különbség az előző típustól azon alkalmazások esetén jelentkezik, ahol az igényelt információ megkeresése erőforrás igényes. A publish-subscribe néven ismert kommunikációs modell lényege, hogy a programok vagy előfizetnek (érdekeltséget jelentenek be) adott témára, vagy üzenetet tesznek közzé az adott a témában. Amint egy program előfizetett egy témára, minden olyan üzenetet meg fog kapni, amit az elosztott alkalmazásban, ebben a témában közzé tettek.

A hagyományos hálózati alkalmazásokban, a kommunikáló folyamatoknak szükségük van a hálózati címekre az adatcsere elkezdéséhez: ha egy folyamat üzenetet akar küldeni a többi folyamatnak, először meg kell tudnia a fizikai hálózati címeiket, majd felépíteni mindegyikkel a kapcsolatot. Ez a modell nem biztosít megfelelő skálázhatóságot, mivel bonyolult (és nagy hibaforrást jelent) a bekonfigurálása. A publish-subscribe kommunikációs modell viszont átlátszó helyfüggetlenséget biztosít, hiszen a programnak elég a témát megjelölni, és elküldeni az üzenetet. A middleware ezek után elirányítja az üzenetet minden olyan programhoz, amely előfizetett az adott témára.

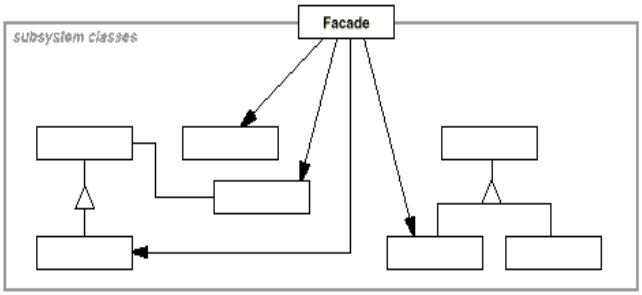
A MOM forgalmazók jellemzően kiegészítő ügynökökkel (agent) együtt implementálják ezt a modellt (például valósidejű adatbázis karbantartó, téma-előjegyzést kérő, programokat listázó ügynökök). Ezáltal tipikusan egy program, amikor közzétesz egy üzenetet, ezt egy ügynökhöz kapcsolódva teszi (nem feltétlenül kell ugyanazon a gépen lennie) és elküldi neki az üzenetet. Ezek után az ügynök fogja elirányítani azt a megfelelő programokhoz. Itt is megmutatkozik a MOM modellek közti különbség: a publish-subscribe modellnek nagy a hibatűrése, mert az ügynökök tudják dinamikusan route-olni az üzeneteket, esetleg képesek rendszerhiba miatt bekövetkező leállásból származó hibát javítani. Nyilvánvaló tehát, hogy ezt a modellt legfőképp magas szinten elosztott rendszerek esetén alkalmazzák, ahol szükség van hibatűrésre és jó teljesítményre is. **Nem használható** viszont olyan esetekben, **amikor a folyamatok hosszú ideig le vannak kapcsolódva a hálózatról**.

VIII.7.3 Elosztott objektum kezelők (Object Request Brokers)

Ezek nemcsak middleware-ek, hanem **alkalmazás fejlesztési mechanizmusok** is.

- nyelv-független, objektum-orientált RPC
- szabványos **interfészek + protokollok** a kommunikáció megvalósításához

Az **elosztott objektumkezelő rendszerek** (COM, CORBA, EJB, .NET) is RPC alapokon nyugszanak de annál magasabb absztrakciós szintet nyújtanak a programozók számára erős OO szemlélettel. A nagy architektúrális elegancia, és a sok új funkció mellett öröklik az RPC-k **mérsékeltebb teljesítményét**. **Gyengébb teljesítmény**, mint a MOM, mert több a hálózati kommunikáció. 7 fajta üzenettípust különböztetünk meg, melyeket absztrakt protokollon (IIOP, GIOP) keresztül



továbbítódnak. Megoldást az jelent, hogy a gyártók **optimalizálnak**, és a tervezők **csökkentik a szemcsézettséget**. A granularitás csökkentésére jó módszer a **Façade** tervezési minta, ahol új objektumot vezetünk be a komplex alrendszer elé, így elrejtve azt. **Előnye**, hogy lazább csatolás lehetséges és így, az alrendszer a klienstől függetlenül módosítható.

VIII.7.4 Tranzakciós rendszerek

A **tranzakció** parancsok atomi (oszthatatlan) csoportja, amelyek **vagy mind sikeresek**, **vagy** akár egy parancs **sikertelensége esetén az összes parancs visszavonásra kerül** (COMMIT vagy ROLLBACK). Ez **távoli erőforrások** esetén is érvényes.

- **ACID** tulajdonságok
  - **Atomic** – vagy mind, vagy egy sem
  - **Consistent** – mindig konzisztens állapot áll fenn
  - **Isolated** – a tranzakciók egymástól függetlenek
  - **Durable** – a konzisztens állapot rendszerhiba után is megmarad
- **Két fő szolgáltatás**
  - Tranzakciók **integritásának** a garantálása: a résztvevő elemek kezdeti állapotáról másolatot készít, és eldobja azt, ha a minden részművelet sikeres volt
  - **Erőforrásmenedzsment** + runtime menedzsment szolgáltatásokat nyújt

Transaction Processing Monitorok, (alkalmazás-szerverek, message brokerek) az **információmozgatás** és **metódus megosztás** (method sharing) támogatását szolgálják. Az együttműködésből inkább **szorosan csatolt** integrációt támogat (pl. a MOM lazán csatolt)
Történet: a mainframe világban **beépített TP Monitor** a processzek + DB elérés koordinálása.
Előnyük, hogy a **centralizált** feldolgozás egyszerűsége miatt csökkentheti a rendszer árát, hibatűrő, sok kliens kiszolgálására képes, így valószínűleg még sokáig fennmarad.
Hátránya, hogy az integrálandó alkalmazásokat módosítani kell**.**
Felhasználási terület: elosztott alkalmazások egy vállalaton belül
Szolgáltatások:

- alkalmazás **kód** elhelyezése (**hostolása**) és **futtatása**
- **metódusok megosztása** alkalmazások között
- üzleti szabályok (**business rules**) betartatása, így **adatintegritás** biztosítása
- alkalmazás **fejlesztés támogatása** (tranzakciós szolgáltatások elérhetővé tétele a kliensek számára procedurális szemlélettel)
- **adatbázis multiplexelés**
  - a connection pooling őse
  - a kliens a TP MW-hez csatlakozik, a TP a DB-hez
  - DB kapcsolatok száma csökkenthető
  - 3 rétegű architektúra alapjai
- **terhelésmegosztás**
  - **több szerver** (esetleg csak **processzor**) között **dinamikusan** megosztja a terhelést
  - **taszkok prioritásos ütemezése**
    - alkalmazás típusa szerint,
    - igényelt erőforrások szerint,
    - igényelt válaszidő szerint, ...

Alkalmazások közti kommunikációt támogató elsőgenerációs alkalmazás szerverek. Alkalmazás **logikát tárol/futtat** és **hozzáférést** biztosít alkalmazásokhoz, adatbázisokhoz, üzenetsorokhoz, stb. Az kapcsolódáshoz alkalmazás fejlesztés szükséges, ezért **jó teljesítmény** érhető

el. Jól skálázható, sok klienst képes kiszolgálni - sor alapú input pufferek, prioritások ütemezése, többszálú végrehajtás terhelésmegosztás használatával -, viszont **magas load**-ot generál. Pl: **BEA Tuxedo**, **MTS**, **CICS**

A tranzakciós monitorokra általában olyan kritikus alkalmazások épülnek, melyek folyamatosan gyors válaszidőt, magas szintű biztonsági rendszert és erős adatbázis integritást igényel. Maga a kommunikációs költség általában kicsi, mert az adatcsere általában egyszerű request/reply alapon megoldható. Mindehhez a TP monitorok alkalmazásfejlesztő eszközöket (adatbázis interfész), rendszeradminisztrációt (biztonság, hangolás) és tranzakció futtatást (időzítés és kihasználtság) biztosítanak.

A TP monitorokat általában nem a szokásos program-program kommunikációban használják, sokkal inkább tranzakciós alkalmazások számára szolgáltatnak egy teljes környezetet, amely kapcsolatban áll a szükséges relációs adatbázisokkal. A TP monitorok használatát akkor kell megfontolnunk, ha tranzakcióinkat sokféle adatbázison át kell koordinálni és szinkronizálni. Ugyanakkor tudni kell, hogy ezek az eszközök drágák és bonyolultak szoktak lenni, így nagy gyakorlottságot igényel a helyes bekonfigurálásuk, ezért a legtöbb TP forgalmazó jelentős szervizrészleget tart fenn.

VIII.7.5 Komponens alapú alkalmazás-szerverek

Az alkalmazás szerverek TP Monitorok **kiterjesztései**: komponens alapú feldolgozás, alkalmazás logika megosztás, végrehajtás. **Konnektorok** back-end erőforrásokhoz (ERP, Legacy) és támogatják a **UI fejlesztés**t. **Web support** jellemző, alkalmazásai többségében **Java alapú**ak. **Domináns szabványok:**

- Enterprise Java Beans (EJB)
- COM+ = COM + MSMQ + MTS (a Microsoft Transaction Server (MTS), amely mostanra része a COM+-nak)

J2EE – EJB	COM+ + MTS + AppCenter
Menedzsment	**környezet**
Életciklus menedzsment	**Tranzakcióképes COM+ komponensek**
Tranzakciókezelés	Hagyományos TP monitor funkciók
Perzisztencia	**Windows** környezet
menedzsment	**Appcenter**: **transaction server** + **message**
Security services	**queueing** – nagy load

Előnyök: TP monitorok előnyei**, OO** paradigma**, komponens alapú** fejlesztés – újrafelhasználás, **webes** interfészek, képes különböző cégek közötti tranzakciók, alkalmazás logika és adattámogatatás használatára a szabványokra támaszkodva. **Hátrány**: **sok programozási** munkát igényel, ha alkalmazásokat akarunk integrálni alkalmazás szerverekkel, különösen, ha a **forrás- és célrendszer kódja is változtatandó.**

VIII.7.5.1 A két komponens technológia összehasonlítása

A komponens jól definiált interfésszel rendelkező, futtatható kódrészlet, mely tetszőlegesen kivehető, beágyazható már megírt környezetekbe.

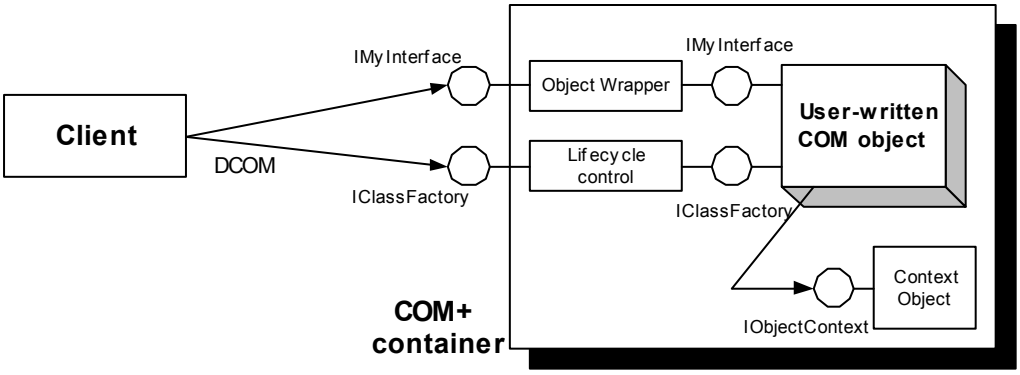
A Java Beanek Java virtuális gépen futnak (Java Virtual Machine - JVM), a COM kód pedig Windows operációs rendszerek alatt. A Java RMI-vel (Remote Method Invocation – távoli függvényhívás) módunk van távoli Java Beaneket megszólítani, míg a DCOM–mal a távoli rendszeren lévő COM komponenseket is futtathatók. Mindkét komponens rendszer képes adatbázisok elérésére is.

Ahhoz, hogy többféle komponens összehangoltan tudjon működni, szükség van egy ún. konténerre. A konténer menedzseli a komponensek életciklusát, valamint számos gyakran használt szolgáltatást nyújt feléjük, sok kódolástól kímélve meg ezzel a fejlesztőt. A konténernek szüksége van bizonyos információkra (pl. tranzakció támogatás) a komponensekről ahhoz, hogy megfelelően tudja

kezelni őket. Ez COM+-ban a COM+ Explorerrel, EJB-k esetén pedig XML leíró fájlokban adható meg a komponensekhez.

A két technológia nagyon hasonlít egymásra architektúrális szempontból, egy lényeges különbség a két megoldás között az objektumok inaktiválásában van. Egy nagyobb elosztott rendszerben sok felhasználó létezhet, akik rengeteg komponenst használnak. Mivel ilyenkor nem megengedhető, hogy az összes már valaha használt objektumot folyamatosan memóriában tartsuk, ezért szükség van az objektumok inaktiválására.

VIII.7.5.2 COM+



A komponens írójának tudatnia kell a konténerrel, hogy az adott komponens milyen tranzakciós izolációs szinteket támogat. Egy COM objektum esetében ezek az alábbiak lehetnek:

- **Requires transaction** – A komponens csak tranzakciós környezetben futhat, így amennyiben a kliens már futtat egy tranzakciót, akkor az objektum ahhoz csatlakozik, amennyiben nem, a kliens létrehoz egy új tranzakciót.
- **Requires new transaction** – A kliens minden esetben új tarnzakciót indít, mihelyst a komponens létrejön.
- **Supports transaction** – A komponens futhat tranzakción belül és kívül is.
- **Does not support transaction** – A komponens nem támogatja a tranzakciós futást.

Egy COM+ objektum inaktiválása kétféleképpen történhet. Definiálható úgy, hogy az objektum minden egyes művelet után inaktívvá váljon. Ebben az esetben a rendszer minden hívás után alapállapotba kerül. Megadható, hogy az inaktiválás a tranzakció befejezésekor történjen. Ez esetben módunk van a komponens többszöri hívásra is a tranzakción belül, miközben az objektum megőrzi állapotát.

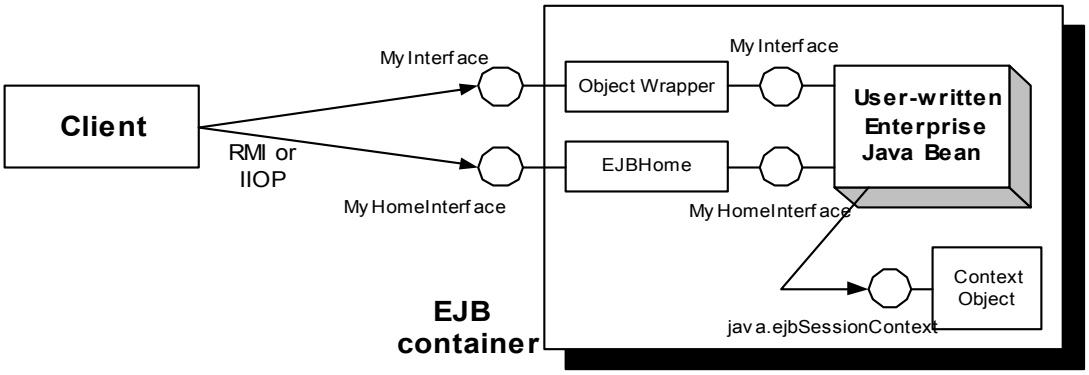
VIII.7.5.3 EJB

Az EJB lényegében egy szabvány, melynek számos implementáció létezik (pl.: BEA, IBM, Oracle stb.) Az EJB-ben a komponensek két fő típusra oszthatók entity és session bean-ekre.
Állapottárolás szempontjából a session beanek tovább kategorizálhatók:

- **állapotmentes (stateless) session beanek** – Működésük hasonló azon COM komponensekéhez melyek minden művelet után inaktívvá válnak.
- **állapottal bíró (stateful) session beanek** – Megőrzik állapotukat a teljes élettartamuk alatt.

Az állapot tárolása a konténer feladata, tőle függ mikor inaktiválja a bean-t és írja ki a háttértárra (adatbázisba) az információkat. Jól meg kell fontolni egy bean-t inaktiválásának idejét, hiszen pl. ha a kliens tranzakciójában visszagörgetés (ROLLBACK) következik be egy hiba miatt,

elvárható a session bean-től, hogy állapota is a tranzakció előtti helyzetet tükrözze. Ezért az EJB állapota csak a kliens tranzakció sikeres befejeződése után menthető. A teljes biztonság érdekében a konténernek kétfázisú commit-ot csinál, hogy összehangolja az EJB commit-ot az adatbázis commit-tal.



VIII.7.6 Egyedi middleware megoldások

Amikor a vállalatok először ráéreztek a middleware réteg szükségességére, megpróbálták a sajátságos kisebb problémáikat a saját fejlesztőikkel megoldani, ahelyett hogy beruháztak volna egy drágább middleware csomagba. Az így elkészült számos middleware-szerű megoldás, amely – bár tökéletesen megfelelt egy speciális célnak –általában nem volt sem skálázható, sem új kihívásokhoz rugalmasan átalakítható. A nagy cégeknél ez komoly hátránnyal is járhat, mivel a folyamatos foltozgatások és bővítgetések végeredményeként, amit főleg olyan emberek végeztek, akik még soha nem írtak middleware szoftvert, a házi készítésű program könnyen drágább lehet a polcról levehető megoldásoknál, és továbbra is skálázhatatlan, vagy nem kellően robosztus maradt.

VIII.8 A middleware kiválasztása

VIII.8.1 Szempontok

- Az **alkalmazás-integrációval szemben támasztott igények** (miért van szükség a middleware-re)
  - **minden alkalmazás képes legyen kommunikálni**
    - bármely más alkalmazással
    - bármely irányban
    - valós idõben
  - **minden releváns információ** tetszőleges alkalmazásban **elérhetõ legyen** más alkalmazások számára
  - **adatok és üzleti folyamatok megosztása**
    - az integrált **alkalmazások** és **adatforrások** között
    - **vállalaton belül** és vállalatok **között** is
    - **integritás** biztosítása
    - újrafelhasználhatóság
  - **minimális módosítások** a már meglévõ alkalmazásokban
  - **korábbi rendszerek** jellemzőinek **kiaknázása**
    - ERP
    - korábbi Middleware
- Már **meglévõ** alkalmazások és **fejlesztendõ** alkalmazások **aránya**
  - alkalmazások **heterogenitása** megkötés
  - új alkalmazások esetén kevesebb a megkötés (a rendszerkomponensek **tervezése a MW-hez illeszkedve)**

- A middleware-t használó szervezeti **egység mérete (**projekt, munkacsoport, osztály, egész cég, több vállalat)
- **Idõbeli korlátok** (kiválasztás, implementáció, használat)
- **Kockázati** tényezők (**ár**, megvalósíthatóság)
- **Szabványok** (vállalati policy-k, nyílt/zárt, hivatalos/de facto)

VIII.8.2 Kiválasztás folyamata

Nincs kiforrott módszertan. Lépések:

1. Middleware kategória kiválasztása (elsődelges cél, jellemzők, gyakorlati alkalmazásuk alapján)
2. Konkrét middleware termék(ek) kiválasztása (megvalósíthatóság, támogatott platformok, támogatott operációs rendszerek, támogatott protokollok)
3. Alkalmazás fejlesztési eszközök kiválasztása (minőségi elvárások figyelembe vételével, pl: konkrét ORB, konkrét RPC, programozási nyelv, IDE)
4. Menedzsment eszközök kiválasztása (adminisztráció, monitorozás)
5. Pilot alkalmazás integrációja (tesztek kiértékelése, esetleg iteráció a 2. lépéstől)

VIII.8.3 Middleware típusok összehasonlítása

	RPC	MOM	ORB	DB	TPM	komponens app. server	Message Broker
Topológia	pont-pont	pont-pont (bonyolult)	több-több	egy-több	több-több	több-több	több-több
Integrálható alk. száma	2	sok	kevés	sok	inf.	több	több
Régi elemek felhasználhatók?	új sw	régi integrálható	új sw	meglévő	meglévő	új sw	meglévő
Kód módosítás	sok	kevés	sok	(nincs) kevés	sok	sok	minimális
Implementáció	rövid idő	adapter van? rövid idő	hosszú	konfig van? gyors	hosszú	hosszú	minimális
Felh. száma	kevés	nagy	korlátozott		nagyon sok	nagy	nagy
Elosztottság	i	heterogén	i	i	centralizált architektúra		
Hálózati komm.	LAN	legkisebb	nagy igény		kicsi	nagy	
Fejlesztő eszk.	n	n	i	n	n	i	n
Vált. kezelés	n	rugalmas	komoly dolog	driver csere	n	sok kódolás	legrugalmasabb
Szabvány	i	JMS	COM(+) .NET Remoting EJB, CORBA	JDBC ODBC ADO	n	J2EE	?
termékek	OS	MQseries MsMQ JMS	COM, .NET, CORBA	tetszőleges DBMS	CICS Tuxedo	.NET server AppCenter J2EE	

IX XML

Az XML jelentése eXtensible Markup Language magyarul bővíthető jelölő nyelv. Ez egy leíró nyelv, melynek célja elsősorban adatok, információk, és ezek struktúráinak leírása. Az SGML (Standard Generalized Markup Language) és a HTML tapasztalataira alapozva hozta létre a webes szabványok kidolgozásáért felelős szervezet a W3C (World Wide Web Consortium).

Az SGML szintén egy általános és szabványos leíró nyelv. Elterjedését erőteljesen gátolta, rendkívüli komplexitása, emiatt ugyanis nagyon nehéz implementálni az SGML-t értő rendszereket. Az SGML tapasztalataira és irányelveire építve hozták létre a HTML-t amely egy korlátozott céllal és funkcionalitással bíró jelölőnyelv az SGML-énál jelentősen egyszerűbb felépítéssel. Ez az egyszerűség tette lehetővé a HTML világméretű elterjedését.

Az XML a HTML-éhoz hasonlóan egyszerű felépítésű, azonban annál általánosabb funkciójú, rugalmasabb, mégis ugyanakkor szintaktikai szempontból szigorúbb is, ami leegyszerűsíti az ipari méretű fejlesztést és hibakezelést.

Az XML kidolgozásakor a fejlesztés szempontjait tíz pontban foglalták össze:

1. Az Interneten jól használhatónak kell lennie
2. Az alkalmazások széles körét kell támogatnia
3. SGML kompatibilisnek kell lennie
4. Az XML-t értő programok fejlesztésének a lehető legegyszerűbbnek kell lennie.
5. Az opcionális lehetőségek nem megengedettek a nyelvben.
6. A dokumentumoknak egyszerűeknek és könnyen érthetőeknek kell lenniük
7. A tervezésnek gyorsan megvalósíthatónak kell lennie.
8. Formálisnak és konzekvensnek kell lennie.
9. Új dokumentum létrehozásának egyszerűnek kell lennie
10. A tömörségnek kicsi a jelentősége

IX.1 Felépítés

Az XML-ben (a HTML-lel ellentétben) nincsenek meghatározva az egyes elemek (Tag) nevei. Ez lehetővé teszi, hogy mindenki a saját maga számára legmegfelelőbb, leginkább beszédes, vagy éppen a saját meglévő rendszereihez leginkább illeszkedő elnevezéseket válasszon.

Példa:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<megjegyzés>
<címzett>Tamás</címzett>
<feladó>Jani</feladó>
<fejléc>Emlékeztető</fejléc>
<törzs>Ne felejtsd el elzárni a vizet! </törzs>
</megjegyzés>
```

Minden XML dokumentumnak egy vezérlési utasítással kell kezdődnie, amelyben legalább azt meg kell adni, hogy a dokumentum melyik XML verzió szerint készült.

Vezérlési utasítás:

<?xml version="1.0" encoding="ISO-8859-2"?>

Egy dokumentumban található adatok egy fa struktúrát alkotnak. Minden dokumentum egy gyökér elemből indul, a fenti példában ez a <megjegyzés> elem. Egy XML dokumentumnak csak egy gyökere lehet, de minden dokumentumnak kötelezően rendelkeznie kell egy gyökér elemmel.

Minden elemhez kötelezően tartozik egy bezáró pár. (</megjegyzés>). Ha egy elem esetében nincs értelme a bezáró párnak, azt az elem vége előtti "/" jellel kell jelezni, például: <BR/>. Egy XML dokumentumban a nyitó és záró elemre a hagyományos zárójelezés szabályai érvényesek, a HTML-ben megengedett ... szerkezet nem érvényes.

Az XML dokumentum bővíthetősége azt jelenti, hogy új elemet és ezáltal új adatot adhatunk az adathalmazhoz anélkül, hogy a régi alkalmazás működése ettől megváltozna (elromlana).

IX.2 Karakterek, elnevezések

Az XML esetén az egyes elemek neveiben illetve az adatokban tetszőleges karakterek felhasználhatók, azzal a kitétellel, hogy az elnevezéseknek és a karakterek felhasználásának meg kell felelniük bizonyos szabályoknak.

- A szabvány szerint az XML megkülönbözteti a kis és nagy betűket (case sensitive). Léteznek megvalósítások, amelyek ebben a tekintetben nem követik a szabványt, de a legtöbb (és a szabványos) implementációk ilyenek.
- A HTML-lel ellentétben az XML értelmezők nem veszik ki az adatokból a többszörös kihagyás jellegű (white space) karaktereket.
- Elemek nevei tartalmazhatnak bármilyen karaktert, a következő szabályok figyelembevételével:
  - Elem név nem tartalmazhat kihagyást (space).
  - Elem név nem kezdődhet számmal valamint aláhúzás karakterrel.
  - Elem név nem kezdődhet az XML karakter sorozattal
  - Elem névben nem ajánlott a mínusz (-) és a pont (.) karakter valamint a szintén nem ajánlott a kettőspont (:) mert speciális jelentése van.

Nincs fenntartott szó, a fentieknek megfelelő bármilyen karakter sorozat szerepelhet egy elem nevében.

Adatok neveiben szintén tetszőleges karakterek lehetségesek azzal a megkötéssel, hogy az elemzett adatokban (parsed data) a kisebb a nagyobb valamint az idéző jeleket kerülni kell – ezek karakter kódokkal vagy speciális egyedek (< > & stb.) felhasználásával adhatók meg.

IX.3 Névterek (Namespaces)

A névterek használata lehetővé teszi a név konfliktusok elkerülését két azonos elnevezésű és különböző célú elem használata esetén. Név konfliktusok számos okból fordulhatnak elő elsősorban azért mert az XML nem tesz megkötést az elemek neveire vonatkozóan, ezért könnyen előfordulhat, hogy amikor két különböző forrásból származó XML dokumentumból hoz létre valaki egy új XML dokumentumot, akkor az új dokumentumban név konfliktus(ok) adódnak.

IX.3.1 Név konfliktus

```
<Dokumentum>
        <méret>4 adat</méret>
        <típus>XML</típus>
        <doc>
        <ruha>
        <típus>póló</típus>
                <szín>sárga</szín>
                <méret>XXL</méret>
                <forma>hagyományos</forma>
        </ruha>
        </doc>
</Dokumentum>
```

Ebben a példában egy dokumentum keret van, amiben elvileg tetszőleges dokumentum lehet beágyazva. Itt egy XML típusú, ruháról szóló 4 adatot tartalmazó dokumentumot tartalmaz, amely azonban szintén rendelkezik <típus> és <méret> elemekkel, ez pedig név konfliktust okoz.

IX.3.2 Megoldás név előtéttel

```
<doc:Dokumentum>
        <doc:méret>4 adat</doc:méret>
        <doc:típus>XML</doc:típus>
        <doc:doc>
        <adat:ruha>
        <adat:típus>póló</adat:típus>
                <adat:szín>sárga</adat:szín>
                <adat:méret>XXL</adat:méret>
                <adat:forma>hagyományos</adat:forma>
        </ adat:ruha>
        </doc:doc>
</doc:Dokumentum>
```

A dokumentum különböző részeit különböző előtéttel egészítettük ki, így biztosítottuk az egyediséget.

IX.3.3 Megoldás névtér használatával

Névtér használata annyiban különbözik az előtét esetén megismerttől, hogy az előtét első előfordulásakor definiálni kell, hogy az egy névtér, és hozzá kell rendelni egy egyedi névtér nevet.

```
<doc:Dokumentum xmlns:doc="http://www.aut.bme.hu/docns.html">
        <doc:méret>4 adat</doc:méret>
        <doc:típus>XML</doc:típus>
        <doc:doc>
        <adat:ruha xmlns:adat="http://www.aut.bme.hu/ruhans.html">
        <adat:típus>póló</adat:típus>
                <adat:szín>sárga</adat:szín>
                <adat:méret>XXL</adat:méret>
                <adat:forma>hagyományos</adat:forma>
        </ adat:ruha>
        </doc:doc>
</doc:Dokumentum>
```

Ebben a példában definiáltunk két névteret. A két névtér nevének egy-egy URL-t adtunk meg. Az XML értelmező ezt az URL-t nem használja semmilyen célra mindössze egy egyedi névként tekinti (tehát ez lehetne egy tetszőleges szó is), bár sok esetben az XML dokumentumok készítői az URL által hivatkozott oldalon helyeznek el információt az adott névteret illetően.

IX.4 Tulajdonságok

Akárcsak a HTML-ben az XML-ben is lehet az egyes elemeknek tulajdonsága, azonban az XML-ben a attribútumok értékeinek mindig kötelezően idézőjelek között kell lenniük. Példa:

```
<megjegyzés dátum="99/12/24">
```

Az idézőjel lehet egyszeres és kettős idézőjel is.

A tulajdonságok hasonló szerepet töltenek be mint az adott elem gyermek elemei, a gyermek elemekkel megegyezően valamilyen információt vagy adatot hordoznak.

Példa ugyanannak az információnak két féle kezelésére:

```
1)
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>

2)
<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

Az iménti két példából mindkettő szabványos, mindkettő ugyanazt jelentheti de inkább az utóbbit ajánlatos használni.

Nincs előírás, hogy mikor kell egy adott információ vagy adat tárolásához gyermek elemet és mikor tulajdonságot használni, azonban a kettő kezelése közötti különbség rá világít a gyermek elemek és a tulajdonságok használatának lehetséges módozataira és irányelveire.

- A tulajdonságok nem tartalmazhatnak egyszerre több adatot (a gyermek elemek igen).
- A tulajdonságok nem bővíthetők olyan egyszerűen mint a gyermek elemek.
- A tulajdonságok adatai nem rendezhetők struktúrákba (a gyermek elemek adatai igen)
- A tulajdonságokat nehezebb kezelni a programokban.
- A tulajdonságok helyességet nehezebb ellenőrizni

Mindezek alapján tulajdonságokat akkor érdemes használni, ha azok nem az adatok, hanem az azokat feldolgozó alkalmazások szempontjából jelentenek plusz információt. Nincs előírás, hogy mikor kell egy információt adatként vagy attribútumként megadni, de ami adatként felhasználandó, azt érdemes adatként megadni.

Példa tulajdonság használatára:

```
<image type="gif" name="picture.gif"/>
```

Ebben a példában a felhasználó számára nem lényeges adat sem a kép neve, sem a típusa, - csak maga a kép - azonban a kép megjelenítéséhez a megjelenítést végző alkalmazásnak szüksége van ezekre az információkra.

Egy másik jellemző alkalmazás amikor tulajdonságokat érdemes használni, ha egy adat rekord halmazban az egyes rekordokat egyedi azonosítóval kell ellátni. Ekkor az azonosító a felhasználó felé ismét nem lényeges adat, kizárólag az adatokat feldolgozó alkalmazás használja.

IX.5 DTD (Document Type Definition)

Egy XML dokumentum tetszőleges elemeket és tulajdonságokat tartalmazhat, valamint tetszőleges felépítéssel rendelkezhet. Mindezek a paraméterek meghatározzák a dokumentum típusát, amelyet leírva a dokumentum típus definícióját, vagy sémáját kapjuk.

Egy dokumentum típus definíciója számos módon felhasználható. Felhasználható több, különböző forrásból származó XML dokumentum „kompatibilitásának" a vizsgálatára (hogy vajon a dokumentumok azonos típusúak, így ugyanabban az alkalmazásban egyaránt felhasználhatóak-e). Felhasználható annak meghatározására, hogy egy készítendő dokumentumnak milyen típussal kell rendelkeznie, milyen konvenciókat kell betartania, hogy egy adott alkalmazásban felhasználható legyen, valamint a típus definíció alapján vizsgálható egy dokumentum érvényessége.

A DTD egy fajta szabványos típus definíció. Az SGML nyelvhez készült, de az XML esetén is kiválóan használható, ma az egyik legelterjedtebb típus definíciós módszer.

Példa:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

IX.5.1 Elemek

A dokumentum típusának definíció megadásakor a definíció nevének a gyökér elem nevével kell megegyeznie. Ezután következik az egyes elemek megadása. Először a gyökér elemet kell megadni, majd a gyermek elemek definíciójára kerül sor.

A "note" elem definíciójakor meghatározásra kerül, hogy négy gyermek elemmel rendelkezik, ez hordozza a struktúra információit. Ugyanitt adható meg az is, ha valamely gyermek elemből több is előfordulhat a dokumentumban.

<!ELEMENT note (to+,from,heading,body)>

A plusz jel a to elemnév után azt jelezheti, hogy több címzett is lehetséges a dokumentumban.

Az egyes elemek tartalmazhatnak további gyermek elemeket, elemzett (#PCDATA – Parsed Character Data), vagy nem elemzett (#CDATA - Character Data) adatokat. A nem elemzett adatok adatrészét az

XML elemző nem vizsgálja, nem értelmezi, ezért ott tetszőleges karakterek – például a kisebb vagy a nagyobb jel - korlátozás nélkül használható.

IX.5.2 Tulajdonságok

A DTD a tulajdonságok leírását is lehetővé teszi a következő módon.

<!ATTLIST elem-név tulajdonság-név tulajdonság-típus alapértelmezett-érték>

Példa:

<!ATTLIST megjegyzés dátum #CDATA "99/12/24">

Az elem-név annak az elemnek a neve amelyhez a tulajdonság tartozik. A tulajdonság-név és az alapértelmezett-érték egyértelmű, a tulajdonság típus pedig a következők valamelyike lehet:

Value	**Explanation**
CDATA	Nem ellenőrzött karakter adat
(en1|en2|..)	Az érték egy számozott lista valamely eleme lehet
ID	Az érték egy egyedi azonosító
IDREF	Az érték egy másik elem egyedi azonosítója
IDREFS	Más elemek egyedi azonosítóinak listája
NMTOKEN	Az érték egy érvényes XML név
NMTOKENS	Az érték érvényes XML nevek listája
ENTITY	Az érték egy egyed
ENTITIES	Az érték egyedek listája
NOTATION	Az érték egy megjegyzés neve
xml:	Az érték előre definiált xml: érték

IX.5.3 Egyedek

A DTD-ben lehetőség van egyedek megadására. Az egyedek kezelése nagyon hasonlít a C nyelv #include illetve #define lehetőségeire, lehetővé teszi külső erőforrásban levő, vagy előre definiált adatok (szövegek, XML dokumentum részletek, stb.) újra felhasználását egy dokumentumban. Az egyedeket az &egyed-név; formában lehet megadni a dokumentumban, ahol az egyed helyett a hozzá rendelt karakter vagy szöveg meg.

Az egyedeket három csoportra oszthatjuk.

1. Előre definiált egyedek.

Egyed	Egyed név
Kisebb jel (<)	Lt
Nagyobb jel (>)	Gt
Angol és – AND – jel (&)	Amp
Egyszeres idézőjel (')	Apos
Dupla idézőjel (")	Quot

2. Normál egyedek

Megadása: <!ENTITY egyednév "hozzá rendelt szöveg">

<!ENTITY IBM "International Business Machines">

Ekkor ahol a dokumentumban az &IBM; karaktersorozat található, ott az International Business Machines szöveg jelenik meg.

3. Paraméter egyedek

<!ENTITY % paraméter-egyed-név "hozzá rendelt szöveg">

A paraméter egyedek abban különböznek a normál egyedektől, hogy csak a DTD-ben használhatók (a DTD felépítésének megkönnyítésére), az XML dokumentumban nem. Hivatkozni rá a %paraméter-egyed-név; utasítással lehet.

A külső egyedek a belső egyedektől annyiban különböznek, hogy a deklaráció tartalmazza a SYSTEM kifejezést, és a hozzá rendelt szöveg helyett egy erőforrás azonosító (URI) mutat a bemásolandó adatra.

<!ENTITY ruha SYSTEM "http://www.aut.bme.hu/ruha.xml">

Lehetőség van a DTD külső (az XML fájlon kívüli) megadására is, ekkor a DOCTYPE elemben jelezni kell, hogy a leírás egy külső fájlban található:

```
<!DOCTYPE note SYSTEM "note.dtd">
```

Ahol a note.dtd fájl tartalmazza a definíciót:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

IX.6 Megjelenítés

Az XML adatok, információk vagy struktúrák leírására szolgáló leíró nyelv. Önmagában nem foglalkozik az adatok megjelenítésének problémájával. Erre számos más módszer létezik. Kezelhető egy XML dokumentum HTML forrásban, különböző stíluslapok segítségével, illetve tetszőleges, internettől, böngészőtől független alkalmazás fejleszthető a dokumentumok kezelésére és megjelenítésére.

IX.6.1 HTML

Egy HTML dokumentumban XML állományt az <XML> Tag segítségével lehet felhasználni.

Példa XML forrás:

```
<?xml version="1.0"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
</CATALOG>
```

Példa HTML állomány:

```
<html><body>
<xml src="cd_catalog.xml"
id="cdcat">
</xml>

<table datasrc="#cdcat" width="100%" border="1">
<thead>
<th>Title</th><th>Artist</th><th>Year</th>
</thead>
<tr align="left">
<td><span datafld="TITLE"></span></td>
<td><span datafld="ARTIST"></span></td>
<td><span datafld="YEAR"></span></td>
</tr></table>
</body></html>
```

Az XML elem fontos paramétere az SRC, amely a forrás XML állományt adja meg. Szintén elhagyhatatlan az ID paraméter, ez alapján lehet ugyanis az állományban az XML adatokra hivatkozni. Ezután a táblázatban megadható az adatforrás azonosítója (datasrc paraméter), majd az egyes adatokra az XML elemnév segítségével lehet hivatkozni.

IX.6.2 CSS (Cascading Style Sheets)

Stílus lapok segítségével is meg lehet jeleníteni XML állományok tartalmát. Ekkor a stíluslapban az egyes XML elemekhez kell stílusokat rendelni. Az előző pont példa XML állományát figyelembe véve egy lehetséges stíluslap lehet a következő:

```
CD
{
color: green;
border: '1pt solid black'
page-break-after:always;
}

TITLE
{
color: red;
background-color:#EFE7D6;
border: '1pt solid black'
}

ARTIST
{
color: #FFFFFF;
background-color:#000000;
}
```

Természetesen ahhoz, hogy az XML állomány a megjelenéshez használjon egy stíluslapot, egy vezérlési utasítást kell beszúrni az adatok elé:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

IX.6.3 XSL eXtensible Stylesheet Language

A stíluslapok használata számos esetben elegendő lehet a megfelelő megjelenés létrehozásához, ugyanakkor problémát jelent, hogy nem nyújt lehetőséget az elemek tetszőleges pozícionálására, ezért kidolgoztak egy kifejezetten az XML igényeihez igazodó stílus leíró nyelvet, amely képes egy XML állomány tetszőleges formában történő megjelenítésére, ez az XSL.

Az XSL egy XML alapú sablont jelent, ami az XML dokumentum megjelenítésére felhasználható. Lehetőséget ad a dokumentum adatok transzformációjára – ami jelentheti bizonyos elemek elhagyását épp úgy mint elemek sorrendjének megváltoztatását, stb. – az adatok tetszőleges pozicionálására – erre a HTML nyelv lehetőségeit használja fel – illetve a stílus lapokhoz hasonló stílus definíciókra.

Példa XSL-re a CD katalógus XML állományának figyelembevételével.

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="CATALOG/CD"
       order-by="+ ARTIST">
      <tr>
        <td><xsl:value-of select="TITLE"/></td>
        <td><xsl:value-of select="ARTIST"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Látható, hogy az XSL is egy XML dokumentum, amely elemeit a konfliktusok elkerülésére az xsl: névtér szerint határozza meg. A <template> elem határozza meg a sablon kezdetét – a bezáró pár pedig a végét –, a match paraméter pedig az XML állomány belépési pontját, ez a jelen példában az állomány gyökere.

A <for-each> elem egy ciklust határoz meg, az elem által meghatározott (a nyitó és záró pár közötti) részt annyiszor hajtja végre, ahányszor a select paraméter által mutatott elem – és gyermek elemei – az XML állományban szerepelnek (azaz itt minden rekordon végig megy). Az order-by a sorrendezést adja meg (feltéve, hogy szükséges sorrendezés), az adatot ami szerint a sorrendezésnek történnie kell és a sorrend irányát.

A <value-of> pár nélküli elem választja ki a rekord megjelenítendő elemét.

Ahhoz, hogy az adott sablont a dokumentum használja, a következő vezérlési utasítással kell azt kiegészíteni:

```
<?xml-stylesheet type="text/xsl" href="cd_catalog.xsl"?>
```

IX.7 Programozás

Az XML-t kezelő alkalmazásokat fejlesztők dolgát megkönnyíti, hogy rögtön kétfajta értelmező is rendelkezésre áll XML dokumentumok programból történő kezeléséhet, értelmezéséhez. Mindkettő elrejti a programozó elől az XML szintaktikáját, így a fejlesztőnek csak a tartalom helyes kezeléséről kell gondoskodnia.

IX.7.1 SAX

A SAX (Simple API for XML) egy eseményvezérelt értelmező. A dokumentumot folyamatosan olvasva a felbukkanó elemeket és adatokat egy-egy metódushívássá konvertálja, a konkrét értelmezést pedig rábízza az alkalmazás készítőjére, akinek feladata az, hogy az értelmező osztályt kiterjesztve megírja az egyes metódusok törzsét, és saját céljai szerint értelmezze az adatokat.

IX.7.2 XML DOM

A DOM (Document Object Model) értelmező az XML dokumentumot egy az egyben beolvassa, és egy objektumokból épített fát állít elő belőle, amely leképezi a dokumentum struktúráját. Ezek után tetszés szerinti elem bármely adata elérhető. Az XML DOM számos programozási nyelvben használható a C++-tól a JavaScriptig ezért kiválóan alkalmazható például kliens oldali web alkalmazások programozására.

Példa XML DOM használatára Javascriptben:

```
<html>
<body>
<script type="text/javascript">
// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("cd catalog.xml")

// Load the XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load("cd catalog.xsl")

// Transform
document.write(xml.transformNode(xsl))
</script>

</body>
</html>
```

A program működése egy XMLDOM objekjum létrehozásával kezdődik, amihez hozzá kell rendelni a kívánt XML állományt. Ezután tetszőleges módon használható az állomány. A jelent példa létrehoz egy újabb XML objektumot, amihez egy XSL sablont rendel, majd megjeleníti az XML állományt az XSL sablon segítségével. Ez a program egy lehetőség tetszőleges adat és sablon állományok összerendelésére.

IX.8 Egyéb alkalmazások

Az XML azon túl, hogy adat leíró nyelv, és lehetővé teszi adatok szabványos kommunikációját, számos alkalmazás leírónyelve is egyben. Ilyen alkalmazás például a már megismert XSL, ami egy megjelenés leíró sablont definiáló leíró nyelv, amely azonban egyszersmind XML alkalmazás.

IX.8.1 XHTML

Az XHTML újabb jó példa az XML alkalmazásokra. Olyan XML alkalmazás amely elem neveit és célját illetően teljesen megegyezik a HTML 4.01 szabványban rögzített HTML nevekkel és képességekkel, ugyanakkor vonatkoznak rá az XML alkalmazások szigorú kritériumai.

IX.8.2 XML Schema

Az XML Schema ismét egy XML alkalmazási nyelv amelynek a célja megegyezik a DTD céljával, tehát típus definíciók leírására szolgál. Kifejezetten az XML-hez készült, a DTD-nél egyszerűbb, ráadásul kínálja azt az előnyt, hogy XML szintaktikájú, tehát használatához (a DTD-vel ellentétben) nincs szükség új szintaxis megtanulására. Ma az XML alkalmazások típus definícióihoz a szabvány már az XML Schema-t ajánlja.

IX.8.3 EbXML

Az ebXML egy szabványos, az egymással kompatibilis, egymással változtatás nélkül kommunikálni akaró eBusiness alkalmazásokhoz kifejlesztett XML alkalmazás nyelv.

X SOAP

A SOAP (Simple Object Access Protocol) egy olyan specifikáció, mely lehetőséget nyújt néhány egyre fontosabbá váló igény kielégítésére, mint a Web szolgáltatások létrehozása, illetve a komponens rendszerek együttműködésének megoldása.

X.1 Igény a Web szolgáltatások létrehozására

A jelen Web a HTML technológián alapul, ami lehetővé teszi weboldalak böngészését. A különböző linkek segítségével rengeteg információhoz való hozzájutásának lehetőségét is megkapjuk, és a HTML oldalak megtekintését bármilyen platform alatt futó Web böngészővel megtehetjük.

A HTML oldalak kitűnőek az információ *megjelenítésére*, így azt emberek milliárdjai nézhetik meg. Az információ kinyerése azonban csak az emberek számára egyszerű egy képként megjelenő, helyesen formázott weboldalból.

Ha szeretnénk például nyaralni menni, akkor megtehetjük, hogy különböző lehetőségek után nézünk a Web-en, sorra nézzük az egyes helyeket, és osztályozzuk különböző szempontok alapján: tengerparton van-e az adott hely, elég meleg van-e, gyerekeinek van-e lehetősége szórakozásra, stb. Rengeteg hely létezik, ahova elutazhatunk, és képtelenség mindet végignézni, vagy akár csak egy töredékét is a lehetőségeknek – így a döntés végül csekély információ alapján történik.

Miért ne bízhatnánk a nyaralási lehetőségeink felkutatását inkább egy számítógépes programra? Jóval több esélye lenne ideális helyet találnia, mint nekünk, hiszen ugyanannyi idő alatt jóval több lehetőséget tud kiértékelni. Az egyetlen probléma, hogy ma az információ HTML lapok formájában található meg, amiből egy számítógépes program nehezen tudja azt kinyerni. Egy HTML nyelvű leírásból megtudhatjuk például, hogy a „26 C” milyen színnel, milyen fonttal, hol, hogyan jelenjen meg, de hogy az egy adott üdülőhely nyári átlaghőmérsékletét jelenti, az már nehezebben deríthető ki – egy szemgolyóval nem rendelkező program számára.

A legkülönbözőbb területeken felmerült hasonló, információ-összegyűjtő igény; igény a Web szolgáltatásokra. Nézzük meg, milyen tulajdonságokkal kell rendelkeznie egy Webes szolgáltatásnak, hogy a fentihez hasonló problémára megoldást nyújtson.

Egyrészt szeretnénk, ha a Web szolgáltatás hasonló technikákra épülne, mint a mostani, így az eddigi módszerekkel, illetve azok kibővítésével elérhetővé válik. Egy teljesen új alapokon nyugvó technika nehezen tudna betörni a Web mellett.

A Webes szolgáltatáshoz szükséges, hogy egy kérdésünkre kapott válaszból az adatok egy program számára is könnyen kinyerhetők legyenek, nem úgy, mint a HTML esetében. Olyan protokollt kell alkalmaznunk, amely az adatok leírásával, és nem ábrázolásával foglalkozik.

Nem következik azonban ebből, hogy bináris adatleírásra célszerű lenne áttérni a szöveges HTML-ről. A szöveges adatábrázolás rengeteg előnnyel rendelkezhet esetünkben, és tudjuk, létezik számítógép által is könnyen értelmezhető szöveges adatleírás – ilyen például az XML.

Fontos, hogy egy Web szolgáltatásnál az interfész, amin keresztül a szolgáltatást elérjük, bármikor lekérdezhető legyen, hiszen lazán csatolt rendszerhez szeretnénk jutni. Ha egy előre rögzített interfészhez kellene ragaszkodni, az esetünkben lehetetlenné tenné a szolgáltatás változtatását, bővítését.

Szintén figyelnünk kell rá, hogy bármilyen platformról, bármilyen rendszerből elérhető legyen a Web szolgáltatás, nem úgy mint pl. a COM vagy a CORRBA esetében.

Web szolgáltatások létrehozását segíti a SOAP is: az adatok leírása XML nyelven történik, mely szöveges, könnyen változtatható struktúrájú, számítógép és emberi feldolgozásra is használható adatleírást eredményez. Mivel a szolgáltatások használatára az igényt egy XML dokumentum formájában jelentjük be, a szolgáltatás elérése platformfüggetlen lesz, szemben az egyedi bináris protokollokkal. Az XML nyelven megfogalmazott szolgáltatás hívása könnyen szállítható a hálózaton a HTTP protokoll segítségével, így az eddig elterjedt módszerekkel is használható megoldást kapunk, ráadásul a hívásunk tűzfalon keresztül is működőképes lesz. A szolgáltatások interfészeinek leírása szintén történhet XML nyelven, valamint a szolgáltatásokat igénybevevő kliens programoknál megoldhatjuk, hogy a hívás előtt kérje le ezt az interfészt, és tartsa be az abba foglalt szabályokat.

Összefoglalva tehát a SOAP segítségével készíthetünk olyan Web szolgáltatásokat, amely megfelel követelményeinknek.

X.2 Együttműködési problémák komponens rendszereknél

A SOAP szintén sikeres lehet, ha különböző komponens rendszerek együttműködése a cél, vagy csak platformfüggetlen elérést szeretnénk biztosítani komponens alapú rendszerünkhöz.

Fontos áttekinteni, hogy milyen problémák merülnek fel, amikor azt szeretnénk biztosítani, hogy egy kliens el tudjon érni egy távoli komponenst a komponens rendszerétől függetlenül.

X.2.1 Különböző adatreprezentáció

Másik számítógépen lévő eljárások hívására régóta igény volt, ezért szabványosítottak különböző RPC (Remote Procedure Call) protokollokat. Az RPC protokollok fontos része, hogy hogyan ábrázolják az adatot. Az adatokat platformtól, nyelvtől független módon kell ábrázolni, hiszen bármilyen programot futtató bármilyen számítógépnek képesnek kell lenni egy adott távoli eljárás meghívására.

Az objektum-orientált rendszerek támogatásához később kialakultak ORPC (Object RPC) protokollok, amik lehetővé tettek távoli objektumok elérését, metódusuk hívását. Az ilyen ORPC protokollok általában RPC-t használnak megvalósításukkor, kiegészítve azt valamilyen módon egy cél objektum azonosításához szükséges adatokkal.

A két legelterjedtebb ORPC protokoll a DCOM, illetve az IIOP (Internet Inter-ORB Protocol), melyet nem meglepően a két nagy komponens rendszer, a COM és a CORBA használ távoli komponensek elérésekor.

Vizsgálva a két protokollt, két fontos különbséget vehetünk észre. Az egyik, hogy mivel minden CORBA objektum egy interfészt támogat, ezért a hívásban az csak implicit módon szerepel, az objektum azonosító így az interfész azonosítását is magában foglalja. COM-nál nem elég azt az információt átvinni, hogy melyik objektumot szeretném elérni, szükség van egy interfész azonosítására is.

A másik, talán az előzőnél is fontosabb eltérés, hogy az adatok ábrázolásához két különböző formátumot használnak. A DCOM a paraméterek átadásához az NDR (Network Data Representation) formátumot használja, míg az IIOP a CDR (Common Data Representation) formátumot. A két formátum nem teljesen egyforma.

X.2.2 Nehézkes kliens-szerver kommunikáció

A COM és a CORBA vizsgálata közben rájöhetünk a következő dologra. Mindkét rendszer jól használható, egyiket sem lehet, és nem is érdemes figyelmen kívül hagyni. Viszont mindkét rendszer hiányosságának felróható, hogy bár két szerver nagyon jól tud kommunikálni egymással, a kliens-szerver kommunikáció nehézkes.

Mind a COM, mind a CORBA olyan megoldást biztosít, ami a kliens rendszerére valamilyen megkötést ad. Ez a COM esetében azt jelenti, hogy (gyakorlatilag) csak Windows operációs rendszert futtató kliens képes elérni egy COM komponenst. CORBA esetében sem jobb a helyzet. Először is szükségünk van egy ORB implementációra, másodszor is, bár az IIOP-t minden ORB képes megérteni, bizonyos magasabb szintű szolgáltatások csak akkor igénybevehetők, ha ugyanazt az ORB terméket használja a kliens.

Mind a COM, mind a CORBA biztosít számunkra egy magas szintű környezetet, ahol egy távoli objektum elérése viszonylag könnyen megtehető. Ha viszont egy kliens ezek nélkül a rendszerek nélkül (COM alrendszer illetve ORB) szeretne elérni egy távoli objektumot, a bonyolult protokollok miatt szinte reménytelen a helyzete.

Egy másik probléma, amely mindkét rendszert érinti, hogy olyan helyzetekben, amikor a kliens az Interneten helyezkedik el, nehézségek lépnek fel. Gondoljunk csak arra az esetre, amikor a klienst egy proxy szerver, vagy egy tűzfal választja el a rendszertől. Bár a COM-ra és a különböző ORB termékekre léteznek egyedi megoldások, ezek nem képesek együttműködésre, és további hátrányuk, hogy konfigurációs hibákra érzékenyek.

X.3 A megoldás

A fentebb említett problémák nem jelentkeznek persze, ha a COM-ot, vagy a CORBA-t szerver számítógépek között használjuk. Lehetséges tehát egy olyan megoldás, ahol a középső rétegben az alkalmazás fejlesztőjének tervezési igénye szerint vagy COM, vagy CORBA rendszert használunk, míg biztosítjuk a komponensek elérését egy más féle módszerrel, ami egységes adatreprezentációt használ, és biztosítja az Interneten lévő klienseknek a könnyű elérést, illetve nem tesz szinte semmilyen megkötést a kliens rendszerére.

X.3.1 Az XML mint adatreprezentációs protokoll

Az XML az NDR-hez és a CDR-hez hasonlóan platformtól független adatleírást tesz lehetővé.

Az XML szöveges leírást biztosít, így szinte bármilyen nyelvből, bármilyen rendszerből, minden többlet szoftver igénye nélkül tudunk kezelni XML fájlokat. Nem beszélve arról, hogy szinte minden nyelvhez/rendszerhez találhatunk olyan segédeszközt, ami megkönnyíti az XML-fájl létrehozását illetve olvasását.

Az XML további előnye, hogy könnyen, tetszőlegesen bővíthető, ez a bővíthetőség kezelhető is a namespace-eknek köszönhetően. Az XML támogat adattípusokat, az alap típusokon kívül pedig létrehozhatunk saját adattípusokat is. (Az XML Schema specifikációnak köszönhetően.)

X.3.2 A HTTP mint adattovábbítási protokoll

A HTTP protokoll alkalmas távoli eljáráshívások megfogalmazására.

Nagy előnye, hogy elterjedt. A HTTP hívásokat általában Web szerverek szolgálják ki, de elképzelhető egyedi HTTP kiszolgáló szerver program írása is.

A HTTP további előnye az RPC-vel szemben, hogy szöveg alapú protokoll. Bárki megfogalmazhat távoli eljáráshívást akkor is, ha szinte semmilyen segédeszköz nem áll a rendelkezésére. (Például egy telnet programmal "kézzel" begépelve is.) Egy példa egy HTTP fejlécre:

```
POST /RPC2 HTTP/1.1
Host: coloman.myksys.hu
Content-Type: text/xml
Content-length: 181
[új sor]
[szöveges tartalom]
```

Mint láthatjuk, a HTTP header a következő információkat tartalmazza: a HTTP kérés metódusa (GET/POST), a kért objektum (például metódusnév vagy RPC kiszolgáló objektum neve), a protokoll verziószáma, a tartalom típusa (például text/plain, text/html, text/xml), illetve a tartalom hossza.

A fejléc után a megadott típussal megfogalmazható egy metódushíváshoz szükséges információk (például metódusnév, paraméterek típusa, értékei). A hívás feldolgozása után a HTTP szerver választ küld, mely a hívás visszatérési értékét tartalmazhatja, vagy valamilyen hibainformációt hordozhat.

X.3.3 A megoldások megvalósítása

A most felvázolt megoldást két általánosan használható specifikáció is támogatja. Az egyik a kicsit egyszerűbb, és ezért korlátozottabb lehetőségekkel bíró XML-RPC, a másik a jóval összetettebb, és ezért kifinomultabb lehetőségeket is magába rejtő SOAP.

X.4 SOAP

A SOAP XML-t használ adatreprezentációs protokollként. Hívások továbbításához leggyakrabban a HTTP-t használják, de lehetséges más protokollok használata is. Az XML-RPC előnyei mellett azonban rendelkezik azzal a kedvező tulajdonsággal is, hogy kellő rugalmassága miatt alkalmas lehet a komponens rendszerek magasabb szintű szolgáltatásainak igénybevételére is.

X.4.1 Envelope

Egy SOAP hívás XML-dokumentuma a következőket kell, hogy tartalmazza. A gyökérelem az `Envelope`, melynek kötelezően meg kell jelennie. Ennek közvetlen leszármazottja egy `Body` elem, és esetleg egy `Header` elem. `Body` elemnek kötelező lennie, vagy közvetlenül az `Envelope` után, vagy ha a `Header` elem is jelen van, akkor az után.

Az `Envelope` namespace-e a "http://schemas.xmlsoap.org/soap/envelope/", a mostani specifikáció [21] szerint, ha az `Envelope`-nak ettől eltérő namespace attribútumot adunk meg, az adott hívás visszautasítandó.

Az `Envelope`-nak lehet még egy `encodingStyle` attribútuma is, amelynél az adatok tárolásával kapcsolatos szabályok rögzítésével foglalkozó sémát adhatjuk meg, ez rendszerint "http://schemas.xmlsoap.org/soap/encoding/", vagy nincs jelen.

X.4.2 Body

A `Body` elem használható egy eljárás hívás paramétereinek, vagy egy eljárás visszatérési értékének csomagolásához, illetve hiba visszajelzéséhez is. Mint már korábban említettük, vagy közvetlenül a `Header` elem után következik, vagy ha az nem létezik, akkor az `Envelope` után.

A `Body` elemben lévő bejegyzéseknek namespace-hez tartozónak kell lenni. A bejegyzésekhez szintén megadható az `encodingStyle` attribútum.

X.4.3 Header

A `Header` rész egy eljárás hívásának kibővítéséhez használható egyéb, előre nem meghatározott funkciókkal. Ilyen tipikus funkciók például a tranzakciókezelés, felhasználó azonosítás.

A `Header` elemben lévő bejegyzéseknek szintén namespace-hez tartozónak kell lenni. A `Header`-ben szereplő elemeknek speciális attribútum is adható.

Az egyik ilyen attribútum a `mustUnderstand`, melynek szerepeltetése 1 értékkel azt jelenti, hogy ha a szerver oldalon az adott fejlécelem nem ismert, nem értelmezhető, akkor a hívást kötelező visszautasítani. Elhagyása esetén a nem ismert elem egyszerűen figyelmen kívül lesz hagyva.

A másik, szintén fontos attribútum az `actor`. Egy SOAP hívás feldolgozása alatt keresztülhaladhat több objektumon is, míg eljut a végcélhoz. Az actor adja meg annak az objektumnak az URI-jét, ami az adott fejlécelemet feldolgozza. Ilyen lehet például a tranzakciókezelő, hogyha a header elem egy tranzakció azonosító. Az actor elhagyása esetén a feldolgozás a végcél feladata.

X.4.4 Adattípusok

Az alap adattípusok a W3C XML Schema specifikációjának második részében találhatók meg. Bizonyos adattípusokat a "http://schemas.xmlsoap.org/soap/encoding/" namespace alatt találunk meg.

Az XML sémákból adódóan a SOAP az adatok típusának és struktúrájának leírására olyan széles lehetőséget biztosít, hogy annak részletes ismertetése meghaladja ezen ismertető kereteit.

X.4.5 Hibakezelés

Ha a szervernek nem sikerült valami miatt végrehajtani egy hívást, hibával kell visszatérnie. Ilyenkor a Body egy `Fault` elemet tartalmaz, aminek a következő részei lehetnek:

- `faultcode`
  A hiba rövid kódja.
- `faultstring`
  A hiba szöveges leírása, mely az embereknek biztosítja az olvashatóságot.
- `faultactor`
  A hibát okozó objektum azonosítója.
- `detail`
  Alkalmazásfüggő hiba jelzésére használt.

X.4.6 HTTP fejléc

Ha HTTP-t használunk a SOAP hívások lebonyolítására, akkor a hasonlók mondhatók el, mint az XML-RPC esetében. Hozzá kell tenni, hogy a SOAP nemcsak a POST metódust támogatja, hanem elképzelhető más kérési módszer is. A SOAP támogatja a HTTP Extension Framework-öt is.

Kérésnél még megadhatunk egy `SOAPAction` mezőt is a HTTP fejlécben, ami a kért metódus nevét, vagy egyéb objektum-azonosítót tartalmazhat. Ezt például a tűzfalak használhatják bizonyos hívások kiszűrésére.

Hiba esetén, tehát amikor a `Body` elem egy `Fault` elemet tartalmaz a válasz fejlécében az "`500 Internal Server Error`" hibaüzenetnek kell megjelennie.

X.4.7 RPC és SOAP

Egy távoli eljáráshíváshoz szükséges információk a következő helyen jelennek meg egy HTTP-n keresztüli SOAP hívásban.

A szerver objektum azonosítója a HTTP fejlécben található meg. A metódushívás egy struktúra, melynek neve a metódus neve. Elemei a hívott metódus `[in]` és `[in,out]` paraméterei, melyeknek típusa megegyezik a hívott függvény paramétereinek típusával.

A válasz szintén egy struktúra, melynek neve (bár nem kötelezően, de ajánlottan) a hívott függvény neve összekapcsolva a “Response” sztringgel. Ez az elem tartalmazza az `[out]` illetve `[in,out]` paramétereket.

A hiba egy, az előzőekben megadott `Fault` struktúrában helyezendő el. Szintén megkötés van a HTTP visszaadott hibakódjára (lásd az előző részt).

A hívások kibővítését a `Header` elemmel írhatjuk le. Itt lehet például megtalálni egy tranzakció azonosítót, amit majd valószínűleg a szerver oldalon lévő tranzakciókezelő dolgoz fel.

X.4.8 Egy példa SOAP hívásra

Az előzőek tükrében nézzünk meg egy lehetséges példát egy SOAP hívásra illetve egy válaszra.

```
POST /Szamolo HTTP/1.1
Host: coloman.myksys.hu
Content-Type: text/xml;
Content-Length: nnnn
SOAPAction: "urn:uuid:DEADF00D-BEAD-BEAD-BEAD-BAABAABAABAA"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle= "http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ISzamolo:Osszead
xmlns: ISzamolo="urn:uuid: ABBABEBE-A2A2-5050-C000-AAABBBCCCDDD">
<a>5</a>
<b>-2</b>
</ISzamolo:Osszead>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Ha sikeres volt a hívás:
HTTP/1.1 200 OK
Content-Type: text/xml;
Content-Length: nnnn

<SOAP-ENV:Envelope xmlns:SOAP-ENV=   .... (lásd fentebb)
<SOAP-ENV:Body>
<ISzamolo:OsszeadResponse
xmlns: ISzamolo=   .... (lásd fentebb)
<c>3</c>
</ISzamolo:OsszeadResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Ha a hívás nem sikerült:
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml;
Content-Length: nnnn

<SOAP-ENV:Envelope xmlns:SOAP-ENV=   .... (lásd fentebb)
 <SOAP-ENV:Body>
  <SOAP-ENV:Fault>
    <faultcode>SOAP-ENV:Application Faulted</faultcode>
    <faultstring>Negativ numbers not allowed.</faultstring>
    <detail>
       <e:myfaultdetails xmlns:e="Valami-URI">
       <message>
             Call the ISzamolo:Substract method for
             substracting numbers.
       </message>
    </detail>
  </SOAP-ENV:Fault>
```

```
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XI Web szolgáltatások

Az Internet népszerűsödésével, hamar nyilvánvalóvá vált, hogy a web modell nagyszerűen alkalmas elosztott rendszerek közötti kommunikáció megvalósítására. Egy XML web szolgáltatás egy olyan önleíró kódrészlet, mely szabványos protokollok segítségével megkereshető, feltérképezhető, meghívható és az eredményt szintén szabványos formában adja vissza.

Az XML web szolgáltatások áttörő sikerüket és gyors népszerűsödésüket annak köszönhetik, hogy a többi elosztott komponens technológiával szemben (DCOM, CORBA, EJB) lazán csatoltak. A web kliensek és szerverek között a kommunikáció egyszerű, MIME-típusú adatok közlekednek a hálózaton, melyek szemantikája fejlécek segítségével módosítható. Egy üzenet célját indirekt módon adhatjuk meg egy ún. URL-lel (Universal Remote Location), melynek köszönhetően a rendszerekben könnyebben valósíthatjuk meg a terheléselosztást és munkamenet követést.

Az egyszerű kommunikációnak köszönhetően lehetőség van a rendszereket inkrementálisan építésére. A szorosan csatolt elosztott objektumalapú rendszerekkel szemben, ahol az összes komponenst egyszerre kell lefordítanunk, itt akkor adhatunk új klienseket és szervereket a hálózathoz, amikor csak szeretnénk. A web szolgáltatások alapötlete az, hogy a web programozásánál megszokott lazán csatolt modellt terjesszük nem böngésző-alapú alkalmazásokra. A céljuk pedig megvalósítani egy olyan platformot, melyen lehetőség van különböző operációs rendszereken, eltérő programnyelveken egymástól függetlenül fejlesztett és fordított alkalmazások, szolgáltatások használatára. Ma leggyakoribb használata HTTP protokollon keresztül történik, ami tulajdonképpen azt jelenti, hogy a definiált funkciókat HTTP kérésekkel aktiválhatjuk.

A tradicionális web-alapú alkalmazások és a web szolgáltatások között három alapvető különbség van:

- A web szolgáltatások adatcseréhez nem MIME üzeneteket, hanem SOAP (Simple Object Access Protocol) üzeneteket használnak. A SOAP egy XML alapú protokoll, mely meghatározza az üzenetek felépítésének szabályait. Egy SOAP üzenet bármilyen adatot tartalmazhat XML formátumban, amelyet a küldő alkalmazás szeretne, ahogy azt a következő példában is láthatjuk:

```
<env:Envelope
xmlns:env="http://www.w3.org/2001/12/soap-envelope">
   <env:Body>
      <ns:Order xmlns:ns="urn:fabrikam-com:orders">
         <item>Wallabee</item>
         <item>Wombat</item>
         <item>Wren</item>
      </ns:Order>
   </env:Body>
</env:Envelope>
```

- A web szolgáltatások nem átviteli protokoll specifikusak. Hiszen a SOAP specifikáció csupán azt definiálja, hogyan kell átvinni egy SOAP üzenetet HTTP protokollon (ez az a megoldás, amit manapság a web szolgáltatások használnak), de ettől függetlenül egy üzenet átvihető akár SMTP vagy TCP prtokollon is a megfelelő definíciók meglétével és azok használatával. Előreláthatólag a protokollfüggetlenség egy óriási előny a jövőre nézve, mivel a HTTP-t nem arra tervezték, hogy hosszú válaszidejű kéréseket, esemény

értesítéseket továbbítson. Ezeknek a funkcióknak a megfelelő támogatásához elengedhetetlen lesz más protokollok alkamazása.

- A web szolgáltatások a web alkalmazásokkal szemben önleíróak; azaz a használt üzenetek, adatok mellett a metaadatokat (adat az adatokról) is tárolják, melyek leírják az egyes üzenetek formátumát, a használt fizikai átvivő protokollokat, a szolgáltatás meghívásához szükséges logikai címzési információkat. A web szolgáltatások üzenetformátumait az ún. XML Schema (XSD) segítségével is leírhatjuk, amely eléggé rugalmas ahhoz, hogy különféle formátumokat is definiálhassunk. Egy web szolgáltatás „viselkedését" az ún. web szolgáltatás leíró nyelvvel (WSDL – Web Services Definition Language) adhatjuk meg. A WSDL segítségével társíhatjuk a különböző üzenetek fogadását funkcióhívásokhoz.

Irodalom jegyzék

1. Martin Fowler: *Patterns of Enterprise Application Architecture*, Addison Wesley, ISBN: 0321127420, 2002
2. Juhász Sándor: *Okos kliens technikák kidolgozása vastag kliensek fejlesztésére*, Kutatási-fejlesztési jelentés, Budapesti Műszaki és Gazdaságtudományi Egyetem, Automatizálási és Alkalmazott Informatikai Tanszék, 2004
3. Dieter Wackerow: *MQSeries Enterprise Application Integration Center*, IBM October 1999 MQ EAI Center
4. Jiawei Han, Micheline Kamber: *Adatbányászat - Koncepciók és technikák*, Panem, ISBN: 9635453949, 2002
5. Kende Mária, Kotsis Domokos, Nagy István: *Adatbázis-kezelés az ORACLE-rendszerben*, Panem, ISBN: 9635453477, 2002