NEMZETI TANKÖNYVKIADÓ
a Sanoma company

Logisztikai rendszerek informatikai architektúrája

Szerzők: Dr. Kovács László (3. fejezet és 5.3, 5.4, 5.5 alfejezetek)

Krizsán Zoltán (4. fejezet)

Szűcs Miklós (1. fejezet, 5.1,5.2 alfejezetek)

Wagner György (2. fejezet)

Lektor: Dr. Kusper Gábor

Korszerű anyag-, nano- és gépészeti technológiákhoz kapcsolódó műszaki képzési területeken kompetencia alapú, komplex digitális tananyag modulok létrehozása és on-line hozzáférésük megvalósítása
(TÁMOP-4.1.2-08/1/A-2009-0001)

Nemzeti Fejlesztési Ügynökség
www.ujszechenyiterv.gov.hu
06 40 638 638

MAGYARORSZÁG MEGÚJUL

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

ÚJ SZÉCHENYI TERV

TARTALOMJEGYZÉK

BEVEZETÉS

Mindannyian tapasztalhattuk, hogy az ”információ hatalom” szólás mennyire igaz a gyakorlati életben, hiszen megfelelő információ birtokában hatékonyabb lesz a döntéshozatalunk. Az értékes információ megőrzése és megtalálása napjainkban szinte elképzelhetetlen információ technológia támogatás nélkül. A tananyagmodul célja áttekintést adni a logisztikai hálózatok megvalósítását támogató információ technológiai architektúrákról és az adatkezelő mechanizmusokról. A segédlet általános ismertetőt ad az informatikai rendszerek informatikai hátterének alapjairól, az alapvető koncepciókról és a megvalósításukat biztosító eszközökről. A tananyag felhasználható minden olyan tárgyban, amely a logisztikai rendszerek informatikai hátterét a vállalati információs rendszerekbe ágyazottan tárgyalja.

A modul öt fő részből épül fel. Az első részben bemutatásra kerülnek logisztikai rendszerek informatikai hátterének alapjai: a kliens-szerver és middleware technológiák, a hálózati szolgáltatások. A második fejezetben az adatcsere hátterét biztosító hálózati elemek bemutatására kerül sor, ahol nem csak az adattovábbítás technológiájára térünk ki, bemutatjuk az adatgyűjtés tipikus eszközeit is. A harmadik rész az információs rendszerek alapját képező adatbázisokat, az adatkezelési és adatcsere módszereket tárgyalja. A negyedik részben a logisztikai berendezéseket irányító és adminisztráló szoftverek típusai, valamint a szoftverek fejlesztési módszertanai, specifikumai kerülnek bemutatásra. Az ötödik fejezetben mintarendszereket ismertetünk a VIR rendszeren belüli logisztikai modulok megvalósítására.

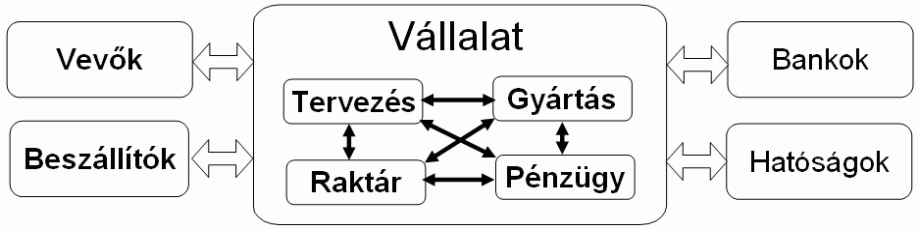
1. INFORMATIKAI RENDSZEREK ARCHITEKTÚRÁJA

1.1. VIR alapfogalmainak áttekintése

A gazdasági versenyben előnyt szerezhet az a vállalat, az a kereskedő, vagy az a bank, amely időszerűbb, pontosabb, jobban körülhatárolt információval rendelkezik a tevékenységével kapcsolatos területeken. A számítógépes és a kommunikációs rendszerek segítségével ma már hatékonyan begyűjthetjük, letárolhatjuk és a megfelelő formában feldolgozhatjuk a működéshez, a döntéshozatalhoz szükséges információkat. A vállalati szintű hatékony információkezelés ma már elképzelhetetlen megfelelő vállalati információs rendszer nélkül.

A vállalati információs rendszerek definiálása előtt nézzünk át röviden néhány olyan alapfogalmat, amelyek segítenek abban, hogy kirajzolódjon a vállalati információs rendszer koncepciója. A vállalat több alrendszerből (tervezés, gyártás…) felépülő szervezet, melynek célja gazdasági tevékenység végzése, profit maximalizálása. A vállalat több egymással kapcsolatban álló alrendszerek együtteseként jelenik meg és megadott külső gazdasági és közigazgatási környezetben működik.

A vállalat nyitott rendszert alkot, vagyis határán kívülről inputokat (rendelések, pénzügyi adatok, szabályok, törvények…) fogad környezetéből, ezeket felhasználva működteti alrendszereit, és állítja elő az outputot, a termékeket. A rendszer általános értelmezésben egymással kölcsönhatásban álló elemek egésze, melyek egy közös cél érdekében működnek. A rendszer működése során bemenetet (input) fogad és egy belső átalakítási folyamat során kimenetet (output) hoz létre (pl. a fentebb felvázolt tanulási rendszer).



1.1. ábra: Vállalat külső-belső kapcsolati rendszere (KEP_A303_I_01_01)
KEP_A303_I_01_01.JPG

A vállalat egy speciális gazdasági rendszerként jelenik meg, mivel valamilyen gazdasági cél elérése érdekében jött létre. Személyek és technikai eszközök szervezett csoportja, mely képes célok

kitűzésére, és a célkitűzésben meghatározott feladatok végrehajtására. A tevékenységének fő mozgató rugója a haszon növelése. A haszon azonban nem mindig csak számszerűsíthető fogalmakban jelenhet meg. Ha a komponensek működését vesszük, nézzük, hogy milyen hasznot hozhat egy megfelelően működő VIR rendszer:

- Nagyobb forgalom,
- Kisebb raktárkészlet,
- Rövidebb átfutási idők,
- Gyorsabb pénzforgás,
- Jobb, gyorsabb döntéshozatal,
- Elégedettebb ügyfelek,
- Előnyösebb vállalati arculat.

1.2. Információs rendszer

A VIR rendszerek az információs rendszerek körébe tartoznak. Az információs rendszerek célja az adatok megszerzésére, tárolására és a tárolt adatok különböző szempontok szerinti feldolgozására, információkká alakítására létrehozott rendszer. A kezdeti információs rendszerek papír alapúak és manuálisak voltak, vagyis a papíron megkapott adatokat emberi munkával dolgozták fel, és az előállított információkat (listákat, kimutatásokat) szintén papíron továbbították. Ma ez a folyamat természetesen számítógépek segítségével történik, melynek eredménye:

- csökkentek a költségek,
- növekedett a sebesség,
- növekedett a feldolgozott adatok mennyisége,
- növekedett a pontosság,
- bővült a kielégített igények köre (többféle jellegű, tartalmú kimutatás).

A vállalati információs rendszerek a kezdeti papír alapú nyilvántartó lapoktól (melyeken főleg a munkaidőt és raktári készleteket tartották nyilván) a mai integrált, modul rendszerű, standard számítógépes rendszerekig nagy átalakuláson, fejlődésen mentek keresztül, nézzük a fontosabb mérföldköveket.

1950-es évek

A papír alapú tárolást és manuális feldolgozást felváltják az elektronikus adatfeldolgozó rendszerek EDP (Electronic Data Processing), melyek a vállalatnál folyó gazdálkodási műveletekből származó adatokat fogadják, tárolják és egyszerű összesítő műveleteket végeznek velük. Az első programok a bérszámfejtés, a könyvelés és a raktározás adatait kezelték, volt kereső funkciójuk (pl. cikkszám alapján kigyűjtötte a raktári tételeket), és előre definiált jelentések készítésére is alkalmasak voltak. A kor számítástechnikai színvonalát a korlátozott számítógépes kapacitások, a lyukkártyák, a kötegelt (batch) adatfeldolgozás, és az ebből adódó offline megoldások jellemzik.

1960-as évek

Bővül a feldolgozott adatok területe, és megváltozik az adatgyűjtés szemlélete. Nem önmagukban az adatokat, hanem az egyes események adatait tárolták (pl. megrendelés: ajánlat, rendelés, visszaigazolás, árugyártás, kiértesítés, megrendelés lezárása). Ezeket a rendszereket tranzakció feldolgozó rendszerek (TPS - Transaction Processing System) nevezzük. Az eddigi területek kibővülnek, megjelennek a gyártásirányítási, gyártástervezési és a marketing szolgáltatások, valamint megjelennek a vállalat speciális igényeit kielégítő programok. A tehetősebb vállalatoknál megjelentek a saját számítógépek, az adattárolásban megjelentek a mágneslemezek, a feldolgozás pedig kezdett on-line módúvá válni.

1970-es évek

Az EDP/TPS rendszerek nem szolgáltattak közvetlenül felhasználható információkat a középvezetők számára, így azok speciális igényeit kielégítendő megjelentek a kimutatásokat, jelentéseket készítő vezetői információs rendszerek (MIS – Management Information System). Az elemzők és döntés-előkészítők statisztikai, modellező, szimulációs munkájának támogatására pedig megalkották a döntéstámogató rendszereket (DSS – Decision Support System). Ekkortájt alakultak ki az első operációs rendszerek, és megszületett az ún. adatbázis-koncepció, amely az adatok tárolását függetleníti a felhasználói programoktól, jelentősen meggyorsítva az adatelérést és lehetővé téve, hogy ugyanazt az adatbázist egyszerre több felhasználó is tudja használni.

1980-as évek

A vállalatok egyre több területét fedték szoftverekkel, és természetesen megjelentek az „All in one” megoldások, az EDP/TPS/MIS/DSS integrációból született rendszerek. Elnevezésük ERP – Enterprise Resource Planning, vagyis vállalati erőforrás tervező (rendszer). Az ERP rendszerek alkalmasak egy adott vállalat teljes üzleti folyamatainak lefedésére, így a folyamatok tervezésére, szervezésére, irányítására és követésére, vagyis ezek a rendszerek tekinthetők az első integrált vállalatirányítási alkalmazásoknak. A számítógépek teljesítménye folyamatosan nőtt, megjelentek a

mainframek, létrejöttek az első hálózatok, kialakultak a PC-k, és az évtized végére számos gyártó állt elő (főleg nagyvállalatoknak szánt) standard, modulokból felépülő ERP rendszerekkel, melyeket „csak” kiválasztani, bevezetni és működtetni kellett.

1990-es évek

A hálózatok elterjedésével és az Internet kialakulásával megjelentek az első szerverek, a PC-k térhódításával pedig elterjedtek az irodai programok (szövegszerkesztők, táblázatkezelők…). Az ERP-k a mainframes technológiáról átálltak a kliens/szerver megoldásra, és komoly szemléletváltáson estek át, melynek lényege: az értékteremtés folyamata nem ér véget a vállalat határainál, valamint, hogy a vállalat csak akkor lehet sikeres, ha megfelelő módon tudja kielégíteni vevőkörének igényeit. Új típusú alkalmazások hozták a megoldást: ebben az időben kezdtek terjedni az adatpiacok és az adattárházak, illetve az ezekre épülő, sokoldalú on-line elemző szoftverek. Ezekkel a technológiákkal újraértelmezték a vezetői döntések támogatást és az egyes szakterületek információigényét kielégítve új informatikai területet nyitottak. Ilyen új terület az Üzleti intelligencia világa. Az Üzleti intelligencia rendszerek (BIS – Business Intelligence System) körébe olyan alkalmazások és technológiák tartoznak, melyek célja, hogy a szükséges adatokhoz való hozzáférés biztosításával, ezen adatok megfelelő tárolásával, interaktív manipulálásával, azokat különböző aspektusokból vizsgálva meghatározzák a komponensek egymáshoz való viszonyát, egymástól való függésének mértékét, és ezzel támogassák a vállalati döntéshozatalt.

2000-es évek

Az ERP rendszerek bővülése a legújabb technológiákkal (BIS, szakértői rendszerek, elektronikus kereskedelem), áttérés a réteges kliens/szerver architektúrára, standard megoldások kis-, közepes- és nagyvállalatok számára. Jelentős fejlődést mutat a rendszerek tervezési módszertana és a megvalósító technológia is. E korszak fő eredményei közé tartozik a szolgáltatás orientált technológiák (SOA) megjelenése és a Internet alapú WS technológiák elterjedése.

A történeti áttekintés után, és azok után, hogy láttuk hogyan változtak, alakultak, fejlődtek a vállalati információs rendszerek, definiáljuk újra a VIR fogalmát.

1.3. VIR rendszer jellemzése

A vállalat környezetére, belső működésére és a vállalat-környezet tranzakcióira vonatkozó információk koordinált és folyamatos begyűjtését, tárolását, feldolgozását és szolgáltatását végző személyek, tevékenységek, és technikai eszközök összessége. A VIR fő összetevői (erőforrásai);

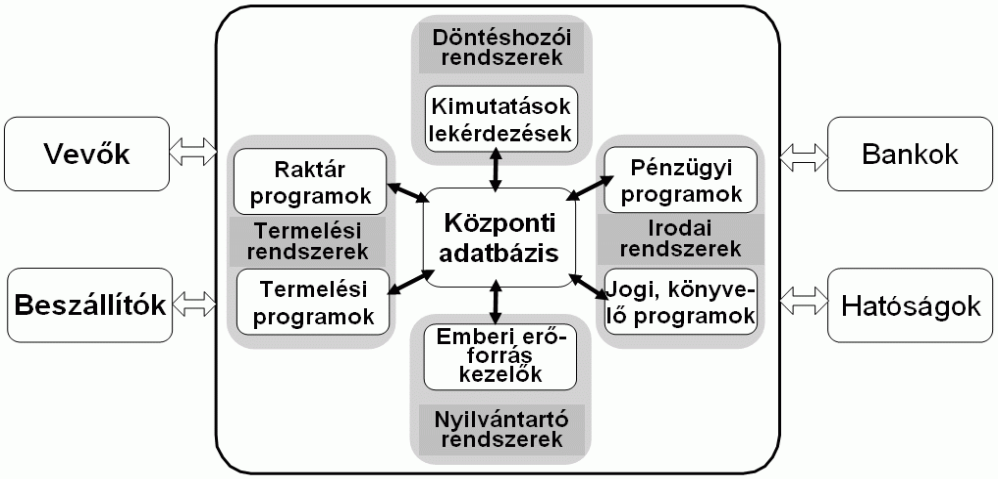
- Az ember, mint működtető, döntés-előkészítő és döntéshozó:
  - A személyzet és mindazon képessége, mellyel az információs rendszert tervezi, működését szervezi, beszerzi/fejleszti, bevezeti, működteti, és működését felügyeli
- Külső és belső adatok, információk:
  - A vállalatnál előforduló adatok a legszélesebben értelmezve (papíralapú, elektronikus, hang, kép, stb.)
- Hardver, szoftver elemek és szervezeti megoldások (ún. orgver).

  Szoftver:

  - Operációs rendszerek, az adatbázis-kezelő rendszerek,
  - Bővebben: a manuális és automatizált eljárások összessége

  Hardver:

  - Számítógépek, hálózati eszközök, multimédiás eszközök,
  - Bővebben: az információs rendszert támogató összes rendelkezésre álló berendezés



1.2. ábra: VIR rendszer külső-belső kapcsolati rendszere (KEP_A303_I_01_02)

KEP_A303_I_01_02.JPG

A vállalati információs rendszerrel szemben támasztott főbb követelmények a következőkben foglalható össze:

- Az üzleti folyamatok elemi eseményeinek feldolgozása és nyomon követése (tranzakció kezelés),
- A különböző funkcionális szervezeti egységektől, ill. vállalati folyamatokból származó adatok rögzítése,
- Egységes vállalati adatbázis karbantartása,
- Különböző szintű vezetői információigények kielégítése,
- Optimalizálás költségre, átfutási időre, erőforrás felhasználásra, készletszintre.

Az elvárásokból következik, hogy egy vállalati információs rendszer csak IT (Információ Technológia) eszközök segítségével valósítható meg. Fontos, hogy a vállalati információs rendszerek integráltak, vagyis minden moduljuk a vállalat méretétől és a munkahelyek fizikai elhelyezkedésétől függetlenül egy egységes rendszert alkot, ezért a vállalat számítógépes infrastruktúrájának vizsgálatakor a teljes rendszert kell áttekinteni.

1.4. VIR architektúrák alaptípusai

Architektúra fogalma alatt egy rendszer egyes alrendszereinek, az alrendszerek kapcsolatának és működési elveinek vázlatszerű, nagyvonalú leírását értjük. Az IT fejlődésével, a hardverek sebességének növelésével, képességeiknek bővülésével (megjelenítők fejlődése, háttértár kapacitás növekedése, hálózati kommunikáció megjelenése), az újabb és újabb szoftveres megoldások megjelenésével együtt változott a vállalati információs rendszerek architektúrája.

Decentralizált architektúra

Az első EDP és a korai TPS rendszerek önálló szigetekként jöttek létre, ekkortájt a számítógépes rendszer még teljesen független volt az adatok keletkezésének és felhasználásának a helyszínétől. Az összegyűjtött adatokat „időnként” off-line módon feldolgozták egy külön apparátussal működtetett (eleinte még nem a vállalat területén található) számítógéppel, az eredmények visszakerültek a megfelelő területre. Ebben az időszakban nem valósult meg a mai értelemben vett adatcsere az egyes helyszínek között.

Centralizált architektúra

A nagy teljesítményű mainframe gépek elterjedésével a tehetősebb vállalatoknál számítástechnikai osztályok jöttek létre. Az osztályok dolgozói üzemeltették a központi számítógépet, ezen futottak az EDP/TPS, később a MIS/DSS modulok, az adatokat a számítógéphez tartozó terminálokon lehetett felvinni, és itt jelentek meg az eredmények is. A felvitt adatok adatbázisokba kerültek, ezek segítségével már volt adatcsere az egyes modulok között. Ilyen körülmények között működtek a korai ERP rendszerek, melyek a hardver elemek gyorsulása és a szoftverek szemléletmód váltása miatt a kezdeti batch feldolgozástól eljutottak a valós idejű működésig.

Lazán csatolt architektúra

A kisebb méretű, olcsóbb számítógépek és a hálózatok megjelenésével elterjedt az a filozófia, mely szerint egy megfelelő összteljesítményt sokkal olcsóbb kisebb számítógépekből és az azokat összekapcsoló hálózatból felépíteni, mint egy mainframe számítógépet megvásárolni. Az egyes területekre a feldolgozási igénynek megfelelő minőségű és mennyiségű számítógép kerül, az igények változásával a gépek áthelyezhetők, számuk változtatható.

Azok a vállalatok, ahol mainframek működtek, kibővítették a gépparkjukat PC-kkel, ezeken futottak bizonyos modulok (irodai programok, MIS, DSS rendszerek…), az adatcserét (ami eleinte szimpla fájlcsere volt) a hálózat segítségével oldották meg. A kisebb cégek nem vásároltak mainframet, csak kisebb gépeket, a hardver architektúra változását pedig követte a szoftverek megváltozása is, az ERP gyártók is megjelentek a hálózatos megoldásokkal.

Az elosztott architektúrában jelent meg a kliens és a szerver fogalma. A klienseken folyt a munkavégzés, a szerverek biztonsági és erőforrás megosztási funkciókat láttak el, a köztük lévő hálózat pedig vállalaton belüli, lokális hálózat volt.

Az akkori filozófia:

1. Jelentkezz be a szerverre (ehhez tudni kellett a szerver nevét, kellett egy felhasználói név és egy jelszó)
2. Keresd meg a szerver számodra megosztott erőforrásait (hely a háttértáron, bizonyos fájlok, nyomtató…)
3. Dolgozz az erőforrásokkal: azt tehetsz, amire a személyre szóló engedélyed kiterjed (láthatsz egy létező fájlt, esetleg el is olvashatod a tartalmát, esetleg felül is írhatod)

Kétszintű kliens/szerver architektúra

Gyorsuló hardverek, új szabványok megjelenése és elterjedése, és a szoftver-írás szemléletmód váltása vezetett a kétszintű kliens/szerver architektúra elterjedéséhez. Tudatosan, az alkalmazásban megoldandó feladatoknak megfelelően bontották két részre a programokat, így egyazon alkalmazásnak két jól elkülöníthető része került a szerverre és a kliensekre. Nézzük először a feladatokat, aztán azt, hogyan lehet ezekből kialakítani a két oldalt.

Az alkalmazások komponensei a megoldandó feladatok alapján:

- Bemeneti, kimeneti funkciók: Adatbevitel, a bevitt és az eredményként kapott adatok megjelenítése a felhasználói felületen.
- Adatfeldolgozási funkció: A bevitt adatok ellenőrzése, megfelelő formára alakítása, az üzleti (ügymeneti) logikának megfelelő adatfeldolgozás és adatmenedzselés (felvitel, módosítás, törlés, lekérdezés), valamint a hibák észlelése és kezelése.
- Adattárolási funkció: A fizikai adattárak kezelése, adatok kiírása, visszaolvasása.

A kliens (angolul client) olyan önállóan is működőképes számítógép, amely hozzáfér egy (távoli) szolgáltatáshoz, amelyet egy másik – számítógép-hálózaton keresztül elérhető – számítógép nyújt. A kliens kéréseket küld a szervernek (adat igény), és fogadja a válaszként kapott adatokat. Az alkalmazás a kliensen keresztül kommunikál a felhasználóval. A kliensnek ismernie kell a szerverek és az általuk biztosított szolgáltatások neveit.

A szerver (kiszolgáló, angolul server) olyan (általában nagyteljesítményű) számítógépet jelent, ami más gépek számára a rajta tárolt vagy előállított adatok felhasználását teszi lehetővé. A szerver passzív, várja a kliensektől a kéréseket, feldolgozza azokat, és visszaküldi a választ. A szerver nem „ismeri” a klienst, és nem tudja hány kliens létezik. Az alkalmazás két oldalának szétválasztása a megvalósított funkciók alapján megkülönböztethető:

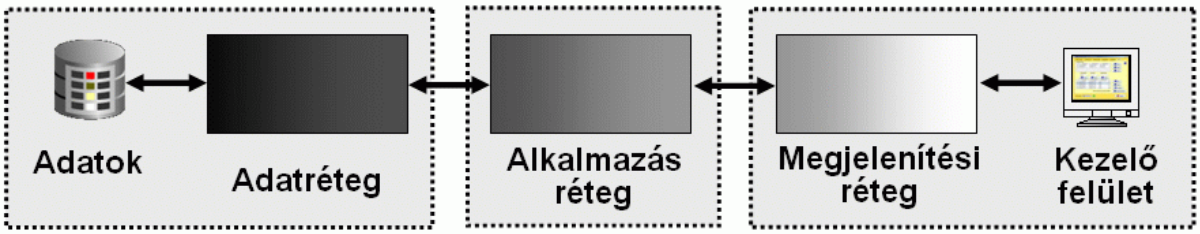
- vastag kliens modell: a szerver csak az adattárolási funkciókat látja el, az adatfeldolgozási funkciókat és a felhasználóval történő kapcsolattartást a kliensen futó szoftver valósítja meg.
- vékony kliens modell: ebben a modellben az adattárolási és az adatfeldolgozási funkciók a szerveren zajlanak, a kliens csupán az adatbevitelért és a megjelenítésért felelős.

A szoftvergyártók a 80-as években jelentek meg kliens/szerver architektúrájú ERP rendszereikkel, és három, akkoriban kialakult és hamar népszerűvé és fontossá vált tulajdonsággal jellemezték azokat:

- Nyílt rendszer: különböző szállítók hardver- és szoftver-elemeiből felépíthető, azokkal bővíthető megoldás. Általában a kliensek voltak sokfélék, mind a hardver, mind az operációs rendszer tekintetében.
- Skálázható: A rendszer képességei új komponensek (szerverek, kliensek) hozzáadásával bővíthetőek.
- Hibatűrő: Ugyanarra a feladatra beállított több szerver, és így az információk többszörözésének lehetősége azt jelenti, hogy ezek a rendszerek bizonyos hardver és szoftverhibákat képesek eltűrni.

Háromszintű kliens/szerver architektúra

A skálázhatóság fokozása és a könnyebb fejleszthetőség újabb szemléletváltást követelt, az alkalmazások három komponensét három különböző, de egymásra épülő „rétegként” megvalósítva kialakult a háromszintű kliens/szerver architektúra.



1.3. ábra:Háromrétegű struktúra komponensei (KEP_A303_I_01_03) KEP_A303_I_01_03.JPG

A rendszer fontosabb rétegtípusai:

Adatréteg: (data tier) Feladata az adatok perzisztens tárolása, és az azokon végezhető elemi műveletek – létrehozás, lekérdezés, módosítás, törlés – támogatása. Leggyakrabban ezt a réteget relációs adatbázis segítségével valósítják meg. Tágabb értelemben az adatréteghez tartozik minden rendszer, amelyből egy alkalmazás adatokat nyer ki.

Alkalmazás réteg: (application tier) A konkrét alkalmazási terület igényeinek megfelelő funkcionalitást biztosítja oly módon, hogy az üzleti szabályok figyelembevételével hívja meg az adatréteg szolgáltatásait. Ezt a réteget üzleti logikai rétegnek is nevezik, bár ez az elnevezés első olvasatban kissé félrevezető, találóbb helyette az ügymeneti réteg elnevezés. Ha például listát kérünk az autókról, más-más információkat vár egy autókereskedő, egy autó szerviz, vagy egy muzeális autókról érdeklődő műgyűjtő. Ez jelenti az üzleti logikát (ügymenetet) az egyes autós

alkalmazásokban. Az alkalmazás réteget gyakran nevezik middleware-nek, köztes (vagy középső) rétegnek, esetleg köztesszoftvernek.

Megjelenítési réteg: (presentation tier) Biztosítja az alkalmazás felhasználói felületét, vagyis a felhasználói beavatkozások hatására meghívja a megfelelő üzleti logikai funkciót, majd a hívás eredményének megfelelően frissíti a felhasználói felületet.

A réteges megoldás előnyei a kétszintű modellel szemben:

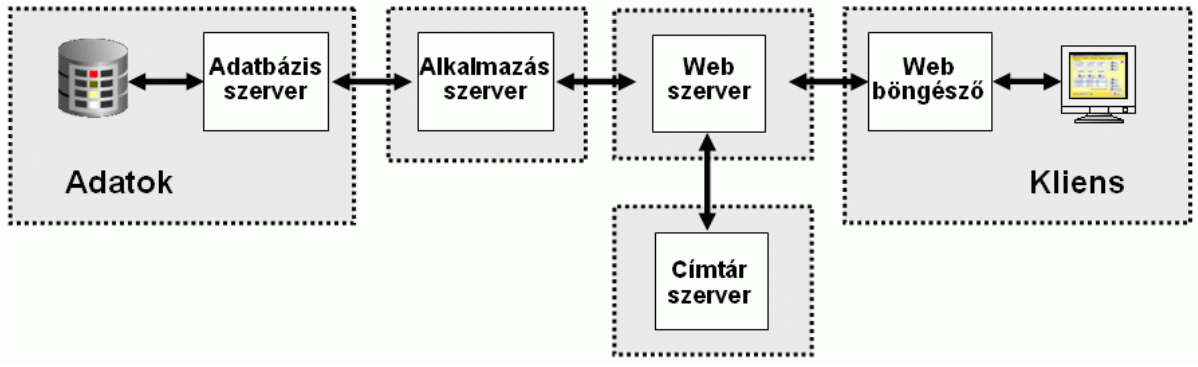
- Az egyes rétegek külön fejleszthetők, könnyebben módosíthatók
- Az alkalmazás független az adatoktól, könnyen le lehet cserélni az adatbázist
- Ugyanahhoz az alkalmazáshoz egyszerűbb és olcsóbb többféle felhasználói felületet létrehozni
- Az adatok nem kerülnek át a kliens oldalra, így nagyobb a biztonság

Többszintű kliens/szerver architektúra

Az internet térhódításával és az állandó rendelkezésre állással megnőtt az igény arra, hogy az egyes vállatok információs rendszereit külső helyszínekről (más vállalatoktól, otthonról, külföldről) is el lehessen érni, ezért a kliens és a szerver közötti kommunikációba beiktattak egy webes réteget.

Web réteg: Ez a réteg fogadja a böngészőktől érkező kéréseket, meghívja a megfelelő alkalmazást, és az alkalmazástól kapott adatokból előállítja a megfelelő weblapot, amit visszaküld a kliensen futó böngészőnek.

Azt az architektúrát, amelyben háromnál több réteg különíthető el, többszintű kliens/szerver architektúrának (multi-tier) nevezzük.



1.4. ábra:Többrétegű struktúra komponensei (KEP_A303_I_01_04) KEP_A303_I_01_04.JPG

Egy vállalat egyik középvezetője munkaidő után, otthonról listát kér a raktári készletről, hogy megtervezze másnapi teendőit. Elindítja böngésző programját, beírja a vállalat web szerverének címét, az ezután megjelenő azonosítás panelen megadja nevét és jelszavát, majd kiválasztja a raktári készlet lekérdezése funkciót, beírja a megfelelő termék nevét, és megtekinti az adatokat. Kérését egy többszintű kliens/szerver architektúrájú információs rendszer szolgálja ki, de valószínű, ő erről semmit sem tud.

Az elosztott rendszerek „áttetszőek” (átlátszóak), a felhasználó nem látja az egyes szervereket, nem tudja azok nevét, helyét a hálózatban. Nem látja, hogy más felhasználók is használják a rendszert, fogalma sincs arról, hogy amikor megadta nevét és jelszavát az azonosítást egy címtár szerver végezte el, és arról sem, hogy a készlet lekérdezés gomb megnyomása után több szerver is azon munkálkodott, hogy megjelenjenek a kért adatok. Az áttetszőség egyszerűvé, felhasználó baráttá teszi az ilyen rendszereket.

1.5. A többszintű kliens-szerver architektúra jellemzői

A kétszintű alapmodell legnagyobb hátránya a funkciók szétválaszthatóságának és továbbfejlesztésének nehézsége, illetve problémás volt a skálázhatóság, a hordozhatóság, és az újrafelhasználhatóság hiánya is. Az első körben kidolgozott és bevezetett megoldáscsomag eredményezte a háromszintű architektúra kifejlődését. A két végpont – a kliens és a szerver – közé beiktatták az alkalmazás szervereket, ide került az üzleti logika a kliens és az adatbázis szerver oldalról is, így az alkalmazás skálázhatóvá, könnyebben fejleszthetővé vált.

A második körben a fejlesztők mind több és több funkciót, támogatást építettek az alkalmazásszerverekbe, a teljes üzleti logikát kisebb, egymással összekapcsolt komponensekre bontották. Újabb és újabb – a programozást megkönnyítő – interfészek születtek, így növekedett a hordozhatóság és az egyes komponenseket más alkalmazásokban is fel lehetett használni. Ezt a réteget – amelyben az alkalmazás szerverek dolgoznak – köztesrétegnek (middleware) nevezzük.

1.5. ábra:Köztesréteg szerepe (KEP_A303_I_01_05) KEP_A303_I_01_05.JPG

A többszintű rendszer főbb jellemzői:

- Egyszerre több száz, több ezer felhasználó használhatja,
- Heterogén rendszerek: különböző technikák működhetnek együtt:
  - Hardverek,
  - Operációs rendszerek,
  - Protokollok,
  - Programozási nyelvek (technológiák)
- Nyílt rendszerek: képesek együttműködésre más rendszerekkel
- Jól skálázható: igény esetén több szerver beállítása az adott feladatra,
- Biztonságos: bizalmas adatok a jobban védett szervereken,
- Kiemelkedően hibatűrő: ha egy szerver kiesik, egy másik azonos feladatú átveheti annak feladatát,
- Az egyes rétegek külön fejleszthetők (specializáció),
  - Jól definiált szolgáltatások
  - Jól definiált interfészek
- Egyszerűen változtathatók az alkalmazások,
- Egy alkalmazáshoz többféle felhasználói felület lehet,
- Egyszerű az adatréteg cseréje,
- Konkurens műveletek könnyen menedzselhetők,
- A kliens többnyire csak egy böngésző program,

Az általános jellemzés után nézzük az egyes rétegeket, és azok fontosabb tulajdonságait

Adatréteg

Az adatréteg tárolja fizikailag az alkalmazáshoz kapcsolódó adatokat, és biztosítja az adatok kezeléséhez szükséges funkciókat. A működési módszer általában a következő: az adatokat a memóriában tároljuk ahelyett, hogy egyből az adatbázisban helyeznénk el, így sokkal gyorsabb a működés. Az eseményeket folyamatosan naplózzák, és időnként lementik a memória tartalmát, frissítik az adatbázist, így rendszerhiba esetén gyorsan visszaállítható a memória tartalma, és folytatható a munka. Jellemzői:

- Perzisztens adattárolás,
- Adatokon végzett műveletek elvégzése:
    - Létrehozás, módosítás, törlés
    - Lekérdezések
- Tágabb értelemben: minden olyan rendszer, amiből az alkalmazásunk adatokat nyer ki
- Ebben a rétegben működnek az adatbázis-szerverek
- Általában relációs adatbázisok, de egyre gyakoribb az objektumorientált megoldás is.

Köztesréteg

Első közelítésben: Egy olyan szoftver(csomag) a kliens és szerver között, mely lehetővé teszi a felhasználó és az erőforrások üzleti logikának megfelelő kommunikációját a hálózaton keresztül. Ez egy olyan elérhető szoftver réteg, mely a heterogén platformok és protokollok hálózati rétege és az üzleti alkalmazás(ok) között helyezkedik el. Leválasztja az üzleti alkalmazásokat bármilyen, a hálózati réteg okozta függőségről, melyet a heterogén operációs rendszerek, hardver platformok és kommunikációs protokollok okoznak.

A köztesréteg további rétegekre bontható:

- Üzleti logikai (ügymeneti) réteg, amely a konkrét alkalmazási terület igényeinek megfelelő funkcionalitást biztosítja oly módon, hogy az üzleti szabályok figyelem-bevételével hívja meg az adatréteg szolgáltatásait.
    - Az üzleti logikai réteget megvalósító programok az alkalmazásszervereken futnak
- Webréteg, amely a böngészőktől érkező HTTP-kéréseket értelmezi, meghívja a megfelelő üzleti logikát, majd pedig megfelelő (HTML, XML, WML) választ generál.
  Akkor szükséges, ha vékony klienseket kell kiszolgálni.
    - A webréteg programjai a web szerverekre kerülnek.

Működési módjuk szerint a köztesréteg alábbi főbb csoportjait különböztethetjük meg:

- **Üzenetközpontú** (Message Oriented Middleware – MOM): Üzenetek küldésével és fogadásával kapcsolja össze a modulokat. Általában aszinkron módon működik: az alkalmazás végzi a saját feladatát, és mikor üzenetet kell küldeni, elhelyezi egy sorban, és halad tovább. Ha megérkezik

a válasz, feldolgozza azt. Működhet szinkron módon is, ekkor a válasz megérkezéséig a küldő alkalmazás várakozik.

- **Távoli eljáráshíváson alapuló** (Remote Procedure Call – RPC): olyan technológia, mely lehetővé teszi egy alkalmazásnak más gépen lévő alkalmazás eljárásainak meghívását. A kliens oldal becsomagolja a hívás paramétereit, és eljuttatja a hívással együtt a szerver oldalnak. A szerver oldal kicsomagolja a paramétereket, és meghívja az alkalmazás eljárását, mint lokális eljárást. Az visszaadja a visszatérési értéket, melyet a szerver becsomagol, és visszaadja a kliensnek, amit az kibont, és visszaadja a hívó félnek.
- **Objektum lekérdező ügynök** (Object Request Broker – ORB): Az objektumorientált metódushívás közvetítők két csoportba sorolhatóak: az egyik az OMG CORBA szabványhoz illeszthető, míg a másik a Microsoft OLE/COM technológiája. A kommunikáció mindkét esetben távoli metódushívásokként megvalósított kérésekből áll.
- **Tranzakció feldolgozó menedzser** (Transaction Processing Monitors – TPM): leveszi a terhet az adatbázis-kezelő rendszer válláról, menedzseli (monitorozza), irányítja a tranzakciókat, biztosítva ezzel az adatintegritást.
- **Adatbázis-kezeléshez kifejlesztett** (Remote Data Access – RDA): Az adatbázis middleware termékek általános és konzisztens elérést biztosítanak sok adatforráshoz, melyek lehetnek relációs, hierarchikus és objektumorientált adatbázis-kezelők is.

Kliens réteg

A mára általánossá vált vékony kliens megoldások esetén a kliens gépen csak egy böngésző szükséges, így nagyon gyenge hardverek is használhatók, semmilyen driver-t sem kell a kliensre telepíteni, és mivel többféle operációs rendszerre is léteznek böngészők, ez a technológia teljesen platform független. Ha változtatni kell az alkalmazáson (pl. verzió frissítés), az csak a szervereket érinti, a sok-sok klienst nem. Jellemzői:

- Biztosítja az alkalmazás felhasználói felületét,
- felhasználói beavatkozások hatására meghívja a megfelelő üzleti logikai funkciót,
- a hívás eredményének megfelelően frissít bizonyos felhasználói felületelemeket.
- Egy rendszerben többféle kliens lehet:
  - Eltérő hardver
  - Többféle operációs rendszer
  - Más-más alkalmazás kliensei
  - A megvalósított funkciók szerint:

- Vékony kliens: csakis a megjelenítéssel és a felhasználóval történő kapcsolattartással foglalkozik
- Vastag kliens: a kliensen több-kevesebb üzleti logikai funkció is található. Minél több, annál „vastagabb” a kliens.

1.6. Szerver típusok

A szerver elemek valamilyen szolgáltatást nyújtanak a kliens programok számára. A többrétegű kliens/szerver architektúrában általában a következő szerver típusok használatosak:

- Web szerverek
- Alkalmazás szerverek
- Adatbázis szerverek

A következőkben az egyes szerver típusok főbb jellemzőit tekintjük át.

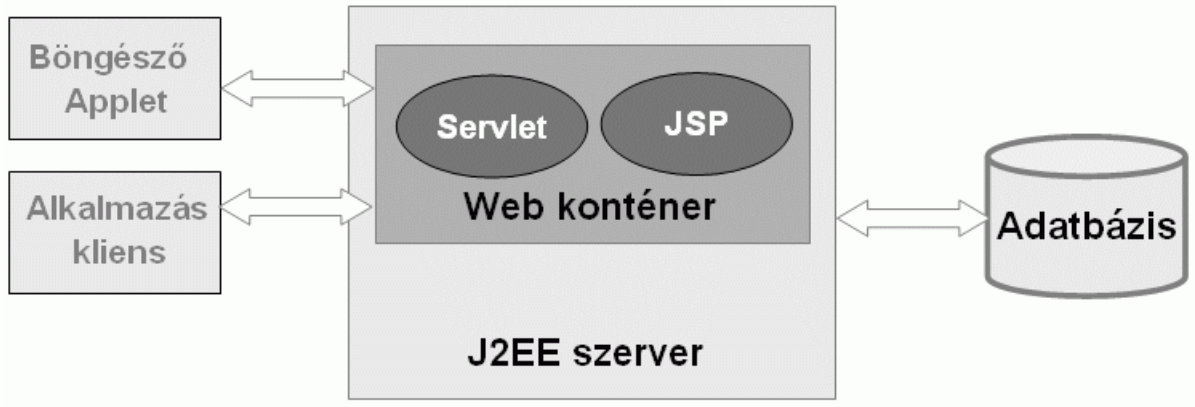
Web szerverek

- A böngészőktől érkező HTTP kéréseket értelmezi,
- kiszolgálja a helyben tárolt adatokra (képek, weboldalak) vonatkozó kéréseket,
- az üzleti logika felhasználását igénylő kéréseket továbbítja az alkalmazás szerverek felé,
- majd pedig megfelelő (tipikusan HTML-, de akár XML-, WML-, tetszőleges bináris formátumú) választ generál.
- Bizonyos esetekben külön réteg, de az alkalmazás szerverre telepítik.

Egy konkrét megoldás az Apache HTTP Server. Az Apache HTTP Server nyílt forráskódú webkiszolgáló alkalmazás, szabad szoftver, mely kulcsfontosságú szerepet játszott a World Wide Web elterjedésében. Szabadon használható, biztonságos, mind üzleti- mind magán célú felhasználásra megfelelő. A következő operációs rendszerekben használható: Unix, Linux, Solaris, Novell NetWare, Mac OS X és Microsoft Windows. Az Apache sok szabványt támogat, az ismertebb, támogatott programnyelv modulok a Perl, a Python, és a PHP. Statikus és dinamikus weboldalak közzétételére egyaránt használják, nemcsak weboldalak, hanem egyéb tartalom publikálására is használható, például tetszőleges fájlok megosztására is.

A Java technológiai megoldása a web szerverek megvalósításra a J2EE szerver web konténerében futó komponensek, melynek elemei:

- Servletek: Java osztályok, amelyek dinamikusan dolgozzák fel a kérést és építik fel a választ
- Java Server Page-ek (JSP): szöveg-alapú dokumentum-vázak, amelyek a servletként lefutva kapják meg a dinamikus tartalmat



1.6. ábra:J2EE webszerver architektúra (KEP_A303_I_01_06) KEP_A303_I_01_06.JPG

A Web konténer lehet a web szerver része, vagy egy önálló alkalmazás (pl. Tomcat), mely biztosítja azt a környezetet, amelyen keresztül a kérések és a válaszok lekezelhetők. Tartalmazza és menedzseli a servlet-eket egész életciklusuk alatt.

Alkalmazás szerverek

Az alkalmazás szervereken futó program modulok oldják meg a konkrét feladatokat, feldolgozzák a klienstől kapott (a webszerver által továbbított) adatokat, eltárolják az adatbázisban, és a kliensek részére adatokat szolgáltatnak az adatbázisból. Röviden: kiszolgálják a kliensek kéréseit.Jellemzőik:

- Futtató környezetként szolgál a rá feltelepített alkalmazások számára
- Az alkalmazásoknak meg kell felelniük bizonyos formai feltételeknek
- Rendszerint van ún. *hot-deploy* könyvtár: ebbe bemásolva az elkészített modulokat, az alkalmazásszerver azokat automatikusan telepíti
- Rendszerint van lehetőség webes menedzsmentre,
- Általában ezen keresztül is lehet modulokat telepíteni

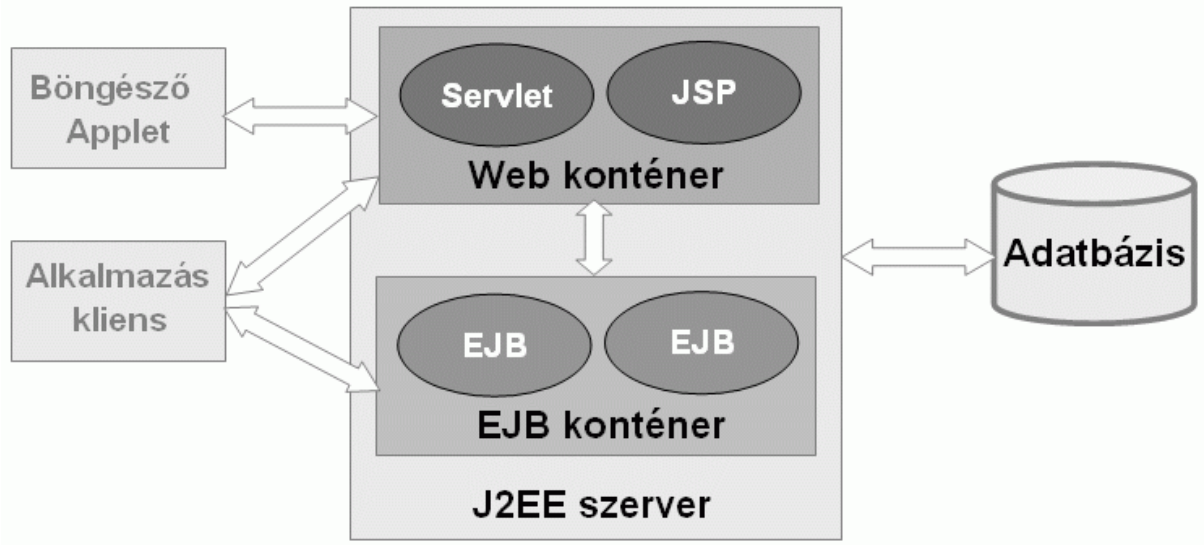
- Általában klaszterekben dolgoznak: Több alkalmazás-szerver példány együttműködik a feladatok elvégzésére
- Authentikáció: felhasználói adatbázis alapján
    - Operációs rendszer felhasználói alapján
    - LDAP alapján
    - Egyéb (tetszőleges adatbázisból)
- Biztonságos kommunikáció: SSL használata

Néhány konkrét alkalmazás szerver:

- Apache Tomcat
- GlassFish
- JBoss
- WebSphere

A Java technológiai megoldása: a J2EE szerver EJB konténerében futó komponensek:

- EJB – Enterprise JavaBeans: üzleti logikai (ügymeneti) komponens, amely a hozzá tartozó Java osztály- és erőforrásfájlokkal egy alkalmazás keretében van telepítve, és más komponensekkel kommunikál.



1.7. ábra:J2EE webszerver architektúra EJB konténerrel (KEP_A303_I_01_07)

KEP_A303_I_01_07.JPG

Az EJB-k szabványos felülettel rendelkező, szolgáltatásokat nyújtó, szerver oldali komponensek, melyeket építőkockaként használhatunk egy összetett alkalmazás megalkotása során. Három fajta EJB létezik:

- Session bean – mely folyamatokat (számítások) testesít meg.
- Entity bean – rekordokat reprezentál (termék, raktárhely…), egy-egy példány egy-egy rekordnak felel meg. A perzisztens adattárolás segítségükkel valósul meg, automatikusan mentik az adatokat az adatbázisba.
- Message-driven bean – amely több együttműködő alkalmazás kommunikációját, az üzenetkezelést megvalósítja meg.

Az EJB-k a relációs adatbázissal az SQL nyelv segítségével kommunikálnak.

Példaként vegyük azt az esetet, amikor egy lelkes vásárló ül a számítógépe előtt, és egy autókereskedő weboldalán gondosan összeválogatta áhított autójának típusát, motorját, színét, és extráit, és megnyomja az Árszámítás nyomógombot. A böngésző http nyelven kérést küld a J2EE szervernek, melyben a választ egy JSP (dokumentum-váz) oldalként alakították ki. A web konténer továbbítja a kérést az EJB konténerhez, ahol aktivizálódik egy session bean, mely az adatbázisból néhány Select parancs segítségével lekéri az adatokat, melyek egy-egy entity bean-be kerülnek. A session bean az entity bean-ekben lévő adatokkal feltölti a JSP-t, amely servletként lefutva visszaküldi a választ a böngészőnek, és megjelenik az autó árlistája.

Adatbázis szerverek

Az adatbázis szerverek olyan programok, melyek az adatbázis adatainak tárolását, módosítását, rendezését, szervezését, adott szempontoknak megfelelő visszakeresését, titkosítását teszik lehetővé. Általában több felhasználós üzemmódban, több szálon, relációs elven, SQL-alapon szolgálják ki az alkalmazásszervereket adatokkal, ezen kívül számos egyéb szolgáltatást nyújtanak:

- **Felügyelet**: A programokhoz általában tartozik néhány parancssoros és grafikus segédprogram, ezekkel könnyen elérhetjük, vagy létrehozhatjuk a táblákat, gyorsan elintézhetjük a felhasználókkal kapcsolatos adminisztrációt, sőt: információkat kaphatunk a kiszolgáló állapotáról vagy a táblák optimalizációjáról is.

- **Tranzakció kezelés**: A tranzakciók segítségével több módosítást végezhetünk el az adatbázisban úgy, mintha egyetlen művelet lenne. Ha elindítunk egy, az elvégzett módosítások nem kerülnek azonnal végrehajtásra, csak akkor, amikor erről külön rendelkezünk (COMMIT parancs), de megoldható a tranzakció ún. visszagörgetése (ROLLBACK parancs), amellyel visszavonjuk a módosításokat, és a tranzakció előtti állapotot állítjuk vissza. A tranzakciók használata nagyobb biztonságot ad, sőt, több felhasználós rendszereknél egyenesen elengedhetetlen.
- **Tárolt rutinok**: Egy tárolt eljárás olyan SQL nyelven írt parancsok összessége, amelyet az adatbázis szerveren rögzítünk, és azután a programozásban használatos módon meghívhatjuk. A megoldás egyik nagy előnye, hogy az adatbázissal kapcsolatos logika valóban az adatbázis szintjére kerülhet, nem teszi nehezen érthetővé a programkódot. A másik, talán ennél is fontosabb előny, hogy itt az elvégzett műveletekben szereplő rekordok nem hagyják el az adatbázis szervert, hanem ott helyben történik meg a feldolgozásuk, így csökken a hálózati terhelés és felgyorsul a végrehajtás.
- **Naplózás**: A naplózás azt jelenti, hogy a szerver futása alatt minden klienstől érkező kérést (kapcsolódás, lekérdezés …) egy szöveges fájlba ír. Ez nem elsődleges védelmi módszer, szerepe inkább az utólagos elemzés, melyből megállapíthatók a betörési kísérletek, bizonyíthatók az esetleges visszaélések.

Egy konkrét megoldás: MS SQL Server, melynek jellemzői:

- 1989-ben jelent meg először
- T-SQL változatot használja, ami az SQL-92 szabvány megvalósítása
- MSSQL szerverek egymás között TDS (Tabular Data Stream) nevű alkalmazásszintű protokollal kommunikálnak.
- ODBC, JDBC, SOAP kapcsolatok
- Beépített OLAP támogatás (Analysis Service)
- Üzenet rendszer támogatás (Messaging System)
- Tükrözés és klaszterezés támogatása
  - Automatikus failover lehetőséggel
- SQL Server Express Edition – ingyenes változata is létezik

1.7. Az SAP VIR rendszer működési struktúrájának áttekintése

Az **SAP** egy világszerte elterjedt vállalatirányítási rendszer főleg a nagy- és középvállalatok számára. 1972-ben jelent meg az R/1 változat, mely egy pénzügyi könyvelő rendszer (EDP) volt. Az R/2 változat egy mainframe gépeken működő, valós idejű „Real Time” (TPS) rendszer, mely fokozatosan fejlődött és bővül, egészen az 1992-ben megjelent R/3 verzióig.

Az R3 egy többrétegű, kliens-szerver struktúrájú, modul rendszerű ERP megoldás, melynek Enterprise változata támogatja az Internet-alapú működést. Több, mint 30 nyelven elérhető, nem iparág-specifikus, tehát bármely fajta vállalat tevékenységénél használható, de léteznek speciális iparági moduljai. Grafikus felülettel, és saját programnyelvvel (ABAP) rendelkezik, az egyes moduljai önállóan képesek működni, így lehetőség van a fokozatos bevezetésre és a bővítésre. A modulok testre szabhatók, ami lehetővé teszi, hogy a felhasználók által igényelt speciális funkciókat be tudják építeni a programba.

Az SAP általában a következő modulokból épül fel:

Számvitel:

- Pénzügyi számvitel (FI),
- Kontrolling (CO),
- Eszközgazdálkodás (AM),
- Projekt rendszer (PS).

Logisztika:

- Értékesítés (SD),
- Anyaggazdálkodás (MM),
- Raktárgazdálkodás (WM),
- Termelésszervezés (PP),
- Minőségbiztosítás (QM),
- Karbantartás (PM).

Humánerőforrás-gazdálkodás:

- Személyügyi adminisztráció (HR),

Iparági megoldások:

- Kiskereskedelmi megoldás (IS – Retail),
- Közszolgálati szektor (IS – U),
- Banki megoldás (IS – B),
- Kórházi megoldás (IS – H),

- Kb. 30 speciális modul.

Az SAP R/3 háromrétegű kliens/szerver architektúrára épül,melynek elemei a megjelenítési réteg, az alkalmazási réteg és az adatréteg. Az egyes rétegek főbb feladatai:

1. Megjelenítési réteg: Grafikus felületű kliens, melynek elnevezése SAPGUI. Csak az adatbevitelt és a megjelenítést végzi (minimális adatellenőrzéssel), az összes kérést továbbítja az alkalmazási réteghez.

2. Alkalmazás réteg: Egy vagy több alkalmazás szerver, melyek az SAP saját programnyelvén, az ABAP-on megírt programokat futtatnak. Eredetileg csak Unix rendszeren futott, később Windows-on is elérhetővé vált.

3. Adatréteg: Az egyes modulok külön-külön adatbázisban tárolták az adatokat, de az egyes tranzakciók minden adatbázisban módosították az értékeket. Bár így jelentősen nagyobb a tárolt adatok mennyisége és a redundancia, a rendszer átlátható, könnyen bővíthető, és gyorsan reagál a lekérdezésekre.

SAP NetWeaver

A 2004-ben bevezetett változatban egy (vagy több) web alkalmazás szerver dolgozik, lehetővé téve ABAP nyelvű és JAVA nyelvű programok futását. Ez tulajdonképpen egy web alapú integrációs és alkalmazási platform, melynek moduljai elérhetők web böngészőn keresztül is. A korábbi adatcentrikus szemléletmód megváltozott, ennek a verziónak a középpontjában az együttműködő, valós idejű folyamatok állnak.

A NetWeaver változat többrétegű kliens/szerver architektúrát használ. A webrétegben működő tranzakciós szerverek teremtik meg a kapcsolatot a kliensek és az alkalmazás rétegben működő ABAP és JAVA alkalmazás szerverek között, melyeket az adatrétegben működő adatbázis szerverek szolgálnak ki adatokkal.

2. INFORMATIKAI RENDSZEREK HARDVER KOMPONENSEI

Az információ feldolgozás hatékonyságának nagyságrendekkel történő emelését eredményezte az a technológiai újítás, mely lehetővé tette az egyes számítógépek erőforrásinak összekötését, az erőforrások egymás közötti hatékony megosztását. „Navigare necesse est …!" („Hajózni szükséges…!"). Az ismert mondás egy hajóskapitány nevéhez fűződik. Aki Interneten web-ezik, az „szörföl"ha nem is a tenger hullámain, de a hálózaton. Ma már természetesnek vehető, hogy még azok is, akik nem feltétlenül informatikai szakemberek, tudják, hogy a számítógépes hálózat is szükséges. Használják a családok elektronikus levelezésre (pl.: freemail, citromail, yahoo, gmail), online beszélgetésre (írásban, vagy szóban, esetleg kameraképpel pl.: skype, msn), illetve közösségi oldalak segítségével egymással való kapcsolattartásra (pl.: facebook, iwiw, myvip, twitter). Egy vállalaton belül a hálózat még inkább indokolt, hiszen nagymértékben lehet vele csökkenteni a cégen belüli papír alapú kommunikációt. Könnyebb az archiválás, az iktatás, és a többi napi tevékenység elvégzése. Vegyük egy egyszerűbb informatikai rendszer összetevőit. Gyorsan belátható, hogy lesz egy vagy több szolgáltatást nyújtó szerver, lesznek a szolgáltatásokat igénybevevő kliensek, és a szerverek eléréséhez elengedhetetlen valamilyen számítógépes hálózat. Az előző példa egy leegyszerűsített rendszer komponenseit mutatta be. A valóságban azonban lényegesen összetettebbek is vannak.

2.1. Számítógép szerverek osztályozása, telepítésük alaplépései

Hálózatot tehát nem öncélúan, hanem vagy valamilyen szolgáltatás biztosítása, vagy annak elérése érdekében építünk ki és használunk. A szolgáltatásokat pedig szerverek biztosítják. Ha egy otthoni felhasználó a saját számítógépén (pl.: Windows XP, Windows 7) megoszt egy katalógust, akkor ezzel létrejött egy szerver, amely innentől kezdve fájl elérési szolgáltatást biztosít függetlenül attól, hogy ezt bárki, bármikor fogja-e használni. Egy ilyen szerver természetesen nehezen hasonlítható össze egy vállalat adatbázis szerverével, amely óránként akár több mint 1000 kérést válaszol meg.

Miből lesz a cserebogár? Vagy inkább miből lesz a szerver? Legalább három meghatározó komponens van. Fontos:

- a hardver (szerver architektúra),
- az operációs rendszer (szerver operációs rendszer és annak megfelelő hangolása) és végül

- a szolgáltatást nyújtó (szerver) alkalmazás.

A cél (hogy milyen szolgáltatást akarunk nyújtani) adott(pl.: Web szerver, File szerver, FTP szerver, SQL szerver, Mail szerver). Elvárható, hogy az ezt biztosítóalkalmazást (pl.: Web szerver esetében a MsInternet Information Services, vagy az Apache, SQL szerver esetében az Oracle, vagy a Ms SQL Server, vagy a MySQL) a várható terhelésnek megfelelően tervezzék meg, és készítsék el. Nagyon eltérő terhelések esetén (és természetesen pénzügyi megfontolások miatt) az alkalmazásokból többféle változat szokott készülni, eltérő terhelhetőséggel, és eltérő árakkal. A választási szempont lehet például az ár, de lehet a szoftver támogatottsága, meglévő rendszerekkel való kompatibilitása.

A helyes operációs rendszer megválasztásával is érdemes foglalkozni. Ha csak a PC-s OS-eket tekintjük, akkor is létezik annak kliens, illetve szerver változata akár Windows-t:

Ms Windows Server 2003 - Ms Windows XP, vagy

Ms Windows Server 2008 R2 - Ms Windows 7,

akár Linux-ot nézünk:

Red Hat Enterprise Linux Server - Red Hat Enterprise Linux forWorkstations, vagy

SUSE Linux Enterprise Server - SUSE Linux Enterprise Desktop.

Ugyancsak fontos feladat a hardver megfelelő kiválasztása. Nem szerencsés egy kliensnek tervezett számítógépet szerver célokra használni. Néhány könnyen belátható, fontosabb eltérés:

- szerver esetében elvárás a hibavédett ECC memória használata, kliens esetében nem;
- szerver esetében a maximális memória lehet akár 512 GB, kliens esetében 8-16 GB.
- szerver esetében elvárás a működésközben cserélhető redundáns tápegység, kliens esetében nem.

Komolyabb szerverek esetében még jelentősebbek az eltérések: max 4 TB memória, 64 db 8 magos processzor, a memória rendszerbusz terhelhetősége több mint 700 GB/sec, áramfelvétele akár 80 A, súlya közel 2000 kg. Ezek (és még sok más) mind olyan eltérések, amiket már tervezéskor figyelembe kell venni. Egy nagyteljesítményű szerver esetében megszokott, hogy éveken keresztül be van kapcsolva, és egyes karbantartási műveleteket bekapcsolt állapotban végeznek el rajta.

Ennek megfelelően megkülönböztetünk szerver kategóriákat:

- mikroszámítógépek (PC-k): az átlagos elvárásoknak megfelelő felépítésűek, 1-2 db x86-x64 processzor, 2-4 merevlemez, 1 LAN csatlakozó, stb. Pl.: NEC, Dell, Hewlett-Packard, Thinkpad;

- miniszámítógépek (midrange): rack-be szerelhetők, speciális tervezésűek, 4-8 db x86-x64 esetleg már saját tervezésű (SPARC, Itanium) processzor, 8-16 merevlemez, 4-8 LAN csatlakozó, redundáns tápegység, stb., könnyen karbantarthatók, távmenedzselhetők. Pl.: SUN, IBM, Hewlett-Packard;

- nagyszámítógépek (mainframe): (különálló kártyákkal, speciális belső buszokkal, klímával, külön üzemeltető személyzettel). Pl.: SUN, IBM, NEC;

- szuper számítógépek (1-10 megaWatt fogyasztás, fokozott hőtermelés, folyadékhűtés, jellemzően saját tervezésű és gyártmányú processzor kártyák több (32-64) processzorral, tároló alrendszerekkel. Pl.: Cray, Hitachi, IBM. Az IBM egy 2009-ben bemutatott szuperszámítógépének számítási teljesítménye 20 petaflop, ami 2 millió notebook teljesítményével vethető össze.).

Az egyes kategóriákat jellemző adatok megvizsgálása esetén többen esnek abba a hibába, hogy nem tulajdonítanak megfelelő fontosságot az egyes jellemzőknek. Bár a PC-s háttértárolók tárolási kapacitása nagyon jelentős mértékben megnőtt az elmúlt pár évben, valamint olvasási és írási sebességük is, még sem vethetők össze egy komolyabb szervertároló rendszerével még akkor sem, ha egyes adataik megegyeznek. A hétköznapi életből véve egy példát talán még jobban szemlélteti a különbséget, ha egy átlagos gépkocsit hasonlítunk össze egy haszongépjárművel. Míg az előbbi jellemző igénybevétele évi 20-30 ezer km (ennél nagyobb igénybevétel esetén gyors amortizálás várható), addig egy haszongépjármű esetében teljesen elfogadott az évi 100 ezer km terhelés éveken keresztül (pl.: egy haszongépjármű esetében 500.000 km-enként van átfogó műszaki felülvizsgálat).

A következőkben nézzük meg, milyen lépésekből áll egy szerver alapú szolgáltatás telepítése. Előfordulhat, hogy egy géppel több szolgáltatást kell biztosítani, vagy olyan szerveren kell újabb szolgáltatást biztosítani, amelyen már fut egy másik. A kérdés, hogy milyen lépésekből áll a telepítés?

Először tisztázni kell a szolgáltatás biztosításához szükséges erőforrásokat. Célszerű minden komponenst kigyűjteni, akár hardveres (memória, HDD, CPU, stb.), akár szoftveres (operációs rendszer, stb.) elvárás van. A második lépés, hogy ellenőrizzük, nincsenek-e megadva

inkompatibilitási problémák (pl.: ütközés vírusölővel). Lehet, hogy korábbi tapasztalatok, felhasználói visszajelzéseknek köszönhetően már ismertté vált, hogy egy adott verziójú operációs rendszeren nem fut a program, vagy fut ugyan, de nem megbízhatóan. Lehetnek nyelvi beállítástól függő működési problémák is. Esetleg valamilyen szoftver kiegészítő (patch) felinstallálására szükség van. Ugyanezt meg kell tenni (ha van) a többi szolgáltatás esetében is. Külön meg kell vizsgálni, hogy a szolgáltatásokat egy gépen futtatva, egymással nem ütköznek-e. Ha igen, akkor két lehetőség van. Vagy két külön gépre telepítjük az egyes szolgáltatásokat, vagy egy gépre ugyan, de az egyre inkább elterjedt virtualizációs technikának köszönhetően ezen a gépen két (vagy több) virtuális gépet futtatva, az egyes szolgáltatásokat biztosító programokat külön virtuális gépekre telepítjük. Ezt a virtualizációt kétféleképpen is meg lehet valósítani. vagy maga az operációs rendszer támogatja a virtuális gépeket (pl.: Microsoft Hyper-V technológia), vagy külön program segítségével (pl.: VMWare Workstation). Mindkét változatnak vannak előnyei, hátrányai. Virtualizációs megoldást választva az összesített hardver igényekhez hozzá kell adni a választott megoldás saját hardverigényét. Célszerű betartani egy úgynevezett „ökölszabályt”, mely szerint egy hardver komponens nem akkor van leterhelve, ha 100-ig van kihasználva, hanem általában már 75%-os terhelés is elég ehhez. Például a fizikai memória (RAM) 92%-os terheltsége esetén az operációs rendszer speciális műveleteket hajt végre, hogy memóriát szabadítson fel, ami erős lassulással járhat. Összegeztük tehát a hardver igényt, meghatároztuk a szoftveres elvárásokat. Következő lépés egy olyan számítógép kiválasztása, amely teljesíti ezeket.

Érdemes előregondolkozva már olyan kiegészítők betervezése is, amelyek az üzemszerű működtetéshez szükségesek (archiváláshoz valamilyen backup eszköz és szoftver, üzembiztonság fokozásához szünetmentes tápegység, stb.). Ennek során a már meghatározott hardver-szoftver komponenseket figyelembe kell venni, szükség esetén módosítani kell. Ha a meghatározott feltételek biztosításra kerültek (van hardver, megvannak a szoftverek), akkor kezdődhet a telepítés. Első lépésként az operációs rendszert kell felinstallálni. Érdemes előtte végig gondolni, hogy célszerű a háttértárolót felosztani (partícionálni). Kell terület magának az operációs rendszer fájljainak (system), kell az átmeneti állományoknak (temporary), kell az alkalmazásoknak (binaries v. programs), kell a felhasználó(k)nak (users), kell(het) speciális memória műveletekhez (swap). Ezek természetesen kerülhetnek mind egy partícióba, de javasolt elválasztani egymástól, és ha van rá mód, akkor külön merevlemezekre. Így várhatóan kisebb mértékű lesz a fájlrendszer töredezettsége, gyorsabb lesz a működése. Egy esetleges újra telepítés is könnyebb, ha az egyes komponensek szét vannak választva.

Amennyiben sikerült a szervert működésre kész állapotba hozni, a cél az, hogy ezt a működőképességet megőrizzük. A tennivalókat különböző szempontok szerint áttekinteni.

A sikeres telepítés után kezdődhet a felhasználók, csoportok, és az egyes katalógusokhoz, fájlokhoz való hozzáférési engedélyek kialakítása. Ezt célszerű előre megtervezni, és nem ad hoc jelleggel intézni. Felhasználókat jellemzően kétféle módon lehet létrehozni egy rendszerben. Az első módszer szerint nevesített felhasználók vannak, akik különböző, (esetenként változó) feladatokat látnak el. A bejelentkezési nevek ekkor a felhasználó valódi nevéből származódnak valamilyen módon (pl.: Fekete Péter → FeketeP, vagy FPeter, esetleg FeketePeter). A másik módszer szerint a bejelentkezési neveket szerepkörökből vezetjük le (pl.: Tervezési Osztály vezetője→ TervOV). Ekkor a felhasználó adatai közé (megjegyzésként) bekerülhet, hogy ki az, aki jelenleg betölti ezt a szerepkört. A különböző hozzáférési engedélyek kialakításakor azt az elvet érdemes követni, hogy hozzáférési engedélyeket ne az egyes felhasználókhoz rendeljük hozzá, mert a tapasztalatok szerint az azonos feladatot ellátó felhasználók általában ugyanazokhoz a fájlokhoz kell hozzáférjenek, és ugyanolyan hozzáférési engedéllyel. Ezt úgy a legegyszerűbb megvalósítani, hogy csoportokat kell létrehozni, és a hozzáférési engedélyeket a csoportokhoz kell rendelni. Majd azokat a felhasználókat, akik azonos szerepkört töltenek be, azokat beletenni ugyanabba a csoportba. Eleinte szokatlannak tűnhet, de normális jelenség, hogy egy felhasználó bizonyos esetekben több csoportnak is tagja lesz. Ekkor ügyelni kell arra, hogy a különböző csoporttagságok miatt mi lesz a felhasználó végső hozzáférési engedélye. Természetesen ez a választott operációs rendszernek is függvénye.

Utolsó lépés a szükséges szoftverek feltelepítése. Ezt is érdemes megfontoltan tenni. Még a telepítés megkezdése előtt ellenőrizni kell (mint korábban a szolgáltatások esetén), hogy a telepíteni kívánt szoftverek nem ütköznek-e egymással, és hogy működésükhöz milyen előzetes feltételeknek kell teljesülnie. Több esetben még az sem mindegy, hogy a szoftvereket milyen sorrendben telepítik fel. Javasolt a telepítés megkezdése előtt a rendszerről a jelenlegi állapotnak megfelelő mentést készíteni, hogy esetleges problémák esetén vissza lehessen állítani az elmentett állapotot

2.2. Szerverek működtetése

Amennyiben sikerült a szervert működésre kész állapotba hozni, a cél az, hogy ezt a működőképességet megőrizzük. A tennivalókat érdemes két csoportba rendezni: hardveresek és szoftveresek.

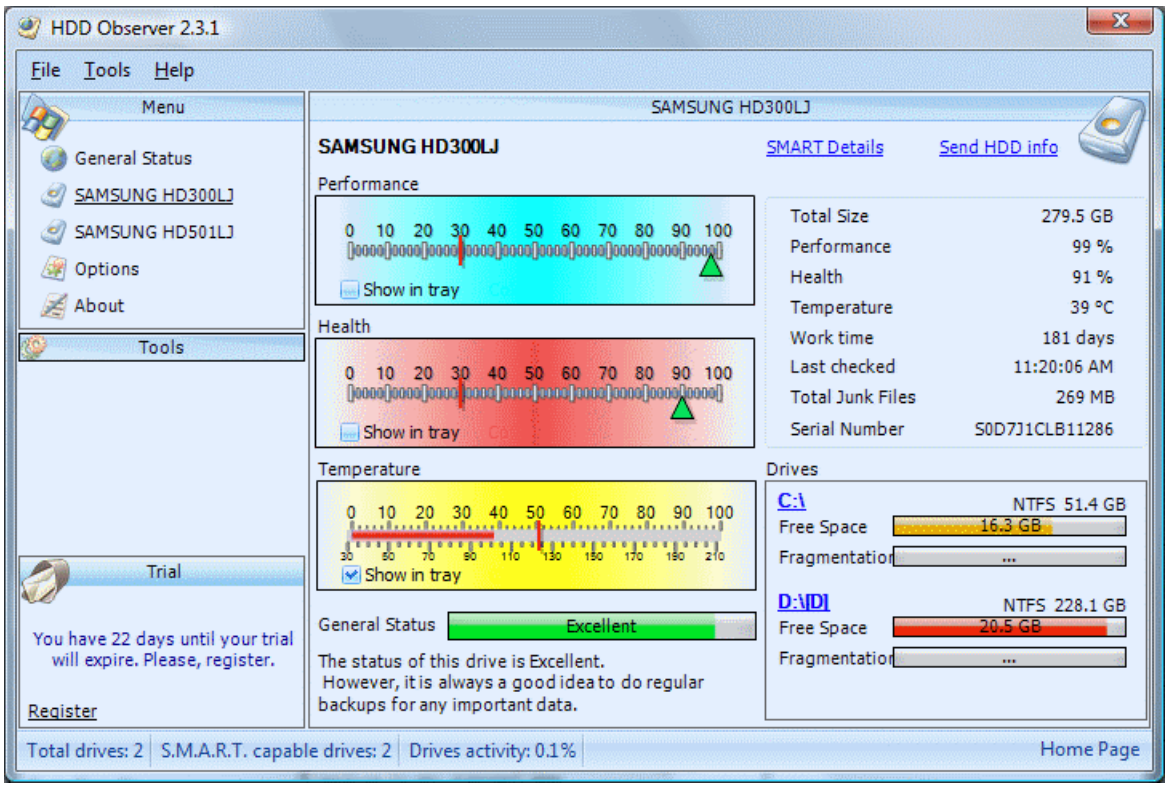
A hardverrel kapcsolatos üzemeltetési kérdések jellemzően két csoportra bonthatók:

- előre tervezhetők (például ilyen a TMK: Tervszerű Megelőző Karbantartás, vagy egy hardverbővítés);
- váratlanok (véletlenszerű elromlás).

Az első esetben komoly segítség, hogy az időpont tervezhető. Akár hétvégére is ütemezhető, amikor a rendszeres napi munkavégzést nem zavarja. Ugyancsak előny, hogy több esetben előzetesen végig lehet próbálni egy tartalék gépen a végrehajtani kívánt műveleteket. Pontosan becsülhető a szükséges időtartam, fel lehet készülni szerelési problémákra, lehet menteni a fontosabb állományokat. Hardver bővítés esetén a tartalék gép szintén nagy segítséget jelent, hiszen előre bele lehet próbálni az új hardvert, és az éles gépről készített mentést a tartalékra feltéve, kiderülhetnek az esetleges inkompatibilitási problémák, vagy a beszerzett hardver gyári hibái. Ebbe a csoportba tartoznak a karbantartási műveletek is (például portalanítás, csapágyak cseréje).

A másik esetet ugyan pontosan nem lehet előre kiszámítani, de a rendszeres mentések segítségével jelentősen csökkenteni lehet az esetleges veszteségeket, és olyan hardverek esetében, amelyeknél monitorozni lehet a működési paramétereket (például a S.M.A.R.T. –(Self-Monitoring, Analysis, and ReportingTechnology)merevlemezek), nem pontosan ugyan, de előre jelezhetők a várható meghibásodások. Azokban az esetekben, amikor hosszabb kiesés nem engedhető meg, eleve lennie kell egy tartalék gépnek, amely vagy folyamatosan működik és besegít az éles gépnek, vagy ott áll a raktárban(és avul). Amikor nem ennyire fontos a folyamatos működés, akkor is lerövidíthető a javítási idő tartalék alkatrészek előzetes beszerzésével.

Szoftverrel kapcsolatos üzemeltetési kérdésekre nehéz általános érvényű, részletes forgatókönyvet megadni. Általában azonban megállja a helyét, hogy az operációs rendszerek, és a komolyabb szoftverek felhasználói könyve tartalmaz olyan utalásokat, hogy adott időközönként milyen rendszergazdai tennivalók vannak. A többi program esetében (ha van rá mód) vásárlás előtt kell ezeket az információkat beszerezni. Amikor az információk már rendelkezésre állnak, akkor ütemtervet kell készíteni, hogy ne maradjon ki semmi. Javasolt erre egy felelős személyt kijelölni.



2.1. ábra: Egy S.M.A.R.T. merevlemez figyelő alkalmazás (HDD Observer)

(KEP_A303_I_02_01) KEP_A303_I_02_01.JPG

2.3. Adatkommunikáció alapelvei

Hálózatot öncélúan nem hoznak létre, mindig valamilyen igény kielégítése a cél. Ez a célok (mozgató rugók) a következők lehetnek:

- erőforrás összevonás, erőforrás megosztás,
- megbízhatóság növelés,
- gazdaságosság növelés,
- speciális szolgáltatás (pl.: kommunikáció).

Míg korábban a költséges erőforrások miatt az erőforrások megosztása volt meghatározó ok, addig mostanában a leggyakoribb motiváció a kommunikáció (elektronikus levelezés, on-line üzenetváltás, IP alapú telefonálás képpel vagy kép nélkül, stb.) biztosítása. Egy meglévő színes lézernyomtató hálózaton keresztül mások számára is hozzáférhető. Ezzel nő az eszköz kihasználtsága, és elegendő belőle kevesebbet (adott esetben csak egyet) venni. Természetesen, ha a hálózat egyszer már kiépült, akkor ugyanazon a hálózaton több szolgáltatást is lehet igényelni illetve biztosítani. Így az adatok több egymástól távol levő helyen történő tárolásával, ha az egyik

helyszínt esetleg katasztrófa éri (árvíz, tájfun, tűz, stb.), akkor a távoli helyen minden adat továbbra is rendelkezésre áll. Nőtt a megbízhatóság. Ha ezeket a tároló helyeket nem ugyanazoknak az adatoknak a tárolására használják, hanem összeadódnak a tárhelyek, akkor erőforrás összevonás történt.

Jelen esetben egymástól távol levő számítógépek összekötése a cél. Általánosságban azt lehet mondani, hogy ezek a számítógépek egymással kétféle kapcsolatban lehetnek. Lehetnek:

- egyenrangúak (peer-to-peer), vagy
- alá-fölérendeltek (kliens-szerver).

A kliens-szerver viszony lehet:

- erősen centralizált, vagy
- gyengén centralizált.

Meghatározó, hogy a hálózat mekkora távolságot fog át. Ennek függvényében a következő kategóriákat szokás használni:

- helyi hálózat (LAN - Local Area Network)
- városi hálózat (MAN - Metropolitan Area Network)
- nagy kiterjedésű hálózat (WAN - Wide Area Network)
- globális hálózat (GAN - Global Area Network)

A fejlődés miatt az egyes kategóriákra jellemző értékek (méret, sebesség, stb.) folytonosan változnak. Általánosságban igaz azonban, hogy a helyi hálózatok tipikusan magán kézben vannak (ez lehet egy vállalat, egy központi szervezet, de akár ténylegesen egy kis iroda is), és nagy átviteli sebességek, kis késleltetési idők jellemzik. Míg a földrészeket összekötő hálózatok nagyobb késleltetésűek. A jellemző sebességek kezdenek egymáshoz közelíteni.

A felhasználók saját számítógépeik előtt ülve különböző programok segítségével használják a hálózatot. A hálózat működtetésében nem vesznek részt, idegen szóval host az elnevezésük. A hálózatot működtető eszközöket kapcsológépeknek, node-nak hívjuk. A node olyan (számító)gép, amely több átviteli vonalhoz kapcsolódik. Feladata az üzenetek irányítása, vagy egyszerűbb esetekben csak továbbítása.

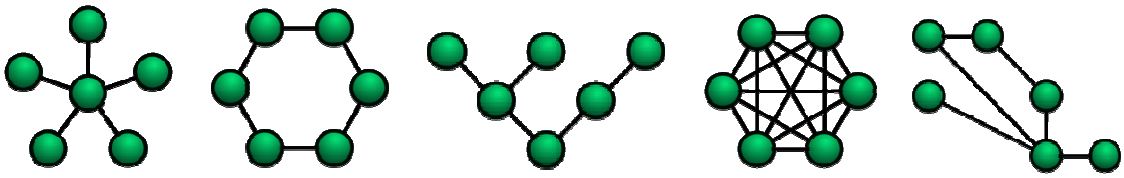
Ahhoz, hogy egy számítógép csatlakozni tudjon a hálózathoz, valamilyen eszközre van szükség. Ez lehet egy hálózati kártya, vagy egy modem. Akár a hálózati kártya, akár a modem, lehet beépítve a

számítógépbe (azaz alaplapi, pl.: a notebook-ok esetében ez a jellemző), és lehet a számítógéphez csatlakoztatható (belső vagy külső csatlakozókon keresztül, pl.: PCI, vagy USB).

A hálózat működésében szerepet vállaló fontosabb eszközök:

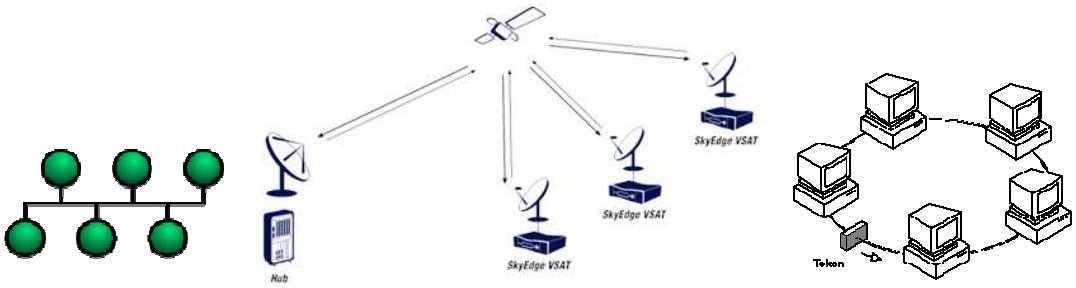
- repeater-ek (jelismétlők)
- bridge-ek (hidak)
- switch-ek (kapcsolók)
- router-ek (forgalomirányítók)
- átjárók (gateway-ek)

A hálózatokat több szempont szerint szokás csoportosítani. Ebből az egyik a hálózat kialakítása (topológiája).



2.2. ábra: Jellemző topológiák pont-pont kapcsolat esetén (KEP_A303_I_02_02)

KEP_A303_I_02_02.JPG



2.3. ábra: Jellemző topológiák üzenetszórás kapcsolat esetén (KEP_A303_I_02_03)

KEP_A303_I_02_03.JPG

Pont-pont kapcsolat esetén mindig két, egymással kapcsolatban levő csomópont kommunikál. Üzenetszórás esetén az üzenetváltás több résztvevő között történik, mint pl.: egy egyetemi előadás esetén. Amikor az oktató előadáson egy diákhoz kérdést intéz, azt egy adott távolságon belül

minden jelenlevő hallani fogja. Természetesen a választ is. Ebből adódik a probléma, ugyanis ha az oktató nem egy adott diákhoz intézi a kérdést, hanem a jelenlevőkhöz, akkor a válasz is több diák felől érkezhet. Ezek a válaszok összemosódnak, érthetetlenek lesznek. Üzenetszórás esetében ezt a problémát meg kell oldani, gondoskodni kell a csatorna kiosztásáról (ki az, aki beszélhet).

A csatorna kiosztása lehet:

- statikus, vagy
- dinamikus, amely lehet
    - centralizált
    - decentralizált

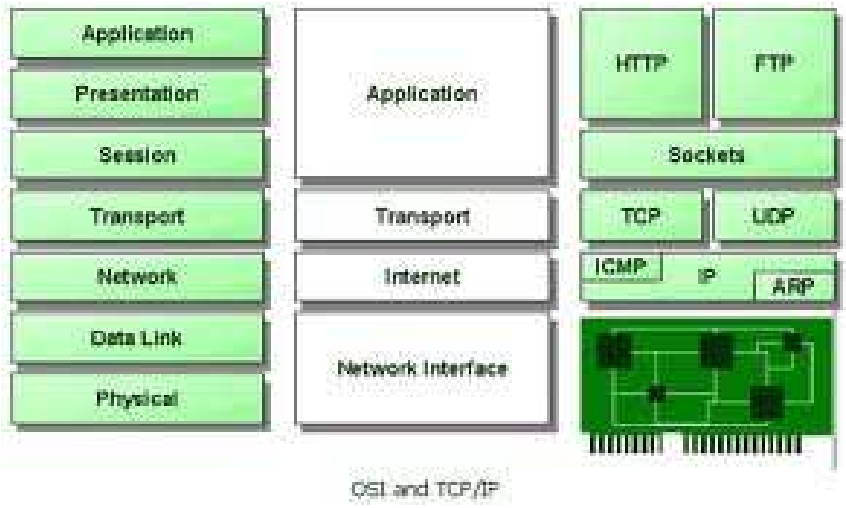
2.4. Rétegzett hálózati architektúrák

A következőkben az alapoktól kezdve, tisztázásra kerülnek hálózati fogalmak, hálózati eszközök, kommunikációs szabályok (protokollok). Mivel a hálózat működése meglehetősen összetett, ezért az egyszerűsítés, az áttekinthetőség, és a könnyebb kezelhetőség érdekében a hálózat megvalósítását rétegekbe (layer) szervezték. A rétegek száma meghatározó. A rétegek kialakításának fontos szempontja, hogy mi lesz a réteg feladata. Ezért a következőket vették figyelembe:

- Az egyes rétegek szolgáltatásokat nyújtanak.
- A szolgáltatást a réteg a közvetlenül felette levő réteg számára nyújtja.
- A szolgáltatást a réteg a közvetlen alatta levő rétegtől igényli.
- A szolgáltatás igénybevételéhez szükséges információkat a rétegek egy felületen (interface) keresztül adják, és a válaszokat is onnan kapják.

A rétegek bevezetésével nem fontos tudni, hogy a szolgáltatás hogy kerül megvalósításra (egy kávéautomatába bedobott pénz, és a gomb megnyomása után senkit nem érdekel, hogy nyílik meg a csap, hogy kerül kiadagolásra a porkeverék, hogy kerül meghatározásra a visszajáró pénz, stb.). A réteg által biztosított funkciókat leegyszerűsítve egy funkcionális elem (entity) nyújtja. A kommunikáció során a kommunikációban résztvevő egyes gépeken különböző módon működő funkcionális elemek lehetnek, de a feladatuk ugyanazon a szinten ugyanaz kell legyen. Ezeket társelemeknek (peerentities) nevezik. A kommunikáció szabályait forgatókönyv (protocol) rögzíti.

A protokoll tehát szabályok halmaza, amely meghatározza például a kommunikáció sebességét, időzítéseket, sorrendiséget.



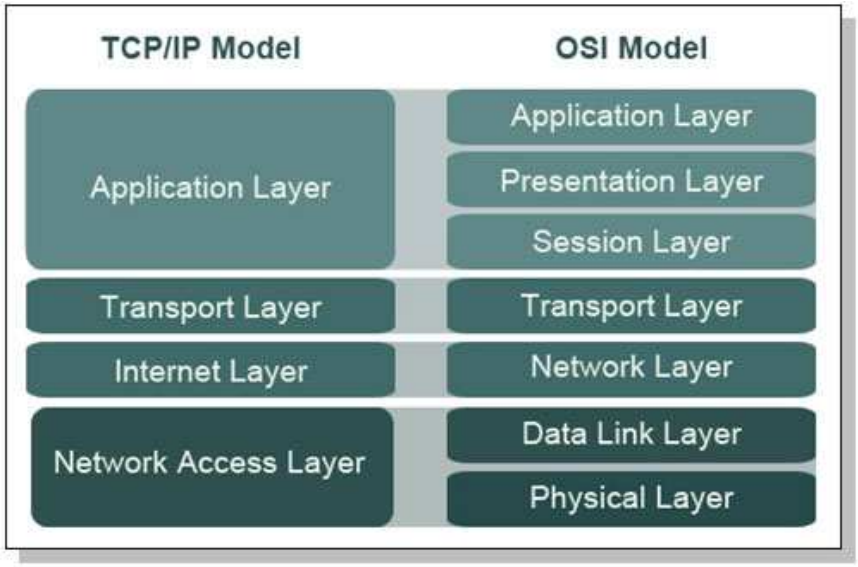
2.4. ábra:Hálózati rétegek (KEP_A303_I_02_04) KEP_A303_I_02_04.JPG

Bár a kommunikáció valójában a rétegek között történik, de az egyes rétegek ezt érzékelhetik úgy, mintha közvetlenül egymással kommunikálnának. Ezt nevezik virtuális kommunikációnak. (Amikor 2 személy egymással telefonon beszélget, akkor bár valójában a telefonkészülék kagylójába beszélnek, és onnan hallják a választ, mégis úgy tekintik, mintha a másik személy ott lenne. A rétegek előnye itt látható először. A telefonkészülék bármikor kicserélhető, egy másikra, például jobb hangminőségűre, vagy vezetékmentes készülékre.).

Belátható, hogyha a feladat adott, de azt sok réteg között osztják el, akkor az egyes rétegekre kevés részfeladat jut. Ha kevés réteget alakítanak ki, akkor a rétegre több feladat jut.

A rétegek kialakításánál szempont lehet a kapcsolat felépítése, lebontása, vagy akár a forgalmazás iránya. Irányát tekintve a forgalom lehet:

- egyirányú (simplex),
- kétirányú, de egyidőben csak egyirányú (half duplex),
- kétirányú (full duplex)



2.5. ábra:Réteg kialakítás DoD illetve OSI ajánlás szerint (KEP_A303_I_02_05)

KEP_A303_I_02_05.JPG

A 2.3. ábrán látható, hogy ugyanarra a feladatra (hálózati kommunikáció) nem csak egy réteg kialakítás létezik. A baloldali részen az amerikai Védelmi Minisztérium (DoD - Department of Defense) ajánlása van, a jobb oldalon pedig a Nemzetközi Szabványügyi Szervezetnek (ISO - International StandardsOrganisation) egy ajánlása, az ún. OSI (OSI - Open System Interconnect . Nyílt rendszerek összekapcsolása) hivatkozási (referencia) modellje látható. Megfigyelhető, hogy bár eltérő a rétegek száma, de ennek ellenére vannak olyan egyes rétegek, amelyek ugyanazt a feladatot látják el (Internet Layer - Network Layer), míg más esetben ugyanazt a feladatot több réteg végzi el (Network Access Layer - PhysicalLayer + Data Link Layer).

A rétegek az információ helyességének ellenőrzésére, illetve feladatuk ellátása érdekében kiegészítő információkat fűzhetnek a felettük levő rétegtől megkapott adatokhoz. Szükség esetén az adatokat kisebb méretűre tördelhetik, sorszámmal látják el, és úgy továbbítják. A célhoz érkezve ezeket a hozzájuk fűzött információ segítségével újra össze kel tudni állítani, és a sértetlenséget (adott esetben) tudni kell ellenőrizni.

A rétegek kialakításának szempontjai:

- jól meghatározott feladatokat hajtsanak végre,
- szimmetrikusak legyenek,
- adott keretek között rugalmasak legyenek,
- teremtsenek szabványokat,

- az interfészen keresztül lehetőleg minél kevesebb információ kerüljön továbbításra.

Alulról felfelé az egyes rétegek szerepe röviden.

Fizikai réteg (PhysicalLayer): az adatok valamilyen fizikai jellemző segítségével (pl.: eltérő feszültség szintek, vagy eltérő fényintenzitás) bitenként kerüljenek átvitelre. Itt kerül meghatározásra (mint interfész) a csatlakozó mérete, a tüskék száma, távolsága, stb.

Adatkapcsolati réteg (Data Link Layer): a bitek összekapcsolásával nagyobb információegység (keret) állítható össze. Ennek ellenőrizhető helyessége, illetve beazonosítható egy kisebb körön belül a címzett. Lehet nyugtát visszaküldeni. "Fizikai" címek vannak csak (a hálózati kártya fizikai címe, az ún. MAC cím).

Hálózati réteg (Network Layer): a keretek összekapcsolásával még nagyobb információegység (csomag) állítható össze. Ebből meghatározható a csomag kézbesítési módja, iránya. Nagyobb távolságra is érvényes (logikai) címet tartalmaz (IP cím).

Szállítási réteg (TransportLayer): többféle hálózati összeköttetés létrehozása. Lehet szolgáltatásként hibamentes, két pont közti csatorna kialakítását igényelni.

Viszony réteg (Session Layer): nevek használata, "párbeszéd" szervezése, szinkronizálás.

Megjelenítési réteg (PresentationLayer): kódrendszerek közti konverzió, tömörítés, titkosítás.

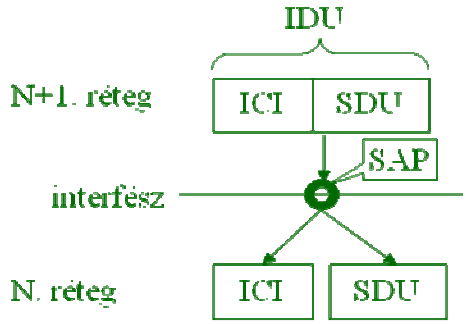
Alkalmazási réteg (ApplicationLayer): magas szintű szolgáltatások biztosítása: pl.: fájl átvitel, elektronikus levelezés, web böngészés. A felhasználó tulajdonképpen itt veszi igénybe a hálózatot tudatosan.

A rétegek kialakításánál tisztázásra került, hogy az egyes rétegek szolgáltatásokat nyújtanak a közvetlen felettük levő réteg számára. Ezt egy interfészen keresztül tehetik meg. Az interfész egy azonban határfelület. A szolgáltatást általában egy ponton keresztül veszik igénybe. Ehhez illeszkedve bevezetésre került egy szolgáltatás elérési pont (SAP - Service Access Point). Minden SAP egyedi azonosítóval rendelkezik.

A 2.4-es ábrán az N+1-ik réteg küldeni akar az N. rétegnek egy adatot (SDU) az interfészen keresztül. Ezért ehhez hozzá teszi az interfészt vezérlő információt (ICU). Ez átkerül a SAP-on keresztül az N. rétegbe. Ott szétválasztódik komponenseire (ICI, SDU). Az N. rétegben ehhez hozzáadódik az N. réteg fejléce. Így abból egy újabb egység keletkezik (PDU). (A rajzon nem látható.) A rövidítések a következőket takarják:

- IDU (Interface Data Unit): interfész adategység. Részei: ICI + SDU
- ICI (InterfaceControllerInformation): interfészt vezérlő információ.
- SDU (Service Data Unit): Szolgálati adat elem a szolgáltatás igénybevételéhez.
- PDU (Protocol Data Unit): protokoll adat elem.

Van összeköttetés alapú (Connection Oriented Service) és összeköttetés mentes (Connectionless Service) szolgálat.



2.6. ábra:A SAP felépítése (KEP_A303_I_02_06) KEP_A303_I_02_06.JPG

Az összeköttetés alapú szolgálat jellemzője, hogy sorrendhelyes kapcsolatot használ (például egy csőbe annak átmérőjével közel megegyező de kisebb golyókat teszünk. A túloldalon ugyanabban a sorrendben kell megérkezniük). Van összeköttetés felépítés, használat, és lebontás. (Másik példa a telefonálás. Nincs szó vagy akár betűcsere.)

Az összeköttetés mentes szolgálat jellemzője, hogy mivel nincs meg az összekötés, ezért az üzeneteket mind el kell látni a célállomás címével. mivel mindegyik egymástól függetlenül kerül továbbításra. Emiatt nem feltétlenül a feladási sorrendben kerülnek kézbesítésre, vagyis az eredeti sorrendet vissza kell állítani. Például a postán ugyanarra a címre több levelet adnak fel. Még az sem garantált, hogy ugyanazon a napon kerülnek kézbesítésre, de megérkezhet akár a feladási sorrendben is.

Mindkettő lehet nyugtázott (megbízható) vagy nyugtázatlan (megbízhatatlan).

A szolgálat igénybevételéhez ún. szolgálat primitíveket (műveleteket) használnak. Az OSI-ban 4 osztály van:

- Kérés (Request): egy funkcionális elem valamilyen tevékenység végrehajtását kéri. (Fentről lefelé irányú.)

- Bejelentés (Indication): egy funkcionális elemet értesíteni kell egy eseményről. (Alulról felfelé irányú.)
- Válasz (Response): egy funkcionális elem válaszolni akar egy eseményre. (Fentről lefelé irányú.)
- Megerősítés (Confirm): egy funkcionális elemet informálni kell a kérésről. (Alulról felfelé irányú.)

Ezek segítségével egy nyugtázás nélküli szolgálat lépései:

- kérés
- bejelentés,

A nyugtázott szolgálat lépései:

- kérés
- bejelentés
- válasz
- megerősítés

A kapcsolat felépítés mindig nyugtázott szolgálat (tárcsázás - telefoncsörgés, bejelentés - csörgés, felveszik - válasz, megerősítés - abba maradt a csörgés).

A kommunikáció lehet nyugtázott ("holnap várlak ebédre" - "azt mondtad, holnap ebédre?"), illetve nyugtázatlan (amikor az egyik csak beszél-beszél-beszél...).

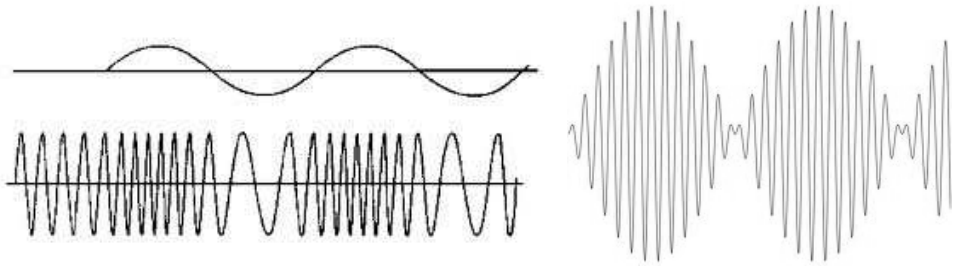
2.5. Rétegzett hálózati architektúrák

Meghatározó az adatátvitelre használt közeg. Ez sokféle lehet: fémes vezető, üvegszál, rádióhullám, stb. A közegtől függően szintén többféle megoldás létezik. Fémes vezető esetén lehet árnyékolatlan csavart érpár, árnyékolt csavart érpár, árnyékolatlan koaxiális kábel, árnyékolt koaxiális kábel, stb.

Az átviteli mód lehet:

- alapsávú (egyetlen fizikai jellemző módosításával): például feszültség szintek (kétféle, négyféle).

- szélessávú (az információt valamilyen vivőre ültetik rá, és annak több jellemzőjét módosítják): például szinuszos vivőhullám, és módosítják a frekvenciát (FM - FrequencyModulation), az amplitúdót (AM - AmplitudeModulation), esetleg a fázist (9. ábra).



2.7. ábra: FM és AM modulált szinuszos jelek (KEP_A303_I_02_07) KEP_A303_I_02_07.JPG

Ha több jellemzőt egyszerre módosítanak, azok egyidejűleg több információ átvitelét teszik lehetővé. Például két frekvencia (10 Hz illetve 20 Hz), és négy amplitúdó érték (1 V, 2 V, 3 V, 4 V) esetén ez 2 * 4 = 8 jelzés (000, 001, 010, 011, 100, 101, 110, 111). Egy 20 Hz-es 3 V-os jel egyértelműen meghatároz 1 lehetséges esetet (például: 111).

A csatorna jellemzője az adatátviteli sebessége (adatmennyiség / idő, például 10 bit/sec), sávszélessége (legmagasabb és legalacsonyabb átvitt frekvenciák különbsége, például 5 MHz), jelzési sebessége (átvitt jelzések száma / átviteli idő, például 200 baud).

A Nyquist tétel:

Ha tetszőleges jelet H sávszélességű aluláteresztő szűrőn engedünk át, akkor a szűrt jelből másodpercenként 2H-szor mintát véve az eredeti jel teljesen helyreállítható. Ebből meghatározható a maximális adatátviteli sebesség:

$$\text{Maximális adatátviteli sebesség} = 2 * H * \log_2 V$$

ahol:

H: a csatorna sávszélessége

V: a jel diszkrét értékeinek száma (jelzések száma)

Zajtalan 3 kHz-es csatorna esetén bináris (kétféle) jelek esetén 3.000 * 2 = 6.000 bit/sec (bps) a maximális adatátviteli sebesség.

Zajos csatorna esetén Nyquist törvény nem használható. Shannon azonban 1948-ban meghatározta a zajjal terhelt csatorna maximális adatátviteli sebességét:

$$\text{Maximális adatátviteli sebesség} = H * \log_2(1 + S/N)$$

ahol:

H: a csatorna sávszélessége

S/N: a jel-zaj viszony (S - Signal, N - Noise)

A jel-zaj viszonyt decibelben szokták megadni. Ezt vissza kell számolni a következő képletből:

$$S/N_{dB} = 10 \log_{10} S/N$$

Ha tehát a csatorna sávszélessége 3 kHz (3.000 Hz), és S/N = 30 dB ($10^{(30/10)} = 1000$), akkor $3000 * \log_2(1 + 1000) = 3000 * 9.967 \approx 30.000$ bps, azaz 30kbps.

A Shannon korlát

Zajos, sávkorlátozott csatornán a maximális adatátviteli sebesség független a jelzések számától, a mintavételezési gyakoriságtól! A gyakorlatban ennek megközelítése is nehéz. Az előző példában kapott 30 kbps a gyakorlatban 9600 bps.

2.6. Hálózati eszközök áttekintése

A most következő felsorolás egyre intelligensebb eszközöket tárgyal. Az egymás után következő eszközök rendelkeznek az előttük levő eszközök tudásával, és kiegészítik azt egy újabb tudással.

Repeater: az információt hordozó jelek tehát valamilyen fizikai közegen keresztül kerülnek továbbításra. Minden közeg esetében számolni kell azzal, hogy a jel a távolság függvényében veszít erősségéből. Egy adott távolság meghaladásakor már nem lesz megkülönböztethetők a jelszintek. Mielőtt ez bekövetkezik, a jeleket meg kell tisztítani a rájuk rakódott zavaroktól, majd fel kell erősíteni. Ezt a feladatot látja el egy repeater (jelismétlő). A repeater a megkapott jeleket nem tudja értelmezni, csak a jelszinten dolgozza azt fel. A hálózatok egyéb jellemzői miatt nem lehet akárhány jelismétlőt egymás után tenni (olyan lenne, mint egy rettenetesen hosszú folyosó, amelyen bárki bárhol beszél, mindenki hallja). Ekkor, ha két (akár egymás mellett levő) személy beszél, a

jelerősítés miatt senki más nem tudna beszélni az egész folyosón, mert nem értenék egymást. A folyosót tehát szelektálni kell.

Bridge: ezt a feladatot egy olyan eszközre bízzák, amelyik már nem csak fizikai szinten képes a jeleket feldolgozni, hanem képes azok tartalmát is megvizsgálni. Ki a feladó, és ki a címzett. A bridge(híd) két hálózati kártyával rendelkezik. Mindegyikkel egy-egy alhálózathoz (folyosó szakaszhoz) csatlakozik. Ha a feladó és a címzett is a bridge azonos oldalán van, akkor a bridge nem ismétli meg az információt a másik oldalra (feleslegesen), ezért a másik oldalonaz elsőtől függetlenül lehet kommunikálni. Ehhez azonban már ismernie kell valamilyen címzési rendszert. Ezen a szinten a hálózati kártya egyedi fizikai címét (MAC cím: Media Access Control) használják. Az ugyanis (jogosan) elvárható, hogy egymáshoz közel levő eszközök valamilyen módon ismerjék egymás fizikai címét (egy vállalaton belül ismerik egymás telefonmellékét, vagy meg tudják azt kérdezni). Ez egy protokoll segítségével kerül meghatározásra (ARP –AddressResolutionProtocol – Címfeloldó protokoll).

Router: nagyobb távolság esetén ez a címzés nem használható. Új, úgynevezett logikai cím került kialakításra. Ez az IP cím. Az az eszköz, amely IP címeket használ az információk feldolgozásához, az a router. A router (forgalomirányító) jellemzően nem két, hanem 3 vagy több hálózati kártyával rendelkezik. A hozzá beérkezett információt feldolgozva, eldönti, hogy azt melyik másik hálózati kártyáján kell továbbítania. A döntést valamilyen szabály rendszernek megfelelően (forgalomirányítási algoritmus) hozza meg. Sok algoritmus létezik.

Gateway: több esetben olyan számítógépek kommunikálnak egymással, amelyen nem azonos „nyelven” beszélnek. Ekkor szükség van egy olyan eszközre, amely a fordítást elvégzi. Ez a gateway, vagy átjáró. Szaknyelven ezt úgy mondják, hogy protokoll konverzió.

Nem esett szó a hub-okról és a switch-ekről. Működésüket tekintve a már felsorolt eszközök valamelyikének általában megfeleltethetők.

2.7. Azonosítási mechanizmusok

Természetesen többféle azonosítási cél létezik. Sok esetben lehet szükség egy számítógép beazonosítására például hálózaton. Ilyen esetben olyan azonosítót kell keresni, amely nem, vagy csak nehezen módosítható, és a kívánt távolságból le is kérdezhető. A hálózati kártyáról korábban volt szó, és ott tisztázásra került a kártya fizikai címe (MAC cím). Ezzel az a probléma, hogy a hálózati forgalmazás során ugyan az információ egységekbe (keretekbe) belekerül a célszámítógép

hálózati kártyájának a MAC címe, de amennyiben a feladó és a célszámítógép nem azonos alhálózaton vannak (például szükség vanmondjuk egyrouter-re a továbbításhoz), akkor a routerhálózati kártyájának a MAC címe fog belekerülni a keretbe célként megadva, hiszen a feladó szempontjából az alhálózaton a router lesz a célállomás. Egy réteggel magasabban ugyan már látszik az IP címből, hogy a router csak egy közbenső továbbító, de a feladó nem fér hozzá (hagyományos módon) a valódi célállomás MAC címéhez. Ugyanígy a feladó MAC címe sem fog eljutni a valódi célállomáshoz, hiszen azt a hozzá legközelebb levő router fogja neki elküldeni, vagyis a valódi célállomás annak a router-nek a MAC címét fogja forrásként megkapni. Ilyen módon tehát a (nem azonos alhálózaton levő) feladó és a célállomás egymás MAC címéhez nem fér hozzá, egymást az alapján azonosítani nem tudják.

Sokkal inkább jellemző a felhasználók azonosítása. Erre szükség van egy adott számítógépre történő helyi vagy távoli bejelentkezéskor (login folyamat), szükség van bizonyos szolgáltatások eléréséhez (például FTP szerverhez történő kapcsolódás, SQL szerverhez történő csatlakozás), illetve szükség van jogosultságok ellenőrzéséhez (például hozzáférési engedélyekkel védett adatbázisok).

Felhasználók azonosítása alapvetően három féle módon történhet:

- tudás alapú
- birtok alapú
- biometriai alapú

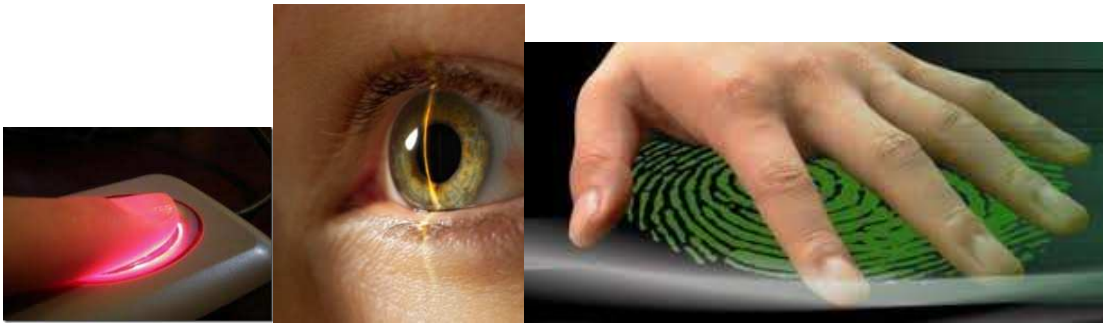
Ha a három módszerből legalább kettőt egyszerre megkövetelünk, akkor azt szigorú azonosításnak nevezzük. Nézzük az egyes azonosítási módokat.

A tudás alapú azonosítás az egyik legelterjedtebb módszer. Amikor egy bank ügyfele a pénzkiadó automatából pénzt akar felvenni, meg kell adni a bankkártyájának a PIN kódját. Amikor egy felhasználó be akar jelentkezni a számítógépére, akkor az operációs rendszer nevet és jelszót kér tőle. A mobil telefonba be kell tenni a szolgáltató SIM kártyáját. A telefon bekapcsolásakor (beállítástól függően) meg kell adni először a telefon kódját, majd sikeres esetben a SIM kártya PIN kódját. A vállalatoknál található telefonközpontok a felhasználók telefonálási költségeinek figyelésére szintén kérhetnek azonosító számsorozatot. Ezek mind tudás alapú azonosító eljárások.

Második módszer a birtok alapú azonosítás. Az egyik leginkább elterjedt módszer a későbbiek során részletesebben ismertetésre kerülő az úgynevezett RFID (RadioFrequencyIDentification). Ez bizonyos feltételek teljesülése esetén lesugározza saját azonosítóját. Használják parkolóba történő

beléptetéshez, laborok, szobák ajtajának nyitására. Egy másik, szintén igen elterjedt birtok alapú azonosítási módszer a korábban már említett bankkártya. Hiszen nem elegendő tudni a PIN kódot, előtte a bankkártyát be kell tenni az automata kártyaolvasójába. Már most látható, hogy mivel ez két azonosítási módszerhez is kötődik, és egy időben kell a kettőt megadni (nem lehet a PIN kódot megadni, majd másnap visszamenve betenni a kártyát az olvasóba), ezért ez szigorú azonosítási módszer. Ugyanígy a mobilszolgáltatók SIM kártyája is szigorú azonosítást kér.

A harmadik csoport, a biometriai azonosítás, igen sokféle lehet (11. ábra). Kezdve a hétköznapi életben is igen régóta használt ujjlenyomat (fingerprint) használatától, az irisz (retina) egyedi jellemzőivel folytatva, a hangfelismerésen (nem beszédfelismerés!) át, egészen a kéz érhálózatáig. De például mostanában már notebook-okba is beépítésre kerül az arcfelismerés, amivel korábban széles körben nem igazán foglalkoztak.



2.8. ábra: Ujjlenyomat, retina és tenyérlenyomat vizsgálat (KEP_A303_I_02_08)

KEP_A303_I_02_08.JPG

Vonalkód technológia

A vonalkód (barcode) használata az 1970-es évek közepére nyúlik vissza. Első alkalmazása egy áruházban történt. Egy jellemzően fehér felületen különböző vastagságú fekete vonalakat lehet látni egymás mellett.

2.9. ábra:Vonalkód (KEP_A303_I_02_09) KEP_A303_I_02_09.JPG

Optikai úton történő leolvasásakor fénnyel megvilágítják a felületet, és a fekete vonalak közötti fehér területről a fény visszaverődik, ezáltal detektálható lesz. Professzionális olvasók esetén tükrökkel megsokszorozott kis teljesítményű lézerfénnyel történik az olvasás. Ennek köszönhetően több pozícióban is lehetséges a leolvasás (nagyobb forgalmú pénztárak esetében tipikusan elterjedt eszköz).

Egy jellemző működési elv a következő: egy számítógépes rendszerbe beviszik az egyes vonalkódokat, majd társítják több, különböző kiegészítő információval (termék megnevezése, darabszáma, ára, gyártó, szállító, stb.).A (például pénztárnál elhelyezett) leolvasóval megállapítják a termék vonalkódját, majd az adatbázisból kiolvassák a társított információkat. Csökkentik a darabszámot, hiszen a vevő elvitt egyet, az árát hozzáadják egy gyűjtőhöz (ez lesz majd a végösszeg), majd a termék megnevezését és árát rányomtatják a blokkra. Eközben azonban kiegészítő információk is gyűjthetők, mint a vásárlás időpontja, a vevő (ha megadja előtte) irányítószáma, a többi termék adata. Később ezeket feldolgozva, megállapítható, általában mikor vesznek édességeket, az egyes városrészekben élők miket vesznek, vagy az egyes termékeket milyen másokkal együtt veszik meg. Így lehet akciókat tervezni célzottan csak egy termékcsoportra, vagy egy városrészre.

Vonalkód technika természetesen még sok más célra is alkalmas. Például cégen belüli leltározásra, gyártás közben az egyes termékek nyomon követésére, stb.

Többféle vonalkód rendszer létezik, amelyeket különböző célokra használnak. Ilyen például: UPC-A, UPC-E, EAN 8. EAN 13, I2OF5, Kód 39, Kód 128, RSS, Data Matrix, QR code, stb.

RF technológia

Az első rádiófrekvenciás (mai szóhasználattal már RFID-nek nevezhető) elven alapuló azonosításra szolgáló technológiát a II. világháborúban Sir Robert Alexander Watton fedezte fel. Véletlenszerű volt a felfedezése annak a ténynek, hogyha egy pilóta himbáló mozgást végez a repülőgéppel,akkor megváltozik a visszavert rádióhullámok alakja. Ekkor a radar képernyőjén megkülönböztethetővé válik a saját és az ellenséges gép. Ez tekinthető a legelső passzív RFID rendszernek: Erre az elvre építve Watton vezetésével kifejlesztésre került az első aktív repülőgép felismerő rendszer, az IFF.

Az IFF rövidítés azóta egy gyűjtőfogalommá, technológiák összességévé nőtte ki magát. Alapelve szerint minden azonosítandó eszköz egy készüléket visz magával, ami a földi állomás által sugárzott jeleket észlelve egy másik, egyedi jelet sugároz vissza, így ez alapján a földi állomás azonosítani tudja azt.

A kereskedelmi alkalmazások az 1960-as évek elején indultak. Az azóta is vezető pozícióbanlevőSensormatic nevű cég élenjárt az RFID megoldások kifejlesztésének területén. Az EAS néven megjelent áruvédelmi lopás gátló rendszer napjainkban is széles körben alkalmazott technológia. A rendszerek 1 bites tag-eket használtak. Ennek megfelelően csak két állapot megkülönböztetésére voltak alkalmasak: a tag megléte, vagy meg nem léte volt megkülönböztethető. Előnye olcsóságából és könnyű használhatóságából adódott. A mikrohullámú, vagy induktív csatolás alapján működő EAS rendszerek vezettek az RFID széles körű elterjedéséhez.

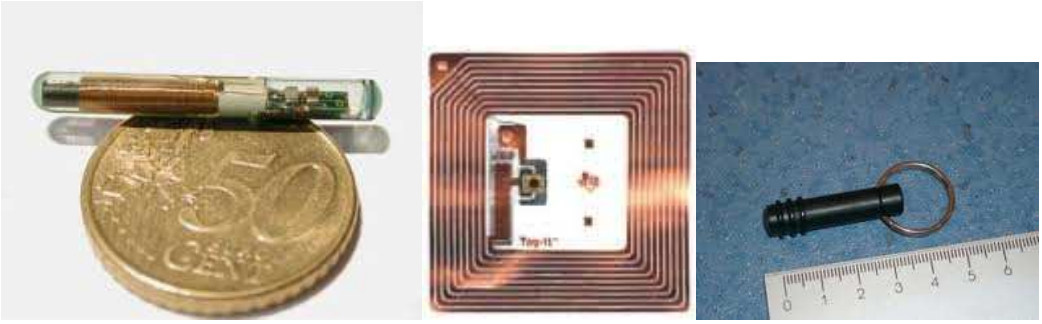
A 70-es években komoly fejlesztések folytak Amerikában, és Európában egyaránt. Elsősorban állatok nyomon követésére készültek alkalmazások, de sok megoldás született jármű- és gyártási folyamatok nyomon követésére is. A gazdák körében népszerű állataik nyomon követése RFID segítségével. Nukleáris eszközök nyomon követésére is kifejlesztésre került egy rendszer a Los Alamos-ikutatóintézetben ezekben az években.

A fejlesztésekbe, kutatásokba egyetemek és cégek is bekapcsolódtak. Az első USA-beli RFID szabadalom Mario W. Cardullonevéhez fűződik, aki 1973. januárban védte le az aktív RFID tag-et, amely újraírható memóriával rendelkezett. Ugyanebben az évben kapta meg Charles Walton találmánya, a passzív transzponder aszabadalmat. Ezzel zárt ajtót lehetett kinyitni, hagyományos kulcs használata nélkül. Ekkor még passzív, 125 kHz-en (LowFrequency - LF) adó RFID transzpondereket használtak. Az olvasó által kibocsátott rádióhullámota transzponder modulálva verte vissza. Ezt a technológiát jelenleg is használják a világon. Idővel a 125 kHz-ről áttértek a 13,56 MHz-es sávra (HighFrequency - HF), ami az egész világonszabad frekvenciasav volt. A nagyobb frekvencia lehetővé tette a nagyobb olvasási távolságot és a gyorsabb adatátvitelt is. A HF rendszerek használataelsősorban Európában terjedt el, főként az újrafelhasználható konténerek es más vagyontárgyak nyomkövetésére. Napjainkban a 13,56 MHz-es RFID rendszereket beléptető, díjfizető, éssmartcard rendszereknél használják.

A rádiós azonosításhoz legalább két eszközre van szükség: egy azonosítandó, valamint egy azonosító berendezésre. Az azonosító valamilyen adatkapcsolatot kezdeményez az azonosítandóval, mely során egyik, vagy mindkét irányba adatátvitel történik. A kommunikáció rádiófrekvencián

zajlik. Mindkét eszköznek tehát rádiós interfésszel is kellrendelkeznie. Egy alap RFID rendszer e szerint minimum két komponensből áll:

- a transzponderből, mely az azonosítani kívánt objektumhoz kötődik (esetleg az objektumban – állat, kutya, marha) helyezkedik el;
- az olvasóból, mely olvasni és/vagy írni is képes a transzpondert.



2.10. ábra: RFID transponder-ek (KEP_A303_I_02_10) KEP_A303_I_02_10.JPG

A fenti rendszer – alkalmazástól függően – kiegészülhet vezérlő számítógéppel, mely esetlegesen több olvasó összehangolt munkáját vezérli, valamint összeköttetést teremt az olvasók és a számítógépen tárolt adatbázisok között. Minden eszközt, mely az olvasó és a végső alkalmazás között helyezkedik el, middleware-nek nevezünk. Az adatbázisok az olvasók zónáiban tartózkodótranszponderekről tárolhatnak információkat. Nem csak lekérdezhetőek, hanem program segítségével írhatók is.

Az olvasó célja rádiós kapcsolat létesítése a transzponderrel, annak azonosítása, valamint adatkapcsolat létrehozása, fenntartása, lezárása az olvasóval, mely során az olvasó és transzponder között információ cserélődik ki. Ennek megfelelően az olvasó mindig tartalmaz egy RF modult (adó-vevő), egy vezérlő egységet, valamint egy csatolóelemet. Sok olvasó egyéb interfészeket is tartalmaz (RS 232, Wi-Fi, USB, stb.) más rendszerekkel való együttműködés céljából. A legtöbb olvasó belső antennával működik, de a drágábbakban külső antennák csatlakoztatására alkalmas port-okat is találunk.

Nagyon sok gyártó létezik, akik az RFID eszközök rengeteg változatát állítják elő. Érdemes ezeket főbb tulajdonságaik alapján csoportokba sorolni.Legelőször is meg kell különböztetnünk a Full Duplex (FDX), a Half Duplex (HDX), valamint a szekvenciális (SEQ) rendszereket. Az FDX, és a HDX rendszerekben atranszponder válasza akkor kerül továbbításra (broadcast), ha az olvasó

RF mezője aktív. Mivel az olvasó által sugárzott jel erősségéhez képest a transzponderé rendkívül gyenge, csakmegfelelő eljárások alkalmazásával lehet azt az olvasóétól elválasztani.

Az RFID transzponderek adattároló kapacitása erásen változó, a néhány byte-tóla néhány KB-ig terjed, bár léteznek 1 bites transzponderek is speciális alkalmazások számára. Az 1 bit éppen elégarra, hogy az olvasótudja, van-e transzponder a zónában, vagy nincs. Mivel ezek atag-ek mikrochip-et sem igényelnek, előállításuk rendkívül egyszerű es olcsó. Előszeretettel használják emiatt lopásjelző berendezésekben. Ha valaki a tag birtokában megpróbálja elhagyni az áruházat (pontosabban a kijáratnál elhelyezett érzékelő kaput), az olvasó érzékeli a transzpondert a zónában, és rögtön jelzést küld. Fizetéskor a tag eltávolításra, vagy deaktiválásra kerül.

A transzponderbe történő írás módja alapján is osztályozhatók az RFID rendszerek. Alegegyszerűbb chipek a gyártás során kerülnek felprogramozásra (általában egyszerű sorozatszámmal), mely a későbbiekben nem módosítható. Az írhatótranszponderek tartalmát ezzel ellentétben az olvasó(író) bármikor megváltoztathatja.

A passzív címkék nem tartalmaznak saját energiaforrást. Az olvasó által kibocsátott rádiófrekvenciás jel elegendő áramot indukál az antennában ahhoz, hogy a lapra épített apró CMOSICfeléledjen, és választ küldjön az adatkérésre. Az antenna tehát speciális tervezést igényel: nem elég, hogy összegyűjti a szükséges energiát, a válaszjelet is közvetítenie kell. A válaszjel általában egy egyedi azonosítószám, de előfordul, hogy a címke egy kis méretűmemóriát (EEPROM) is tartalmaz, és lekérdezéskor ennek tartalmát is továbbítja az olvasó felé. A passzív lapkák rendkívül aprók, 2005-ben a 0.4x0.4 milliméteres felületű, papírnál is vékonyabb címke a kereskedelemben kapható legkisebb darab. A passzív lapok hatótávolsága 2 millimétertől néhány méterig terjed, azaz ekkora távolságból olvasható ki a tartalmuk a használt frekvenciától függően. Alacsony előállítási költségének köszönhetően jelenleg ez a legelterjedtebb típus.

A fél-passzív azonosítók annyiban térnek el a passzív társaiktól, hogy tartalmaznak egy apró, beépített elemet. Ez lehetővé teszi, hogy az IC folyamatosan üzemeljen. Nincs szükség az antenna energiagyűjtő kialakítására, ezért azt adásra optimalizálják. Ennek köszönhető, hogy az ilyen típusú azonosítók válaszideje jobb, és az olvasási hibák aránya kisebb, mint passzív társaik esetén.

Az aktív RFID címkék vagy jeladók beépített energiaforrással rendelkeznek, melyek elegendő energiát biztosítanak bármilyen IC üzemeltetéséhez és a jeladáshoz is. Nagyobb hatótávolságot (akár 10 méter) és memóriakapacitást biztosíthatnak passzív változatuknál, némelyik még a vevő által küldött adatok rögzítésére is képes. Néhány aktív címke impulzusszerűen üzemel, hogy

takarékoskodjon az energiával. A jelenleg kapható legkisebb aktív RFID jelző nagyjából fémpénz méretű.

Az RFID rendszerek működési frekvenciájuk alapján is megkülönböztethetők. Aműködési frekvencia alatt az olvasó által adott jel vivőfrekvenciáját értjük. A transzponderadási frekvenciája általában nem lényeges, az esetek többségében az olvasóéval megegyezik.

- LF (30-300 kHz),
- HF (3-30 MHz),
- UHF (0,3-3 GHz),
- valamint mikrohullám (>3GHz).

3. INFORMATIKAI RENDSZEREK ADATBÁZIS KEZELÉSE

3.1. Adatkezelés szintjei

A vállalati információs rendszerek egyik alapvető sajátossága, hogy működése egy megfelelő szerkezetű és tartalmú adatrendszeren alapszik. Gondoljunk például egy raktárnyilvántartó rendszerre. Ebben a rendszerben adatként tartjuk nyílván többek között a vállalat raktárkészletét, a raktár felépítését, a raktárhoz kapcsolódó mozgásokat és a raktárban dolgozó személyeket. A rendszerben tárolt adatok jelenítik meg a vállalat állapotát, az adatokon keresztül láthatjuk és ellenőrizhetjük a vállalat működését. Az adatok aktualitásának megőrzéséhez az üzleti folyamatokban végbement műveleteket az adatok szintjére le kell képezni az információs rendszer segítségével. A számítógépes információs rendszer nagy előnye, hogy hatékonyan és gyorsan vissza tudunk keresni információt az adatrendszerből.

Az adatkezelés megvalósítása során a kezelő programnak igen sok nehézséget kell megoldania. Ezen nehézségekből most néhányat kiemelünk a problémák megvilágításra.

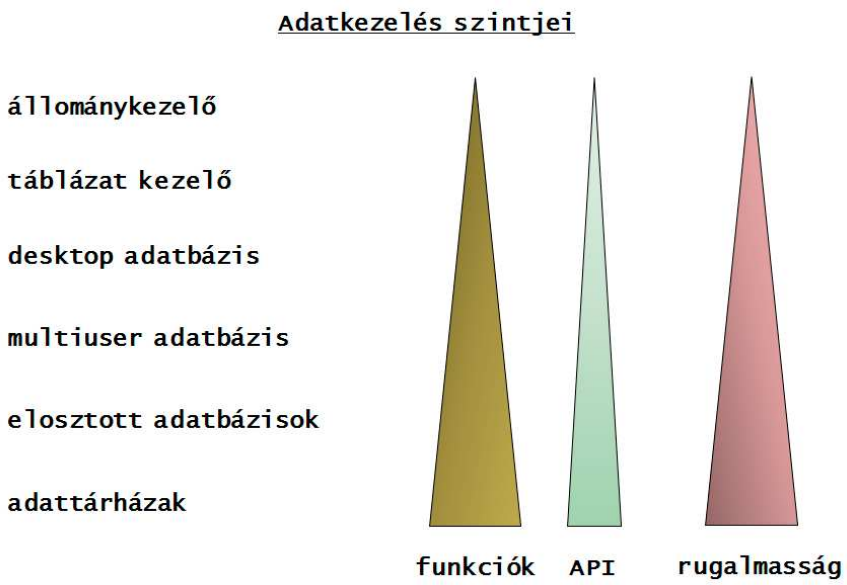
- Az adatkezelés egyik fontos jellemzője, hogy a számítógépi programrendszereknek egyre növekvő adatmennyiséggel kell megbirkózniuk. A növekvő adatmennyiség elsődlegesen nemcsak a tárolásnál jelent gondot, hanem az adatok hatékony kezelésénél, hiszen sokkal lassabban lehet nagyobb adatmennyiségben keresni, mint kis adatmennyiségben.
- Az információ feldolgozására készített számítógépes programoknál az adatok különböző strukturáltságban kerülnek letárolásra. Az adatok lehetnek lazább szerkezetben, vagy szigorúbb, finomabb struktúrában letárolva. A rendszernek tehát fel kell készülnie a különböző szerkezetek kezelésére.
- Az adatoknak helyesnek kell lenniük, azaz tükrözniük kell az üzleti életben érvényes állapotokat és megkötöttségeket. Emiatt az adatkezelés szintjén is biztosítani kell olyan szabályok alkalmazását, amelyek a felvehető adatértékek korlátozásán keresztül érvényesíti az üzleti megkötéseket.
- Az adatokat megfelelő védelmi mechanizmussal kell őrizni az illetéktelen felhasználás megakadályozására.

Mindezen összetett feladatok hatékony ellátására napjainkban általános és speciális célú adatkezelő és adatbázis kezelő rendszerek állnak rendelkezésre.

Az adatkezelés a feladat méretétől és jellegétől függően több különböző szinten valósulhat meg.

A legegyszerűbb esetben a szokásos közvetlen állománykezelés használható. Előnye, hogy rugalmas, az aktuális egyedi igényekhez igazodik. Hátránya viszont az, hogy csak igen korlázotott szolgáltatást biztosít, az igényelt elemeket saját program megírásával lehet megoldani.

A következő szint a táblázatkezelők szintje, amiben egy tipikus feladatcsoportra már készen állnak a megvalósított funkciók. Ezen megoldás korlátját az jelenti, hogy csak kis adatméretet tud kezelni és nem biztosított a konkurrens hozzáférés megvalósítása sem.



3.1. ábra Adatkezelés szintjei (KEP_A303_I_03_01) KEP_A303_I_03_01.JPG

Ezen igényeket az adatbázis kezelő rendszerek tudják támogatni. Az adatbázisokban centralizáltan tárolódnak a kapcsolódó adatok. Az adatbázisoknál is több egymásra épülő szint létezik. Az egyszerűbb, személyi adatbázisoktól kezdve igen széles a kínálat, vannak elosztott adatbázisok is. Az adatbázisok adatbázisának tekinthetők az adattárházak, melyek napjaink méretben vezető adattároló redszerei a palettán.

A számítógép használata során több adatformátummal is találkozhattunk már. Hiszen egy levél, e-mail adatformátuma szemmel láthatóan eltér egy kép adatformátumától. Az eltérés azonban nemcsak a látott formátumban mutatkozik meg, hanem a mögöttes tárolási formátumban is. Egy szám, mint például 23, igen sokféleképen lehet letárolva az adatkezelő rendszer szintjén. A következőkben egy áttekintést kaphatunk a fontosabb formátumtípusokról:

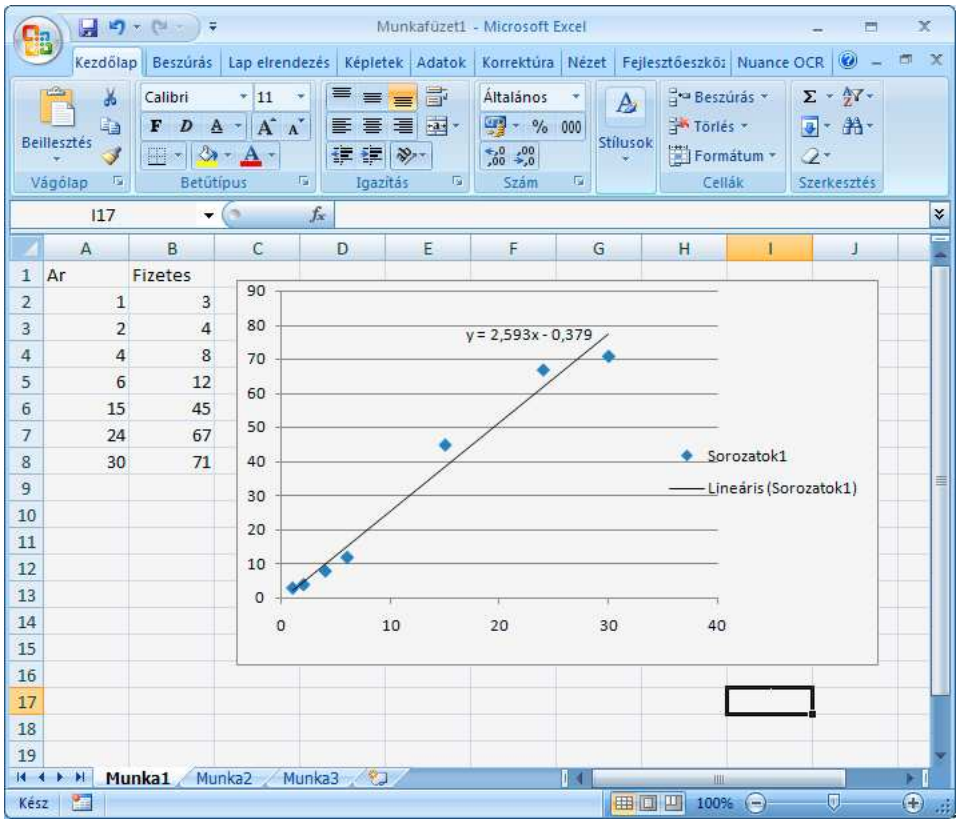
- karakterszintű tárolás: a tárolt adatformátum és a látott adatformátum megegyezik, a tárolón az egyes látott karakterek tárolódnak le. Természetesen ezen karakterek valamilyen karakterkódolási rendszeren keresztül kapják meg a tárolási formátumot.
- kódolt tárolás: a szöveget előbb egy titkosítási vagy tömörítési algoritmus segítségével átalakítjuk, s a kapott szöveget tároljuk le.
- szövegszerű tárolás: a letárolás az emberi szabadszöveget közvetlenül tárolja le. Előnye, hogy az emberi felhasználó számára igen sok hasznos információt közvetít. Mivel a szabadszövegben a nyelvtani megkötéseken kívül más formai megkötés nem fordul elő, szabadszövegnek, nem strukturált is nevezzük ezt a formát. A *szövegszerű* tárolásnál a dokumentumok, könyvek, cikkek alkotják a legkisebb elérési egységet, s a dokumentum rögzített belső adatstruktúra nélkül, ömlesztve tartalmazza az információt. A fenti tárolás a számítógépes feldolgozás során tehát viszonylag nagyobb helyigénnyel jelentkezik, s másrészt igen körülményes a szövegben tárolt információk automatikus, program által történő kigyűjtése.
- adatszerű, strukturált formátum: az adatok csak a lényeges információ elemeket adják meg. Például csak egy számot adok meg, de nem írom körül, mit is jelent az a szám. Ilyenkor más módon lehet az értékhez a tartalmat hozzárendelni. Az *adatszerű* tárolásnál az információk megadott struktúra szerint sokkal kisebb adatelemekre szétbontva kerülnek elhelyezésre, minden adatelemhez a struktúrában jelentést és formátumot is hozzárendelve. A táblázatok az adatszerű tárolás tipikus képviselői. A strukturált adatkezelés fő előnye a tömörség és a feldolgozás egyszerűbb algoritmizálhatósága.
- A *szemi-strukturált* rendszerek az előbb említett típusok között foglalnak helyet. Az ilyen jellegű dokumentumokban rendszerint létezik egy lazább, viszonylag nagyobb terjedelmű struktúra, melyen belül azonban az adatok lazább formában, szövegszerűen is elhelyezkedhetnek.

A számítástechnika induló korszakában az erőforrások korlátos volta miatt csak a strukturált adatokban lehetett automatizált információkezelést megvalósítani. Az informatikai technológia fejlődése révén napjainkban már a szabadszöveges források feldolgozása a fejlesztések egyik fő iránya.

A strukturált adatkezelés előnye, hogy hatékonyan automatizálható és kellő gyakorlással a felhasználók számára is feldolgozható. A strukturált adatelemek között az egyik legelterjedtebb

formátum a táblázat formátum. A táblázatnál az adatokat sorokba és oszlopokba rendezzük. Az oszlopok rendszerint egy tulajdonságot egy mennyiséget jelölnek, míg az egyes sorok az egyes egyedeket, kísérleteket szimbolizálják. Egy sort rekordnak is neveznek.

A táblázat formátum kis méretek esetén gyors áttekintést tesz lehetővé a felhasználó számára. A táblák átalakításával módosíthatjuk a látott tartalmat, például módosíthatjuk a rekordok sorrendjét és új számított értékeket hozhatunk be a táblázatba. A táblázatok kezelésére külön kezelőprogramok jöttek létre, melyekből az egyik legismertebb az Excel program.



3.2. ábra Excel adatkezelés (KEP_A303_I_03_02) KEP_A303_I_03_02.JPG

Az Excel egyre több lehetőséget rejt magába az adatok hatékony kezelésre, melyekből itt most az alábbiakat emeljük ki.

- a mezők konstansokat és formulákat is tárolhatnak
- a táblázat tartalma közvetlenül módosítható
- a rekordok sorba rendezhetők
- a rekordok mezőérték alapján szűrhetők
- a rekordokból aggregált értékek képezhetők
- a mezőkhöz megszorítások, értékellenőrzések köthetők

- a táblázat adatiból egyszerűen készíthetők grafikonok
- összetett adatkezeléshez makro programok készíthetők
- a táblázathoz beviteli űrlapok hozhatók létre
- a táblázat külső adatforráshoz is kapcsolható

Az Excel hatékony és könnyű kezelhetőséget biztosít az alapfunkciók elvégzéséhez. Ha azonban egy összetettebb feladatot kell elvégezni, akkor bizony itt is szükség van a programozói ismeretre, hiszen ezen extra elemeket rendszerint csak a makrókon keresztül lehet megvalósítani. Példaként nézzük azt az esetet, amikor egy mező egyediségét kell biztosítani. Ilyen egyedi értékű mezőnek tekinthető többek között az autók rendszáma vagy a dolgozók TAJ száma. Az egyediséget ellenőrző rutinba egy olyan ciklust kell beépíteni, amely egyenként sorba veszi a meglévő rekordokat és az ottani mezőértéket összehasonlítja a most felvitt új mezőértékkel. Ha nem talál egyezést, akkor egyedinek tekinti az új mezőt. Rögzített táblaméret esetén az alábbi kódrészlet szolgál a vizsgált ellenőrzésre:

```
Sub ellen1()
ss1 = 0
For i = 3 To 10
   k1 = Sheets(1).Cells(i, 1)
   s1 = 0
  For j = i + 1 To 10
     k2 = Sheets(1).Cells(j, 1)
     If k1 = k2 Then
        s1 = 1
    End If
  Next j
  If s1 = 1 Then
     ss1 = 1
    MsgBox (k1 & " nem egyedi")
  End If
Next i
If ss1 = 0 Then
   MsgBox ("egyediseg rendben")
End If
End Sub
```

Tehát az Excel jellegű táblázatkezelő rendszereken sem könnyű az összetett feladatok megoldása, és emellett még további jelentős korlátokkal is számolnunk kell egy információs rendszerbeli alkalmazása esetén. Az összetettebb programozás csak kisebb problémának tekinthető, a nagyobb probléma az egyes igényelt adatkezelési funkciók korlátai jelentik. A következő táblázat ezen korlátokat összesíti.

- Nagy adatmennyiség kezelése: a táblázatkezelőkben rendszerint korlátozott a kezelhető rekordok darabszáma a táblázaton belül. Emiatt a több százmillió rekordot tartalmazó esetekben és a dinamikusan növekvő méretű táblázatok esetében tárolási problémát jelent ez a megkötés.
- Párhuzamos hozzáférés: Az adatforrást a nagyobb rendszerekben egyidejűleg többen is használni fogják. Például egy banki nyilvántartó rendszerben egyszerre többen is indíthatnak tranzakciót pénzfelvételre, a számlatörzsre vonatkozólag.
- Dinamikus adatértékek: a táblában az alapadatok mellett helyet kaphatnak a származtatott értékek is. Például a bevétel és a kiadás független mezők különbségéből meghatározható a nyereség mező értéke is. A táblázatkezelők esetében viszonylag nagyobb erőforrást igényel a dinamikus mezők kezelése, emiatt a nagyobb adatkezelő rendszerekben más egyéb módszert, mint például előszámítások kezelése is bevetnek a hatékonyság növelésére.
- Kapcsolatok nyilvántartása: Az adatrendszerekben csak ritkán elegendő egyetlen táblázat az adatok tárolására. Hiszen egy banki nyilvántartó rendszerben is külön táblázat kell többek között a bankszámlák, az ügyfelek, a hitelek és a dolgozók nyilvántartására. Az adatrendszer egyik lényeges vonása, hogy a különböző táblákban tárolt adatelemek között kapcsolat áll fenn. A kapcsolat szolgál az egymáshoz rendelődő rekordok összerendeléséhez, ez alapján lehet az egyik elemből a kapcsolt elemet elérni.
- Megszorítások kezelése: A VIR adatrendszereknél alapvető követelmény az adatrendszer helyessége. A helyesség biztosítása azt is jelenti, hogy az adatrendszer nem engedi az adatok érékét érvénytelen értékűre beállítani. Ehhez előbb definiálni kell az érvényesség kritériumait az adatkezelő rendszer nyelvében megfogalmazva, és a megadott megkötéseket a rendszernek automatikusan ellenőriznie kell minden adatváltozás esetén. A táblázatkezelő rendszerekben ezen szabályokat csak a nehézkesebb makró programokon keresztül érvényesíthetjük.

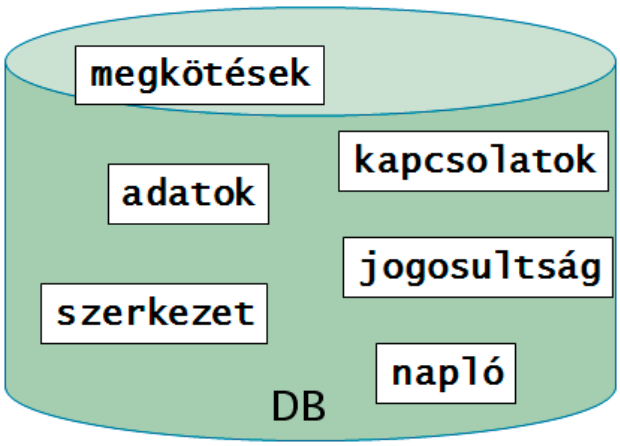
- Hatékony lekérdezés: Az adatkezelő rendszer fő feladata a felmerülő adatlekérdezési igények hatékony kiszolgálás a letárolt adatokra építve. Nagy rekordszám esetén a hatékony rekordkeresés megvalósítása jelentős problémát okoz. A kis adatméreteknél megszokott szekvenciális keresési módszerek nagy adatrendszerben nem alkalmazhatóak, hatékonyabb módszerekre bevezetésére van szükség. A leggyakoribb megoldás erre a feladatra az adatok indexelése, melyre nem biztosítanak lehetőséget a hagyományos táblázatkezelők.
- Nyílt kapcsolati felület: Az adatrendszert alapvetően hosszú távra tervezik, emiatt biztosítania kell a hozzáférés függetlenségét. Ez alatt azt értjük, hogy az adatrendszer több programozási nyelvből is elérhető, tehát az üzleti logikát megvalósító rétegeket megvalósító programok kicserélhetők, frissíthetők az adatréteg helybenhagyása mellett. A táblázatkezelők ugyan ma már rendelkeznek ilyen szabvány lehetőséggel, de egyrészt ez korlátozott hatáskörű és ezen felület formátuma, szintaktikája eltér a közvetlen elérés formátumától.
- Adatvédelem: Mivel az adatrendszer több személy, több részleg adatait is tárolja, szükség van ez adatok elérhetőségének kontroljára. Alapesetben mindenki csak a hozzá tartozó, általa létrehozott adatokat láthatja. Ezen elv szigorú betartása viszont azt eredményezné, hogy az adatok nem lehetnének megoszthatók a felhasználók között. Mivel a megosztás fontos követelmény, az adatok elérhetőségét finomabb szabályozást biztosító védelmi rendszerrel kell szabályozni. Ezen funkciók is csak korlátozottan állnak rendelkezésre a táblázatkezelő rendszerekben.
- Adatvesztés elleni védelem: Az adatok magas szintű rendelkezésre állása az egyik legfontosabb követelmény a megbízható vállalati működéshez. Mivel a termelés is folyamatos, az VIR rendszernek és a kapcsolt adatrendszernek is folyamatoson kell működnie. A rendelkezésre állás több részproblémát is magába foglal, mint például a bejött parancsok lehetőség szerinti mielőbbi végrehajtását és az adatok folyamatos meglétét. Ez utóbbi kritérium kiszolgálására szükséges az adatok lementése. A táblázatkezelők alapvetően nem nyújtanak ehhez a feladathoz saját eszközt.

3.2. Adatbázis-kezelés alapfogalmai

A korábbi elemzés alapján látható, hogy szükség van egy magasabb szintű adatkezelést biztosító adatrendszerre is. Ezen rendszerek lesznek az adatbázis kezelő rendszerek. A nagytömegű adatok igényelt szinten történő kezelését biztosító rendszereket *adatbázis kezelő* rendszereknek nevezik.

E témakör első alapvető fogalma az *adatbázis* fogalma. A fogalom definícióját az adatbázisokhoz rendelhető legfontosabb tulajdonságok megadásával írhatjuk le. Mindenekelőtt nézzük meg, mit kell tárolni az adatbázisban. Nyilvánvaló, hogy a modellezett valóságban szereplő egyedeknek és kapcsolataiknak szerepelniük kell. A felhasználó ezekkel az adatokkal fog dolgozni, ezen adatok kezelésére készülnek a különböző felhasználói programok. Ezeket az adatokat szokás tényleges, elsődleges adatoknak is nevezni. Az adatkezeléssel szemben felállított követelmények kielégítéséhez ezen adatok önmagukban nem elegendőek, gondoljunk csak arra, hogy a hatékony adatkeresés indexstruktúrát vagy hash szerkezetet igényel, vagy például az adatvédelem biztosításához szükség van a hozzáférési jogok tárolására és az adatmásolatok megőrzésére. Ezek a szerkezetek az elsődleges adatokra vonatkozó információkat tárolnak, ezért nevezik ezen adatokat metaadatoknak, tehát adatokra vonatkozó adatoknak. Ezen megfontolások alapján a következő definíciót adjuk meg:

> *Adatbázis*: egy olyan integrált adatszerkezet, mely több különböző objektum előfordulási adatait adatmodell szerint szervezetten perzisztens módon tárolja olyan segédinformációkkal, ún. metaadatokkal együtt, melyek a hatékonyság, integritásőrzés, adatvédelem biztosítását szolgálják. Az adatbázis szó rövidítésére gyakran használják az angol rövidítését, a DB-t.



3.3. ábra Adatbázis komponensei (KEP_A303_I_03_03) KEP_A303_I_03_03.JPG

Az adatbázisok elvileg tetszőleges méretűek lehetnek. Az elsődleges adatok száma nullától, az üres adatbázistól, a végtelen értékig terjedhet. Az elméletileg végtelen kapacitást a gyakorlatban a rendelkezésre álló hely, vagy éppen a belső tárolási struktúra korlátozza.

Az adatbázis, mint a fentiekből kitűnik, egy összetett adatstruktúrának tekinthető. Az adatstruktúra viszont az alkalmazások passzív elemeit jelenti, és kell egy algoritmus, egy program, amellyel felhasználhatók ezek az adatok, életre kelthetők az információk. Így az adatbázishoz kapcsolódnia kell egy kezelő programnak, amit *adatbázis kezelő rendszernek* neveznek. Az adatbázis kezelő rendszer értelmezése jóval egységesebb, mint az adatbázis értelmezése volt.:

> *Adatbáziskezelő rendszer*: Az a programrendszer, melynek feladata az adatbázishoz történő hozzáférések biztosítása és az adatbázis belső karbantartási funkcióinak végrehajtása. Az adatbáziskezelő rendszer rövidítése az angol elnevezés alapján: DBMS.

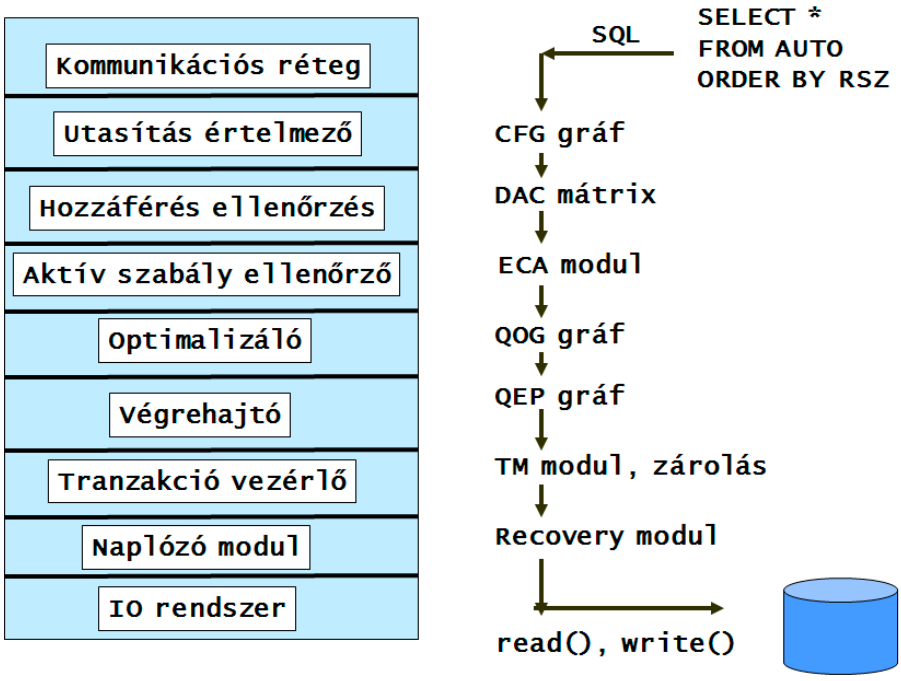
Az adatbázis kezelő rendszernek kell gondoskodnia a már korábban említett integritási, hatékonysági és védelmi feltételek megőrzéséről. Az adatbázis kezelő rendszer emiatt egy bonyolult programrendszernek tekinthető, mely sok funkcióját, összetettségét tekintve leginkább az operációs rendszerekhez hasonlítható. Az integritási, hatékonysági és védelmi feltételek ellenőrzését és betartatását az adatbázis kezelő rendszer a háttérben végzi el, mintegy a felhasználó közvetlen utasítása vagy éppen tudta nélkül. Mindez azért történik így, hogy a felhasználó véletlenül vagy szándékosan se tudja elrontani az adatbázist. Az adatbázis helyessége megőrzésének fontossága miatt definíciónkban külön kiemeljük az adatbázis kezelő rendszer ezen tulajdonságát:

A DBMS és az operációs rendszer hasonlata annyiban is helytálló, hogy mindkettő egy alsó szoftverréteget valósít meg amit, a felhasználó nem közvetlenül, hanem segédprogramokon keresztül ér el. Az adatbázis kezelés esetében is a felhasználó nem közvetlenül a DBMS-t kezeli, hanem egyéb segéd- és alkalmazói programokat futtat, melyek majd a DBMS-en keresztül érik el az adatbázisban tárolt adatokat. Maguk a DBMS rendszereket forgalmazó cégek is készítenek ilyen segédprogramokat, de egyedi fejlesztéssel is létrehozhatunk adatbázisbeli adatokat kezelő programokat. A DBMS-hez integráltan tartozó segédprogramok angol rövidítése UIT (User Interface Tools). Ezek alapján egy hatékony adatkezelő rendszernek tartalmaznia kell egy adatbázist, egy adatbázis kezelő rendszert, valamint alkalmazói és segédprogramokat.

Az első adatbázis kezelő rendszerek az 1960-as években jelentek meg, elsőként a *hálós adatmodell* alapjait, majd nem sokkal rá megjelent a *hierarchikus adatmodellt* dolgozták ki. Az első hálózatos, konkurens hozzáférést biztosító adatbank 1965-ben jelent meg az IBM-nél, és a SABRE nevet

kapta. Az adatbázis kezelő rendszerek maguk is jelentős fejlődésen mentek keresztül; jelentősen megváltozott a használati módjuk, az általuk támogatott adatmodell jellege. Az induló időszak hierarchikus, majd hálós adatmodelljei után az 1970-es években indult el hódító útjára a ma legelterjedtebb adatbázis kezelő típus, a *relációs adatbázis kezelés*. Évtizedünkben az adatbázis kezelés területén is új elvek jelentek meg, mint például az objektum orientáltság, a logikai programozás, az XML adatkezelés, valamint a szöveges és multimédia adatbázis kezelő rendszerek.

Az adatbázis kezelő rendszer egy összetett, több funkciót megvalósító szoftvernek tekinthető. A működésének belső logikáját egy réteges szerkezettel célszerű ábrázolni. A DBMS-ek legfontosabb komponenseinek vizsgálatánál a DBMS-t rendszerint két nagy struktúraegységre bontják: egy felhasználóhoz közeli rétegre (*Data System*), és egy a hardverhez kapcsolódó rétegre (*Storage System*). Míg a Data System feladata az adatok adatmodell szerinti kezelése, a Storage System az adatok fizikai tárolási struktúrájával dolgozik. A DBMS-en belül a két réteg között intenzív kommunikáció folyik, mindkettő a felhasználó és az adatbázis közötti adatcsatorna szerves része. A Data System fogadja a felhasználó utasításait, majd értelmezi az utasítások végrehajthatóságát és meghatározza az utasításhoz tartozó fizikai műveletsort. Ez a műveletsor kerül át a Storage System-hez, amely saját IO rendszerében elvégzi a fizikai adatátviteli lépéseket, ügyelve a konkurens hozzáférésből és a védelmi szempontokból adódó feladatokra.



3.4. ábra Adatbázis-kezelő rendszer komponensei (KEP_A303_I_03_04) [KEP_A303_I_03_04.JPG]

Mindkét réteg tovább bontható funkcionális elemeire. Elsőként vegyük a Data System legfontosabb komponenseit:

- *DC komponens*, melyet már ismertettünk és melynek feladata az interface biztosítása a segédprogramok, a felhasználók felé.

- *Utasításértelmező*. E komponens feladata egyrészt az utasítások szintaktikai ellenőrzése, másrészt az utasítások tartalmi, végrehajthatósági vizsgálata. Ehhez az adott adatmodell, adatkezelő nyelv ismerete mellett szükség van a kezelt adatbázis adatbázismodelljének az ismeretére is. Itt most nem az adatok ismeretére gondolunk, hanem az adatstruktúrára, illetve a védelmi és integritási feltételekre. Ezek az adatok pedig a már korábban említett metaadatok közé tartoznak. A DBMS az adatbázishoz tartozó metaadatokat az adatszótárban, Data Dictionary-ban tárolja. Az értelmezés feladatát, az eltérő jelleg miatt gyakran külön választják az adatdefiníciós és adatkezelő értelmezőkre.

- *Optimalizáló*. Mivel a felhasználóktól összetett utasítások is érkezhetnek, és egyidejűleg több utasításcsoport is állhat végrehajtás alatt, nem lényegtelen az elemi utasítások végrehajtási sorrendje. Egy utasítás ugyanis rendszerint több elemi részlépésre bontható fel, több utasítás esetén a generált elemi utasítások végrehajtási sorrendje is tág határok között változhat, tehát egyazon feladatcsoporthoz több elemi utasítássorozat is tartozik, s ezen sorozatok mind helyigényben, mind gyorsaságban különbözhetnek egymástól. Emiatt a DBMS egyik lényeges feladata az optimális elemi utasítássorozat kiválasztása.

- *Végrehajtó*. Az optimális elemi utasítássorozat végrehajtása történik ebben a modulban. Az utasítás-végrehajtás vezérlése mellett e komponens feladata az elemi utasítások végrehajtási kódjainak a tárolása, felhasználása is. Ebben a modulban is történik döntéshozatal, ugyan már sokkal szűkebb hatáskörben, mint az előző komponensnél, ugyanis bizonyos műveleteknél az algoritmusváltozat kiválasztása, csak az előző elemi lépés eredményének ismeretében történik meg. Az elemi utasítások végrehajtása során rekordszintű IO utasítások állnak elő, melyek feldolgozása már a Storage System feladata lesz.

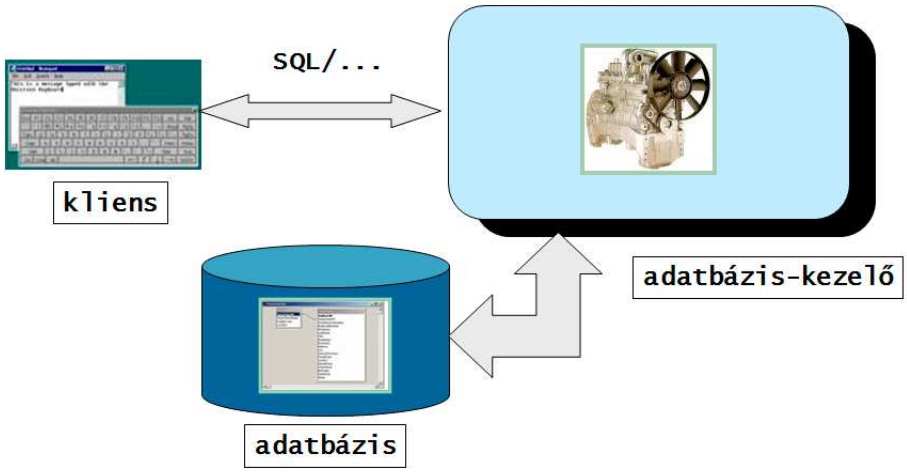
A Storage System legfontosabb komponensei:

- *IO rendszer*. Mint már említettük, a DBMS specifikus igényei miatt saját IO rendszert, bufferkezelést, helyfoglalási mechanizmust építenek be a legtöbb fejlettebb DBMS-be. Ez a rutinkönyvtár az OS IO kezelésénél magasabb szintű, a DBMS bufferhez kapcsolódó

rutinokat tartalmaz. Ezek a rutinok már közvetlenül hívhatják az OS alacsony szintű IO rutinjait.

- *Konkurens hozzáférés vezérlés*. Majd a későbbiekben látni fogjuk, hogy milyen eszközök állnak rendelkezésre a megosztott erőforrások kezelésére. E modul tartalmazza mindazokat az alacsony szintű adatstruktúrákon értelmezett módszereket, melyek az osztott hozzáférés vezérléséhez szükségesek.

- *Adatvédelmi rendszer*. E modul feladata a különböző adatsérülések, rendszer leállások okozta veszteségek minimalizálása, az adatok megfelelő védelmének biztosítása. Ez a komponens is több olyan kisebb részből áll elő, mint pl. a rendszeres háttérmentés végzésére szolgáló rutin.



3.5. ábra Adatbázis rendszer komponensei (KEP_A303_I_03_05) KEP_A303_I_03_05.JPG

A DBMS belső struktúrájával közvetlenül sem a felhasználó, sem az alkalmazó programozó nem fog találkozni, számukra rejtve maradnak a DBMS összetettségének jelei. Ami egy felhasználót igazában érdekel, az a DBS felhasználói kapcsolattartása, azon segédprogramok rendszere, melyeken keresztül elérhetők az adatbázisban tárolt adatok.

3.3. Adatmodell szerepe

Az adatmodell az adatok logikai tárolási formátumát határozza meg: olyan vázat ad, melybe az adatok majd beletölthetők lesznek. Az adatmodell megadása eszerint egy szerkezetleírást jelent. Ezt az elképzelést azonban meg kell még toldani annyival, hogy az adatrendszer ismerete nem csak az

adatszerkezet ismeretét, hanem az adatok kezelési módjának az ismeretét is magában foglalja. Ezért az adatmodellbe a statikus szerkezet leírás mellett a dinamikus, az adatokon értelmezett műveleteket is beleértik. Az adatmodell megadásánál mind a szerkezet, mind a műveletek megadása logikai szinten, s nem fizikai szinten történik, ezért az adatmodell egy elvontabb absztraktabb, formálisabb leírást jelent. Szokás az adatok felvehető értékeinek körét meghatározó megkötéseket leíró nyelvet is adatmodell komponensei közé bevenni. Összegezve tehát az adatmodell olyan matematikai formalizmus, mely az adatok és az adatokon értelmezett műveletek leírására szolgál. Az egyes adatmodellek a kiválasztott formalizmus jellegében különböznek egymástól, a deduktív adatbázisok például logikai formalizmust használnak fel.

Több adatmodell is létezik, de ezekből négy terjedt el igazán a gyakorlati életben: a hierarchikus, a hálós, a relációs és az objektum-orientált adatmodellek. Ezek közül a relációs adatmodell a legnépszerűbb ma, a hálós modell kezd háttérbe szorulni, míg a hierarchikus már inkább a múlté, az objektum-orientált modell pedig csak a jövőben válik igazán piacéretté.

A *hierarchikus adatmodell* az adatokat egy hierarchikus faszerkezetben tárolja. E fa mindegyik csomópontja egy rekordtípusnak felel meg. A hierarchikus modell alapja, hogy a gyakorlati életben a szervezetek vagy éppen a struktúrák nagyon gyakran hierarchikus felépítésűek, gondoljunk csak a vállalati hierarchiára vagy egy gyártmány alkatrészeinek hierarchiájára. Emiatt természetesnek tűnik, hogy a modellezés megkönnyítésére a valóságban leggyakrabban használt, hierarchikus modellt hozzuk létre. Ez a modell a gyakorlati alkalmazások során fejlődött ki, ezért nincs olyan elméleti megalapozottsága mint a későbbi adatmodelleknek. A modellhez kapcsolódó DML nyelvek mind rekordorientált adat megközelítést alkalmaztak. A bonyolultabb kapcsolatok ábrázolása csak kerülőutakon lehetséges. A modell előnye, hogy a hierarchikus szerkezet egyszerűen leírható, és tárolása a mágnesszalagos tárolási formához is jól illeszkedik.

A *hálós adatmodell* a hierarchikus modell továbbfejlesztése, amely jobban illeszkedik a bonyolultabb kapcsolatok ábrázolásához is. Ebben a modellben az egyedek között tetszőleges kapcsolatrendszer, egy kapcsolatháló alakítható ki. Az adatszerkezet leírása, mivel a háló tetszőleges nagy lehet, nem egy adategységgel, hanem több kisebb, hierarchikus felépítésű adategységgel történik. Ehhez a modellhez is rekordorientált adat megközelítést alkalmaztak a DML kialakításakor. A hálós modellen alapuló DBMS-ek igen elterjedtek a nagygépes környezetekben, hiszen a hálós modell nagy adatmennyiségek viszonylag gyors feldolgozását teszi lehetővé. A kezelőnyelv bonyolultsága, viszonylag merevebb szerkezete gátolta szélesebb körben történő elterjedését.

A *relációs adatmodell* sokkal rugalmasabb szerkezetet biztosít az elődeihez viszonyítva. Az adatbázis azonos rekordtípusokat tartalmazó táblákból épül fel, ahol minden tábla teljesen egyenértékű, s nincs semmilyen, az adatdefiníciókor véglegesen lerögzített kapcsolat, váz, mint ami az előző modelleknél előfordult. A relációs modellben az egyedek közötti kapcsolatok az adatértékeken keresztül valósulnak meg. A relációs modellben a táblákon értelmezett műveletek ugyan halmazorientáltak, de számos olyan implementáció létezik, melyben rekordorientált műveletek használhatók. A modell elterjedése az egyszerűségének és rugalmasságának köszönhető.

Az *objektum-orientált modell* célja az objektumorientáltság szemléletmódjának alkalmazásával minél valósághűbb adatmodellt megalkotása. Az egyedek ugyanis sokkal szemléletesebben írhatók le az objektumokkal, mint a relációs modellben szereplő rekordokkal. Az objektum orientáltság a megvalósult rendszerekben lehet teljes vagy részleges. A részleges OODBMS-ek rendszerint csak strukturálisan objektum-orientáltak, a funkcionális, aktív elemek csak a teljes OODBMS-ekben jelennek meg. Az OODBMS-ek elterjedését az egységes elméleti alapok hiánya és az implementációs nehézségek fékezik.

3.4. Relációs adatmodell

A relációs adatmodell napjaink legelterjedtebb adatmodellje. A modell alapjait 1970-ben Codd fektette le. A megtervezett modellben egy igen egyszerű, könnyen megtanulható leírási módot sikerült megvalósítani. *Egyszerűségének* következtében gyorsan népszerűvé is vált a felhasználók körében, és sok implementációja született meg a személyi számítógépek piacán is. Másrészről az *elméleti megalapozottság* a kutatók, a szakemberek szimpátiáját is kiváltotta, s ez a modell számos új fejlesztési projekt alapját képezi. A relációs adatmodell fontos előnye az egyszerűség mellett a rugalmasság. A relációs adatmodell a következő strukturális elemekből épül fel:

- *mezők*
- *rekordok*
- *relációk*
- *adatbázis*

A relációs modellben a mező elemi értékű lehet, tehát egy elemi érték tárolására szolgál. Egy banki mintarendszer esetén összes elemi adat egy-egy külön mezőbe fog tárolásra kerülni. Külön mező lesz a dolgozó nevére, az ügyfél születési évére vagy a bankszámla számlaszámára. Minden elemi adat tehát mezőhöz fog rendelődni. A mezőket nevükkel és a hozzájuk rendelt domain és integritási

feltétel megadásával azonosíthatjuk. Egy autó egyednél, például a rendszám tulajdonsághoz rendelhetjük a rendszámot azonosító mezőt, melyhez a magyar viszonyoknál maradva egy magyar-rendszámok domain tartozhat. E domain adattípusa a hatkarakteres sztringek halmaza lehet, melyben az első három karakter betű, az utolsó három pedig számjegy, azaz a formátuma "XXX999" alakú lesz. A sémában több mező is szerepelhet ugyanazzal a domainnal. Ennek előnye, hogy jobban érzékelhetők a mezők közötti kapcsolatok, másrészt hatékonyabbá teszi a karbantartást is, hiszen ekkor több mező helyett csak egyetlen egy domiannél kell az esetleges változtatásokat (pl. rendszám alakjának megváltoztatása) átvezetni.

Mivel minden elemi adat egy-egy mezőbe kerül, az adatbázisban nagyon sok mező lesz tárolva. A felhasználónak azonban nemcsak a mezők értékeinek ismerete fontos, hanem a mezők közötti kapcsolatok ismerete is. Vagyis azt is kell tudnunk, mely mezőértékek tartoznak egy egyedhez. Az adatbázisban tehát együtt kell tárolni az azonos objektumhoz tartozó mezőket. Egy összetartozó mezőcsoportot nevezünk rekordnak. A mezők között kiemelkedő szerepet játszanak a rekordot meghatározó kulcs mezők. Az adatbázisban az azonos jellegű objektumokhoz azonos rekordszerkezet fog tartozni. Így például minden ügyfélhez ugyanazon rekordséma, ugyanazon mezőlista fog társulni.

A következő egység a reláció, amely egy táblázatnak is tekinthető, amelyben az azonos szerkezetű rekord előfordulások foglalnak helyet. A táblázat könnyen átlátható egység, melyben a sorok jelentik a rekordokat, s az oszlopokban az egyes mezők helyezkednek el.

A rekordok közötti kapcsolatok ábrázolása egészen újszerű módon valósul meg, nevezetesen az értékeken keresztül. Az alapelv a következő: minden rekordnak, vagyis sornak van olyan mezője vagy vannak olyan mezői, amelyek értéke egyértelműen meghatározza az illető rekordot. Ha egy másik rekord kapcsolódik ezen rekord előforduláshoz, akkor a kapcsolat jelzésének módszere az, hogy a kapcsolódó mezőbe beteszünk egy olyan mezőt, amelynek értéke a hivatkozott rekord azonosító értéke. Így a két rekord megfelelő mezőinek értékegyezősége fogja jelezni az összetartozást. Nos, ezen megoldásba belegondolva látszik, hogy itt egyáltalán nem volt fontos a hatékonyság, hiszen ez a megoldás sokkal több időt igényel a kapcsolatok feltárásakor, mint akár a pozíció, vagy akár a pointer alapú kapcsolódás. Mégis miért választottak egy rosszabbnak tűnő megoldást? Azért mert így sokat nyerhettek a rugalmasság oldalán. A modellben igen hatékonyan lehet módosítani a kapcsolatokat, s az adatok lekérdezésekor is rugalmasan össze lehet kapcsolni az egyes relációkat. Igen fontos már itt megjegyezni, hogy az adatok lekérdezésekor minden reláció *egyenértékű*, nincs alá- vagy fölérendelt rekord, mint a korábbi modellekben.

A relációs modellhez kapcsolódik néhány olyan alapfogalom is, amely nem közvetlenül az adatszerkezethez kapcsolódik, hanem az adatbázis integritásához. Az integritási szabályok célja az adatbázis előfordulások lehetséges, megengedett körének behatárolása. Az *integritási szabályok* tehát az adatbázisban lévő adat előfordulásokra adnak megszorításokat. Az integritási szabályokat aszerint csoportosíthatjuk, hogy milyen szinten fogalmazzák meg a megkötéseket. A relációs adatmodellben az alábbi négy szintet szokás megkülönböztetni:

- mező szint
- rekord szint
- reláció szint
- adatbázis szint

A *mező szinten* egy mezőre vonatkozó érték előfordulások körét lehet megadni. A megkötést vagy egy logikai kifejezéssel lehet megadni, amely minden lehetséges domain értékre igaz vagy hamis értéket ad vissza, vagy annak előírásával, hogy a mezőben tárolt érték nem lehet üres. Az adatbázisba csak olyan mezőértékek tárolhatók le, melyekre a kijelölt feltétel igaz értéket ad vissza. Az a megkötés például, hogy a rendszám első három karaktere nem lehet szám, domain szintű integritási feltételt jelent, hiszen az ellenőrzés csak egyetlen egy domaint érint, egy önálló mező ellenőrzési feltételt szab ki.

A mezők esetén az egyik fontos megkötés, hogy a mező maradhat-e kitöltetlen vagy nem lehet üres. Az *üres, kitöltetlen érték*, amit az adatbázis kezelésben a *NULL* szimbólummal jelölünk, egy önálló érték a relációs modellben, mégpedig fontos szerepet játszó érték. A NULL érték nem azonos a 0 értékkel, mert ez utóbbi értékes, valós adat lehet. Az RDBMS-ek rendszerint külön jelzővel jelölik, ha egy mező még nem kapott értéket, tehát ha nincs benne letárolt adat. Ekkor mondjuk, hogy a mező a NULL értéket tartalmazza.

Mező szintű megkötések:

- a mező nem maradhat kitöltetlen

- értékellenőrzés

A *rekordszint* esetén egy teljes rekord elfogadhatóságát döntjük el. Az ellenőrzési feltételben a relációsémában szereplő mezők szerepelhetnek. Az integritási feltétel célja az egy rekordon belül egymáshoz kapcsolódó mezők értékeinek vizsgálata. Példaként vehetünk egy olyan megkötést, egy minta autókereskedő nyilvántartási rendszerből, mely szerint a katalizátoros autóknál az A vagy B

adókulcs alkalmazható. E feltétel ellenőrzéséhez elegendő egyetlen egy rekord előfordulást önmagában vizsgálni.

Rekord szintű megkötés:

- értékellenőrzés

A *reláció szintű* ellenőrzéshez a teljes relációt, azaz több rekord előfordulást is át kell vizsgálni. Az a megkötés például, hogy a mezőben ugyanaz az érték nem fordulhat elő többször a relációban, csak úgy ellenőrizhető, ha ismerjük a relációban tárolt összes mezőértéket. Így az egyediséget előíró feltételt reláció szintű integritási feltételnek tekintjük. A reláció szintű ellenőrzési feltétel megfogalmazódhat egy aggregációs értékre vonatkozó megkötésben, például amikor előírjuk, hogy az autók átlagéletkora 12-nél több nem lehet.

A legfontosabb reláció szintű megszorítás az elsődleges kulcs megkötés. Mint már ismert, az egyedek megkülönböztetése tulajdonság értékeik alapján történik, s a helyesen megtervezett adatmodellben minden egyedtípushoz léteznie kell olyan tulajdonságcsoportnak, mely egyértelműen meghatározza az egyed előfordulásait. Az ilyen mezőcsoportot nevezik elsődleges *kulcsnak*. A kulcs értéke egyedi és mindig tartalmaz értékes adatot.

Reláció szintű megkötések:

- elsődleges kulcs feltétel teljesülése

- a mező értéke egyedi

Az *adatbázis szintű* megkötések esetében a feltétel több relációban szétszórtan elhelyezkedő mezőkre vonatkozik. Ekkor az ellenőrzéshez több reláció adatait is át kell olvasni. Erre példa az a megkötés, amikor előírjuk, hogy az autó típuskódjának szerepelnie kell a típusok reláció valamely rekordjának kód mezőjében.

Adatbázis szintű megkötések:

- idegen kulcs feltétel teljesülése

- összetett értékellenőrzés

Az integritási feltételek egy másik szempont szerinti csoportosításában az osztályozás úgy történik, hogy a feltétel az adatbázis egy konkrét állapotára vagy egy állapot átmenetére vonatkozik. Az *állapotra vonatkozó* feltételeknél egy konkrét adatbázis előfordulást vizsgálunk, függetlenül a

korábbi vagy későbbi adatbázis állapotoktól. Az a feltétel például, hogy egy mező értéke nem lehet üres, kitöltetlen, állapot integritási feltételnek tekintendő. Ha a feltétel azonban az állapotok megváltozását érinti, akkor egy *állapotátmenet* integritási feltételt kapunk. Erre példa lehet az a megkötés, mely szerint a fizetés értéke nem nőhet 25 százaléknál jobban. Ezt a feltételt úgy lehet ellenőrizni, hogy módosításkor az adatbázis módosítás előtti és módosítás utáni állapotából a két fizetésértéket összehasonlítjuk, és megállapítjuk a változás mértékét.

Az adatbázis rendszerek folyamatos és megfelelő működéséhez megfelelő felügyeletet és menedzselést kell biztosítani. Az adatbázis kezelő rendszer rendszergazdását szokás DBA-nak (Database Administrator) nevezni. A DBA tevékenységi köre már a telepítés előtt megkezdődik és folyamatosan végigköveti a rendszer működését. A telepítés előtt a rendszer tervezését kell irányítania, meg kell határoznia, milyen paraméterű rendszer kerüljön bevetésre.

3.5. Relációs adatbázis-kezelő rendszerek áttekintése

A rendszer telepítése kapcsán az egyik alapvető kérdése, hogy mely adatbázis kezelő rendszer (DBMS) kerüljön bevetésre. A piacon ugyanis több gyártó van jelen, ha gazdagnak nem is nevezhető is gazdag, de több lehetőséget kínáló termékskálával. A lehetséges rendszereket két fő csoportba oszthatjuk szét:

- Ingyenes termékek:
  A termék szabadon letölthető a gyártó honlapjáról és szabadon felhasználható. Itt ügyelni kell arra, hogy a letölthető termékre a gyártó milyen felhasználást engedélyez. Vannak ugyanis olyan rendszerek, ahol a gyártó csak a termék tesztelését, betanulását engedélyezi, de a fejlesztést már tiltja. Más esetekben a fejlesztés még engedélyezett, de a piaci értékesítés már csak a fizetős változattal jogszerű. A nagy DBMS gyártó cégek sorba jelentkeztek ilyen szabadabb felhasználású termékkel, melyeket rendszerint XE, azaz Express Edition változatnak neveztek el. A fontosabb szabadabb felhasználású termékek:
    - mySQL
    - Postgres
    - HSQLDB
    - Oracle XE

- SQLServer XE

- DB2 XE

- Fizetős termékek:

  A fizetős termékek esetében komoly licenszdíjat kell fizetni a felhasználásért. A kifizetett díjért cserébe rendszerint jelentős többletérték kaphatunk vissza. A fizetős változat mellett az alábbi érvek szólnak:

  - garancia a rendszer funkcionalitására
  - többlet szolgáltatások:
    - dokumentációk
    - hatékonyság javító modulok
    - nagyobb méretek kezelése (memória, adattábla)
    - adathordozó
    - védelmi elemeket javító modulok
    - menedzselést támogató modulok
  - termék használat támogatás (esetleg külön díj ellenében)

A DBMS termék kiválasztás után következő kérdés a DBMS verziójának, a kategóriájának a meghatározása. Egy adott DBMS termék ugyanis eltérő funkcionalitásban kapható a piacon. Példaként az SQLServert véve, az alábbi verziók közül lehet választani:

- Express Edition
- Workgroup Edition
- Developer Edition
- Standard Edition
- Enterprise Edition

Az előbb említett Express Edition elsődlegesen egyfelhasználós környezetre és tanulási céllal készült. A rendszerből hiányoznak számos kényelmi szolgáltatások és jelentős méretkorlátokkal kell számolni.

A Workgroup Edition kis vállalkozások számára készült, alapszinten támogatja a több processzort, van elemi DBA menedzseri modul, de nincs benne több hatékonyság és megbízhatóság támogató rész.

Developer Edition: Az adatbázis fejlesztők részére készült, amely támogat minden lehetséges funkciót, (mindent ami az Enterprise szinten elérhető), viszont a termék csak fejlesztési időszak alatt használható. Az elkészült adatbázis VIR alkalmazásához ez a szint már nem elegendő.

Standard Edition: Normál nagy és középvállalatok részére biztosít adatkezelési motort. Elsődlegesen csak a támogatott CPU-k számában és a particionálás hiányában különbözik a legnagyobb verziótól.

Enterprise Edition: A teljes funkcionalitást támogató verzió.

A fenti tervezési funkció mellett a DBA-hoz még tovább fontos és szabványos tevékenységkörök tartoznak, mint például:

- felhasználók karbantartása
- védelmi rendszer felügyelete
- objektumok paraméterezése
- adatmentések elvégzése
- adatbázis helyreállítása
- működési paraméterek beállítása, hibák kijavítása

A legnagyobb veszély az adatbázis működése során az, ha elveszik az aktuális adatbázis úgy, hogy nem is pótolható. Az adatvesztés a teljes nyilvántartási rendszer összeomlását jelenti, ami szinte pótolhatatlan veszteség mind a cég mind VIR szempontjából. A veszély súlyát jól mutatja, hogy a részeges leállások, időleges leállások is több tíz vagy száz millió Forint veszteséget okozhatnak a cégnek. Ezen veszteségek elkerülésére gondoskodni kell az adatok lehetőség szerinti védelméről, az adatveszteség minimalizálásáról.

A DBA egyik alapfeladata az aktuális adatbázis állapot lementése. A mentés, vagy BACKUP funkció több módon is elvégezhető. A legfontosabb mentési módok:

- file szintű mentés (például a COPY OS parancs használata)
- DBMS szintű mentés

A file szintű mentés esetén a DBA az adatbázishoz tartozó állományokat átmásolja egy külön lemezre. Ezen mentés igen egyszerűen elvégezhető, azonban van egy szigorúbb előfeltétel: a másolás alatt a DBMS nem működhet, senki nem dolgozhat a rendszerrel. Ha ugyanis dolgoznának a rendszerrel, a mentés inkonzisztens és használhatatlan lenne.

A DBMS szintű mentés esetén a segédeszközök lehetővé teszik a folyamatos munka alatti mentést is. A konzisztencia biztosításához a mentést több állományba szétbontva végzik el. A lehetséges mentési módok:

- teljes mentés: minden adat
- napló mentés: csak a tevékenységlista mentődik, az adat nem
- különbségi mentés: csak a módosult adatok mentődnek
- részleges mentés: az adatbázis egyes szeletei mentődnek csak

A mentés elvégzése után gondoskodni kell a másolatok megfelelő őrzéséről. A másolatok felhasználásra rendszerint két esetben kerül sor:

- ha megsérül az aktuális adatállomány

- ha az adatbázist egy korábbi állapotra kell visszahozni.

A sérült állapot helyreállítását nevezik RECOVERY folyamatnak. A helyreállítás indítása történhet manuális és automata módon is. Ez utóbbi eset akkor következik be, ha a DBMS indításkor érzékeli, hogy sikertelen volt az utolsó rendszerleállás.

Az említett mentési-helyreállítási folyamat egyik gyenge pontja, hogy mind mentés és mind a helyreállítás viszonylag több időt vesz igénybe, hiszen speciális indítási pontja van. Emiatt viszonylag hosszabb lehet a rendszer kiesési ideje. A veszteség további csökkentése érdekében vezették be a tükrözés mechanizmusát. Ekkor a rendszerben a dolgozó, fő adatbázis mellé két további adatbázist vesznek fel, melyek az alábbi funkciókat látják el:

- tüköradatbázis: ide másolódik a munkaadatbázis tartalma.
- irányító adatbázis: figyeli a munkaadatbázis állapotát, s szükség esetén a tükör adatbázist teszi meg munkaadatbázissá.

A tüköradatbázis tartalma automatikusan frissül, aktualizálása úgy történik, hogy a munkaadatbázisban végbement műveletek a naplón keresztül átjutnak a tükör adatbázishoz és ott is lefutnak a tevékenységek. Az irányító adatbázis megadott gyakorisággal ellenőrzi a munkaadatbázis érvényességét, s ha hibát talál, a két adatbázis szerepét felcseréli: a tüköradatbázis lesz a fő adatbázis, a korábbi munkaadatbázis lesz a tüköradatbázis.

A fenti mechanizmus mellett további módszerek is léteznek a veszteség minimalizálására. A DBMS kiesés csökkentését például a Cluster mechanizmusokkal lehet elérni.

3.6. Adatátviteli mechanizmusok áttekintése

Az információs rendszeren belül az adatok több különböző helyen is tárolódhatnak, több különböző helyen is felhasználásra kerülhetnek. Vegyünk például egy rendelés feldolgozását. A rendelés jöhet például elektronikus levélben a felhasználótól valamelyik kihelyezett egységhez. Az egységhez történő beérkezés után a rendelés adatait továbbítják a központba. A központban a feldolgozás után értesítést küldenek a raktárba a tranzakció végrehajtására. A raktárból nyugtázást küldenek a központba a rendelés teljesítéséről, s onnan egy válaszlevél indul ki a rendelő felés, miközben a szállító felé is üzenet indul a szállítási feladat igénylésére. A fenti kis példából látható, hogy a VIR rendszeren belül az adatok folyamatos áramlásban vannak, a hálózat különböző csomópontjai között mozgnak. A fenti adatáramlás egyik fontos technikai elemei a hálózat működése mellett az adatok megfelelő értelmezése, azaz a megfelelő adatformátum használata.

Az adatok relációs tárolása esetén a táblázat tartalmát kell átküldeni a mások oldalra. Az adatok átküldése DBMS szinten több különböző formában is megvalósulhat:

- két DBMS szoros összekapcsolása
- laza adatkapcsolat
- áttételes adatkapcsolat

A szoros kapcsolat azt jelenti, hogy a két DBMS rendszer közvetlenül cserél adatokat egymással. Ehhez közvetlen kapcsolatot kell kiépíteni a két oldal között. A kapcsolat révén az egyik oldal közvetlenül elérheti a másik oldal adattábláit. Ezáltal lehetővé válik egy olyan SQL parancs végrehajtása, amelyben a megadott táblák különböző adatbázisból származnak. A külső tábla kijelölése több különböző formában történhet DBMS-től függően. Az Oracle esetében például egy DATABASE LINK objektum használható, amely mögött az adatbázis kapcsolat él. Ekkor a táblára történő hivatkozáskor a tábla neve mögött szerepeltetni kell a kapcsolati objektum azonosítóját. Az SQLServer esetében regisztrálni kell a külső szervert , és a SQL parancsban a tábla neve előtt adhatjuk meg a szerverazonosító nevét. Mindkét esetben a végrehajtó DBMS átküldi a kérést a másik oldalra, majd feldolgozza a visszaküldött rekordhalmazt.

A laza kapcsolat esetén egy közvetítő állományt hoznak létre, amely egy későbbi, tetszőleges időpontban kerül át a mások oldalra végrehajtásra. A közvetítő állomány átmásolása hagyományos úton, a felhasználó közvetítésével megy végbe. Az egye DBMS rendszerek tömörített és saját egyedi eljárással kódolt formátumot hoznak létre, amelyet csak az ilyen típusú kezelők értenek meg. Más esetekben az adatbázis kezelő rendszer egy szabadon olvasható, szöveges formátumban adja

vissza az adatbázis tartalmát. Az ilyenkor előálló SQL parancssor némi igazítás után több különböző adatforrásnál is felhasználható.

A közvetett kapcsolat esetén egy middleware terméken keresztül megy az adatáram. Ekkor adatformátum konverzióra is sor kerülhet a transzformáció során. Ez a megoldás legrugalmasabb abban a tekintetben, hogy különböző rendszerek, eltérő adatkezelési formátumú adatforrások is összeköthetők. Az adatokat a közvetető egy köztes, szabványosított formátumban viszi az egyes csomópontok között. Ezen megoldás egyik ismertebb képviselője a SOAP technológia, melyet az XML formátum kapcsán fogunk részletsebben elemezni.

EDI szabvány

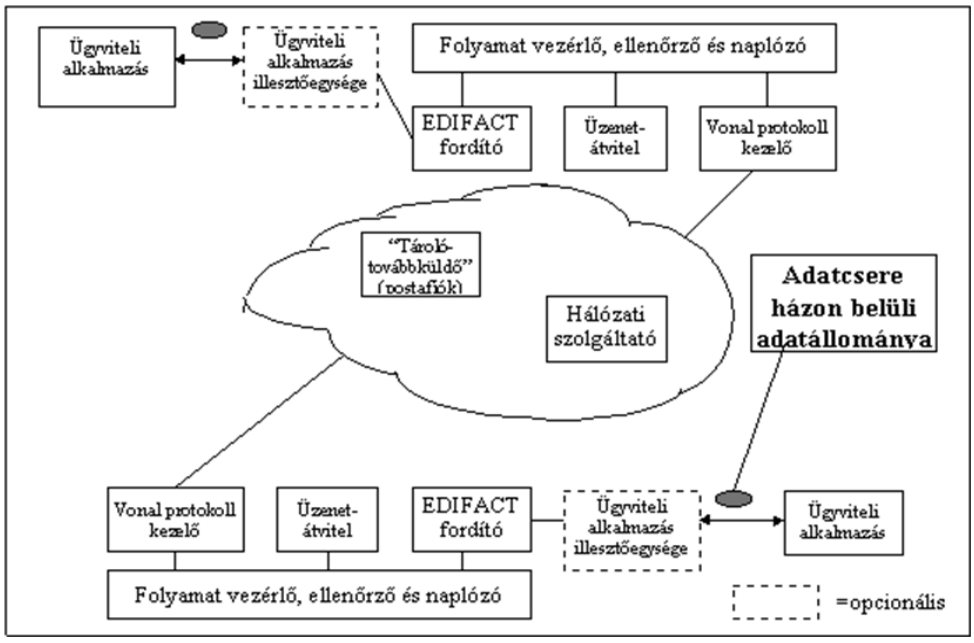
A vállalatoknak a közvetlen adatbázis szintű adatkapcsolat mellett általánosabb adatkapcsolati szolgáltatásra is szükségük van. Ezen üzleti adatcsere esetében hasznos szolgáltatás lenne, ha magas szinten automatizálhatók lennének az üzenetek feldolgozása. Ehhez a bejövő üzenetek tartalmának megértése szükséges. Mivel a szabadszöveges, természetes nyelvi üzenetek teljesebb megértése még a távolabbi jövő feladata, ezért másmódon kell lehetővé tenni az üzenettartalmának automatizált feldolgozását.

A választott megoldás alapelve, hogy a szabadszöveges üzenetet egy kódolt szöveggel helyettesítjük. A kódolás egy tartalom alapú kódtábla alapján történik. A kódtábla az üzleti üzenetek tipikus részeit tartalmazza. A kódolás egyik fontos szabványa az EDI (Electronic Data Interchange) szabvány. Az EDI szabvány fő sajátossága, hogy megpróbál általános érvényű leni, ezért a kódtáblájában minden lehetséges üzenetfajtára kitér. Az EDI szerepét jól jellemzi, hogy az egységes kódtábla kialakítása az ENSZ égisze alatt megy végbe.

Az EDI rendszeren belül az üzenetváltás az alábbi forgatókönyv szerint megy végbe:

- üzenetek megadása űrlapon keresztül, minden beviteli elemhez egy megadott értelmezés, jelentés társul
- az üzenet EDI kódformátumra történő konvertálása
- az üzenet csomagokba szervezése
- adatvédelmi elemek érvényesítése
- hálózati réteg felé továbbítás
- üzenet fogadása

- EDI tartalom kicsomagolása
- tartalom feldolgozása (konvertálás vagy rutinok triggerelése)



3.6. ábra EDI rendszer komponensei (KEP_A303_I_03_06) *KEP_A303_I_03_06.JPG*

Az EI szabvány az alábbi elemkre tér ki:

- komponensek típusai (szintaxis)(az EDI üzeneteknek milyen adatelemei vannak)
- üzenetek struktúrája (EDI csomagok felépítése)
- adatszótár (funkciókör leíró, az EDI kódok és az üzleti jelentés összekapcsolása)
- kiegészítő kódok (védelmi és egyéb rétegek)

Logikailag az üzenet az alábbi egységeket tartalmazza:

- legkisebb egység az elemi adat (például az ember családneve)
- összetett adat (például a teljes név)
- adatszegmens (több összekapcsolt adatelem, mint egy rekord) (például ember adatai)
- üzenet űrlap, egy logikai egység, egy tevékenységet ír le (például egy ember felvétele a céghez)
- funkcionális csoport (azonos űrlapok együttese)
- adatcsomag (egyszerre elküldött adatmennyiség / csoportok

Az EDI üzenetben a kódolás rövidítéseken keresztül valósul meg. A következő táblázat egy kódtábla részletet mutat be:

Kód	db	jelentés
UNH	1	Message Header
BGM	1	Explanation of message function
DTM	1	Date/time of preparation
LOC	up to 10	Port of loading, port of discharge
RFF	up to 10	References with the consignment
TDT	1	Vessel, carrier details
EQD	up to 999	Container type, shipper/carrier
EQN	1 per EQD	Number of containers
FTX	up to 9 per EQD	General container information
UNT		Message Trailer

A fenti kódokat paraméterekkel ellátva adódik ki az üzenet kódolt alakja. Példaként vegyünk egy áru behajózási üzenetet kódolt alakját:

UNH+19134+IFTMCS:D:98B:UN:ENET30'
BGM+770+19134+9'
DTM+137:20011110:203'
LOC+33+USLGB:::LONG BEACH‘
EQD+CN+++2'
EQN+4'
FTX+AAI+++20 foot containers, food quality'
UNT+16+19134'

Az EDI szabvány előnye, hogy a egyszerűen visszakódolható és értelmezhető az üzenet a kódtábla birtokában. Hátránya viszont az, hogy a kódtáblának teljesnek kell lennie a használhatósághoz. Mivel az üzleti tevékenységek köre folyamatosan bővül, a kódtábla is folyamatos bővítést igényel. Ennek viszont az az ára, hogy az EDI modulok állandó fejlesztést igényelnek.

Az EDI alapú üzenetváltás feltételezi, hogy a kódtábla tartalmazza mindazon tartalmi elemet, amiket az üzenetben küldeni szeretnénk. Mind említettük, ez sajnos igen nagy költségvonzattal és

implementációs nehézségekkel jár. Emiatt az EDI nem válhatott teljesen általánossá, az üzeneteknek csak egy részét, igaz fontos részét fedi le. A többi üzenet fajta kezelésére másfajta megoldást kellett kidolgozni. A legsikeresebb megoldássá ezen a téren az XML formátum vált.

XML nyelv

A XML nyelv az 1998-as évben vált szabvánnyá. Maga a nyelv a jelölőnyelvek (Markup Language) családjába tartozik, közvetlen őse az SGML nyelv, közeli rokona a HTML nyelv. A jelölőnyelvek fő jellemzője, hogy a szöveget jelölő elemekkel bővíthetjük ki, ahol a jelölő elemek tetszőleges információt hordozhatnak a közrefogott szövegrészről. Az XML nyelv sajátossága, hogy a szabvány alapvetően csak a jelölések formátumára ad megkötést, a jelölőelemek neve nem rögzített a szabványban. Emiatt lehetőség van arra, hogy az XML dokumentumban a szöveget a tartalmára utaló jelöléssel lássuk el.

A jelölőelem szintaktikailag a < és > jelek között adjuk meg. Itt is vannak nyitó és záróelemek, valamint üres elemek. Az elemekhez attributumok, azaz elemtulajdonságok rendelhetők. Egy XML elemnek teljes egészében bele kell ágyazódnia a szülőelembe. A következő példa egy könyvleírás XML alakban történő megadását mutatja be.

```
<?xml version=”1.0" ?>
<könyvek>
  <könyv ikod="2">
      <cim> &X; </cim>
       <ev>2003</ev>
       ISBN1234
   </könyv>
</könyvek>
```

Az XML dokumentumot akkor tekintünk szabványos dokumentumnak, ha teljesít bizonyos elemi formai követelményeket. A szabvány szerinti előírásoknak megfelelő dokumentumokat helyesen formált dokumentumoknak nevezik.

A helyesen formáltság az alábbi előírásokat jelenti:

- csak egyetlen gyökérelem van

- minden elem teljesen befoglalt a szülő elemben

- a kéttagú elemnek van nyitó és záró tagja

- az egytagú elem jele <nev attributumok />

- az attribútumok értékét macskaköröm jelek között kell megadni

- elem neve tetszőleges egytagú név

- attributum neve tetszőleges egytagú név

- a dokumentumot egy <?xml ..> feldolgozó elemnek kell kezdenie

A jelölőelemek egyértelmű azonosítása végett lehetőség van az elem felhasználási kontextusát is megadni. Ez egyértelműsítés a névtér használatán keresztül valósul meg. A névtér az elem neve előtt szerepel prefixként. A névtér előtti prefixhez tartozó teljes névtér specifikáció az xmlns jellemzőn keresztül adható meg.

```
<books:cim xmlns:books=”http://a.b.hu/konyvek”>
        Egri csillagok
</books:cim>
```

A névtér nem szükségszerűen mutat egy létező URL címre. A névtér azonosító tetszőleges szöveg lehet, célszerű az egyediség megőrzésére gondolni a kiválasztásakor. A névtér a dokumentumban csak a prefixen keresztül szerepelhet. A prefix a specifikáló xmlns –t tartalmazó elemben és annak leszármazottaiban érvényes. A leszármazottakat foglalja magába a szülőelem.

A névtér elsődleges szerepe, hogy jelzi a feldolgozó programnak mely elemek szólnak neki, mely elemek tartoznak hozzá. Az XML dokumentumban előfordulnak olyan szövegek is, melyek tartalmaznak foglalt karaktereket. Egy krakter azért foglalt, mert a szabványban neki már rögzített jelentése van. Ilyen karakter a < jel, hiszen ez a tagok határoló jele. A normákl szövegben nem fordulhat elő ilyen elem. Ha ilyen karaktert kell megadni a szövegben, akkro azt csak egy helyettesítő jelen keresztül lehet megoldani. A helyettesítés formátuma:

&név;

A név a helyettesítőt azonosítja.

A XML szabvány legfontosabb előnyei:

- platform független tárolási mód

- szabad struktúra kialakítás

- szabad jelentés társítás

- gazdag szabvány feldolgozó keretrendszerek

Az XML dokumentum fő előnye, hogy az adatok értéke mellett a az adatok jelentését is meg lehet adni az elemnéven keresztül. Az XML szabvány másik fő előnye, hogy tetszőleges hierarchia alakítható ki belőle, nincs megkötés a mélységre vagy komplexitásra.

Az XML világhoz tartozó feldolgozó programok közül az alábbiakat emeljük ki:

- XML strukúra ellenőrzése: DTD és XMLSchema
- XML állomány konverziója. XSLT
- XML feldolgozása kezelő programokból: SAX, DOM
- XML adatkezelés: xQuery

Az XML elterjedése hatalmas méretek öltött, szinte minden területen XML formátumú tárolási mód is megtalálható. A DTD szabvány egy egyszerűsítet séma nyelv, melynek célja a dokumentumban megadható XML hierarchia felépítésének korlátozása. A DTD.ben megadható, hogy egy elemnek milyen gyerekelemei lehetnek, azaz milyen elemekből állhat össze a dokumentum. A gyerekelemeknél a sorrend megadása mellett megadható az egyes elemeksó számossági előírása is.

<ELEMENT konyv (cím, ar, ev?, kiado, témakör*) >

A további fontos alkalmazások közül most egyet emelünk ki, a SOAP szabványt. A SOAP szabvány az alkalmazások világhálón keresztüli szabad és rugalmas üzenetváltásra szolgál.

A SOAP által elvégzendő feladatok:

- üzenetek csomagolása

- funkciók kódolása

- adatok konvertálása

- hibajelzések kódolása

A SOAP üzenet úgynevezett boríték egységben megy át a másikoldalra. A boríték fej és törzsrészből épül el. Az alábbi mintából jól látszik, hogy a fejrész (Header) megadja a megcímzett célszolgáltatást. A törzsrész (Body) megadja a igényelt rutin nevét (itt most a Caculator nevű alkalmazást) és megadj a híváskor átadandó paramétert (itt a paraméter neve n1 és értéke 3).

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/”>

<SOAP-ENV:Header>
        <t:transId xmlns:t=“http://a.com/trans”>345</t:transId>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
        <m:Add xmlns:m=“http://a.com/Calculator”>
        <n1>3</n1>
        </m:Add>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A visszaküldött válasz egy hasonló felépítésű XML csomagban fog visszajutni a partnerhez.

4. INFORMATIKAI RENDSZEREK SZOFTVER FEJLESZTÉSE

4.1. Szoftver fejlesztés alapjai

Informatikai rendszerek fejlesztésének eredménye általában egy komplex, heterogén rendszer, amely már létező rendszereket használ vagy szolgál ki. Sokan azt hiszik, hogy a fejlesztés kimerül a programozásban, de attól sokkal több.

A programkészítés egy absztrakciós folyamat, amelyben a valós világban létező jelenséget (a megoldandó problémát) valamilyen programozási eszköz absztrakciós szintjén képezünk le. A program a valós világ egy szeletének működő modellje. Ebben a szemléletben a valós világ egy absztrakt modelljét (analízis modell) kell leképezni egy programozási eszközre. Ezt a folyamatot megkönnyíti az, ha az analízis modell elemei könnyen leképezhetők nyelvi elemekre.

Miután a megrendelő cég elemezte működését, folyamatainak, termékeinek pontos költségét, egy új rendszer kifejlesztésével bíz meg egy céget:

- ha az új rendszer költséghatékonyabban,
- és/vagy biztonságosabban fogja tudni elvégezni a régi feladatot, vagy
- új feladatok merültek fel, és a régi rendszer már nem tudja elvégezni azokat.

A cél, hogy mindenki elégedett legyen, azaz a megrendelő a megfelelőrendszert kapja meg a magadott határidőre. A fejlesztő cégnek pedig az, hogy tervezhető ráfordítással pénzügyi haszna keletkezzen. A fejlesztőnek – bár komoly ráfordítást igényel – dokumentálnia kell a projekt minden fázisát, menedzselnie kell a csapatmunkát. Amikor egy csapat kifejleszt egy rendszert, akkor a megfelelő információk és a saját tapasztalataik alapján döntenek valamilyen technológia alkalmazása mellett, valamilyen hardver elemek (szerver, érzékelők, célhardverek) használatával. Manapság a szoftver fejlesztés csapatban történik, a megbízhatóság és hatékonyság érdekében. A fejlesztők a lehető legtöbb elemet újrahasznosítják (algoritmusokat, osztályokat, alrendszereket), ezért legtöbbször használnak tervezési mintát. A jól dokumentált elvek, kódok a kiszámítható létező elemeket használó hatékony fejlesztés elengedhetetlen feltételei. Mivel manapság nagy rendszerek osztott logikán alapulnak (teljesítmény elosztás, megbízhatóság, biztonság miatt), a rendszer tervezéséhez először modelleket alkotunk. Sok esetben a modellek alkotása (vagy későbbi elemzése) során felbukkannak az ellentmondások az esetleges hibák, hiányosságok. Minél hamarabb derül ki egy elv elgondolás hibája, annál hamarabb, kisebb költségen javítható. Manapság

a nagy rendszereket részeire (alrendszereire) bontjuk, definiáljuk azok közötti interfészeket, majd a részeket külön-külön megvalósítjuk. A részeket könnyebb átlátni, ezért a fejlesztés csak a részek problémáira koncentrál. A részeket a végén (az előre definiált módon) illesztjük. Egyik előnye, hogy a részeket párhuzamosan lehet fejleszteni (nem kell várni a függőségek miatt). Ha egy alrendszer egy másiktól függ, és még nincs kész, akkor egy demó alrendszer kifejlesztésével

4.2. Programozási technikák

Monolitikus programozás

"Egy programozó - egy feladat - egy program" alapelvekre épülő, mára teljesen használhatatlanná váló programozási megközelítés, amelyet a számítástechnika hajnalán alkalmaztak. Kisebb feladatokra még használható, de termék készítésére már nem. Kezdetben a programozási feladatok egyszerűek voltezért működött ez a modell. Az "egy programozó" a megoldások algoritmusait átlátta, a programokat egymaga lekódolta. Manapság a nagy rendszereket részekre osztják, akik nem ismerik a megvalósítás pontos menetét. Csak rendszerben és alrendszerben gondolkodnak. A programok általában lineáris felépítésűek voltak, kevés elágazással és ciklussal, nem volt modularitás (tulajdonképpen egy nagy függvényt irtak), a rétegezettség teljes hiányában. A kódok újrafelhasználása nem volt szokás, ha előjött hasonló feladat, akkor a létező kódot átírták. A programokat viszonylag gyorsan meg lehetett írni, ezért nem is volt különösebb belső struktúrájuk.

Moduláris programozás

A moduláris programozás azt jelenti,hogy a problémát olyan részfeladatokra bontjuk, amelyeknek a bonyolultsága már nem okoz gondot, amit már egy modulban - de magát a modult továbbra is monolitikusan leprogramozva - meg tud írni egy programozó, azaz csökkentjük a probléma bonyolultságát. Ha több ember dolgozik egy munkán, akkor az elvégezendő feladatot szintén részekre kell bontani, és a részeknek az összekapcsolását, összekapcsolódását meg kell tervezni. Itt is az a megfelelő megoldás, hogy a programokat bontsuk modulokra. A részek közötti együttműködési felületet interfésznek hívjuk, a programozási módszert moduláris programozásnak.

Strukturált programozás

Strukturált programozás esetén a program alapvetően programblokkokból áll. A blokkok vezérlési szerkezetekkel építhetők össze, alapvetően rákövetkezéssel, elágazással, és ciklussal.

Procedurális programozás

Procedurális programozásról beszélünk, ha a programozási feladat megoldását egymástól többé kevésbé független alprogramokból (procedure) építjük fel. Az alprogramok általában a programozási nyelvben is jól körülhatárolt formában jelennek meg. Rendszerint egy alprogram feladatát a nevében írjuk le, és a működés pontos feltételeit paraméterként adjuk meg minden alprogramnak minden futtatáskor. Sok esetben az alprogram eredményét egy érték jelzi, amit visszatérési értéknek nevezünk. Egy osztást végző alprogram esetén például az osztandó és az osztó a paraméterek, a hányados pedig a visszatérési érték.

Egy procedurális program futása lényegében hívások és paraméterátadások sorozataként fogható fel. A főprogram meghív egy alprogramot valamilyen paraméterekkel, és annak visszatérési értékei (esetleg az ún. kimeneti paraméterek értékei) függvényében további alprogramokat hívhat meg, amik szintén használhatnak alprogramokat a részfeladatuk elvégzéséhez.

Objektumorientált programozás

Objektumorientált programozásnak nevezzük azt a szoftverfejlesztési módszert, melyben a problémát (rendszert) a való világ tárgyaihoz (objektumaihoz) hasonlatosan önálló, önmagukban zárt, de egymással interakcióra képes elemekre bontva oldjuk meg. A megközelítés tükröződik az e módszer szerint kialakított program forráskódjában is. Az egyes objektumokhoz tartozó műveletek kódja és adatterületük leírása általában egymáshoz közel, de az eltérő típusú objektumok kódjai élesen elválasztásra kerülnek egymástól. Ez nagyban növeli a kód átláthatóságát és bővíthetőségét.

Az objektum egyediséggel rendelkező diszkrét entitás, melynek a vázát az osztály szolgáltatja, jellemzői:

- attribútumok (adattagok), műveletek (operációk): az attribútumok együttese szolgáltatja az objektum állapotát, ennek időbeli változása az objektum jellemzője.
- műveletek(operations): ezek modellezik az objektum viselkedését.

Objektum orientált programozás alapelvei:

1. Osztály (class) egy önálló hatáskörrel rendelkező absztrakt adattípus, amely
   •adattagokból(az attribútumok modellezésére) és
   •módszerekből(a műveletek modellezésére) áll.

Ez a minimális követelmény, de egyes nyelvek új elemeket is bevezettek (C#-ban definiálhatunk tulajdonságot, eseményt is). Valójában az objektumok közös elemeit definiálja. Garantált, hogy ha az osztályból bármennyi objektumot hozunk létre, akkor mindegyiknek az osztályban definiált elemei lesznek.

2. Objektum (object) egy osztály egy működőképes példánya, aminek állapota van (időről időre változhat).Egy adott osztályból tetszőleges számú objektum példányosítható. Szinonim a file processz fogalmakkal.
   •Minden objektum természeténél fogva különbözik az összes többitől (memória más helyén található).
   •Egy adott osztályból példányosított valamennyi objektumnak ugyanolyan lehetséges viselkedés módjai (műveletei) vannak, de saját állapotuk van.
   Programozás technikai szempontból egy típus, azaz tetszőleges számú változó hozható létre belőle, amely referenciákat tárolhat egy objektumra.

3. Egységbezárás (encapsulation)
   Az osztály az adatait és a módszereket egy egységgé teszi.
   •az adatok és a módszerek lokálisak. Saját hatáskörrel (osztály hatáskör) rendelkezik az osztály. Ha valamely elemet privátként definiálunk, akkor a külvilág számára elérhetetlen, viszont a többi osztály elem hozzáférhet (pl.: metódus).
   •Egyes módszerek, amelyek az osztály felületét szolgáltatják, használatához szükséges információk nyilvánosak (név, paraméterlista, visszatérési érték), azonban a megvalósítás módja, a függvény törzse nem látható.

4. Információrejtés (informationhiding)
   Egy objektum adatai a külvilág (más objektumok) számára hozzáférhetetlenek, privátok.
   •Egy objektum a külvilággal csak az interfészén keresztül tarthatja a kapcsolatot. Interfész: a külvilág számára elérhető módszerek együttese. Az interfész – legtöbb programozói nyelv esetén – csak publikus módszereket tartalmazhat. A programozó megteheti, hogy nyilvános adattagokat is definiál, de akkor az ellenőrzése nélkül lehet az osztály változóit állítani, és így az objektum egy érvénytelen állapotba kerülhet. Aki az osztályt elkészíti, az tudja pontosan, hogy az osztálynak mely eleme mit reprezentál, mi a feladata, milyen értékeket vehet fel. Ha valaki át akarja állítani kívülről, akkor csak a publikus függvényeken keresztül teheti. A nyilvános függvény azonban csak akkor írja a megfelelő adattagokat át, ha az új érték helyes. Az is lehetséges, hogy egy függvény egyszerre több adattagot is

átállít. Ha az adattagok nyilvánosak, akkor a használóé lenne a felelősség, hogy az objektum ne kerüljön érvénytelen állapotba, és ha nem ismeri pontosan a felépítését, akkor hibát eredményezhet. Másik ok, hogy ha több helyen használják az osztályt (jól), és meg kell változtatni a beállítás menetét, akkor több helyen kell, ami újabb veszélyforrást rejt magában.

•A módszerek implementációja rejtett.

5. Üzenet (message)

Az objektummal valókommunikáció módja. Az osztály egy metódusa egy másik metódusból (sajátból vagy akár másik osztályéból) meghívható.

A módszerek aktivizálását (invocation) jelenti. Minden programozási nyelvben definiálni kell a program belépési pontját, ami általában egy függvény. Sok esetben már ez is egy osztály speciális függvénye, ahol objektumokat hozunk létre és fv.-eket hívunk meg. A cél az lenne, hogy létrehozzuk a megfelelő osztályokat, majd egy fő osztályból létrehozzuk a megfelelő objektumokat. A mai modern operációs rendszerek esemény vezéreltek, azaz a felhasználó reakciói, vagy hardver megszakítások hatására események keletkeznek. A mi feladatunk metódusok kijelölése az eseményekre. Ezekben az esetekben a fejlesztői rendszer készítői egy SDK-tadnak a programozó kezébe, amiben már számos osztály és metodika rendelkezésre áll. A recept azonban nem változott objektumok és üzenetváltás. (Sok esetben a megvalósítás részleteit nem látjuk, de a háttérben objektumok használják egymást, csak mi nem látjuk.)

6. Öröklődés (inheritance)

Hierachikus kapcsolat(rendszer).rendszert lehet kialakítani. Lehetőség van arra, hogy egy osztály definiálása során egy ősosztály adjunk meg. Ilyenkor az osztály a megadott ősosztály leszármazottja lesz.

•A leszármazott osztály örökli az ős osztály elemeit (mindegyiket), de egyeseket nem érhet el közvetlenül.

•Az örökölt módszereket felül definiálhatja a saját függvényeivel (pontosítja azt).

•Új (saját, csak rájellemző) adatokat és módszereket definiálhat.

•Egy leszármazott osztály csak bővítheti, adatokat vagy módszereket. Pontosíthatja az őst, de nem utasíthat el örökölt és módszereit.

7. Polimorfizmus (polymorphism)

Bizonyos elemek viselkedése attól a környezettől függ, amelyben alkalmazzuk.

A gyakorlatban ez azt jelenti, hogy egy nyelvi elem (például egy kódrészlet) attól függően, hogy hol alkalmazzuk és hogyan alkalmazzuk, más-más működést eredményezhet. A környezet lehet paraméterlista (azonos függvénynév, eltérő paraméterszignatúra), operátorok esetén operandus, virtuális függvényes esetén a pointerben, referenciában tárol aktuális objektum típusa.

Az információrejtés és az öröklődés miatt a legtöbb oop nyelv legalább 3 hozzáférési kategóriát használ:

Megnevezés	Leírás
private (saját)	Az elem csak az adott osztály hatáskörben használható.
protected (védett)	Az elem az adott osztály és a leszármazott osztály elemei számára elérhető.
public (nyilvános)	Bárhol használható ahol az osztály ismert.

Vannak nyelvek amelyek többet definiálnak, pl.: a C# bevezeti a szerelvény (assembly) szintű elemeket is. Ha valahol olyan elemet használnánk, amit nem lehet minden esetben fordítási hibát eredményez. Ha valamely programozási nyelv támogatja mindegyik alapelvet, akkor objektum orientált nyelvről beszélünk.

4.3. Szoftver projekt paraméterei, fázisai

A szoftver fejlesztése során több fázison keresztül fejlesztjük ki a végleges rendszert. Több módszertan létezik, amely ajánlásokkal, elméleti és gyakorlati javaslatokkal segítik a fejlesztést, de a legtöbb ezek közül hat lépést határoz meg.

Analízis

Ezen fázis fő feladata a megrendelő követelményeinek begyűjtése, rendszerezése. Problémát jelent, hogy a felhasználó sokszor nem tudja pontosan, hogy mit is akar. Másik probléma, hogy a szakterületen járatlan fejlesztők és a megrendelő között félreértések lehetnek, vagy információ hiány léphet fel. A megrendelő nem említi, hiszen ezt „mindig így szoktuk csinálni”, a fejlesztő viszont nem ismeri az íratlan szabályokat. Az információ begyűjtésére különböző technikák léteznek kérdőívek, riportok, beszélgetés, stb.. Célszerű azokat a személyeket felkeresni, akik a jelenlegi rendszerbe a munkát végzik, és megfigyelni munkájukat, elbeszélgetni velük.

Specifikáció

Ebben a fázisban az előzőleg azonosított funkciókat tovább részletezzük. A kifejlesztendő rendszer funkcionális és nem funkcionális követelményei alapján meghatározzuk az alkalmazandó technológiát. Ezen fázis végén létrejön egy dokumentum, amely részletesen definiálja, hogy a rendszernek milyen szolgáltatásai lesznek, milyen módon kell működjön, milyen válaszidőkkel. Ezen dokumentum az alapja az elszámolásnak, amit általában aláírják a megrendelő és a rendszert fejlesztő képviselői is. Ezek után a megrendelő nem támaszthat újabb követelményeket a rendszerrel szemben. Ez sokszor nehéz feladat, hiszen a megrendelőnek merülnek fel újabb ötletei, de ha mindezt elfogadnánk, akkor két nagy hibát eredményezhetne:

- a kifejlesztett rendszer több ráfordítást igényelne, többlet költséget eredményez, de az árat már rögzítettük, így a készítőknek nem hoz hasznot,
- Egy későbbi ötlet alapjaiban rengeti meg a már megtervezett, esetlegesen félig kifejlesztett rendszer.

Ebben a fázisban a követelmények mellett rögzítjük a teszt eseteket és a sikeresség feltételeit, forgatókönyveket. Itt érdemes definiálni a felhasználói felületet is, mivel ez döntően befolyásolhatja az implementációt, amit a következő lépésben tervezünk meg.

Tervezés

Az előző lépésben dokumentált rendszer (diagramok, szöveges dokumentumok) teljes részletességű leírása a cél. Ebben a fázisban a cél: adatbázis szkriptek üres törzsű forráskódok létrehozása.

Implementáció

Az előző fázisok következményeiként generált forrás kódok függvény törzseit kell kitölteni. Ha itt új osztályokat hozunk létre, vagy új publikus függvényre van szükség, akkor vagy rossz az elgondolásunk vagy rosszak a tervek.

Tesztelés

A specifikációs fázisban definiált teszt esetek futtatása, teszt jegyzőkönyvek kitöltése a feladat. Nem elég olyan rendszert átadni, hanem pontosan azt a rendszert kell átadni. Tesztek futhatnak korábbi fázisokban is – pl. egységteszt – itt azonban a végső elfogadási tesztet futtatják.

Karbantartás, használat

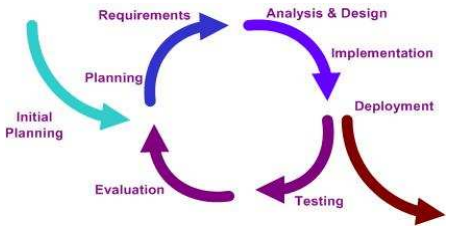
A szoftver átadása után egyes rendszereket folyamatosan karban kell tartani. Nem jellemző a szoftverekre a kopás, de az elavulás igen jelentős. A gyorsabb hardver, és operációs rendszer, a nagyobb méretű memória hatására újabb és újabb felhasználót kiszolgáló automatizmusok jelennek meg, újabb hardvereket szereznek be, amelyeket adoptálni kell a rendszernek. Vannak olyan alkalmazási területek, amelyek gyorsan változnak – például a könyvelés – ahol a rendszerek jól működnek, csak a helyes működés fogalma változik folyamatosan.

4.4. VIR projektek életciklusai, fejlesztési módszertan

Hagyományos, vízesés

Kezdeti, eredeti módszer, amely fázisok szekvenciájaként definiálja a fejlesztést. Egy lépés sem hagyható ki. Kezdetben hatékony volt, manapság nem alkalmazzák, csak az utódait. Nagy probléma, hogy nincs visszacsatolás, azaz, ha valamely pillanatban hiányosságok, ellentmondások jönnek elő, akkor a javítás vagy lehetetlen, vagy nagy költségek árán lehetséges. Nagy tapasztalat és szakmai tudás szükséges, hogy a hibák hamar kiderüljenek.

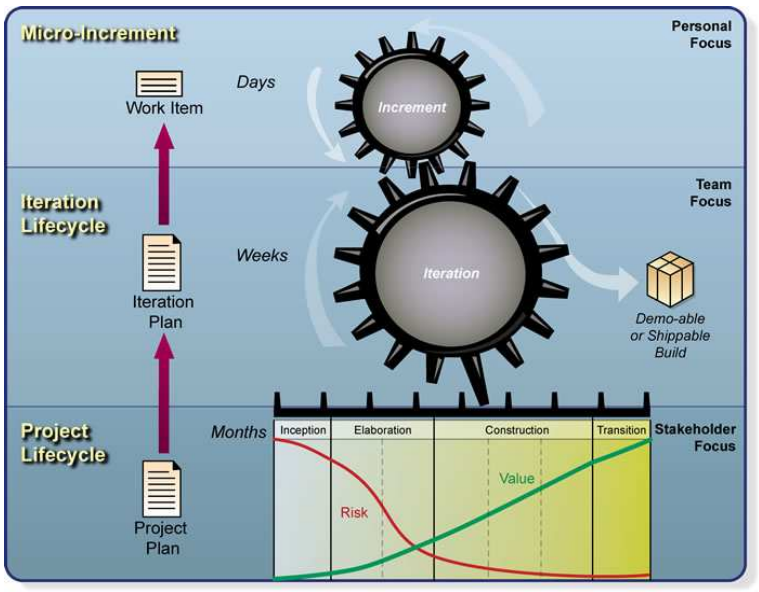
Evolúciós



4.1. ábra Evolúciós modell (KEP_A303_I_04_01) KEP_A303_I_04_01.JPG

RUP, OpenUP módszer

A Rational cég által kifejlesztett módszertan, amely számos dokumentum mintát is ad a fejlesztéshez.



4.2. ábra RUP (KEP_A303_I_04_02) KEP_A303_I_04_02.JPG

Az **Előkészítés (inception)** fázisában a rendszer eredeti ötletét olyan részletes elképzeléssé dolgozzuk át, mely alapján a fejlesztés tervezhető lesz, a költségei pedig megbecsülhetőek. Ebben a fázisban megfogalmazzuk, hogy a felhasználók milyen módon fogják használni a rendszert és hogy annak milyen alapvető belső szerkezetet, architektúrát alakítunk ki.

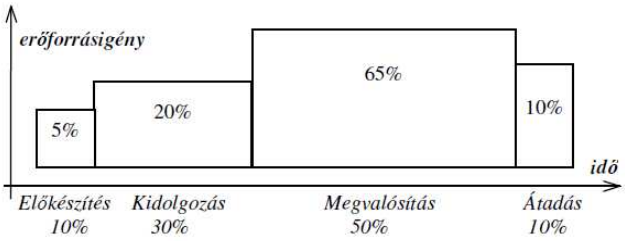
A **Kidolgozás (elaboration)** fázisában a használati módokat, a „használati eseteket” részleteiben is kidolgozzuk, valamint össze kell állítanunk egy stabil alaparchitektúrát (architecturebaseline). A UnifiedProcess készítőinek a képe alapján a teljes rendszer egy testnek tekinthető, csontváznak, bőrnek és izmoknak. Az alaparchitektúra ebből a bőrrel borított csontváz, mely mindössze a minimális összekötő izomzatot tartalmazza, annyit, amennyi a legalapvetőbb mozdulatokhoz elegendő. Az alaparchitektúra segítségével a teljes fejlesztés folyamata ütemezhető és a költségei is tisztázhatóak.

A **Megvalósítás (construction)** során a teljes rendszert kifejlesztjük, beépítjük az összes „izomzatot”.

Az **Átadás (transition)** a rendszer bétaváltozatának kipróbálását jelenti, mely során néhány gyakorlott felhasználó teszteli a rendszert és jelentést készít annak helyességéről vagy a hibáiról és hiányosságairól. A rendszer javítása a rendszer módosítását, majd ezt követően újabb tesztelést jelent.

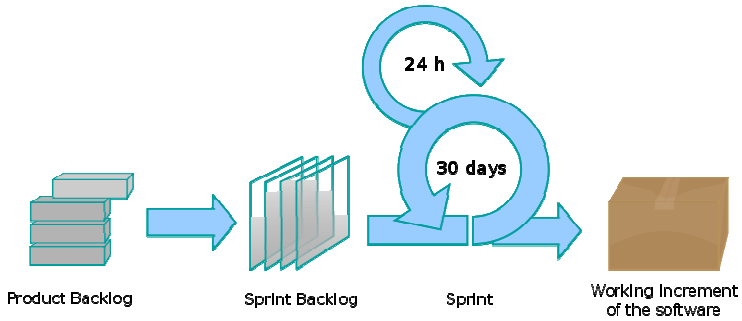
Minden fázis iterációkból épül fel, legalább egyből. Minden iteráció végén előáll egy rendszer, amely a végső rendszer szolgáltatásainak részét nyújtja, de már működik.

A módszer definiálja, hogy a fejlesztés során milyen dokumentumokat, diagramokat, forráskódokat — összefoglaló néven produktumokat (artifact) — készítsünk el. Az elkészítendő produktumok természetesen a megfelelő tevékenységekkel állíthatók össze, amely tevékenységeket pedig adott szakismerettel rendelkező személyek („dolgozók”), adott sorrendben hajthatnak végre. A tevékenységek, azok időbeli sorrendje és az azt végrehajtó dolgozók együttesen egy munkafolyamattal (workflow) írhatóak le.



4.3. ábra RUP modell fejlesztési szakaszok (KEP_A303_I_04_03) KEP_A303_I_04_03.JPG

Agilis módszer



4.4. ábra Agilis fejlesztési módszertan (KEP_A303_I_04_04) KEP_A303_I_04_04.JPG

Egy projekt általában egy nagy ötlettel kezdődik, amihez több kisebb kiegészítő ötlet társul. Ezt a terméket definiáló listát nevezzük itt *product backlog* listának. A projekt elindul, mielőtt még kidolgoznánk az egész rendszer részleteit. Ennek az az előnye, hogy hamar elkezdünk fejleszteni, valamint a változó követelményeket adoptálni tudjuk. Egy prioritásos lista a kezdet, ahol a fontosságot a B*usiness Value* határozza meg, ami azt jelenti, hogy azok helyezkednek elöl, amik a legtöbbet jelentik a termék számára, és ezekkel kezdünk el foglalkozni először.

A P*roduct Ownernek* (csoport(ok) főnök) képviseli és tulajdonolja ezt a listát. Ő definiálja a teljes rendszer képét a fejlesztőcsapatnak egy *kick-off* meeting keretében, hiszen mindenkinek tisztában kell lennie a termékkel. Ezt követően elkezdődnek az iterációk, amiket a Scrumban *sprintek*-nek nevezünk. A product owner és a fejlesztőcsapat összeül, és ennek a listának egy, a tetején lévő részét, odaadja a csapatnak, A fejlesztők ezek után valamilyen módszer szerint (ami lehet idő, de jobb a ráfordítás) megbecsli a feladatokat. Ezután a csapat határozza meg, hogy mennyi új elem fér bele az adott időintervallumba. Innentől nevezik ezt a listát *sprint backlog listának*. Innentől kezdve az aktuális szakasz (sprint) nem módosítható. Ennek az elemei a*story-k.*

A lista elemeit *taszkokra* bontják, amelyek már tovább nem bonthatók tovább. Egy story akkor lesz kész, amikor a hozzá tartozó összes taszk készen van. Az ideális sprint 30 naptári nap. A sprint végén a fejlesztők bemutatják az aktuális sprint eredményeit. Itt a GUI a mögöttes logikai és minden, ami szükséges egyszerre és akkor készül, amikor kitűzték.

4.5. Modell alkotás menete

Az On-line tranzakció feldolgozás OLTP (On Line TransactionProcessing), ahol cél a banki, működési folyamatok elemeinek, eseményeinek „azonnali” feldolgozás (adatbevitel, feldolgozás, tárolás, megjelenítés, lekérdezések). Az alkalmazás jellegét a következő táblázat foglalja össze:

Tulajdonság	OLTP	OLAP
Adatmennyiség	kisebb, tranzakciónkénti adatok	hatalmas adathalmaz
Feladat	az üzleti feladatok során keletkező napi, pillanatnyi adatok követése	Segítség a tervezésben a probléma megoldásban, a döntéshozásban.
Optimalizálás	rövid tranzakciók gyors kezelése, adatmódosítás	időnkénti jobok futnak, amelyek időigényes

Tulajdonság	OLTP	OLAP
Adatmodell	relációs adatbázis	nem mindig relációs modell
Feldolgozandó adat	általában kicsi, néha nagy	hatalmas
Adatforrás	homogén	heterogén
Felhasználók száma, hozzáférése	sok, olvasás és írás	kevés, általában csak olvasás
Műveletek	gyakori adatmódosítás, kevés adattal	lekérdezés hatalmas adattak, számos relációval
Rendelkezésre állás	állandó	időszakos

1. táblázat OLTP és OLAP alkalmazások használati jellege

Ebből a táblázatból látható, hogy az OLTP általában kis méretű adattal dolgoznak nagy rendelkezésre állás mellett, azonnali (vagy gyors) reakciók jellemzik, míg az OLAP hatalmas adatokkal dolgozik, akár több napig, de azt csak alkalmanként

Adatfolyam modell

Az adatfolyam-modellezés célja általában véve az, hogy egy adott információs rendszerről átfogó képet nyújtson, együtt ábrázolva a rendszer folyamatait és adatait, azaz részletesebben:

- A rendszerhatárok kijelölése
- A rendszer külső objektumainak meghatározása
- Kifelé és befelé áramló főbb információk meghatározása
- Belső információ-áramlás
- Információ-tároló helyek meghatározása
- Információt feldolgozó, átalakító folyamatok meghatározása
- Az adatfolyam-modellezés konkrét céljai az elemzés különböző fázisaiban:

Jelenlegi fizikai	A követelmények azonosítása (hiányosságok, új funkciók).
Jelenlegi logikai	Továbbvihető logikai folyamatok azonosítása, a rendszerszervezési alternatívák kiindulópontja.

Rendszerszervezési alternatíva	A felhasználói döntés előkészítése, átfogó kép kialakítása a lehetőségekről.
Igényelt rendszer	Funkciók, események meghatározásának kiindulópontja.

- Az adatfolyam-modell többszintű, hierarchikusan elrendezett adatfolyam-ábrák és a hozzájuk kapcsolódó elemi folyamatok leírásai, külső egyedek leírásai és bemenet/kimeneti leírások összessége. Minden adatfolyam-modellhez tartozó termék esetén meg kell jelölni az adott adatfolyam-modell változatát (jelenlegi fizikai, jelenlegi logikai, rendszerszervezési alternatíva, igényelt)

A technika rövid leírása

Az adatfolyam-modellezési technikát az elemzés legkorábbi fázisaitól kezdve a követelményspecifikáció elejéig (az igényelt rendszer adatfolyam-modelljéig) lehet használni. A megvalósíthatósági elemzés során átfogó kép kialakítása miatt van rá szükség, ami a jelenlegi környezet és az igényelt környezet vázlatos leírását jelenti, általában egy, esetleg két szintű adatfolyam-ábrák segítségével, a kiegészítő leírások nélkül.

A jelenlegi rendszer vizsgálata során először a jelenlegi fizikai adatfolyam-modell készül el, ami azon kívül, hogy közös fogalmakat alakít ki a működési területről a felhasználók és elemzők között, elsősorban a problémák, hiányosságok azonosítására szolgál. A fizikai modell már tartalmazza az összes kiegészítő leírást az adatfolyam-ábrák mellett.

Ezt a fizikai adatfolyam-modellt azután, összevetve az elkészült logikai adatmodellel, meg kell szabadítani a fizikai kényszerűségektől. Ezt hívják logikalizálásnak, vagy más szóval racionalizálásnak. Ennek során létre kell hozni a logikai adattár-egyed megfeleltetést, ami kapcsolatot létesít az eddig párhuzamosan fejlesztett logikai adatmodell és a logikai adatfolyam-modell között. Az így létrejött logikai adatfolyam-modell már a jelenlegi rendszer logikai képét mutatja, ami egy sor problémát eleve feloldhat (pl. többszörös adattárolás), de nem ez a célja, hanem az, hogy a jelenlegi rendszer továbbvihető, az új rendszerben felhasználható logikai folyamatait ábrázolja.

A logikai adatfolyam-modellt felhasználva a rendszerszervezési alternatívák kialakítása a következő fázis az adatfolyam-modellezés felhasználásában. Itt, hasonlóan a megvalósíthatósági elemzéshez, átfogó kép kialakítása a cél, ami segít az egyes alternatívák közötti különbségek bemutatásában. Itt sem kell kiegészítő leírásokat készíteni. Az alternatívákhoz tartozó adatfolyam-ábrák már általában logikai rendszerek képét mutatják, mivel a különböző logikailag lehetséges rendszerek működését

kell leírniuk. (Mint alternatíva, szerepelhet a jelenlegi rendszer fenntartása, aminek lehetnek fizikai vonatkozásai.)

A követelményspecifikáció elején, a választott rendszerszervezési alternatíva adatfolyam-ábráit ki kell egészíteni az új, eddig nem ábrázolt működésekkel (a követelményjegyzék alapján), illetve a mögöttes leírásokkal, a logikai adatfolyam-modellből kiindulva. A jelenlegi logikai adatmodellel meg kell teremteni a kapcsolatot egy megfelelően (át)alakított logikai adattár- egyed megfeleltetés létrehozásával. Az így létrejövő jelenlegi rendszer adatfolyam-modell az utolsó lépés az adatfolyam-modellezés használatában. Ezt a modellt a funkció meghatározás során kell majd felhasználni, mint a rendszer funkcióinak és eseményeinek a meghatározásában segítő fontos kiindulópontot.

Az adatfolyam-modellezési technika hasznos, mert az elemzés korai kezdeteitől fogva eszközt nyújt a felhasználók és az elemzők párbeszédéhez. Nem formális technika, azaz könnyen előállítható ábrákat produkál, az ábrák érthetőek, a hierarchikus elrendezés miatt adott részletességi szinteken könnyen áttekinthető ábrázolási módot nyújtanak. Az előnyeiből következnek a lehetséges hátrányai is, azaz a könnyű előállítás és a párbeszédes jelleg miatt az elemző esetleg túl részletes ábrákat készít, olyan dolgokat is ábrázolva, mint pl. sorrendiségi, időzítési információk, lekérdezések, fizikai feldolgozási részletek. Ezek az információk fontosak, de az elemzés illetve tervezés későbbi fázisaiban lesznek részletesen ábrázolva, az adatfolyam-modellezés során felmerülő ilyen típusú információk megfelelő helye a követelményjegyzék.

A technika által létrehozott vagy módosított termékek a következők:

- Adatfolyam-modell
- Adatjegyzék
- Logikai adattár-egyed megfeleltetés

Adatfolyam-modell

Az adatfolyam-modell a következő termékekből épül fel:

- Kontextusábra
- Adatfolyam-ábrák (hierarchikus halmaz)
- Elemi folyamatok leírása
- Külső egyedek leírása
- Bement/ Kimenet leírások

Az elemi folyamatok leírása az ábrákon szereplő azon folyamatokat írják le, amelyek tovább már nem bomlanak, tehát az ábrák alapján részleteikben nem értelmezhetők. A cél az, hogy a későbbi funkcióleírást ki lehessen alakítani. Az elemi folyamat leírásának utalnia kell az elérendő adatokra (a logikalizálás után erre a logikai adattár-egyed megfeleltetés utal majd), a működési szabályokra ("ha a folyószámlán szereplő összeg a kivét hatására nulla alá menne, akkor a Kivét folyamatnak ezt vissza kell utasítania"), a különböző lehetséges bemenetekre vonatkozó működési szabályokra ("A felvételi utalvány hatására a folyamat ellenőrzi a folyószámlát és kiadja a nyugtát, az egyenleg lekérdezés hatására a folyamat kiírja a folyószámla egyenleget").

Ha a leírás túl hosszú lenne, akkor át kell gondolni az elemi folyamat szétbontásának lehetőségét.

Az olyan elemi jellegű feldolgozási folyamatok leírását, amelyek több elemi folyamatra nézve közösek, közhasznú folyamatokként lehet felvenni az elemi folyamatok leírásai közé. Ezeket és a használó elemi folyamatokat kölcsönös egymásra hivatkozásokkal kell ellátni.

A külső egyedek leírásai minden külső egyedről leírják annak felelősségi körét vagy funkcióit a rendszerben, illetve a rendszerhez kapcsolódásának módját, ha ez lényeges (pl. egy külső számítógépes rendszer esetén a kapcsolódási felületet, interfészt)

A bemenet/kimenet leírások az alsó szintű, rendszer határait átlépő adatfolyamokat írják le, felsorolva az adatfolyam adatelemeit. Nem kell szerkezeti részleteket kifejezni (pl. ismétlődő adatelem csoportok vagy kötelezőség/ opcionalitás), de ha felmerülnek ilyenek, megjegyzésként fel lehet venni őket.

Adatjegyzék

Minden olyan elemi adatról, ami a rendszerhatárokat átlépő adatfolyamokon utazik, egy adatelem-leírást kell készíteni. Ebben az adatelem nevén kívül olyan információk kapnak helyet, amelyek az elemzés során kiderülnek, mint például ellenőrzési szabályok, alapértékek, számított értékek számításának módja, esetleg az adatelem mérete, példaértékek felsorolása. A több adatfolyamban is szereplő adatelemeket természetesen csak egyszer kell leírni, ez az egyik fő célja ennek a terméknek.

Logikai adattár-egyed megfeleltetés

Ez a termék a logikalizálás után minden adattárhoz hozzárendeli a kapcsolódó logikai adatmodellbeli egyedeket.

Az adatfolyam-modell a következő négy alapvető objektum típust használja:

Külső egyedek	A rendszeren kívüli objektumok
Folyamatok	Az információkat átalakító feldolgozási folyamatok
Adattárak	Az információk tárolási helyei
Adatfolyamok	Az információk áramlásának útvonalai

Ezen felül használható még a fizikai rendszer modelljében az anyagáramlás és anyagtár, ami az információn kívüli konkrét anyagáramlást ábrázolja (pl. alkatrészek raktározása, íróeszközök vételezése stb.)

Külső egyedek

A külső egyedek olyan objektumok, amik a rendszeren kívül vannak, és onnan információt kapnak vagy oda információt továbbítanak. Ezek lehetnek munka- illetve szerepkörök, mint Raktáros, Adminisztrátor vagy Jóváhagyó, külső szervezetek, mint MNB egy bank esetében vagy Parlament egy minisztérium esetében, külső információs rendszerek, mint Bérszámfejtés, Törvénytár, az információs rendszert használó belső szervezetek, mint Könyvelés, Propaganda osztály stb.

A külső egyedeket egy fektetett ovális jelöli. Minden külső egyedet egy kisbetű azonosít, ha a külső egyedek száma nagy, akkor két betű is használható. Ha egy ábrán egy külső egyed sok információáramláshoz kapcsolódik, akkor meg lehet sokszorozni, hogy a vonalak kereszteződését megakadályozzuk. Ilyenkor az összes előfordulást egy ferde vonallal meg kell jelölni.

Egy felsőszintű ábrán szereplő külső egyed egy alsóbb szintű adatfolyam-ábrán felbomolhat. Ilyenkor az azonosító betűt ki kell egészíteni egy sorszámmal. Pl. "c - Vezető" felbomolhat: "c1 - Osztályvezető", "c2 - Csoportvezető" külső egyedekre.

Az információs rendszeren kívül eső objektumok az adatfolyam-ábrákon csak külső egyedek lehetnek.

4.5. ábra Külső egyedek (KEP_A303_I_04_05) KEP_A303_I_04_05.JPG

Folyamatok

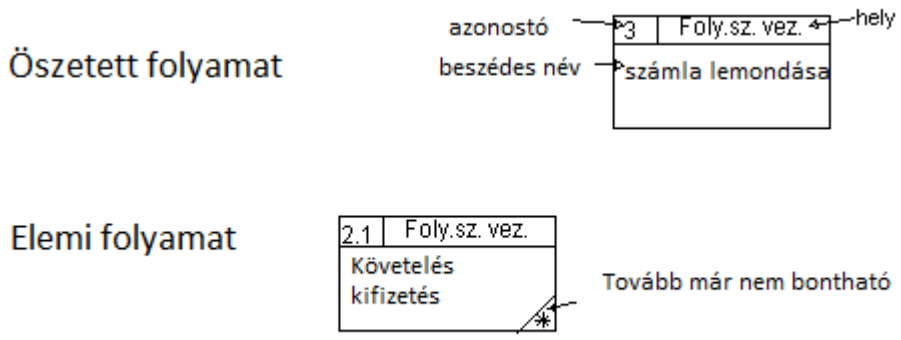
A folyamatok olyan átalakító tevékenységek, melyek a bemenő adatokat kimenő adatokká alakítják.

A folyamatokat egy doboz jelöli, a felső részén két kisebb, elválasztott területtel (azonosító és hely). Minden folyamatnak van egy azonosító sorszáma, de ez nem utal semmilyen sorrendiségre. Minden folyamatnak van egy neve, aminek lehetőség szerint egy aktív tevékenységet kifejező ige képzős alakját kell tartalmaznia. Jó nevek például: "Számla összeállítás", "Kérvény ellenőrzés", "Irat továbbítás", "Folyószámla tranzakciók felvitele". Rossz nevek ezzel szemben: "Számla kezelés", "Kérvény feldolgozás", "Irat nyilvántartás", "Folyószámla tranzakciók kezelése".

A fizikai modell folyamatain meg lehet jelölni a fizikai helyet is, ahol az a folyamat végbemegy, ami általában egy szervezeti egység, vagy egy munkakör neve lehet.

A folyamatok felbomolhatnak, ami tulajdonképpen az adat folyamábrák hierarchiáját kialakítja. A felső szinten szereplő folyamatok mindegyikéhez lehet rajzolni egy külön ábrát, ami az adott folyamat egyszerűbb alfolyamatait ábrázolja. Az ilyen alsóbb szintű folyamatokat a tartalmazó folyamat azonosítójával és egy azon belüli sorszámmal lehet azonosítani. Pl. a felső szinten szereplő "11 - Számla feldolgozás" alsó szinten felbomolhat "11.1 - Számla létrehozás", "11.2 - Számla iktatás" és "11.3 - Számla kiküldés" nevű folyamatokra.

A tovább nem bomló folyamatokat a jobb alsó sarokban csillaggal kell jelölni. Ezek lesznek az elemi folyamatok.



4.6. ábra Folyamatok (KEP_A303_I_04_06) KEP_A303_I_04_06.JPG

Adattárak

Az adattárak azok a helyek, ahol az adatok nyugvópontra jutnak a rendszeren belül. Egyik végén nyitott téglalap jelöli őket. Egy azonosítóval és egy névvel rendelkeznek. A rajz áttekinthetősége miatt ugyanazon adattárat meg lehet ismételni. Ilyenkor minden egyes előfordulást egy függőleges vonallal meg kell jelölni. A fizikai rendszer adattárai konkrét helyeket jelölnek, pl. Iratgyűjtő, Iktató könyv vagy egy adott számítógépes adatállományt (ha létezik). A logikalizálás után az adattárak már semmilyen fizikai tárolásra történő utalással nem rendelkezhetnek.

Kétféle adattár lehet: Állandó (vagy fő) adattár és átmeneti adattár. A fő adattárakat egy 'M' vagy 'D' betű, és egy tetszőleges egyedi szám azonosítja. A 'D' a számítógépes adattárra utal, az 'M' pedig a manuális, azaz kézi adattárra (ez utóbbit csak a jelenlegi fizikai ábrákon lehet használni). Az átmeneti adattárakat a 'T' (tranziens) betű és egy szám azonosítja, és olyan helyeket jelölnek, ahol csak ideiglenesen tartózkodnak az adatok, a bekerülés után a következő, ami történhet velük, az a kikerülés. Ha egy átmeneti adattár egyben manuális is, azt egy zárójeles 'M' jelöli a 'T' után.

Ha egy adattár egy alsóbb szintű ábrán jelenik meg, egy adott folyamat belsejében, akkor azt a betűjel után a folyamat azonosítója, egy '/' és egy sorszám azonosítja. Pl. a 2 folyamat belsejében egy adattárat a D2/1 azonosíthat. Ha egy szinttel lejjebb is van egy belső adattár, pl. a 2.1 folyamatban, akkor azt a D2.1/3 azonosíthatja.

Az adattárak alsóbb szinten felbomolhatnak. Ilyenkor az azonosítójuk a felbontott adattár azonosítójából és egy betű kiegészítésből áll.

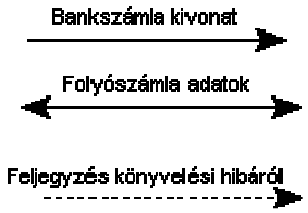
Adatfolyamok

A rendszerben mozgó információt az adatfolyamok fejezik ki, amiket nyilak jelölnek. A felső szintű ábrán csak a fontosabb adatáramlásoknak kell megjelenni, a részletek az alsóbb szintű ábrákon fejezhetők ki. Az alsóbb szintű ábrákon szereplő, az adott ábra határait átlépő adatfolyamoknak a felsőbb szintű ábrán is meg kell tudni feleltetni valamilyen adatfolyamot. Ez jelentheti azt, hogy felsőbb szinten egy adatfolyam alsóbb szinten többfelé bomlik. Kétirányú nyíl is használható, de csak felsőbb szintű ábrákon, annak kifejezésére, hogy alsóbb szinten bemenő és kimenő adatfolyamok is léteznek.

A rendszerhatárt át nem lépő, ún. információ áramlás is jelölhető az ábrákon, szaggatott nyíllal. Ez természetesen csak külső egyedek között lehet, és akkor érdemes használni, ha az ábrát érthetőbbé teszi.

Minden adatfolyamhoz tartozhat egy név, ami röviden utal a tartalmára.

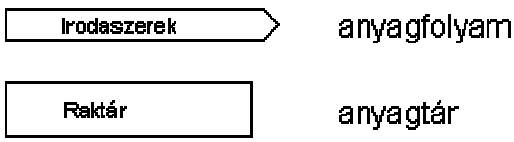
Az adatok a rendszeren belül csak egy folyamat hatására mozoghatnak, azaz nem létezhetnek közvetlenül adattárak közötti, illetve külső egyedek és adattárak közötti adatfolyamok.



4.7. ábra Adatfolyamok (KEP_A303_I_04_08) KEP_A303_I_04_07.JPG

Anyagáramlás

A fizikai anyagok áramlásának kifejezésére két objektum típus szolgál. Az egyik az anyagáramlás, ami egy belül üres, esetleg névvel ellátott széles nyíllal van jelölve. A másik az anyagtár, amit egy zárt téglalap jelöl. Anyagok áramlását csak a fizikai adatfolyam-ábrákon szabad jelezni, ha nincs megfelelő információ-áramlás, vagy kifejezőbb így az ábra. A logikalizálás során adatáramlással kell helyettesíteni.



4.8. ábra Anyagáramlás és anyagtár (KEP_A303_I_04_09) KEP_A303_I_04_08.JPG

4.6. BPMN diagram

Az üzleti folyamatok modellezése (Business ProcessModeling) az a fejlesztési lépés, ahol a vállalat jelenlegi folyamatait analizálják és kidolgozzák azok továbbfejlesztett módozatait. Alapvetően menedzserek készítik, alapvető cél a folyamatok hatékonyságának és minőségének a javítása. Az objektum orientált megközelítésben ez a specifikációs fázis első alapvető lépése.

A BPMN (Business ProcessModelingNotation) üzleti folyamatok modellezésének jelölésrendszerét definiálja az üzleti modellezés során. Gyakran használják web szervizek modellezésére is. EZ egy szabvány (OMG által felügyelt). Nagyon hasonlít az UML aktivitás diagramjára, de általánosabb, és több elemet ismer, komplex rendszereket is képes leírni. Elsődleges célja egy egységes jelölésrendszer definiálása, amely könnyen érthető minden üzleti szereplő számára. A szereplők lehetnek:

- analízist végző személyek – akik elemzik a rendszer folyamatait,
- programozók – akik implementálják a rendszert,
- menedzserek – akik felügyelik a rendszer fejlesztését.

Egy közös nyelv, amely áthidalja ezen három szereplő alapvetően eltérő szemléletét és feladatát. Másodlagos célja a komplex feladatok hatékony leírása, amit hatékonyan le lehet fordítani üzleti futtató nyelvre. Valójában egy irányított gráf, aminek a csomópontjaiban tennivalók vannak, az élek pedig kijelölik a sorrendet. Az üzleti tevékenységek folyamatának modellezését a következőképpen teszi lehetővé:

- modellezzük az eseményeket, amelyek beindítanak egy folyamatot,
- magát a folyamatot, amit aktíválnak az események vagy más folyamat befejezése után következnek,
- a folyamat eredményeit.

A döntési pontok és a vezérlés újboli egyesülései az átjárók. A folyamatok alfolyamatokból állhatnak, amelyeket újab diagram írhat le. Ha egy folyamatnak biztosan nincs alfolyamata, akkor taszknak nevezzük.A folyamat „az elvégzendő tevékenységek” hálózata (gráfja) a modell legfelső szintű eleme, ennek részleteit mutatja be a diagram. Ha a diagramon egy újabb folyamat van (amit majd egy másik diagram ír le), akkor egy „+” jel van a belsejében. Ha nincs, akkor az a „tenni való” egy taszk.

Támogatja sávos jelölést, amivel azt is definiálhatjuk, hogy ki végzi a cselekményt. Alpvetően három esemény létezik:

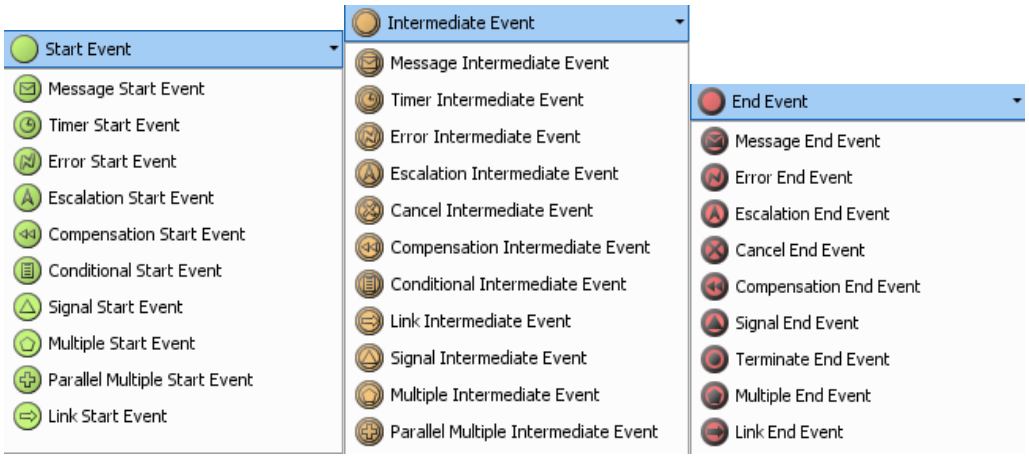
- start esemény, aminek hatására indul egy folyamat,
- közbenső esemény, valahol a komplex esemény folyam közben keletkezik,
- vég esemény, ami a keletkezik ha a folyamat leáll.

Ezeknek lehetnek módozatai, amiket a táblázatban is láthatunk. A módozatok komplex események jelölését szolgálják, ilyenek lehetnek: időzítő, üzleti szabály, hiba, feltételes esemény, … A komplex eseményeket a három alapeseménye belsejében elhelyezett ikonokkal jelöljuk.

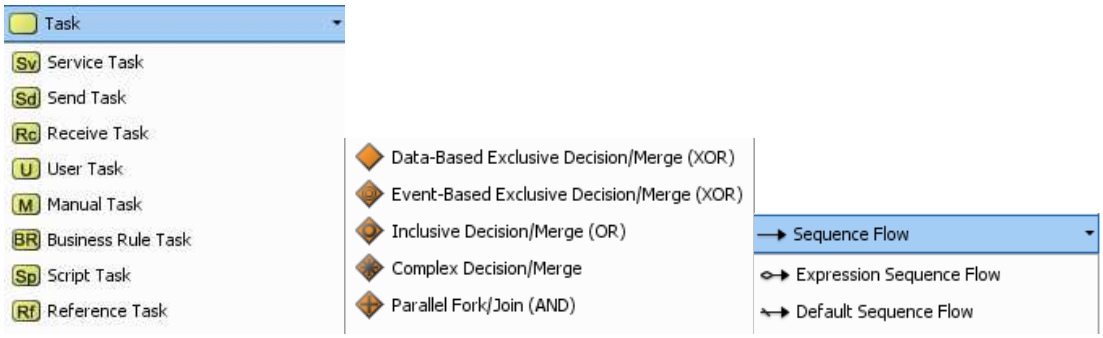
A következő elemeket definiálja (a lista nem teljes, csak az egyszerű elemekre szorítkozunk):

Megnevezés	Leírás	Jelölés
Esemény	Állapotváltozás Ok-hatás Eseménytípusok: Start, Intermediate, End	Jelölésük: lásd 4.09. ábra
Tevékenység	Atomi/összetett Taszk/alfolyamat	Jelölésük: lásd 4.10. ábra
Tevékenység csopot		pontvonallal rajzolt kerekített sarkú téglalap
Átjáró	Szekvencia konvergencia/divergencia	Jelölésük: lásd 4.10. ábra
Szekvencia	Tevékenységek sorrendje a folyamatban (nincs vezérlési folyamat a BPMN-ben)	Jelölésük: lásd 4.10. ábra
Üzenet	Két független folyamat részvevő közötti	szaggatott vonal

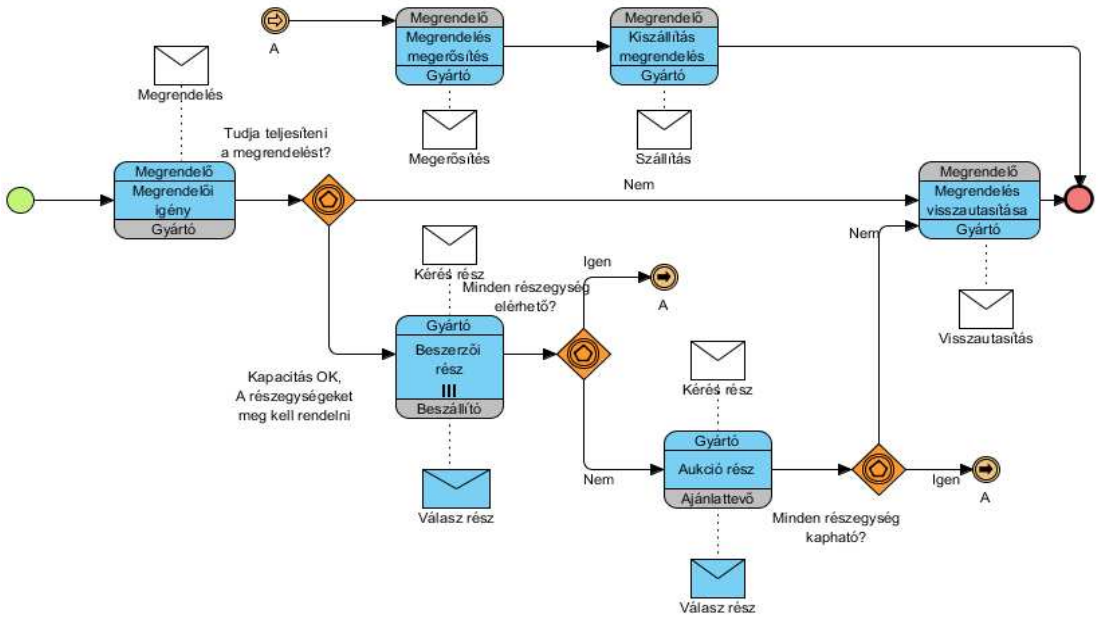
	információcsere	
Asszociáció	Adat hozzárendelés	pontvonal



4.09. ábra Események jelölő elemei (KEP_A303_I_04_09) KEP_A303_I_04_09.JPG



4.10. ábra Tevékenységek, átjárók, szekvenciák jelölő elemei (KEP_A303_I_04_10) KEP_A303_I_04_10.JPG



4.11. ábra Minta BPMN modell (KEP_A303_I_04_11) KEP_A303_I_04_11.JPG

Osztott technológiák

Manapság egyre inkább valamilyen osztott technológiát használnak, mert alkalmazásával biztosított a(z)

- központi logika (nehezebb kijátszani, feltörni),
- központi adatbázis (könnyen menthető),
- heterogén felépítés támogatása,
- olcsó, könnyen cserélhető elemek,
- robosztus, hibatűrő rendszer.

Alapvetően két fajta osztott rendszert különböztetünk meg: hagyományos és webalkalmazás(nem webalkalmazás) alapút. Egyik esetben mind a kliens, mind a szerver egy konzol vagy desktop alkalmazás, másik esetben a kliens leggyakrabban egy böngésző, a logika pedig a szerver oldalon fut, valamilyen konténerben. Gyakran keverednek a megvalósítás során Az első kategóriába tartozik a CORBA, az ICE, és a WCF, a másodikba pedig a WCF és a J2EE. A WCF mind a két kategóriába sorolható, mert minkét megoldáshoz vannak elemei. A webes megoldásra jellemző, hogy szabványokat (SOAP, SOA, …) implementáló, egymásra épülő rétegeken folyik a kommunikáció, ezért lassabb is.

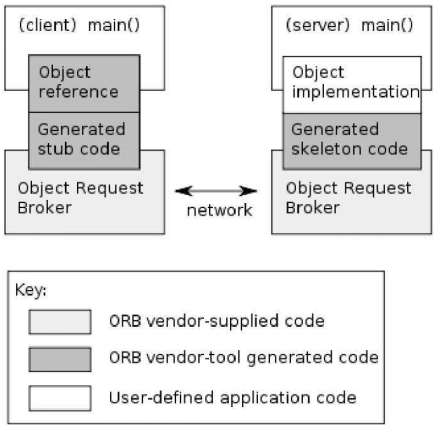
4.7. CORBA architektúra

A CommonObjectRequestBrokerArchitecture (röviden: CORBA) egy mechanizmus az objektumok közti metódushívási minták egységesítésére, függetlenül névtérbeli és fizikai helyzetüktől. Egy interfész definíciós nyelvet *(InterfaceDefinitionLanguage, IDL)* használ, mely segítségével leírja, hogy az objektum milyen interfészeket kínál a külvilágnak. Az így kapott generált IDL kód alapján számos programozási nyelvre készíthető leíró kódrész. A következő nyelveket támogatja: Ada, C, C++, Lisp, Ruby, Smalltalk, Java, COBOL, PL/I, Python. Léteznek más nyelvekre nem teljesen szabványos megoldások *ORB (objectrequestbroker)* implementációk által. Ezek a nyelvek: Perl, Visual Basic, Erlang, és Tcl.

A CORBA specifikációja előírja, egy ORB névszerver meglétét, melyen keresztül az alkalmazás más objektumokat felderít. Az ORB feladatai közé tartozik az objektum referenciák számontartása, objektum (és referencia) példányosítás, objektum életciklusának szabályzása, stb. Az Object Adapter regisztrálja a generált kód osztályainak példányait is. Ezek az osztályok a felhasználói IDL kód lefordításának eredményei, melyek a magas szintű interfészdefiníciókat öntik operációs rendszer és nyelvspecifikus osztályok alapjaivá, így téve felhasználhatóvá alkalmazásokhoz. Ez a lépés mindenképpen szükséges a CORBA infrastruktúrájú objektumokkal folytatott kommunikációhoz.

Egyes programozási nyelvek esetén könnyebb, másoknál kényelmetlenebb az IDL használata. Például a JAVA implementáció (a JAVA nyelv természetének köszönhetően) nagyon megkönnyíti és leegyszerűsíti a CORBA használatát. A C++ már nem ennyire triviális, de a CORBA minden funkcióját támogatja. A C implementáció még ennél is "nehézkesebb".

A nyelvspecifikus leíró kód generálásához szükség van a fejlesztő közreműködésére, vagyis IDL kód írására, mely reprezentálja majd az objektumhoz tartozó interfészeket. Általában a CORBA implementációk mellé jár egy IDL fordító, mely valamilyen nyelv specifikus kódot generál. Majd a célnyelv fordítója segítségével készíthetőek az alkalmazásokhoz felhasználható objektumfájlok. Az alábbi modell mutatja a CORBA felépítését a programkódok szempontjából.



4.12. ábra CORBA felépítése

(KEP_A303_I_04_12)

KEP_A303_I_04_12.JPG

Az IDL fájlban deklarált interfészeket az IDL-fordítók nyelv specifikus forrásfájlokra fordítják le. A szerver implementálja ezeket a függvényeket / metódusokat (skeleton file). így képessé válik a távoli metódushívások kiszolgálására. A kliens beépíti a generált stub kódot. így válik képessé távoli metódushívások kérésére. A szükséges hálózati, és egyéb folyamatokat a hálózati middleware transzparensen biztosítja.

A nyelv- és platform független távoli metódushívás (RemoteMethodInvocation, RMI) specifikáción túl a CORBA rendelkezik más gyakran használt szolgáltatásokkal is. Távoli kivételek kezelése, timeout,különböző hálózati protokollok.

A kliens és a szerver kifejezés nem annyira az alkalmazás adott részének biztos elnevezése, mint inkább szerepek megjelölése, melyeket az alkalmazás részei egy adott kérés erejéig betölthetnek. A kliens aktív entitás, szolgáltatásokra vonatkozó kérést küld a szervernek. A szerver passzív entitás, szolgáltatásokat nyújt válaszként a kliens kérésekre.
Gyakran a szerverek nem "tiszta" szerverek abban az értelemben, mert szerverként működnek egy adott kliens számára, de kliensként működnek egy másik szerver számára, hogy teljesítsék a saját kliensük kérését. Hasonlóképp gyakran a kliensek sem "tiszta" kliensek abban az értelemben, hogy csak kéréseket küldenek egy objektumnak. Ehelyett a kliensek gyakran kliens-szerver keverékek. Például egy kliens indíthat egy hosszan futó műveletet egy szerveren, a művelet indításának részeként pedig küldhet egy visszahívó (callback) objektumot a szervernek, melyet a szerver arra használ, hogy értesítse a klienst, ha a művelet véget ért. Az ilyen szerepcsere sok rendszerben

általános, így a kliensszerver rendszerek gyakran pontosabban meghatározhatók, mint peer-to-peer rendszerek.

4.8. További architektúra

ICE architektúra

Az ICE objektum-orientált middleware platform. Ez alapjában véve azt jelenti, hogy az ICE eszközöket, API-kat és programozói könyvtárakat biztosít objektum-orientált kliensszerver alkalmazások fejlesztéséhez. Az ICE alkalmazások alkalmasak arra, hogy heterogén környezetben használják őket: a kliens és a szerver íródhat különböző programozási nyelven, különböző operációs rendszereken és számítógép architektúrákon futtathatók, illetve hálózati technológiák nagy választékán keresztül kommunikálhatnak. Az alkalmazások forráskódja hordozható függetlenül a fejlesztőkörnyezettől.

Az ICE futtatókörnyezet nagy része beállítható tulajdonságokon keresztül. A tulajdonságok név-érték párok, mint pl. "ICE.Default:Protocol=tcp". A tulajdonságok jellemzően szövegfájlokban tárolódnak, és az ICE futtatókörnyezet olvassa be őket, hogy beállítson több lehetőséget, mint pl. a threadpool(elérhető fonalak száma) méret, a tracinglevel (nyomkövetési szint) és sok más beállítási paraméter.

Minden ICE objektumnak van egy csatolófelülete adott számú művelettel. A csatolófelületek, a műveletek és az adattípusok, melyeket a kliens és a szerver egymással cserél, a Slice *(SpecificationLanguagefor ICE)* nyelv segítségével vannak meghatározva. ASlice lehetővé teszi a kliens-szerver kommunikáció olyan meghatározását, mely független a programozási nyelvektől. A Slice definíciókat egy fordító az adott programozási nyelvhez való API-ra fordítja, azaz az API-nak az a része, mely az adott csatolófelületekről és típusoktól függ, generált kódból áll.
Azokat a szabályokat, melyek meghatározzák, hogy egy Slice szerkezetet hogyan kell egy adott programozási nyelvre fordítani, nyelvi leképezéseknek nevezik. Pl. a C++ leképezésnél egy Slice szekvencia egy STL vektorként jelenik meg, míg a JAVA leképezésnél egy Slice szekvencia egy JAVA tömbként. Ahhoz, hogy meghatározzuk, hogy egy adott Slice szerkezethez tartozó API hogyan néz ki, csak a Slice definícióra és a nyelvi leképezési szabályok ismeretére van szükség. A szabályok elég egyszerűek és általánosak ahhoz, hogy feleslegessé váljon a generált kód olvasása ahhoz, hogy megértsük, hogyan kell használni a generált API-t. Természetesen el lehet olvasni a

generált kódot. Mindazonáltal ez nem hatékony, hiszen a generált kód nem feltétlen alkalmas emberi felhasználásra. Jelenleg az ICE rendelkezik nyelvi leképezéssel a C++, Java, C#, Python, Objec-tive-C és a kliens oldalon a PHP és Ruby nyelvhez.

Az ICE olyan RMI protokollal rendelkezik, mely TCP/IP-t vagy UDP-t használhat alacsonyabb szintű átvitelként. Ezen túl az ICE lehetővé teszi az SSL (titkosított csatorna) használatát az átvitelhez, így minden kommunikáció a kliens és a szerver között titkosítható. Az ICE protokoll meghatároz:

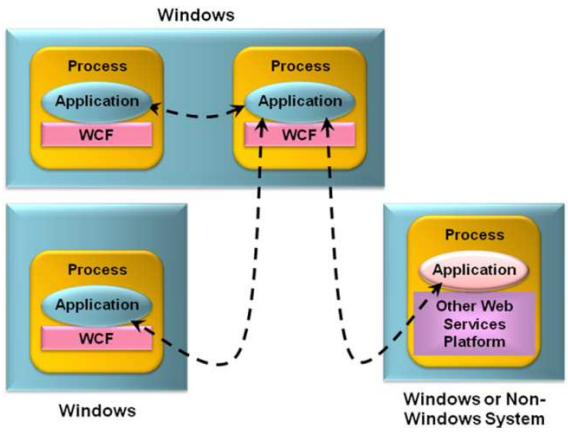
- Számos üzenettípust, mint pl. kérés és válasz üzenettípusok.
- Egy protokoll állapotgépet, mely meghatározza, hogy milyen sorrendben válthatók különböző üzenettípusok a kliens és a szerver között, együtt a TCP/IP ehhez tartozó kapcsolatlétesítés és szakadás szemantikájával.
- Kódolási szabályokat, melyek meghatározzák, hogy az egyes adattípusokat hogyan kell reprezentálni a hálózaton.
- Egy fejlécet minden üzenettípushoz. mely olyan részleteket tartalmaz, mint az üzenettípus, az üzenet mérete, a használt protokoll és kódolási változat.

Az ICE támogatja a hálózati tömörítést is: egy konfigurációs paraméterrel beállítható. hogy minden hálózati forgalom tömörítve legyen, hogy sávszélességet takarítsunk meg. Ez hasznos, ha az alkalmazás nagy mennyiségű adatot cserél a kliens és a szerver között. Az ICE protokoll alkalmas arra. hogy nagy hatékonyságú eseménytovábbító mechanizmust építsünk, mert megengedi, hogy egy üzenetet anélkül továbbítsunk, hogy ismernénk az üzenet belső részleteit. Ez azt jelenti, hogy az üzenetkezelő csomópontoknak nem kell az üzeneteket unmarshaling-olni és remarshaling-olni (ki- és becsomagolni) egyszerűen bájtok átlátszatlan pufferekként továbbíthatnak üzeneteket.

Az ICE protokoll támogatja a kétirányú kapcsolatokat is: ha egy szerver egy kliens által küldött visszahívó objektumnak szeretne küldeni egy üzenetet. a visszahívás történhet abban a kapcsolatban, melyet eredetileg a kliens hozott létre. Ez a sajátság különösen fontos, ha a kliens tűzfal mögött van, mely a kimenő kapcsolatokat engedi, de a bemenőket nem.

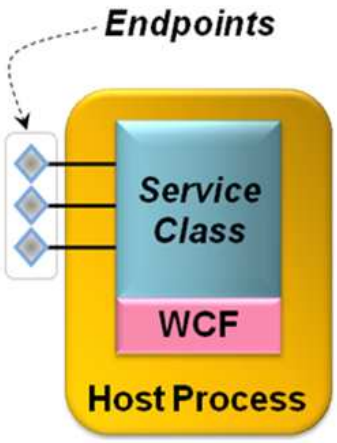
WCF

A Microsoft technológiája, amely lehetővé teszi a szervív orientál alkalmazások készítését. A következő ábrán láthatjuk a koncepcióját:



4.13. ábra WCF struktúra (KEP_A303_I_04_13) KEP_A303_I_04_13.JPG

Több féle adatátviteli protokollt is támogat (SOAP, bináris, …), így más rendszerben megírt szolgáltatásainkat is használhatjuk (pl. java). Végre kilépett a saját operációs rendszer világából és használhatunk más technológiákat. Kompatibilis a régi saját technológiájával a .NETremoting-al is, így a régi rendszerünket is elérhetjük, ha abban írtuk. Két fázisúcommit tranzakciót vezetett be, ahol a műveleteket visszagörgethetjük, vagy véglegesen érvényesíthetjük. Támogatja a RESTful web szervizeket, ahol a kérés nem SOAP, hanem egyszerű http GET vagy POST, így a szükséges sávszélesség kisebb. A szolgáltató alkalmazás készítésének 3 komponense van:



4.14. ábra WCF HOST process

(KEP_A303_I_04_14)

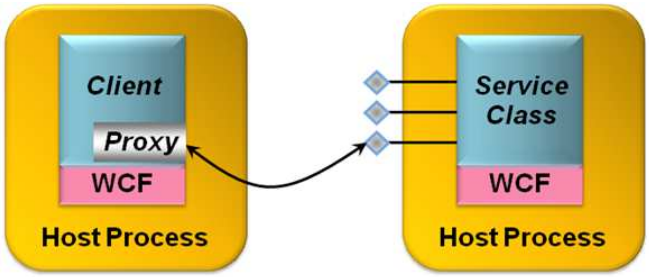
KEP_A303_I_04_14.JPG

e

- Szerviz osztály: bármelyik támogatott nyelven implementálható. Ez függvényeket tartalmaz, ami implementálja az üzleti logikát. Általában egy külön szerelvényben van, ami dll.
- Hosztprocess: beágyazza a szerviz osztály, lehetővé teszi annak az osztott használatát.
- Egy vagy több végpont (endpoint), ami biztosítja a szolgáltatás elérhetőségét. Ehhez csatlakoznak a kliensek. Definiálni kell a protokollját, és a pontos címét, portját (URL) és a szerződését (EndpointABC-jét, azaz Address-Binding-Contract).

Először létre kell hozni egy szerződést, amiben definiálják a szolgáltatást. A szolgáltatást használók fejlesztését már ezen a ponton el lehet kezdeni ennek a szerződésnek a birtokában.

A fogyasztó alkalmazás (kliens) tartalmaz egy proxy-t, amit a kapott szerződés alapján generálnak. A proxy osztály metódusait hívva valójában a távoli objektum fog műveleteket végezni.



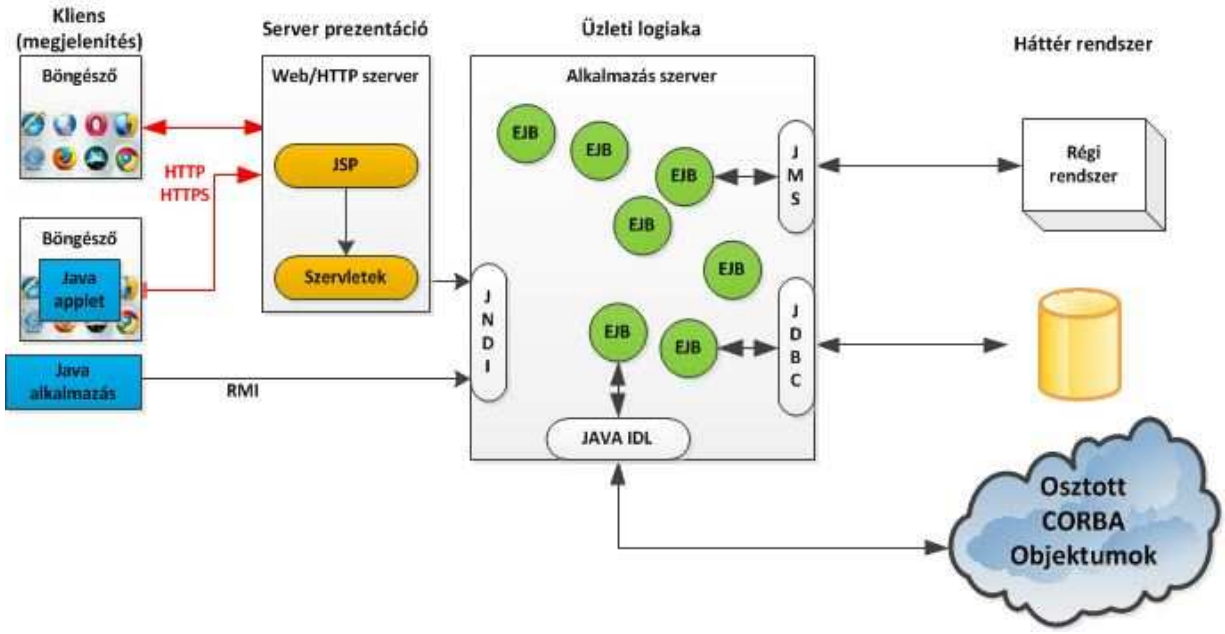
4.15. ábra A fogyasztó és szolgáltató komponensek együttműködésének részletei WCF-ben

(KEP_A303_I_04_15) KEP_A303_I_04_15.JPG

J2EE

A rendszer fejlesztése során a platform függetlenség, a heterogén elemek támogatása nagyon fontos. A JAVA nyelv biztosítja ezt a lehetőséget, sőt a J2EE egy komponensgyűjteményt és API-t is ad a

vállalati rendszerek fejlesztéséhez. A J2EE-vel készíthető rendszer architektúráját a következő ábra illusztrálja:



4.16. ábra N-tier alkalmazás architektúrája (KEP_A303_I_04_16) KEP_A303_I_04_16.JPG

Fejlesztők, rendszer szervezők számára a következő előnyöket nyújtja a endszer:

- Az üzleti logika önálló komponensek formájában implementált.
- A megjelenítési réteg könnyen testre szabható.
- Az üzleti logikát lehet használni konzol, és desktop alkalmazásokból egyaránt.
- A régi rendszerrel való kapcsolatot egy komponensen keresztül lehetővé teszi.
- A más nyelveken írt alkalmazásokhoz saját CORBA implementációja van.

A mi feladatunk a kék (JAVA applet és alkalmazás), a narancssárga (JSP, szervlet), és a zöld (EJB) elemek elkészítése. Manapság a legtöbb vállalati alkalmazás böngésző alapú, így az alkalmazás és appletkészítése csak ritkán fordul elő. Az EJB-k (Enterprise Java Bean) a vállalat logikáját megvalósító komponensek, amelyek bármikor lecserélhetőek, és használhatóak akár webalkalmazásból, akár sima JAVA alkalmazásból, akár bármilyen alkalmazásból a CORBA segítségével. Az EJB-k automatikusan bekerülnek. A legújabb verzióban kényelmesen annotációkkal tudjuk definiálni, használni az EJB-ket. Ha a rendszerhez kérés érkezik a következő módszer javasolt a kiszolgálásra:

1. A szervlet feladata általában az adatok fogadása a klienstől, azok meglétének és helyes értékének ellenőrzése.
2. Ha a művelet bemeneti paraméterei megfelelőek, akkor pedig meghívja a megfelelő EJB komponens(eke)t.
3. Az EJB már csak jó adatokat kapván elvégzi a műveleteket (adatbázis, fájlműveletek, ..).
4. A szervlet átirányít a megfelelő JSP oldalra, átadva annak az EJB által szolgáltatott adatokat.

Az ábrán látható ívelt oldalú téglalapokat (nincs kitöltve) JMS, JDBC, JNDI, JAVA IDL, a j2EE rendszerrel kapjuk, csak használni kell azokat.

Programozástechnikai szempontból a JSP egy kényelmes kiíratást biztosít. Az alapértelmezett mód a html, ha speciális jeleket teszünk a kódba (<% %>), akkor áttér java módba. Minden esetben átfordul szervletre, de ezt a rendszer automatikusan elvégzi. Ha csak műveletek kell végeznünk kevés kimenettel, akkor ezt használjuk. A JDBC egy kényelmes adatbázis független módját biztosítja az adatbázisaink elérésének, így ha áttérünk másikra, akkor nem kell teljesen átírni a kódot, elég csak néhány helyen. A JMS (**Java Message Service)** az üzenetküldést teszi lehetővé a komponensek között.*Lazán csatolt kommunikáció,* azaza szoftverkomponensek nem közvetlenül egymássalkommunikálnak, hanem egy köztes (üzenetkezelő) komponens létrehozás és beállítása után. Ennek az előnye, hogy az üzenetek küldőinek nem kell pontosan ismerniük a fogadókat, mert minden kommunikáció az üzenetsoron keresztül történik. Két modellt támogat, a P2P és feliratkozó / publikálót. Egyik esetben a küldő megadott pontosan ismeri a címzettet, és egy megfelelő sorba helyezi el üzenetét, amit a másik fél majd kiolvas onnan.

JNDI (*Java Naming and DirectoryInterface*) egy név szolgáltatás, ami lehetőség nyújt, hogy elemeket névvel lássunk el, és hierarchiába rendezzük.

Módszerek összehasonlítása

Arra a kérdésre, hogy melyik módszert is használjuk, nincs egyértelmű válasz. Abban az esetben, ha terület a fejlesztőknek ismert, és a leendő felhasználó türelmetlen, akkor az agilis módszer kiváló. A baj sok esetben az, hogy egy látszólag kis kérés, felhasználói igény óriási átszervezését eredményezi a rendszernek, ha arra nem számítottak a fejlesztők. A RUP valahol középen helyezkedik el a vízesés és az agilis között. A RUP esetében több lépésben – ahol minden lépés eredménye egy félkész rendszer – készül az alkalmazás, de minden esetben figyelembe veszik a végső rendszert. A következőkben röviden összefoglaljuk az említett fejlesztési módszertanokat.

Tulajdonság	RUP	Agilis
Reakciók a felhasználó igényeire	+++	+++++
Kód újraszervezésének szükségessége	++	-----
Folyamatos tesztelés	+++	+++++
Célok	+távlati és rövidtávú	-csak rövidtávú
Módszer teljessége	++++++	+++

Technológia

Függ attól, hogy milyen rendszert kell tovább fejleszteni, milyen környezetben fog majd futni a rendszer, melyik nyelv / technológia ismert a projekttagok által. A következő táblázatban összehasonlítjuk az említett technológiákat néhány szempont alapján (nem térünk ki mindenre, és a saját tapasztalatomat tükrözi):

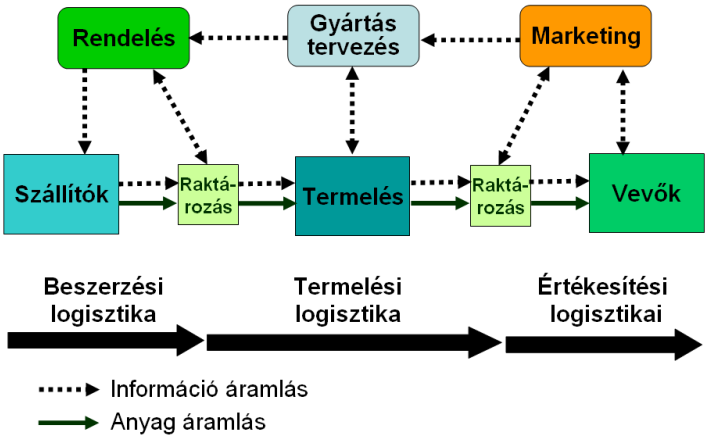
Tulajdonság	CORBA	ICE	WCF	J2EE
Programozhatóság	-	++	+++	+++
Sebesség	+++++	++++	+++	++
Platform függetlenség	+++++	+++++	-	+++++
Dokumentáció, segítség	+++++	++	+++++	+++++

5. LOGISZTIKAI MODULOK A VIR RENDSZEREKBEN

5.1. Speciális perifériák a logisztikai VIR rendszerekben

A logisztikai folyamat magába foglalja az ellátási lánchoz tartozó összes tényezőt a beszerzéstől a termelés kiszolgálásán keresztül, a raktározást, a készletgazdálkodást, az áruelosztást és az értékesítést is. A logisztikai folyamatot három nagy területre bonthatjuk:

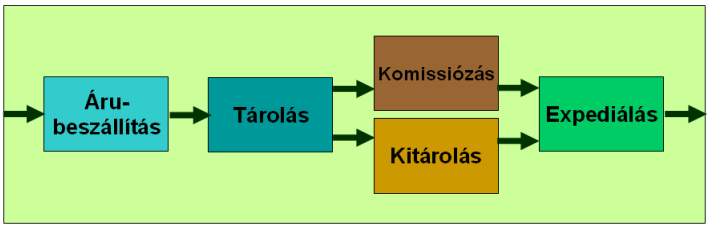
- Beszerzési logisztika, amely a vállalati tevékenységhez szükséges inputokat (alapanyagok, segédanyagok, félkész termékek) szerzi be és bocsátja a termelés rendelkezésére. A feladatokhoz tartozik a megfelelő beszállítók kiválasztása, a szállítási kondíciók meghatározása, a beérkezés ütemezése, beleértve a fenti tevékenységekkel kapcsolatos anyag- és információáramlást.
- Termelési logisztika, amely a termelési folyamat közben biztosítja az anyagok áramlását. Funkciója az anyagoknak a termelési folyamatba történő belépésénél kezdődik, és a késztermék raktárba érkezésével fejeződik be. Az anyag-, és információáramlás végig követi a termelési folyamat minden fázisát, beleértve az egyes fázisok közötti esetleges közbenső tárolást, várakozást is.
- Értékesítési logisztika, amely a termelésből kikerülő termékeket a piacon való értékesítés számára megfelelő módon biztosítja. Feladata a megrendelések teljesítéséhez, azaz az előállított termékeknek a logisztikai elvek betartásával a megrendelőhöz juttatása. Tevékenységi körébe tartozik a megrendelés szerinti komissiók összeállítása, az áruk elosztási hálózatának kialakítása, az áruelosztó járatok megszervezése, esetleges közbenső elosztó raktározás feltételeinek biztosítása.



5.1. ábra Logisztikai folyamatok (KEP_A303_I_05_01) KEP_A303_I_05_01.JPG

A raktározás szerepe a logisztikai ellátási láncban folyamatosan nő, hiszen az ellátási lánc egymást követő szakaszai éppen a közöttük lévő raktárakban kapcsolódnak egymáshoz és a raktárak lényegi csomópontját képezik az áru-és az információáramlásnak egyaránt. A raktárak általában nemcsak tárolják az anyagokat, hanem sok egyéb járulékos tevékenységet is elvégeznek, pl:

- Ellenőrzik a beérkezett alapanyagok mennyiségét, minőségét.
- Komissióznak, vagyis több árufajtából állítják össze a kiszállítási csomagokat.
- Expediálnak, vagyis a vevő vagy a szállító igényeinek megfelelő egységcsomagokat képeznek, melyeket szükség esetén becsomagolnak és feliratoznak.



5.2. ábra Logisztikai folyamatok (KEP_A303_I_05_02) KEP_A303_I_05_02.JPG

Ez a kiemelkedő szerep kihívást jelent a raktárak számára, amelynek csak akkor tudnak megfelelni, ha technológiájukkal, technikáikkal és szervezési módszereikkel az ellátási lánc stabil tagjává válnak.

A raktári információs rendszerek eredetileg a raktári készlettel gazdálkodtak, meg tudták mondani, hogy melyik anyagból mennyi van raktáron, sőt, általában azt is, hogy az anyag hol található a raktárban. A betárolási, kitárolási funkciók nélkülözhetetlen közreműködője volt az emberi erőforrás, az átfutási idő órákban, esetleg napokban volt mérhető. A cél a megrendelések teljesítése volt. A raktár rutinszerűen működött, az eredmény – kimaradt teljesítések, hibás kiszállítások – általában műszak végén derült ki.

A mai, korszerű raktári információs rendszerek a készleten kívül már több erőforrást menedzselnek, a raktári tároló helyeket, a raktári gépeket, és természetesen az alkalmazottakat. Integráltak, ami azt jelenti, hogy nemcsak az informatikai folyamatokat, hanem a fizikai anyagmozgatást is felügyelik. Online kapcsolatban vannak a vállalati információs rendszerekkel, hiszen csak így oldható meg,

hogy a vállalkozás ne pusztán raktárra gyártson, hanem fizetőképes igényeket elégítsen ki, vagyis rövid átfutási idejű megrendeléseket teljesítsen.

A korszerű raktári információs rendszerektől elvárják a gyors, hibamentes működést, ehhez természetesen megfelelő berendezésekre van szükség.

Állványos dinamikus tárolási rendszerek

Jellemzőjük, hogy egy-egy tárolási egység elhelyezése vagy kiemelése esetén, az állványon levő áru egy része vagy egésze is változtatja helyzetét. Főbb változataik:

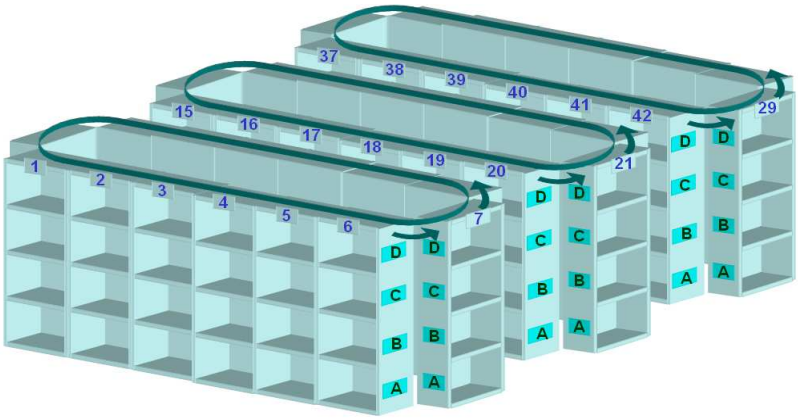
Utántöltős állványos tárolás

Olyan átjárható állványos tárolási rendszer, ahol a tárolási egységek alátámasztó hossztartók lejtős kialakításúak: így a tárolókban a tárolási egységek a nehézségi erő segítségével a betárolási oldal felől a kitárolási oldal felé haladnak. Egy tároló helyen egyféle anyagból több tárolási egység van, a FIFO elvet (az előbb beérkezett áru előbb kerüljön ki) maga a tároló biztosítja. Ez a megoldás a raktári információs rendszerrel nem irányítható, csak támogatható. Meghatározható a betárolandó alkatrész tárolási helye, a kitárolandó alkatrészek helye, tárolási egységei, és meghatározható egy optimális útvonal az egyes tárolóhelyek bejárásához.

Körforgó állványos tárolás

Egymással összekapcsolt tálcák, polcok vagy egyéb tartóelemek mozognak függőleges („páternoszter”-rendszer) vagy vízszintes („karusszel”-rendszer) irányba. A működtetéskor valamennyi egység megindul a pályán, és mindaddig mozgásban marad, amíg a kívánt tárolási egységet tartalmazó állványrekesz az átadóhelyre nem érkezik.

A rendszer működése jelentős energia befektetést igényelhet, ebből következik, hogy ez a technika elsősorban kisméretű, nagy értékű termékek, alkatrészek és szerszámok esetén gazdaságos.A megoldás szoftveresen irányítható, a betárolláskor nemcsak közli a tárolóhely kódját a rendszer, hanem előhozza azt, kitárolláskor nem útvonalat, hanem a kitárolási sorrendet optimalizál, így jelentősen csökken az átfutási idő. Ebben a megoldásban az emberi tényezők minimálisak, gyakorlatilag nulla a tévedés, a nem megfelelő alkatrész kitárolásának az esélye.



5.3. ábra Körforgó állványrendszer (KEP_A303_I_05_03) KEP_A303_I_05_03.JPG

Magasraktározási rendszerek

A magasraktározási rendszerek darabáruk olyan állványos tárolási rendszerei, amelyekben a tárolási magasság az általános célú emelőtargoncák által elérhető átlagos tárolási magasságot meghaladja, és az áruknak állványokba helyezését, illetve levételét onnan az állványok közötti folyosókban mozgó felrakógépek vagy felrakótargoncák végzik.

A torony elv a kis alapterületen történő maximális tárolást biztosítja, az ilyen rendszerek magassága általában 6-30 méter közötti. A középmagas raktárak 6-10 méter magas állványokkal rendelkeznek, a magasraktárak 10 méter felettiekkel. Az automatizáltsági szintjük szerint megkülönböztethető:

- gépesített,
- részlegesen automatizált,
- teljesen automatizált

magasraktár.

A teljesen automatizált magasraktárak automata betároló és kitároló rendszerrel, kezelő nélkül működnek. A rendszer síneken mozgó, lift rendszerű járművel oldja meg a gyors és pontos rakodólap vagy tároló doboz berakást és kiszedést. A rakományt a munkahelyről felveszi, a kijelölt helyre viszi és berakja, ill. a polcról kiszedi, és a komissiózó helyre szállítja. Általában minden polcrendszer-folyosóban működik egy-egy felrakógép, amelyek egymástól függetlenül vezérelhetőek.

Nagyon gyors, és pontos a rakatmozgatás, egyszerű a rakatok azonosítása, mely nagyban segíti a pontos nyilvántartást és a kitárolást. Egyes megoldásokban lehetőség van a rakatok dupla vagy akár a tripla mélységű tárolására is. Nagy előnye ennek a megoldásnak, hogy így kevesebb folyosóra, kevesebb gépre van szükség, tehát olcsóbb beruházás és üzemeltetés valamint nagyobb az adott területre betárolható rakatok száma. A teljes automatizálás miatt a balesetveszély gyakorlatilag ki van zárva. Nem befolyásolja a rendszer működését semmilyen időjárási tényező vagy körülmény, sőt nincs szükség világításra, gépészetre, (közművekre), adott esetben még hőmérséklet temperálásra sem.

A magasraktári nyilvántartó és vezérlő program a VIR-től kapott adatok alapján vezérli a raktár működését. A betárolást követően az adatok megjelennek a VIR-ben mint raktári készlet, a gyártásütemezést követően pedig a szükséges alkatrészek kiszedési listája alapján történik a kitárolás.

5.2. Raktári folyamatok irányítása

Ahhoz, hogy egy raktár az elvárt színvonalon tudjon működni olyan irányítási rendszert kell kiépíteni, amely integrálni tudja a fizikai anyagmozgatási folyamatokat az anyagokkal kapcsolatos információk áramlási folyamataival. Az integráció legfőbb jellemzője a valós idejűség, ami azt jelenti, hogy a raktári tranzakciókhoz kapcsolódó információk a végrehajtás időpontjában azonnal a folyamatirányítás rendelkezésére állnak.

A raktári alapfolyamatok alatt az áruk be- és kiszállítását, a raktáron belüli mozgatását, valamint készletezésével kapcsolatos teendőket értjük. Ezen folyamatok végrehajtása közben különböző műveleteket végzünk, melyek valamilyen célfüggvénnyel optimalizálhatók (pl. legrövidebb átfutási idő, legrövidebb bejárási útvonal …). A folyamatok irányítása a munkavégzés egyes műveleteinek optimális megtervezését, és a kialakított feladatsor megfelelő végrehajtásának betartatását jelenti. Az információs rendszer csak akkor tud megfelelően működni, ha egyértelmű jelölési rendszereket vezetnek be mind az árukra, mind a tárolóhelyekre, mind az elvégzendő feladatokra.

A hatékony működéshez az is szükséges, hogy a vállalat teljes informatikai rendszerében egységesek legyenek a jelölések. Sok probléma okozója lehet, ha egy meglévő vállalat irányítási rendszert kibővítenek egy új raktári modullal, és az eredeti 6 karakterek kódok mellett a raktárban 13 karakteres vonalkódokat használnak, vagy ha egy új raktárépületben ugyanazokat a kódokat használják a raktárhelyekre, mint a régiben.

Nézzünk meg, bizonyos raktári funkciókat hogyan támogat egy korszerű raktári információs rendszer:

- Betárolás: Az áru beérkezésekor bevételezik, és betárolás alatti státusszal látják el. Bár az áru az átvételi területen van, mennyisége növeli a raktári készletet, és ha szükséges, azonnal kiadható. Bevételezéskor a rendszer az áru kódja alapján meghatározhatja a tárolási helyet, pl. az áru jellegének, mennyiségének alapján, a szokások alapján (az adott áru mindig ugyanoda kerül), vagy az aktuális üres raktárhelyek alapján. A betároláshoz személyre szólóan megadhatóak az egyes feladatok, meghatározható az áruk megfelelő betárolási sorrendje, és optimalizálható a raktárhelyek felkeresési útvonala.
- Kitárolás: Több feladatból feltételeknek megfelelően összesíthető, azután személyre szólóan kiválasztható az egyes áruk köre és darabszáma (pl. adott időre kitárolandó áruk, vagy adott célhelyre kerülő áruk), a raktárhelyek alapján optimalizálható a bejárási útvonal, lefoglalhatók a szükséges tételek, így véletlenül nem adhatók ki másnak.
- Figyelmeztetések: A rendszer figyelmeztethet a rövidesen lejáró szavatossági idejű termékekre, kigyűjtheti azok adatait. Olyan esetekben, amikor a raktári készlet egy bizonyos darabszám vagy százalékos érték alá csökken, a rendszer figyelmeztethet az utánrendelésre, a fogyás alapján meghatározhatja az utánrendelési mennyiséget.
- Hatékony statisztikák: A korszerű rendszerekben naprakész statisztikák állnak rendelkezésre, pl. az áruk raktárban töltött idejéről, a raktárhelyek állapotáról és kihasználtságáról, az egyes személyek által elvégzett feladatokról…

Targoncarobotok

A nagytételű anyagmozgatásban egyre nagyobb szerep jut az automatikus vezérlésű járműveknek (automatic guided vehicle, AGV), a targoncarobotoknak, robotkocsiknak.

A korai robotok intelligenciája csak az indulás, megállás funkciókra, az irányváltásra, és a mágnesszalag segítségével kijelölt pályán történő mozgásra korlátozódott. A kocsik önálló azonosítóval rendelkeznek, de az általuk szállított alkatrészek is egyedileg azonosítottak, általában vonalkóddal vagy rádiófrekvenciás kóddal (RFID).

A kocsikat egy központi kocsivezérlő-rendszer irányítja, amely mindig tudja, hogy melyik robot hol található, és milyen alkatrészeket szállít.

A központi kocsivezérlő-rendszer a gyártósortól kapott információk alapján tervezi meg az egyes robotok pályáját, és azt is, hogy milyen alkatrészek, milyen sorrendben kerüljenek az egyes a járművekre.

Általában olyan gyártósoroknál használnak alkatrész kiszállító targoncarobotokat, ahol az alkatrészek nagy tömegűek, szalagon nehezen mozgathatók, vagy veszélyes technológiával történik a gyártás, pl. öntödék, hegesztősorok, festő részlegek.

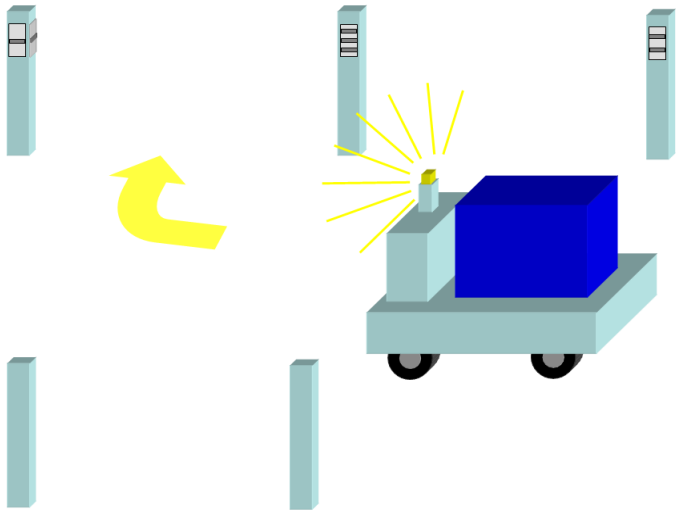
Egyes gyárakban a kocsik mozgó szerelőegységként működnek, ilyenkor nem alkatrészeket szállítanak, hanem elindulnak egy váz elemmel, és minden egyes állomáson megállnak, ahol kezelő leolvasóval ellenőrzi a vonalkódot, majd az egységre felszereli a szükséges alkatrészt.

A kocsi az utolsó alkatrész beszerelése után a raktérre megy, ahol a teljesen összeszerelt egységet átteszik a kiszállító raklapra. A kocsi ezután kész a következő szerelési feladat elvégzésére.

A korai AGV-k kötött pályán, induktív nyomvonalvezetéssel mozogtak. A kocsik pályáját egy mágneses szalaggal jelölték ki, amely könnyen lerakható és áthelyezhető volt. Az útvonalak megváltozásához át kellett helyezni a mágneses szalagokat.

Nagy áttörést jelentettek a radaros és lézeres navigációs rendszerek megjelenése. Itt a jármű tetején egy lézer-navigációs egység található. Ez a készülék másodpercenként néhány 10000 nem a látható fény tartományába eső impulzust ad ki. Ezek a fénynyalábok az útvonal mentén fix pontokon rögzített tükrökről visszaverődnek. A visszaverődés érzékelésével tájékozódik a robot. A pálya módosítása nem hardveresen, hanem szoftveresen, koordináták megadásával történik. Az ilyen módon vezérelt robotok intelligensebbek elődeiknél, fedélzeti számítógépük képes a két pont közötti legrövidebb útvonal kijelölésére és az útvonal önálló bejárására. A központi kocsivezérlő-rendszer pedig figyeli az összes robot pozícióját, és az összeütközés elkerülésére érdekében képes a kevésbé fontos alkatrészt szállító robotot megállítani addig, amíg a másik robot elhagyja az adott szektort.

Az AGV-kből és a központi kocsivezérlő-rendszerből nyert információk alapján egy grafikus alkalmazás segítségével, az üzem térképén, nyomon lehet követni az egyes targoncák pozícióját, állapotát.



5.4. ábra Targoncarobot (KEP_A303_I_05_04) KEP_A303_I_05_04.JPG

Kommunikáció a robotokkal

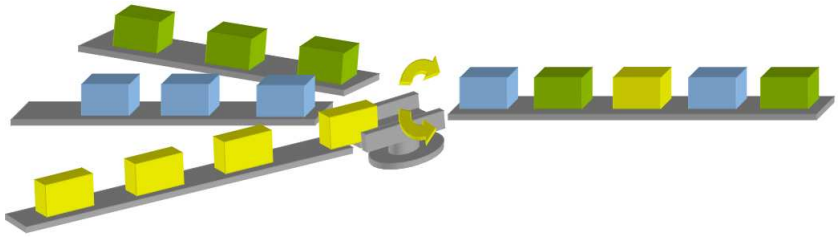
A központi kocsivezérlő-szerver az AGV-kkel rádiófrekvencián kommunikál. Az adó-vevő egységek elhelyezése a targoncák által bejárt terület teljes lefedésének figyelembe vételével történik.

A szerver az AGV-nek alapvetően szállítási parancsot ad, valamint blokkolni tudja az adott kommunikációs pont után. Amikor a targonca áthalad egy kommunikációs ponton, jelzi ezt a szerver felé. Ezen kívül státuszinfót is küld a saját állapotáról (mi lesz a következő kommunikációs pontja, mi a célja, meg van-e rakva, akkumulátor töltöttség, stb.).

Szállítópálya rendszerek

A raktári területek, szintek és épületek közötti szállítás hatékony eszközei. Olyan helyeken alkalmazhatók hatékonyan, ahol ugyanazon pontok között, hosszú, változatlan pályán kell nagy mennyiségű anyagot szállítani. Nemcsak egyszerű anyagtovábbításra, hanem speciális tevékenységek elvégzésre is használhatók:

Összehordás, ahol az alrendszer az elosztó központ különböző helyeiről kapja a termékeket, és azokat meghatározott összetételűre “összehordva” egy folyamban a sorolásra továbbítja.



5.5. ábra Szállítópálya rendszer (KEP_A303_I_05_05) KEP_A303_I_05_05.JPG

- Sorolás, ahol beazonosítják a terméket (súly szerinti azonosítás, vonalkód, RFID), és az egyes termékeket megfelelő távolságra helyezik el egymástól, a lehető a legnagyobb osztályozási sebesség eléréséhez.
- Szétválasztás, amely a sorolásról kapott adatok alapján a terméket a rendeltetése szerinti helyre továbbítja. A soroláson azonosított tételt relés vezérlés, programozható logikai vezérlés vagy PC-vezérlés kíséri figyelemmel a terelőig. Egy érzékelő a terelőt beindítja, és a terelő az adott tételt kellő sebességgel az előírt helyre irányítja.
- Elszedés, amely a célba érkezett termékeket lerakja a szállítópályáról, pl. egy másik szállító eszközre, vagy egy feldolgozási műveletre.

Az irányítás az anyagmozgató rendszerek esetében az egyes folyamatok elindítását, a már elindított folyamat befolyásolását, valamint a folyamat leállítását jelenti. Az egyes folyamatok irányításhoz:

- Meg kell ismerni az anyagmozgatási folyamat és a kiszolgált alapfolyamat állapotát, észlelni kell a szükséges adatokat, rögzíteni, és továbbküldeni feldolgozásra.
- A lehetőségek figyelembe vételével meg kell határozni a lehetséges cselekvési változatokat, vagyis fel kell dolgozni az adatokat, és információkká kell alakítani.
- A lehetséges cselekvési változatok közül ki kell választani a legmegfelelőbbet, vagyis optimalizálni kell a lehetséges megoldásokat, és el kell dönteni, hogy melyiket alkalmazzuk.
- Végre kell hajtani a kiválasztott cselekvési metódust, vagyis be kell avatkozni az anyagmozgatási folyamatba. Ez történhet vezérléssel, vagy a futó folyamat paramétereinek módosításával, vagyis szabályozással.

Az irányításhoz szükséges információs folyamatok: az adatok észlelése, rögzítése, tárolása, továbbítása, feldolgozása (információkká alakítása), igény esetén megjelenítése, valamint célfüggvények segítségével történő optimalizálása kiválasztó algoritmusok használatával. Az

információknak ki kell terjedniük mind a kiszolgált folyamatok paramétereire (pl. termelési, raktári vagy szállítási), mind az anyagmozgató rendszer állapotára.

Az anyagmozgatás emberi irányítása esetében az irányító személy kezelőszervek (pl. nyomógombok, kapcsolókarok) igénybevételével indítja, leállítja, szabályozza az egyes gépeket, gépcsoportokat, illetve irányítja az anyagmozgató rendszer működését.

Munka közben a munkás érzékeli, elolvassa vagy tudomásul veszi a kapott utasításokat, elvégzi a szükséges ellenőrzéseket a különböző kijelzőkről érkező akusztikai vagy optikai jelzések alapján, összehasonlítja, összehangolja egymással a kapott utasításokat és a gyűjtött információkat, képzettsége, tudása és a munka közben szerzett tapasztalatai alapján döntéseket hoz, végül a megfelelő kezelőelemek révén működésbe hozza a beavatkozó szerveket.

Az anyagmozgatás számítógépes irányítása lényegében ugyanezt a tevékenység-sorozatot látja el egy program segítségével. A program megkapja az anyagmozgató rendszer meghatározott pontjain felszerelt érzékelőktől érkező adatokat, feldolgozza a folyamatosan kapott adatokat, és ha a rendszer pillanatnyi állapota ezt szükségessé teszi, végrehajtja a programban rögzített lépéseket.

Ha a számítógép az információk feldolgozása során azt állapítja meg, hogy a beavatkozás nem lehetséges, vagy valamilyen technikai probléma miatt (pl. nincs kapcsolat az eszközzel) a megfelelő parancs nem adható ki, akkor vészjelzéssel figyelmezteti az illetékes személyt a közvetlen beavatkozás szükségességére.

5.3. Metrikák szerepe

A VIR rendszerek működésének egyik fontos eleme a működés, a rendszer állapotát leíró jelzők karbantartása és kiértékelése. Ezen jelzőket szokás a rendszer metrikájának is nevezni. A metrika szó ezen kontextusban mérőszámot jelent, azaz azon mérőszámokat, melyek alkalmasak a rendszer működésének jellemzésére. A metrika több szempontból is fontos elem, hiszen a kvalitatív verbális jellemzések sokszor szubjektív elemeket hordoznak, így nem teszik lehetővé a tárgyilagos értékelést. A metrika használata mellett szóló fontosabb érvek:

- számszerűsíti a mennyiségeket, mutatókat
- mennyiségi modelleket tesz lehetővé

- objektív mutató
- konvertálható mutatók
- matematikai formulák alkalmazhatók
- optimalizálást tesz lehetővé
- szabályozást tesz lehetővé

Emiatt a vállalat működésének leírására célszerű metrikákat bevezetni. A kiválasztott metrikák esetében fontos, hogy azok

- lényeges működési paramétereket fedjenek le
- mérhetőnek kell lennie
- megbízhatónak kell lennie
- elérhetőnek kell lennie
- ellenőrizhetőnek kell lennie
- lehetőleg bázis metrika legyen, amely a meghatározza a többi metrikát

Mivel a vállalat működése olyan összetett folyamat, melyben több résztevékenység kap helyet, ezért a metrikákat is osztályozhatjuk az érintett tevékenységkört illetően is. A Naylor féle vállalati modell alapján a fontosabb tevékenység és metrika körök:

- erőforrások (humán, eszköz, anyag, energia,…)
- termelés (mennyiség, érték, idő,…)
- kontrolling (tervek, teljesítés, felhasználás,..)
- minőségbiztosítás
- irányítás
- tervezés
- vezérlés

A metrikák egyik fontos jellemzője, hogy azok között összetett kapcsolatrendszer alakulhat ki. Az egyik területen bevezetett metrika kihathat más területeken felvett metrikákra is.

A kapcsolatnak több módozata lehetséges:

- aktivizáló, stimuláló (pozitív hatás)
- gátló kapcsolat (negatív hatás)
- szinkron kapcsolat
- aszinkron kapcsolat

A fenti kapcsolatrendszer, függőségi rendszer ismerete több szempontból is fontos, hiszen ennek ismeretében lehet eldönteni, hogy hogyan változtathatjuk az egyes metrikákat egy adott cél érdekében,a nélkül, hogy nemkívánt, káros mellékhatások lépnének fel. A kapcsolatrendszer leírása egy szabályozási gráffal adható meg.

A metrikák kiválasztásánál az alábbi lépéseket célszerű bejárni:

- A működési célok és irányelvek kijelölése
- Figyelj oda a lehetőségekre és igényekre
- Az irányadó kulcsmetrikák meghatározása
- Értsd meg a feladatot, költségelemzés
- A kulcsmetrikákra irányuló stratégiák kijelölése
- Dolgozd ki a részleteket
- A metrikák közötti kapcsolatok elemzése
- Ellenőrizd az elgondolásaidat
- A metrika orientált taktikák kidolgozása, ellenőrzési pontok beépítése

A fenti lépések célja, hogy az előzőekben megfogalmazott kívánalmaknak megfelelő metrikák kerüljenek kiválasztásra. Hiszen a lényeges metrikák köre a működést meghatározó célokból, irányelvekből származtathatók le. A logisztikai rendszerekben több sajátos metrika terjedt el. A legfontosabb megvalósított logisztikai célok a következők:

- az áru forgási sebességének növelése, vagy átfutási idejének csökkentése,
- kihasználtságok növelése,
- készletszint csökkentése,
- készlethiány elkerülése,
- előrejelzési pontosság növelése,
- növekvő eladás,
- költségcsökkentés,
- kapacitás jobb kihasználása,
- vevőszolgálat fejlesztése
- megbízhatóság növelése,
- raktár hatékonyabbá tétele
- fejlődő kapcsolat,
- adatbeviteli hibák elkerülése,
- emberi beavatkozástól mentes számítógépek közötti adatátvitel,

- hibás vagy téves megrendelés megszűntetése.

A felhasználandó információk a következő nagy csoportba sorolhatóak:

- törzsadatok, partner és termékinformációk,
- tervezés, előrejelzés adatai,
- megrendelés információi,
- szállítás adatai,
- számlázás tulajdonságai,
- tranzakciók, kifizetési adatok,
- jelentések és tervek, tervezési üzenetek,
    - leltárkészlet jelentés,
    - értékesítési előrejelzés,
    - értékesítési jelentés,
- szolgáltatási üzenetek,
- általános üzenetek,
- egyéb üzenetek,

5.4. Elterjedt logisztikai modulok

A VIR alkalmazások a következő logisztikához kapcsolódó technológiákat használják:

- egységes tárolóhely és rakományazonosító eljárások és rendszerek,
- automatikus és szabványosított helymeghatározó rendszerek,
- szabványosított jármű- és küldeménykövetési rendszerek,
- operatív irányítási szolgáltatások a megrendelések kezelése komplex logisztikai irányításának támogatásához,
- a közvetlenül az adatok beérkezésekor végzett interaktív előszűrésen alapuló feldolgozási technológiával történő adatelemzés,
- operatív döntéshozatalt támogató keresési, rendezési, jelentéskészítési és tervezési funkciókkal támogatott vezetői információs szolgáltatás,
- termelésütemező eljárások,
- a logisztikai rendszer kapacitásának, működési paramétereinek, stratégiáinak meghatározására szolgáló szimulációs eljárások.

Termék- és rakományazonosító rendszer: A rendszer feladata, hogy a logisztikai láncban áramló anyagokkal kapcsolatos információkat (pl. az anyag típusa, az anyaggal kapcsolatos feladat beazonosítása stb.) a lánc bármely pontján, az áramlási folyamat tetszőleges időpontjában minden kétséget kizáróan fel lehessen ismerni. Ezekhez a következőek szükségesek:

- az anyagokkal kapcsolatos különböző információk bekódolását elősegítő kódrendszerek, amelyek segítségével előállított kódokat termékazonosító segédeszközökön (vagy benne) helyeznek el,
- az áramló anyagokon feltüntetett (vagy felszerelt) kódokat leolvasni képes adatfelvételi eszközökre.

Jármű- és küldeménykövetési rendszer: A járműkövető rendszer lényege, hogy a szabványosított helymeghatározó rendszerek által szolgáltatott adatokra támaszkodva on-line, vagy off-line módon különböző szoftveres platformok segítségével képesek a különböző térben mozgó objektumok pillanatnyi helyzetének koordinátáit megjeleníteni szöveges vagy grafikus formában, ezáltal folyamatos on-line vagy off-line információt szolgáltatni a járművek mozgását illetően a diszpécser központba.

A rendszer felépítése a következő:

- a szabványosított helymeghatározó rendszer,
- diszpécser központ + diszpécserek,
- a diszpécser központban kiépített hardver és szoftver rendszer,
- a mozgó objektumokon elhelyezett hardver és szoftver rendszer,
- on-line követés esetén, a járművön elhelyezett mobil, mikroszámítógép által működtetett terminál és a diszpécser központ közötti kommunikációt megvalósító műholdbázisú, vagy földbázisú kommunikációs rendszer,
- grafikus megjelenítés esetén a vizsgált terület relatíve pontos vektorgrafikus digitális térképére, amelyet a megjelenítéshez a diszpécser központban üzemeltetett szoftver használ fel.

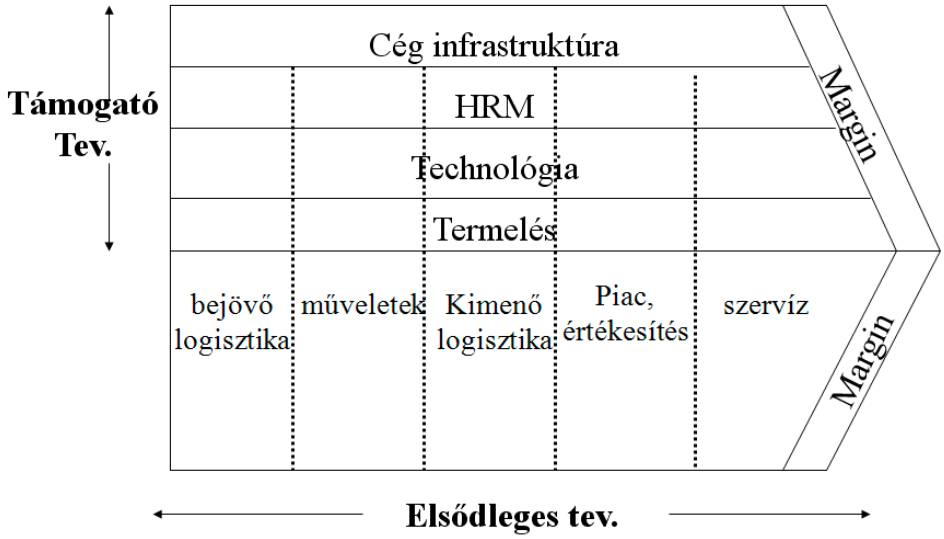
5.6. ábra Adatkapcsolatok az informatikai rendszerben (KEP_A303_I_05_06)

KEP_A303_I_05_06.JPG

A logisztikai célú operatív irányítási szolgáltatások gyakorlatilag magukba foglalhatják az összes olyan funkciót, amelyek a logisztikai rendszerek operatív üzemeltetése szempontjából felmerülhetnek, és hozzájárulhatnak a megrendelés kezelési folyamat hatékonyságának növeléséhez. Ezeknek a szolgáltatásoknak a fő célja olyan (az egyes funkciók működtetése szempontjából lényeges) paraméterek kiszámítása, amelyek segítségével az egyes részterületek optimális költség szinten, hatékonyan üzemeltethetők. Az integrált vállalatirányítási rendszerek lehetőséget nyújtanak arra vonatkozóan, hogy a korábban különböző szoftveres megoldások által kezelt funkciók optimalizálása egy egységes központosított adatbázis felhasználásával történhessen meg. A vállalatirányítási rendszerek integráltsági fokát nagymértékben meghatározza az a tényező is, hogy milyen mértékben valósul meg a rendszer szintű integráció bizonyos operatív irányítási szolgáltatások vonatkozásában.

A logisztikai modulok alapvető szerepet játszanak a VIR különböző moduljaiban, mint az a klasszikus Porter féle értéklánc modellben is jól látható. Az elsődelges tevékenységekben több modulban is jelentős szerepet kap a logisztika, hiszen mind a bemenő mind a kimenő szállítási feladatok az értéklánc zerves részét képezik.

A Naylor féle vállalati funkció modellnél is jól megfigyelhető a logisztika szerepének fontossága a többi tevékenység modul vonatkozásában. A logisztikai modul egyik fő jellemzője, hogy átfogó jellegű, hiszen nemcsak a termelési vetületben szerepel, hanem mindhárom funkcióághoz köthető.



5.7. ábra Porter-féle értéklánc modell (KEP_A303_I_05_07) KEP_A303_I_05_07.JPG

A funkcionális részterületek az alábbiak lehetnek:

- a készletgazdálkodás, ahol a cél a termékszintű készletezési mechanizmus optimális működtetése (termékenként meghatározott készletezési stratégiák, valamint az azokhoz kiszámított optimalizált szabályzó paraméterek segítségével),
- a szállításirányítás, amelynek segítségével a járműpark optimális módon diszponálható (eszköz és feladat hozzárendelése matematikailag optimalizált körjáratok alkalmazásával),
- a komissiózási rendszer szervezése, amely a vevői megrendelések kigyűjtését különböző matematikailag optimalizált árukigyűjtési stratégiákkal vezényli (statikus vagy dinamikus komissiózási módszerek optimalizálása matematikai algoritmusokkal).

Interaktív előszűrésen alapuló feldolgozási technológiával történő adatelemzés: Az on-line adatelemzés lehetővé teszi a rendelkezésre álló üzleti adatoknak hatékony és gazdaságos felhasználását, vagyis az információkkal való hatékony gazdálkodást. A technika alkalmazása a vállalatoknál onnan ered, hogy a végrehajtási folyamatok információ technológiai követése rendelkezésre áll, vagyis a folyamatokból származó információk kinyerése megoldott, ezáltal megvalósítható az adatmodellezés, a rendelkezésre álló adatok segítségével pedig az adatelemzések.

Integrált logisztikai információs rendszer: Az adatok az értékalkotási – logisztikai lánc mentén való intenzív cseréje lerövidítheti az átfutási időt, csökkentheti a készleteket és növelheti a szolgáltatás színvonalát. Az a korszerű rendszer, amit integrált logisztikai információs rendszernek nevezünk, a logisztikai – értékalkotási láncban résztvevő tetszőleges partnereket képes összekapcsolni. A logisztikai adatbusz kapcsolja össze a logisztikai és a termelésirányítási rendszer résztvevőit, és gyűjti magába ezekből az összes új feladást, előjelentést, készletváltozást, megrendelést, valamint az összes határidő és mennyiség változását. A termelővállalatok és partnereik adatkapcsolatban vannak egymással. Az egymással határidőt egyeztető önálló üzlettársak széles csoportjából virtuális vállalatok alakulnak ki. Minden logisztikai folyamat, így a megrendelések, az előjelentés, a készletváltozások, a lekérdezések, a feladások, az igénylések és a szállítások egy közös logisztikai adatbázisba kerülnek. A letisztított logisztikai folyamatból az integrált logisztikai információs rendszer először egy átfogó logisztikai hálózatot, más szóval ellátási hálót épít. A logisztikai háló átszövi mindegyik résztvevő vállalatot, és egy olyan közös hálózatot hoz létre, amelyben elmosódnak a vállalati határok. A logisztikai adatbázis a virtuális vállalat valamennyi logisztikai folyamatát tárolja. Az adatállomány azonban további felhasználói lehetőségeket is felkínál. Segítségével kialakíthatók olyan új logisztikai szolgáltatások, amelyek a vállalati léptékű tervezést, sőt optimalizálást támogatni tudják.

Folyamatos készletfeltöltési rendszer: A folyamat az áru mozgását végigköveti a gyártótól a vevőig, vagy a végfogyasztóig. Illetve az ellentétes irányban a visszacsatolást is kezelni tudja. A rendszer megvalósítja a fontosabb logisztikai célokat.

- áru forgási sebességének növelése,
- készletszint csökkenése,
- készlethiány elkerülése,
- raktár hatékonyságának növelése.

A beszállító dönt a megrendelő készletének feltöltési időpontjáról, vagyis a készletszintjét tudja változtatni. Az ellátási láncon belül ez egy igen hatékony módszer a készletek csökkentérére. Az áru a lehető legkevesebb erőforrás igénybevételével kerüljön a megrendelő raktárába. A megrendelések és a kiszállítások automatikusan történnek. Így alkalmas készletszint követésére és előrejelzésére.

Együtt dolgozva a beszállító által működtetett készletgazdálkodási rendszerrel az információáramlás szempontjából a következő tevékenységek jelentkeznek:

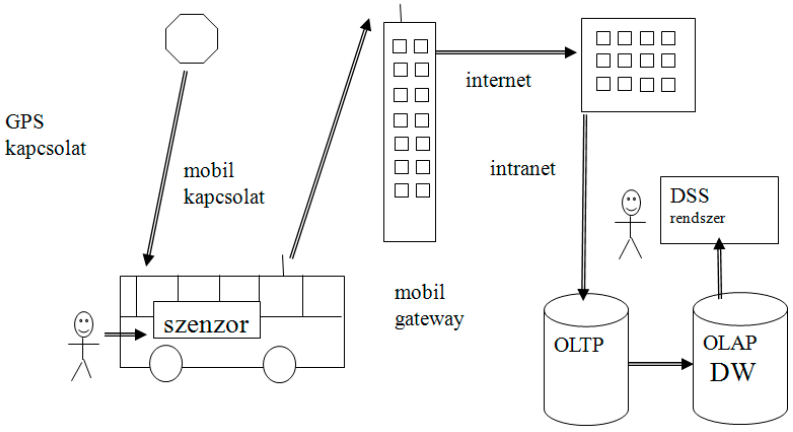
- készletszint értesítésének fogadása a megrendelőtől,
- értékesítési előrejelzés fogadása a megrendelőtől,
- készletfeltöltési megrendelések generálása,
- feladási értesítés küldése a megrendelőnek,
- értékesítési jegyzék fogadása a megrendelőtől,
- számlaküldés a megrendelőnek.

5.5. LTS mintarendszer

Az úgynevezett Live Traffic Services (LTS) rendszerek lényege, hogy a forgalomban résztvevő járművek aktuális működési adatokat kapnak és adnak az irányító rendszerhez kapcsolódva. A rendelkezésre álló adatok alapján az irányító rendszer optimalizálni tudja a jármű tevékenységét. A rendszer megvalósításának egyik fő nehézsége a rendszer összetett volta, hiszen jelenleg külön-külön egység kell az egyes komponensek működéséhez. A GPS rendszerek szerepe a járművek és az egyéb közlekedésben résztvevő elemek helyzetének meghatározásában áll. A GPS rendszerek vizsgálatánál a legfontosabb elemzési szempontok:

- rendelkezésre állás (milyen lefedettségű a GPS rendszer)
- kompatibilitás (a kapott adatok automatikus feldolgozása, továbbítása)
- költség
- pontosság

A funkcionalitási vizsgálatunkban a GPS-t mint általános navigációs rendszert értjük, mivel az összes rendelkezésre álló rendszer biztosítani tudja a helymeghatározás alapfunkcióját.



5.8. ábra LTS rendszer architektúra (KEP_A303_I_05_08) KEP_A303_I_05_08.JPG

A forgalmi adatok begyűjtésének több alternatív módozata van. Az adatgyűjtő rendszer főbb komponensei:

- szenzorok
- adattovábbítók
- adattárolók
- adatfeldolgozók

A szenzorok esetében lehet beszélni

- helyhez kötött
- járműhöz kötött
- személyhez (utashoz) között

A helyhez kötött eszközök esetében viszonylag nagyobb szerelési költségekkel kell számolni, de olcsóbb az üzemeltetés. A mobil eszközök technikailag újabb szintet jelentenek, egyénre szabott megvalósításokkal.

Az egyes távoli eszközök által nyerhető információk:

Szenzor típusa	Információ jellege
Helyhez kötött	
	Pozíció validálása
	Videokép a jármű állapotáról
	Várakozó utasok jelzése
Járműhöz kötött	
	Utas felszállása / leszállása
	Sofőr hang üzenetei
	Jármű pozíciója
	Sofőr állapota
	Jármű, motor állapota
Utashoz kötött (kártya)	
	Utas azonosítása
	Utasok lakhelyének megismerése
	Utas szolgáltatás fogyasztás mérése

A rendszer architektúra elemei

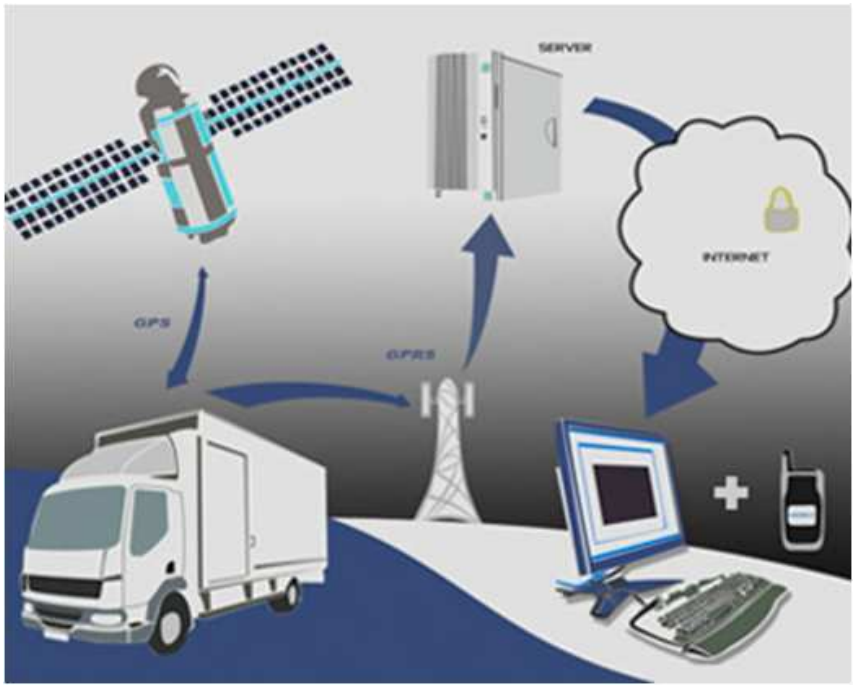
- GPS vevők a járműveken
- Utasszámláló szenzorok a járműveken
- Jármű állapot szenzorok a járműveken
- Vezető állapot szenzorok a járműveken
- Rádió kommunikációs eszközök a járműveken
- Utasszámláló szenzorok a megállókban
- Utastájékoztató eszközök a megállókban
- Rádió kommunikációs eszközök a megállókban
- Rádió kommunikációs eszközök a központban

- Adattisztító modul a központban
- Adatgyüjtő rendszer a központban
- OLTP adatforrás a központban
- Kapcsolat a VIR/ERP rendszer felé
- OLAP adattárház a központban
- Adatelemző modul a központban
- Kapcsolat külső adatforrások felé

A hely (illetve helyzet) meghatározó eszközök segítségével bármikor megállapítható, hogy egy adott jármű hol tartózkodik. Alapesetben ugyan kommunikáció csak a járműben levő fix, vagy ideiglenes eszköz és a műhold között van, azonban a meghatározott pozíció továbbadható. Ehhez olyan kommunikációs csatornákat érdemes használni, amelyek minél kisebb infrastruktúra kiépítését igényli, optimális esetben meglévőre épül.

GSM alapú kommunikáció

A szállító eszköz pozíciója néhány byte-on elfér. A GSM szolgáltatók igen jó meglévő lefedettséget biztosítanak (akár városon kívül is). Ezt a lefedettséget, és hálózatot lehet felhasználni vagy SMS alapú, vagy GSM alapú vezetékmentes hálózati kommunikációval (GPRS). Ez a változat szinte egyáltalán nem igényel infrastruktúra kiépítést. Mivel a járművekről csak igen kevés adat kerül továbbításra, ezért költségvonzata nem jelentős, és a városon kívül levő megállókban is élvezni lehet ennek előnyét. Ott is követhető a közeledő busz, becsülhető az érkezési idő, szükség szerint egyes ellenőrzések is megvalósíthatók. Összetettebb rendszerek is kiépíthetők, amelyek ha detektálják, hogy a jármű késésben van, az útvonalán levő közlekedési lámpákat zöldre állítják.



5.9. ábra Kommunikációs rendszer komponensei (KEP_A303_I_05_09) KEP_A303_I_05_09.JPG

Rádiójelekkel történő kommunikáció

Készült olyan kutatási jelentés, és teljes implementációs dokumentáció, amely rádiójelek segítségével valósítja meg a kommunikációt a buszállomás és a járművek között (Jian Jinrong: Smart Bus Station Sign, Oriental Institute of Technology). Ez városi közlekedésben elég jól kivitelezhető, viszont nem egy elterjedt kommunikációs forma. Az egyedi kivitelezés miatt ára magasabb, megbízhatósága kisebb lehet, ráadásul a teljes rendszert az üzemeltetőnek kell fenntartania.

Vezetékmentes és vezetékes hálózat

A közlekedési eszközök a mobilitás érdekében továbbra is vezetékmentes hálózatot használnának, mely kiszolgálója (WiFi Access Point) szintén a megállóba van telepítve. A megálló azonban a központtal vezetékes hálózaton van összekötve. Így kevésbé érzékeny az esetleges környezeti viszonyokra, mint pl.: időjárás. Mivel a vezetékes hálózat nagyobb sebesség biztosítására képes, ezt ki lehet használni úgy, hogy a felszálló utasok intelligens jegyének, bérletének adatai a buszon

felszálláskor lekérdezésre kerülnek, majd egy gyors központi ellenőrzés után a felszállás jóváhagyható, korlátozható, megtagadható.

A járművekről begyűjtött adatokat az alábbi tranzakció rekordokba lehet szervezni:

- időpont
- hely
- mozgás (fel vagy leszállás)
- járatazonosító

A le és felszállási tranzakció adatok alapján meghatározhatók az egyes útszakaszokra vonatkozó forgalmi adatok:

- időszak
- útszakasz
- fő
- járatazonosító

A forgalom elemzésére és előrejelzésére többek között használhatók az egyes statisztikai aggregációs formulák, mint például az

- átlag
- szórás

Minden járműről történő fel- és leszállásnál egy tranzakciórekord keletkezik az alábbi adatokkal:

- mozgásjelző (le vagy felszállás)
- dátum
- hely
- járatazonosító
- utasazonosító

A begyüjtött tranzakció adatokból előfeldolgozás után az alábbi módosított rekordok kerülnek előállításra:

Szakaszforgalom tranzakció:

- hónap
- hét napja
- óra
- szakasz

Reláció forgalom rekord:

- hónap
- hét napja
- indulás óra
- induló hely
- célhely
- járat

A feltöltött szakasz tranzakció adatkocka és a járatterv adatok alapján meghatározhatók a szűk és bő keresztmetszetek. A járatterv adatok tartalmazzák az egyes járatok indulási adatait. A járatok kapacitásainak ismeretében kihasznált kapaciás adatok megjeleníthetők az elemzésben.

Az egyes szakaszok kihasználtságainak ismeretében meghatározhatók a járatok különböző kihasználtsági adatai az egyes időszakokban.

- járatkapacitás és szakasz kapacitás viszonya
- kihasználtság alakulása az egyes időszakokban
-

Az összegyűjtött kihasználtsági adatok alapján lehet előrejelzést készíteni a várható forgalomra. Az előrejelzés alapvetően a korábban mért adatok alapján történik. Az előrejelzés mellett lehetőség van a járatok ütemezésének optimalizálására is. A járatok forgalmi adatainak ismeretében optimálási feladatként jelennek meg az alábbi problémák:

- a legjobb, optimális kihasználtságot megadó járat ütemterv (a módszernél előbb a meglévő forgalmi adatok alapján az utasokat egy intervallumhoz rendeljük, ahol az összerendelés egy adott eloszlás alapján megy végbe)
- menetidő jobb, pontosabb meghatározása
- működési költségek (üzemanyag, munkabér,..) optimalizálása